



Firebird Referenz Dokumentation

Copyright © 2007, 2008, 2009, 2010, 2011 : Alle Beiträge unterliegen dem Urheberrecht ihrer jeweiligen Autoren, sofern nicht anders angegeben.

Übersicht

Dies ist das Inhaltsverzeichnis der deutschen Firebird Referenz. Für den vollständigen Firebird Dokumentationsindex, besuchen Sie bitte:

<http://www.firebirdsql.org/en/documentation/>

Die Homepage des Firebird Dokumentations-Unterprojektes finden Sie hier

<http://www.firebirdsql.org/en/development-documentation/>

Inhaltsverzeichnis

| | |
|--|----|
| Firebird 2.5 Sprachreferenz | 1 |
| 1. Über die Firebird SQL Sprachreferenz | 1 |
| Thema | 1 |
| Urheberschaft | 1 |
| Aktualisierungen der Sprachreferenz | 1 |
| Die Geburt des Bigbooks | 2 |
| Mitwirkende | 2 |
| Anmerkungen | 3 |
| 2. SQL Sprachstruktur | 5 |
| Hintergrund zu Firebirds SQL-Sprache | 5 |
| SQL Bestandteile | 5 |
| SQL-Dialekte | 6 |
| Fehlerbedingungen | 7 |
| Grundelemente: Statements, Klauseln, Schlüsselwörter | 7 |
| Bezeichner | 8 |
| Literele | 9 |
| Operatoren und Sonderzeichen | 9 |
| Kommentare | 10 |
| 3. Datentypen und Unterdatentypen | 12 |
| Integer-Datentypen | 14 |
| SMALLINT | 14 |
| INTEGER | 14 |
| BIGINT | 15 |
| Fließkomma-Datentypen | 15 |
| FLOAT | 16 |
| DOUBLE PRECISION | 16 |
| Festkomma-Datentypen | 16 |
| NUMERIC | 17 |
| DECIMAL | 17 |
| Datentypen für Datum und Zeit | 18 |
| DATE | 19 |
| TIME | 19 |
| TIMESTAMP | 19 |
| Operationen, die Datums- und Zeitwerte verwenden | 19 |
| Zeichendatentypen | 21 |
| Unicode | 21 |
| Client-Zeichensatz | 21 |
| Spezielle Zeichensätze | 21 |
| COLLATION | 22 |
| Zeichenindizes | 23 |
| Zeichendatentypen im Detail | 24 |
| Binärdatentypen | 25 |
| BLOB Untertypen | 26 |
| BLOB Specifics | 26 |
| ARRAY Type | 27 |
| Spezialdatentypen | 28 |
| SQL_NULL-Datentyp | 28 |
| Datentyp-Konvertierungen | 30 |
| Explizite Datentyp-Konvertierung | 30 |
| Implizite Datentyp-Konvertierung | 34 |

| | |
|---|-----|
| Benutzerdefinierte Datentypen—Domains | 35 |
| Domain-Eigenschaften | 35 |
| Domain-Überschreibung | 36 |
| Domains erstellen und verwalten | 36 |
| 4. Allgemeine Sprachelemente | 39 |
| Ausdrücke | 39 |
| Konstanten | 40 |
| SQL-Operatoren | 43 |
| Bedingte Ausdrücke | 46 |
| NULL in Ausdrücken | 48 |
| Unterabfragen | 49 |
| Prädikate | 51 |
| Behauptungen | 51 |
| Vergleichs-Prädikate | 51 |
| Existenzprädikate | 62 |
| Quantifizierte Unterabfrage-Prädikate | 66 |
| 5. Statements der Data Definition (DDL) | 68 |
| DATABASE | 68 |
| CREATE DATABASE | 68 |
| ALTER DATABASE | 72 |
| DROP DATABASE | 75 |
| SHADOW | 76 |
| CREATE SHADOW | 76 |
| DROP SHADOW | 78 |
| DOMAIN | 78 |
| CREATE DOMAIN | 79 |
| ALTER DOMAIN | 83 |
| DROP DOMAIN | 87 |
| TABLE | 87 |
| CREATE TABLE | 88 |
| ALTER TABLE | 101 |
| DROP TABLE | 108 |
| RECREATE TABLE | 109 |
| INDEX | 110 |
| CREATE INDEX | 110 |
| ALTER INDEX | 113 |
| DROP INDEX | 115 |
| SET STATISTICS | 116 |
| VIEW | 117 |
| CREATE VIEW | 117 |
| ALTER VIEW | 120 |
| CREATE OR ALTER VIEW | 122 |
| DROP VIEW | 123 |
| RECREATE VIEW | 123 |
| TRIGGER | 124 |
| CREATE TRIGGER | 124 |
| ALTER TRIGGER | 130 |
| CREATE OR ALTER TRIGGER | 133 |
| DROP TRIGGER | 134 |
| RECREATE TRIGGER | 134 |
| PROCEDURE | 135 |
| CREATE PROCEDURE | 135 |

| | |
|---|-----|
| ALTER PROCEDURE | 140 |
| CREATE OR ALTER PROCEDURE | 143 |
| DROP PROCEDURE | 144 |
| RECREATE PROCEDURE | 144 |
| EXTERNAL FUNCTION | 145 |
| DECLARE EXTERNAL FUNCTION | 146 |
| ALTER EXTERNAL FUNCTION | 148 |
| DROP EXTERNAL FUNCTION | 149 |
| FILTER | 150 |
| DECLARE FILTER | 150 |
| DROP FILTER | 152 |
| SEQUENCE (GENERATOR) | 153 |
| CREATE SEQUENCE | 153 |
| ALTER SEQUENCE | 154 |
| SET GENERATOR | 155 |
| DROP SEQUENCE | 156 |
| EXCEPTION | 157 |
| CREATE EXCEPTION | 157 |
| ALTER EXCEPTION | 158 |
| CREATE OR ALTER EXCEPTION | 159 |
| DROP EXCEPTION | 159 |
| RECREATE EXCEPTION | 160 |
| COLLATION | 161 |
| CREATE COLLATION | 161 |
| DROP COLLATION | 164 |
| CHARACTER SET | 165 |
| ALTER CHARACTER SET | 165 |
| ROLE | 166 |
| CREATE ROLE | 166 |
| ALTER ROLE | 167 |
| DROP ROLE | 167 |
| COMMENTS | 168 |
| COMMENT ON | 168 |
| 6. Statements der Data Manipulation Language (DML) | 171 |
| SELECT | 171 |
| FIRST, SKIP | 172 |
| Die SELECT-Spaltenliste | 174 |
| Die FROM-Klausel | 177 |
| Joins | 183 |
| Die WHERE-Klausel | 191 |
| Die GROUP BY-Klausel | 194 |
| Die PLAN-Klausel | 198 |
| UNION | 202 |
| ORDER BY | 204 |
| ROWS | 208 |
| FOR UPDATE [OF] | 210 |
| WITH LOCK | 211 |
| INTO | 214 |
| Common Table Expressions („WITH ... AS ... SELECT“) | 215 |
| INSERT | 219 |
| INSERT ... VALUES | 220 |
| INSERT ... SELECT | 221 |

| | |
|--|-----|
| INSERT ... DEFAULT VALUES | 222 |
| Die RETURNING-Klausel | 222 |
| Einfügen in BLOB-Spalten | 223 |
| UPDATE | 223 |
| Verwendung eines Alias | 224 |
| Die SET-Klausel | 225 |
| Die WHERE-Klausel | 226 |
| Die ORDER BY- und ROWS-Klauseln | 227 |
| Die RETURNING-Klausel | 228 |
| Aktualisieren von BLOB-Spalten | 229 |
| UPDATE OR INSERT | 229 |
| The RETURNING clause | 230 |
| DELETE | 230 |
| Aliases | 231 |
| WHERE | 232 |
| PLAN | 232 |
| ORDER BY und ROWS | 233 |
| RETURNING | 234 |
| MERGE | 235 |
| EXECUTE PROCEDURE | 237 |
| „Ausführbare“ gespeicherte Prozedur | 237 |
| EXECUTE BLOCK | 238 |
| Eingabe- und Ausgabeparameter | 241 |
| Statement-Terminatoren | 241 |
| 7. Prozedurale SQL-Anweisungen (PSQL) | 243 |
| Elemente der PSQL | 243 |
| DML-Anweisungen mit Parametern | 243 |
| Transaktionen | 243 |
| Modulstruktur | 244 |
| Gespeicherte Prozeduren | 246 |
| Vorteile von gespeicherten Prozeduren | 246 |
| Varianten der gespeicherten Prozeduren | 246 |
| Erstellen einer gespeicherte Prozedur | 247 |
| Anpassen einer gespeicherte Prozedur | 247 |
| Löschen einer gespeicherte Prozedur | 248 |
| Gespeicherte Funktionen (Stored Functions) | 248 |
| PSQL-Blöcke | 249 |
| Trigger | 249 |
| Reihenfolge der Ausführung | 250 |
| DML-Trigger | 250 |
| Datenbank-Trigger | 251 |
| Trigger erstellen | 251 |
| Trigger ändern | 252 |
| Trigger löschen | 253 |
| Schreiben des Body-Codes | 253 |
| Zuweisungs-Statements | 254 |
| DECLARE CURSOR | 255 |
| DECLARE VARIABLE | 257 |
| BEGIN ... END | 260 |
| IF ... THEN ... ELSE | 261 |
| WHILE ... DO | 263 |
| LEAVE | 264 |

| | |
|---|-----|
| EXIT | 266 |
| SUSPEND | 266 |
| EXECUTE STATEMENT | 267 |
| FOR SELECT | 273 |
| FOR EXECUTE STATEMENT | 276 |
| OPEN | 277 |
| FETCH | 279 |
| CLOSE | 280 |
| IN AUTONOMOUS TRANSACTION | 281 |
| POST_EVENT | 282 |
| Abfangen und Behandeln von Fehlern | 283 |
| Systemausnahmen | 283 |
| Benutzerdefinierte Ausnahmen | 283 |
| EXCEPTION | 284 |
| WHEN ... DO | 286 |
| 8. Eingebaute Funktionen und Variablen | 290 |
| Kontextvariablen | 290 |
| CURRENT_CONNECTION | 290 |
| CURRENT_DATE | 290 |
| CURRENT_ROLE | 291 |
| CURRENT_TIME | 291 |
| CURRENT_TIMESTAMP | 292 |
| CURRENT_TRANSACTION | 293 |
| CURRENT_USER | 293 |
| DELETING | 294 |
| GDSCODE | 294 |
| INSERTING | 294 |
| NEW | 295 |
| 'NOW' | 295 |
| OLD | 296 |
| ROW_COUNT | 296 |
| SQLCODE | 297 |
| SQLSTATE | 298 |
| 'TODAY' | 298 |
| 'TOMORROW' | 299 |
| UPDATING | 300 |
| 'YESTERDAY' | 300 |
| USER | 300 |
| Skalarfunktionen | 301 |
| Funktionen zum Arbeiten mit Kontextvariablen | 301 |
| Mathematische Funktionen | 304 |
| Functions for Working with Strings | 317 |
| Date and Time Functions | 333 |
| Funktionen zur Typumwandlung | 337 |
| Funktionen für Bitweise Operationen | 340 |
| Funktionen zum Arbeiten mit UUID | 344 |
| Funktionen zum Arbeiten mit Generatoren (Sequenzen) | 345 |
| Bedingte Funktionen | 346 |
| Aggregatfunktionen | 351 |
| AVG() | 351 |
| COUNT() | 352 |
| LIST() | 353 |

| | |
|--|-----|
| MAX() | 354 |
| MIN() | 355 |
| SUM() | 356 |
| 9. Transaktionskontrolle | 357 |
| Transaktions-Statements | 357 |
| SET TRANSACTION | 357 |
| COMMIT | 363 |
| ROLLBACK | 365 |
| SAVEPOINT | 366 |
| RELEASE SAVEPOINT | 367 |
| Interne Sicherungspunkte | 368 |
| Sicherungspunkte und PSQL | 368 |
| 10. Sicherheit | 370 |
| Benutzerauthentifizierung | 370 |
| Besonders privilegierte Benutzer | 370 |
| RDB\$ADMIN-Rolle | 372 |
| Administratoren | 375 |
| SQL-Anweisungen für die Benutzerverwaltung | 376 |
| SQL-Berechtigungen | 380 |
| Der Objekthinhaber | 381 |
| Statements zur Erteilung von Privilegien | 381 |
| Anweisungen zum widerrufen von Privilegien | 387 |
| Anhang A: Zusatzinformationen | 392 |
| Das Feld RDB\$VALID_BLR | 392 |
| Funktionsweise der Invalidierung | 392 |
| Ein Hinweis zur Gleichheit | 394 |
| Anhang B: Fehlercodes und Meldungen | 395 |
| SQLSTATE Fehlercodes und Beschreibungen | 395 |
| SQLCODE und GDSCODE Fehlercodes und Beschreibungen | 403 |
| Anhang C: Reservierte Wörter und Schlüsselwörter | 438 |
| Reservierte Wörter | 438 |
| Schlüsselwörter | 439 |
| Anhang D: Systemtabellen | 443 |
| RDB\$BACKUP_HISTORY | 444 |
| RDB\$CHARACTER_SETS | 445 |
| RDB\$CHECK_CONSTRAINTS | 445 |
| RDB\$COLLATIONS | 446 |
| RDB\$DATABASE | 447 |
| RDB\$DEPENDENCIES | 447 |
| RDB\$EXCEPTIONS | 448 |
| RDB\$FIELDS | 449 |
| RDB\$FIELD_DIMENSIONS | 452 |
| RDB\$FILES | 452 |
| RDB\$FILTERS | 453 |
| RDB\$FORMATS | 454 |
| RDB\$FUNCTIONS | 454 |
| RDB\$FUNCTION_ARGUMENTS | 455 |
| RDB\$GENERATORS | 457 |
| RDB\$INDICES | 457 |
| RDB\$INDEX_SEGMENTS | 458 |
| RDB\$LOG_FILES | 459 |
| RDB\$PAGES | 459 |

| | |
|---|-----|
| RDB\$PROCEDURES | 460 |
| RDB\$PROCEDURE_PARAMETERS | 461 |
| RDB\$REF_CONSTRAINTS | 462 |
| RDB\$RELATIONS | 463 |
| RDB\$RELATION_CONSTRAINTS | 464 |
| RDB\$RELATION_FIELDS | 465 |
| RDB\$ROLES | 466 |
| RDB\$SECURITY_CLASSES | 467 |
| RDB\$TRANSACTIONS | 467 |
| RDB\$TRIGGERS | 468 |
| RDB\$TRIGGER_MESSAGES | 469 |
| RDB\$TYPES | 470 |
| RDB\$USER_PRIVILEGES | 470 |
| RDB\$VIEW_RELATIONS | 471 |
| Anhang E: Monitoring Tables | 473 |
| MON\$ATTACHMENTS | 474 |
| Verwendung von MON\$ATTACHMENTS um eine Verbindung zu beenden | 475 |
| MON\$CALL_STACK | 475 |
| MON\$CONTEXT_VARIABLES | 476 |
| MON\$DATABASE | 477 |
| MON\$IO_STATS | 478 |
| MON\$MEMORY_USAGE | 479 |
| MON\$RECORD_STATS | 480 |
| MON\$STATEMENTS | 481 |
| MON\$STATEMENTS zum Stoppen einer Abfrage nutzen | 481 |
| MON\$TRANSACTIONS | 482 |
| Anhang F: Zeichensätze und Collations | 484 |
| Anhang G: Lizenzhinweise | 490 |
| Anhang H: Dokumenthistorie | 491 |



Firebird 2.5 Sprachreferenz

Release 1

Dmitry Filippov
Alexander Karpeykin
Alexey Kovyazin
Dmitry Kuzmenko
Denis Simonov
Paul Vinkenoog
Thomas Woinke
Dmitry Yemanov

11. März 2018, Dokumentversion 1.000-de - Deutsche Übersetzung

Firebird 2.5 Sprachreferenz

Release 1

11. März 2018, Dokumentversion 1.000-de - Deutsche Übersetzung
von Dmitry Filippov, Alexander Karpeykin, Alexey Kovyazin, Dmitry Kuzmenko, Denis Simonov, Paul Vinkenoog, Thomas Woinke und Dmitry Yemanov

Die Quelle des meist kopierten Referenzmaterials: Paul Vinkenoog

Weitere Quelle kopierten Referenzmaterials: Thomas Woinke

Übersetzung ins Deutsche: Martin Köditz, Mark Harris

Copyright © 2017 Das Firebird Projekt und alle beteiligten Autoren, unter der [Public Documentation License Version 1.0](#).

Bitte vergleichen Sie auch [Lizenzhinweise](#) im Anhang.

Zusammenfassung

Dieser Band ist eine Zusammenstellung von Themen rund um die Firebird SQL-Sprache, geschrieben von Mitgliedern der russischsprachigen Firebird Benutzer- und Entwickler-Gemeinschaft. Im Jahr 2014 erreichte diese Zusammenstellung ihren Höhepunkt in einem russischsprachigen Referenzhandbuch. Auf Veranlassung von Alexey Kovyazin wurde eine Kampagne unter den weltweit verteilten Firebird-Benutzern gestartet, um Mittel für eine professionelle Übersetzung ins Englische freizumachen. Hieraus sollen wiederum Übersetzungen in andere Sprachen unter der Schirmherrschaft des Firebird Dokumentations-Projektes erstellt werden.

Inhaltsverzeichnis

| | |
|--|----|
| 1. Über die Firebird SQL Sprachreferenz | 1 |
| Thema | 1 |
| Urheberschaft | 1 |
| Aktualisierungen der Sprachreferenz | 1 |
| Die Geburt des Bigbooks | 2 |
| Mitwirkende | 2 |
| Anmerkungen | 3 |
| 2. SQL Sprachstruktur | 5 |
| Hintergrund zu Firebirds SQL-Sprache | 5 |
| SQL Bestandteile | 5 |
| SQL-Dialekte | 6 |
| Fehlerbedingungen | 7 |
| Grundelemente: Statements, Klauseln, Schlüsselwörter | 7 |
| Bezeichner | 8 |
| Literele | 9 |
| Operatoren und Sonderzeichen | 9 |
| Kommentare | 10 |
| 3. Datentypen und Unterdatentypen | 12 |
| Integer-Datentypen | 14 |
| SMALLINT | 14 |
| INTEGER | 14 |
| BIGINT | 15 |
| Fließkomma-Datentypen | 15 |
| FLOAT | 16 |
| DOUBLE PRECISION | 16 |
| Festkomma-Datentypen | 16 |
| NUMERIC | 17 |
| DECIMAL | 17 |
| Datentypen für Datum und Zeit | 18 |
| DATE | 19 |
| TIME | 19 |
| TIMESTAMP | 19 |
| Operationen, die Datums- und Zeitwerte verwenden | 19 |
| Zeichendatentypen | 21 |
| Unicode | 21 |
| Client-Zeichensatz | 21 |
| Spezielle Zeichensätze | 21 |
| COLLATION | 22 |
| Zeichenindizes | 23 |
| Zeichendatentypen im Detail | 24 |
| Binärdatentypen | 25 |
| BLOB Untertypen | 26 |
| BLOB Specifics | 26 |
| ARRAY Type | 27 |
| Spezialdatentypen | 28 |
| SQL_NULL-Datentyp | 28 |
| Datentyp-Konvertierungen | 30 |
| Explizite Datentyp-Konvertierung | 30 |

| | |
|---|-----|
| Implizite Datentyp-Konvertierung | 34 |
| Benutzerdefinierte Datentypen—Domains | 35 |
| Domain-Eigenschaften | 35 |
| Domain-Überschreibung | 36 |
| Domains erstellen und verwalten | 36 |
| 4. Allgemeine Sprachelemente | 39 |
| Ausdrücke | 39 |
| Konstanten | 40 |
| SQL-Operatoren | 43 |
| Bedingte Ausdrücke | 46 |
| NULL in Ausdrücken | 48 |
| Unterabfragen | 49 |
| Prädikate | 51 |
| Behauptungen | 51 |
| Vergleichs-Prädikate | 51 |
| Existenzprädikate | 62 |
| Quantifizierte Unterabfrage-Prädikate | 66 |
| 5. Statements der Data Definition (DDL) | 68 |
| DATABASE | 68 |
| CREATE DATABASE | 68 |
| ALTER DATABASE | 72 |
| DROP DATABASE | 75 |
| SHADOW | 76 |
| CREATE SHADOW | 76 |
| DROP SHADOW | 78 |
| DOMAIN | 78 |
| CREATE DOMAIN | 79 |
| ALTER DOMAIN | 83 |
| DROP DOMAIN | 87 |
| TABLE | 87 |
| CREATE TABLE | 88 |
| ALTER TABLE | 101 |
| DROP TABLE | 108 |
| RECREATE TABLE | 109 |
| INDEX | 110 |
| CREATE INDEX | 110 |
| ALTER INDEX | 113 |
| DROP INDEX | 115 |
| SET STATISTICS | 116 |
| VIEW | 117 |
| CREATE VIEW | 117 |
| ALTER VIEW | 120 |
| CREATE OR ALTER VIEW | 122 |
| DROP VIEW | 123 |
| RECREATE VIEW | 123 |
| TRIGGER | 124 |
| CREATE TRIGGER | 124 |
| ALTER TRIGGER | 130 |
| CREATE OR ALTER TRIGGER | 133 |
| DROP TRIGGER | 134 |
| RECREATE TRIGGER | 134 |
| PROCEDURE | 135 |

| | |
|---|-----|
| CREATE PROCEDURE | 135 |
| ALTER PROCEDURE | 140 |
| CREATE OR ALTER PROCEDURE | 143 |
| DROP PROCEDURE | 144 |
| RECREATE PROCEDURE | 144 |
| EXTERNAL FUNCTION | 145 |
| DECLARE EXTERNAL FUNCTION | 146 |
| ALTER EXTERNAL FUNCTION | 148 |
| DROP EXTERNAL FUNCTION | 149 |
| FILTER | 150 |
| DECLARE FILTER | 150 |
| DROP FILTER | 152 |
| SEQUENCE (GENERATOR) | 153 |
| CREATE SEQUENCE | 153 |
| ALTER SEQUENCE | 154 |
| SET GENERATOR | 155 |
| DROP SEQUENCE | 156 |
| EXCEPTION | 157 |
| CREATE EXCEPTION | 157 |
| ALTER EXCEPTION | 158 |
| CREATE OR ALTER EXCEPTION | 159 |
| DROP EXCEPTION | 159 |
| RECREATE EXCEPTION | 160 |
| COLLATION | 161 |
| CREATE COLLATION | 161 |
| DROP COLLATION | 164 |
| CHARACTER SET | 165 |
| ALTER CHARACTER SET | 165 |
| ROLE | 166 |
| CREATE ROLE | 166 |
| ALTER ROLE | 167 |
| DROP ROLE | 167 |
| COMMENTS | 168 |
| COMMENT ON | 168 |
| 6. Statements der Data Manipulation Language (DML) | 171 |
| SELECT | 171 |
| FIRST, SKIP | 172 |
| Die SELECT-Spaltenliste | 174 |
| Die FROM-Klausel | 177 |
| Joins | 183 |
| Die WHERE-Klausel | 191 |
| Die GROUP BY-Klausel | 194 |
| Die PLAN-Klausel | 198 |
| UNION | 202 |
| ORDER BY | 204 |
| ROWS | 208 |
| FOR UPDATE [OF] | 210 |
| WITH LOCK | 211 |
| INTO | 214 |
| Common Table Expressions („WITH ... AS ... SELECT“) | 215 |
| INSERT | 219 |
| INSERT ... VALUES | 220 |

| | |
|--|-----|
| INSERT ... SELECT | 221 |
| INSERT ... DEFAULT VALUES | 222 |
| Die RETURNING-Klausel | 222 |
| Einfügen in BLOB-Spalten | 223 |
| UPDATE | 223 |
| Verwendung eines Alias | 224 |
| Die SET-Klausel | 225 |
| Die WHERE-Klausel | 226 |
| Die ORDER BY- und ROWS-Klauseln | 227 |
| Die RETURNING-Klausel | 228 |
| Aktualisieren von BLOB-Spalten | 229 |
| UPDATE OR INSERT | 229 |
| The RETURNING clause | 230 |
| DELETE | 230 |
| Aliases | 231 |
| WHERE | 232 |
| PLAN | 232 |
| ORDER BY und ROWS | 233 |
| RETURNING | 234 |
| MERGE | 235 |
| EXECUTE PROCEDURE | 237 |
| „Ausführbare“ gespeicherte Prozedur | 237 |
| EXECUTE BLOCK | 238 |
| Eingabe- und Ausgabeparameter | 241 |
| Statement-Terminatoren | 241 |
| 7. Prozedurale SQL-Anweisungen (PSQL) | 243 |
| Elemente der PSQL | 243 |
| DML-Anweisungen mit Parametern | 243 |
| Transaktionen | 243 |
| Modulstruktur | 244 |
| Gespeicherte Prozeduren | 246 |
| Vorteile von gespeicherten Prozeduren | 246 |
| Varianten der gespeicherten Prozeduren | 246 |
| Erstellen einer gespeicherte Prozedur | 247 |
| Anpassen einer gespeicherte Prozedur | 247 |
| Löschen einer gespeicherte Prozedur | 248 |
| Gespeicherte Funktionen (Stored Functions) | 248 |
| PSQL-Blöcke | 249 |
| Trigger | 249 |
| Reihenfolge der Ausführung | 250 |
| DML-Trigger | 250 |
| Datenbank-Trigger | 251 |
| Trigger erstellen | 251 |
| Trigger ändern | 252 |
| Trigger löschen | 253 |
| Schreiben des Body-Codes | 253 |
| Zuweisungs-Statements | 254 |
| DECLARE CURSOR | 255 |
| DECLARE VARIABLE | 257 |
| BEGIN ... END | 260 |
| IF ... THEN ... ELSE | 261 |
| WHILE ... DO | 263 |

| | |
|---|-----|
| LEAVE | 264 |
| EXIT | 266 |
| SUSPEND | 266 |
| EXECUTE STATEMENT | 267 |
| FOR SELECT | 273 |
| FOR EXECUTE STATEMENT | 276 |
| OPEN | 277 |
| FETCH | 279 |
| CLOSE | 280 |
| IN AUTONOMOUS TRANSACTION | 281 |
| POST_EVENT | 282 |
| Abfangen und Behandeln von Fehlern | 283 |
| Systemausnahmen | 283 |
| Benutzerdefinierte Ausnahmen | 283 |
| EXCEPTION | 284 |
| WHEN ... DO | 286 |
| 8. Eingebaute Funktionen und Variablen | 290 |
| Kontextvariablen | 290 |
| CURRENT_CONNECTION | 290 |
| CURRENT_DATE | 290 |
| CURRENT_ROLE | 291 |
| CURRENT_TIME | 291 |
| CURRENT_TIMESTAMP | 292 |
| CURRENT_TRANSACTION | 293 |
| CURRENT_USER | 293 |
| DELETING | 294 |
| GDSCODE | 294 |
| INSERTING | 294 |
| NEW | 295 |
| 'NOW' | 295 |
| OLD | 296 |
| ROW_COUNT | 296 |
| SQLCODE | 297 |
| SQLSTATE | 298 |
| 'TODAY' | 298 |
| 'TOMORROW' | 299 |
| UPDATING | 300 |
| 'YESTERDAY' | 300 |
| USER | 300 |
| Skalarfunktionen | 301 |
| Funktionen zum Arbeiten mit Kontextvariablen | 301 |
| Mathematische Funktionen | 304 |
| Functions for Working with Strings | 317 |
| Date and Time Functions | 333 |
| Funktionen zur Typumwandlung | 337 |
| Funktionen für Bitweise Operationen | 340 |
| Funktionen zum Arbeiten mit UUID | 344 |
| Funktionen zum Arbeiten mit Generatoren (Sequenzen) | 345 |
| Bedingte Funktionen | 346 |
| Aggregatfunktionen | 351 |
| AVG() | 351 |
| COUNT() | 352 |

| | |
|--|-----|
| LIST() | 353 |
| MAX() | 354 |
| MIN() | 355 |
| SUM() | 356 |
| 9. Transaktionskontrolle | 357 |
| Transaktions-Statements | 357 |
| SET TRANSACTION | 357 |
| COMMIT | 363 |
| ROLLBACK | 365 |
| SAVEPOINT | 366 |
| RELEASE SAVEPOINT | 367 |
| Interne Sicherungspunkte | 368 |
| Sicherungspunkte und PSQL | 368 |
| 10. Sicherheit | 370 |
| Benutzerauthentifizierung | 370 |
| Besonders privilegierte Benutzer | 370 |
| RDB\$ADMIN-Rolle | 372 |
| Administratoren | 375 |
| SQL-Anweisungen für die Benutzerverwaltung | 376 |
| SQL-Berechtigungen | 380 |
| Der Objekthinhaber | 381 |
| Statements zur Erteilung von Privilegien | 381 |
| Anweisungen zum widerrufen von Privilegien | 387 |
| Anhang A: Zusatzinformationen | 392 |
| Das Feld RDB\$VALID_BLR | 392 |
| Funktionsweise der Invalidierung | 392 |
| Ein Hinweis zur Gleichheit | 394 |
| Anhang B: Fehlercodes und Meldungen | 395 |
| SQLSTATE Fehlercodes und Beschreibungen | 395 |
| SQLCODE und GDSCODE Fehlercodes und Beschreibungen | 403 |
| Anhang C: Reservierte Wörter und Schlüsselwörter | 438 |
| Reservierte Wörter | 438 |
| Schlüsselwörter | 439 |
| Anhang D: Systemtabellen | 443 |
| RDB\$BACKUP_HISTORY | 444 |
| RDB\$CHARACTER_SETS | 445 |
| RDB\$CHECK_CONSTRAINTS | 445 |
| RDB\$COLLATIONS | 446 |
| RDB\$DATABASE | 447 |
| RDB\$DEPENDENCIES | 447 |
| RDB\$EXCEPTIONS | 448 |
| RDB\$FIELDS | 449 |
| RDB\$FIELD_DIMENSIONS | 452 |
| RDB\$FILES | 452 |
| RDB\$FILTERS | 453 |
| RDB\$FORMATS | 454 |
| RDB\$FUNCTIONS | 454 |
| RDB\$FUNCTION_ARGUMENTS | 455 |
| RDB\$GENERATORS | 457 |
| RDB\$INDICES | 457 |
| RDB\$INDEX_SEGMENTS | 458 |
| RDB\$LOG_FILES | 459 |

| | |
|---|-----|
| RDB\$PAGES | 459 |
| RDB\$PROCEDURES | 460 |
| RDB\$PROCEDURE_PARAMETERS | 461 |
| RDB\$REF_CONSTRAINTS | 462 |
| RDB\$RELATIONS | 463 |
| RDB\$RELATION_CONSTRAINTS | 464 |
| RDB\$RELATION_FIELDS | 465 |
| RDB\$ROLES | 466 |
| RDB\$SECURITY_CLASSES | 467 |
| RDB\$TRANSACTIONS | 467 |
| RDB\$TRIGGERS | 468 |
| RDB\$TRIGGER_MESSAGES | 469 |
| RDB\$TYPES | 470 |
| RDB\$USER_PRIVILEGES | 470 |
| RDB\$VIEW_RELATIONS | 471 |
| Anhang E: Monitoring Tables | 473 |
| MON\$ATTACHMENTS | 474 |
| Verwendung von MON\$ATTACHMENTS um eine Verbindung zu beenden | 475 |
| MON\$CALL_STACK | 475 |
| MON\$CONTEXT_VARIABLES | 476 |
| MON\$DATABASE | 477 |
| MON\$IO_STATS | 478 |
| MON\$MEMORY_USAGE | 479 |
| MON\$RECORD_STATS | 480 |
| MON\$STATEMENTS | 481 |
| MON\$STATEMENTS zum Stoppen einer Abfrage nutzen | 481 |
| MON\$TRANSACTIONS | 482 |
| Anhang F: Zeichensätze und Collations | 484 |
| Anhang G: Lizenzhinweise | 490 |
| Anhang H: Dokumenthistorie | 491 |

Tabellenverzeichnis

| | |
|---|-----|
| 3.1. Übersicht der Datentypen | 12 |
| 3.2. Verfahren zum Speichern von reellen Zahlen | 17 |
| 3.3. Arithmetische Operationen für Datums- und Zeitdatentypen | 20 |
| 3.4. Collation für den Zeichensatz UTF8 | 23 |
| 3.5. Maximale Indexlänge nach Seitengröße und Zeichengröße | 24 |
| 3.6. Konvertierungen mit CAST | 31 |
| 3.7. Formatierungsargumente für Datum- und Zeit-LiteraleDate and Time Literal | 32 |
| 3.8. Literale mit vordefinierten Datums- und Zeitwerten | 32 |
| 3.9. Regeln zum Überschreiben von Domain-Eigenschaften in Spaltendefinitionen | 36 |
| 4.1. Beschreibung der Ausdruck-Elemente | 39 |
| 4.2. Vorrang der Operatortypen | 43 |
| 4.3. Vorrang arithmetischer Operatoren | 44 |
| 4.4. Prioritäten der Vergleichsoperatoren | 45 |
| 4.5. Prioritäten logischer Operatoren | 45 |
| 5.1. CREATE DATABASE Statement-Parameter | 69 |
| 5.2. ALTER DATABASE Statement-Parameter | 73 |
| 5.3. CREATE SHADOW Statement-Parameter | 76 |
| 5.4. DROP SHADOW Statement-Parameter | 78 |
| 5.5. CREATE DOMAIN Statement-Parameter | 80 |
| 5.6. ALTER DOMAIN Statement-Parameter | 84 |
| 5.7. CREATE TABLE Statement-Parameter | 89 |
| 5.8. ALTER TABLE Statement-Parameter | 103 |
| 5.9. DROP TABLE Statement-Parameter | 109 |
| 5.10. CREATE INDEX Statement-Parameter | 110 |
| 5.11. Maximale Anzahl Indizes pro Tabelle | 112 |
| 5.12. Maximale indizierbare (VAR)CHAR-Länge | 112 |
| 5.13. ALTER INDEX Statement-Parameter | 114 |
| 5.14. DROP INDEX Statement-Parameter | 115 |
| 5.15. SET STATISTICS Statement-Parameter | 116 |
| 5.16. CREATE VIEW Statement-Parameter | 117 |
| 5.17. ALTER VIEW Statement-Parameter | 121 |
| 5.18. CREATE OR ALTER VIEW Statement-Parameter | 122 |
| 5.19. DROP VIEW Statement-Parameter | 123 |
| 5.20. RECREATE VIEW Statement-Parameter | 123 |
| 5.21. CREATE TRIGGER Statement-Parameter | 125 |
| 5.22. ALTER TRIGGER Statement-Parameter | 131 |
| 5.23. DROP TRIGGER Statement-Parameter | 134 |
| 5.24. CREATE PROCEDURE Statement-Parameter | 136 |
| 5.25. ALTER PROCEDURE Statement-Parameter | 141 |
| 5.26. DROP PROCEDURE Statement-Parameter | 144 |
| 5.27. DECLARE EXTERNAL FUNCTION Statement-Parameter | 146 |
| 5.28. ALTER EXTERNAL FUNCTION Statement-Parameter | 149 |
| 5.29. DROP EXTERNAL FUNCTION Statement-Parameter | 150 |
| 5.30. DECLARE FILTER Statement-Parameter | 151 |
| 5.31. DROP FILTER Statement-Parameter | 152 |
| 5.32. CREATE SEQUENCE CREATE GENERATOR Statement-Parameter | 154 |
| 5.33. ALTER SEQUENCE Statement-Parameter | 154 |
| 5.34. SET GENERATOR Statement-Parameter | 155 |
| 5.35. DROP SEQUENCE DROP GENERATOR Statement-Parameter | 156 |
| 5.36. CREATE EXCEPTION Statement-Parameter | 157 |

| | |
|---|-----|
| 5.37. ALTER EXCEPTION Statement-Parameter | 158 |
| 5.38. CREATE OR ALTER EXCEPTION Statement-Parameter | 159 |
| 5.39. DROP EXCEPTION Statement-Parameter | 160 |
| 5.40. RECREATE EXCEPTION Statement-Parameter | 161 |
| 5.41. CREATE COLLATION Statement-Parameter | 162 |
| 5.42. Spezifische Collations-Attribute | 162 |
| 5.43. DROP COLLATION Statement-Parameter | 165 |
| 5.44. ALTER CHARACTER SET Statement-Parameter | 165 |
| 5.45. CREATE ROLE Statement-Parameter | 167 |
| 5.46. COMMENT ON Statement-Parameter | 169 |
| 6.1. Argumente für die FIRST- und SKIP-Klauseln | 172 |
| 6.2. Argumente der SELECT-Spaltenliste | 174 |
| 6.3. Argumente der FROM-Klausel | 178 |
| 6.4. Argumente für JOIN-Klauseln | 184 |
| 6.5. Argumente der GROUP BY-Klausel | 194 |
| 6.6. Argumente der PLAN-Klausel | 199 |
| 6.7. Argumente für die ORDER BY-Klausel | 204 |
| 6.8. Argumente für die ROWS-Klausel | 208 |
| 6.9. Wie TPB-Einstellungen das explizite Sperren beeinflussen | 212 |
| 6.10. Argumente für Common Table Expressions | 215 |
| 6.11. Argumente für die Parameter des INSERT-Statements | 219 |
| 6.12. Arguments for the UPDATE Statement Parameters | 224 |
| 6.13. Arguments for the UPDATE OR INSERT Statement Parameters | 229 |
| 6.14. Argumente der DELETE-Statement-Parameter | 231 |
| 6.15. Argumente der MERGE-Statement-Parameter | 235 |
| 6.16. Argument der EXECUTE PROCEDURE-Statement-Parameter | 237 |
| 6.17. Argument der EXECUTE BLOCK-Statement-Parameter | 239 |
| 7.1. SET TERM-Parameter | 245 |
| 7.2. PSQL Block Parameters | 249 |
| 7.3. Zuweisungs-Statement-Parameter | 254 |
| 7.4. DECLARE CURSOR-Statement-Parameter | 255 |
| 7.5. DECLARE VARIABLE-Statement-Parameter | 258 |
| 7.6. IF ... THEN ... ELSE Parameters | 262 |
| 7.7. WHILE ... DO Parameters | 263 |
| 7.8. LEAVE-Statement-Parameter | 264 |
| 7.9. EXECUTE STATEMENT-Statement-Parameter | 268 |
| 7.10. FOR SELECT-Statement-Parameter | 273 |
| 7.11. FOR EXECUTE STATEMENT-Statement-Parameter | 276 |
| 7.12. OPEN Statement Parameter | 277 |
| 7.13. FETCH-Statement-Parameter | 279 |
| 7.14. CLOSE-Statement-Parameter | 280 |
| 7.15. IN AUTONOMOUS TRANSACTION Statement Parameter | 281 |
| 7.16. POST_EVENT Statement Parameter | 282 |
| 7.17. EXCEPTION-Statement-Parameter | 284 |
| 7.18. WHEN ... DO-Statement-Parameter | 287 |
| 8.1. CURRENT_TIME Parameter | 292 |
| 8.2. CURRENT_TIMESTAMP Parameter | 292 |
| 8.3. RDB\$GET_CONTEXT-Funktionsparameters | 302 |
| 8.4. Kontextvariablen im SYSTEM-Namespace | 302 |
| 8.5. RDB\$SET_CONTEXT-Funktionsparameter | 303 |
| 8.6. ABS Funktionsparameter | 304 |
| 8.7. ACOS Funktionsparameter | 305 |

| | |
|---|-----|
| 8.8. ASIN Funktionsparameter | 305 |
| 8.9. ATAN Funktionsparameter | 306 |
| 8.10. ATAN2-Funktionsparameter | 306 |
| 8.11. CEIL[ING]-Funktionsparameter | 307 |
| 8.12. COS Funktionsparameter | 307 |
| 8.13. COSH Funktionsparameter | 308 |
| 8.14. COT Funktionsparameter | 308 |
| 8.15. EXP Funktionsparameter | 308 |
| 8.16. FLOOR Funktionsparameter | 309 |
| 8.17. LN Funktionsparameter | 309 |
| 8.18. LOG-Funktionsparameter | 310 |
| 8.19. LOG10 Funktionsparameter | 310 |
| 8.20. MOD-Funktionsparameter | 311 |
| 8.21. POWER-Funktionsparameter | 312 |
| 8.22. ROUND-Funktionsparameter | 312 |
| 8.23. SIGN Funktionsparameter | 313 |
| 8.24. SIN Funktionsparameter | 314 |
| 8.25. SINH Funktionsparameter | 314 |
| 8.26. SQRT Funktionsparameter | 315 |
| 8.27. TAN Funktionsparameter | 315 |
| 8.28. TANH-Funktionsparameter | 315 |
| 8.29. TRUNC-Funktionsparameter | 316 |
| 8.30. ASCII_CHAR Funktionsparameter | 317 |
| 8.31. ASCII_VAL Funktionsparameter | 317 |
| 8.32. BIT_LENGTH Funktionsparameter | 318 |
| 8.33. CHAR[ACTER]_LENGTH Funktionsparameter | 319 |
| 8.34. HASH Funktionsparameter | 320 |
| 8.35. LEFT-Funktionsparameter | 320 |
| 8.36. LOWER FunktionsparameterS | 321 |
| 8.37. LPAD-Funktionsparameter | 322 |
| 8.38. OCTET_LENGTH Funktionsparameter | 323 |
| 8.39. OVERLAY-Funktionsparameter | 324 |
| 8.40. POSITION-Funktionsparameter | 325 |
| 8.41. REPLACE-Funktionsparameter | 326 |
| 8.42. REVERSE Funktionsparameter | 327 |
| 8.43. RIGHT-Funktionsparameter | 328 |
| 8.44. RPAD-Funktionsparameter | 329 |
| 8.45. SUBSTRING-Funktionsparameter | 330 |
| 8.46. TRIM-Funktionsparameter | 331 |
| 8.47. UPPER Funktionsparameter | 332 |
| 8.48. DATEADD-Funktionsparameter | 333 |
| 8.49. DATEDIFF-Funktionsparameter | 334 |
| 8.50. EXTRACT-Funktionsparameter | 335 |
| 8.51. Typen und Bereiche der EXTRACT-Ergebnisse | 336 |
| 8.52. CAST-Funktionsparameter | 337 |
| 8.53. Mögliche Typkonvertierungen mit CAST | 338 |
| 8.54. BIN_AND-Funktionsparameter | 341 |
| 8.55. BIN_NOT Funktionsparameter | 341 |
| 8.56. BIN_OR-Funktionsparameter | 342 |
| 8.57. BIN_SHL-Funktionsparameter | 342 |
| 8.58. BIN_SHR-Funktionsparameter | 343 |
| 8.59. BIN_XOR-Funktionsparameter | 343 |

| | |
|--|-----|
| 8.60. CHAR_TO_UUID Funktionsparameter | 344 |
| 8.61. UUID_TO_CHAR-Funktionsparameter | 345 |
| 8.62. GEN_ID-Funktionsparameter | 346 |
| 8.63. COALESCE-Funktionsparameter | 347 |
| 8.64. DECODE-Funktionsparameter | 347 |
| 8.65. IIF-Funktionsparameter | 348 |
| 8.66. MAXVALUE-Funktionsparameter | 349 |
| 8.67. MINVALUE-Funktionsparameter | 350 |
| 8.68. NULLIF-Funktionsparameter | 350 |
| 8.69. AVG-Funktionsparameter | 351 |
| 8.70. COUNT-Funktionsparameter | 352 |
| 8.71. LIST-Funktionsparameter | 353 |
| 8.72. MAX-Funktionsparameter | 354 |
| 8.73. MIN-Funktionsparameter | 355 |
| 8.74. SUM-Funktionsparameter | 356 |
| 9.1. SET TRANSACTION-Statement-Parameter | 358 |
| 9.2. Kompatibilität der Zugriffsoptionen für RESERVING | 362 |
| 9.3. COMMIT Statement Parameter | 364 |
| 9.4. ROLLBACK Statement Parameters | 365 |
| 9.5. SAVEPOINT Statement Parameter | 367 |
| 9.6. RELEASE SAVEPOINT-Statement-Parameter | 368 |
| 10.1. Parameter für die RDB\$ADMIN Rollen GRANT und REVOKE | 373 |
| 10.2. Administrator („Superuser“) Eigenschaften | 375 |
| 10.3. CREATE USER Statement Parameter | 377 |
| 10.4. ALTER USER Statement Parameter | 378 |
| 10.5. DROP USER Statement Parameter | 380 |
| 10.6. GRANT Statement Parameter | 382 |
| 10.7. Liste der Berechtigungen auf Tabellen | 385 |
| 10.8. REVOKE Anweisung Parameter | 388 |
| B.1. SQLSTATE Fehlercodes und Meldungen | 395 |
| B.2. SQLCODE und GDSCODE Fehlercodes und Meldungen (1) | 403 |
| B.3. SQLCODE und GDSCODE Fehlercodes und Meldungen (2) | 412 |
| B.4. SQLCODE und GDSCODE Fehlercodes und Meldungen (3) | 421 |
| B.5. SQLCODE und GDSCODE Fehlercodes und Meldungen (4) | 428 |
| D.1. Liste der Systemtabellen | 443 |
| E.1. Liste der Monitoringtabellen | 473 |
| F.1. Zeichensätze und Collations | 484 |

Kapitel 1

Über die Firebird SQL Sprachreferenz

für Firebird 2.5

Diese Firebird SQL Sprachreferenz ist das erste zusammenhängende Dokument, das alle Aspekte der Abfragesprache betrachtet, die von Entwicklern für die Kommunikation ihrer Anwendungen mit dem Firebird Relation Database Management System benötigt werden. Diese Sprachreferenz hat eine lange Vorgeschichte.

Thema

Das Kernthema dieses Dokuments ist die vollständige Implementierung der SQL-Abfragesprache. Firebird entspricht weitestgehend den internationalen Standards für SQL in Bezug auf Datentypen, Datenspeicherstrukturen, Mechanismen zur referentiellen Integrität sowie Fähigkeiten zur Datenmanipulation und Zugriffsrechte. Firebird beinhaltet außerdem eine robuste Prozedursprache —procedural SQL (PSQL)— für gespeicherte Prozeduren, Trigger und dynamisch ausführbare Code Blöcke. Dies sind die Themen, die in diesem Dokument behandelt werden.

Urheberschaft

Das Material für die Zusammenstellung dieser Sprachreferenz wurde durch die Open Source Community der Firebird-Entwickler über die letzten 15 Jahre zusammengetragen. Die Übergabe der Interbase 6 Open Source-Codebasis im Juli 2000 durch das (damaligen) Inprise / Borland-Konglomerat war herzlich willkommen. Allerdings kam diese Übergabe ohne Rechte zu bestehenden Dokumentationen. Seit dem Fork der Code-Basis durch die Besitzer für private und kommerzielle Zwecke, wurde klar, dass die nichtkommerzielle Firebird Open Source-Community niemals Nutzungsrechte hierfür erhalten würde.

Die zwei wichtigen Bücher des InterBase 6-Sammlung waren das *Data Definition Guide* und die *Language Reference*. Während ersteres die Untermenge der Data Definition Language (DDL) abdeckte, war der Rest im zweiten Buch enthalten. Glücklicherweise war es jahrelang für Firebird-Benutzer möglich, die PDF-Handbücher online zu finden.

Aktualisierungen der Sprachreferenz

Das *Data Definition Guide*, welches die Erstellung und Wartung der Metadaten für Datenbanken behandelt, war über Jahre hinweg „gut genug“: Die Data Definition Language eines DBMS ist üblicherweise stabil und wächst

nur langsam im Vergleich zur Data Manipulation Language (DML), die der Abfrage dient und die PSQL für Serverbasierte Programmierung nutzt.

Der Vorsitzende des Firebird Dokumentations-Teams, Paul Vinkenoog, nahm sich der Aufgabe an, die große Masse an Verbesserungen und Neuerungen in der DML und PSQL im Rahmen der Veröffentlichungen zu dokumentieren. Paul war persönlich verantwortlich für das Zusammentragen sowie Zusammenstellen und, zum Großteil, Schreiben der kumulativen Reihe der „Language Reference Updates“—eine für jede Hauptversion seit v.1.5.

Die Geburt des Bigbooks

Ab etwa 2010 arbeitete Paul, mit Firebird Projektleitung Dmitry Yemanov und einem Dokumentationskollegen Thomas Woinke, an der Planung und dem Aufbau einer kompletten SQL-Sprachreferenz für Firebird. Sie begannen damit auf Basis der LangRef-Updates, die voluminös ist. Es würde eine große Aufgabe sein und für alle Beteiligten ein Freizeitersatz.

Dann, in 2013-4, zeigten zwei wohltätige Unternehmen — MICEX und IBSurgeon — erbarmen und beschäftigten drei Autoren, die sich der liegeengebliebenen Arbeit annahmen und die Firebird 2.5 Language Reference in russisch veröffentlichten. Sie schrieben den Großteil der fehlenden DDL-Abschnitte und erstellten, übersetzten oder verwendeten DML- und PSQL-Material aus den LangRef Updates, russischsprachigen Support-Forums, Firebird Release Notes, README-Dateien und anderen Quellen. Ende 2014 war die Aufgabe weitestgehend erfüllt, in Form eines Microsoft Word-Dokumentes.

Übersetzung . . .

Die russischen Sponsoren erkannten, dass ihre Anstrengungen mit der weltweiten Firebird Community geteilt werden mussten, und fragten einige Projektmitglieder eine Crowd-Funding-Kampagne zu starten, welche den russischen Text professionell ins Englische übersetzen sollte. Dieser übersetzte Text würde dann bearbeitet und in das projektkonforme DocBook-Format übertragen sowie als Zusatz in die offene Dokumentationsbibliothek der Firebird-Projektes aufgenommen werden. Hier wären die Quellen für weitere Übersetzungen in andere Sprachen verfügbar.

Die Kampagne zum Einsammeln der Gelder startete Ende 2014 und war erfolgreich. Im Juni, 2015, begann der professionelle Übersetzer Dmitry Borodin mit der Übersetzung des russischen Textes. Durch ihn gelangte der Text zur Überarbeitung und DocBook-Konvertierung an Helen Borrie—und hier ist *Die Firebird SQL Sprachreferenz für V.2.5*, von ...allen!

. . . und noch mehr Übersetzungen

Nach dem Erscheinen der DocBook-Quellen im CVS, werden unsere vertrauensvollen Übersetzer hoffentlich mit der Arbeit an deutschen, japanischen, italienischen, französischen, portugiesischen, spanischen und tschechischen Übersetzungen beginnen. Gewiss ist, dass wir nie genug Übersetzer haben, so bitten wir Sie, die Firebirder, welche der englischen Sprache mächtig sind, über die Übersetzungsarbeiten in Ihre Muttersprache nachzudenken. Auch die Übersetzung von Abschnitten ist hilfreich.

Mitwirkende

Direkter Inhalt

- Dmitry Filippov (writer)
- Alexander Karpeykin (writer)
- Alexey Kovyazin (writer, editor)
- Dmitry Kuzmenko (writer, editor)
- Denis Simonov (writer, editor, coordinator)
- Paul Vinkenoog (writer, designer)
- Thomas Woinke (writer)
- Dmitry Yemanov (writer)

Ressourcen Inhalt

- Adriano dos Santos Fernandes
- Alexander Peshkov
- Vladyslav Khorsun
- Claudio Valderrama
- Helen Borrie
- and others

Übersetzungen

- Dmitry Borodin, megaTranslations.ru
- Martin Köditz, it & synergy GmbH

Bearbeitung und Konvertierung des englischen Textes

- Helen Borrie

Anmerkungen

Die erste komplette Sprachreferenz wäre ohne die Finanzierung nicht möglich gewesen. Wir danken allen Beteiligten für Ihren Beitrag.

Sponsoren und andere Spender

Sponsoren der russischen Sprachreferenz

[Moscow Exchange](#) (Russland)

Moscow Exchange ist die größte Börse in Russland sowie Osteuropa, gegründet am 19 Dezember 2011, durch den Zusammenschluss der Börsengruppen MICEX (gegründet 1992) und RTS (gegründet 1995). Moscow Exchange zählt zu den 20 weltweit führenden Börsen im Handel von Anleihen und Aktien, als zu den 10 größten Börsenplattformen für Handelsderivate.

[IBSurgeon \(ibase.ru\)](#) (Russland)

Technischer Support und Entwickler administrativer Tools für das Firebird DBMS.

Sponsoren des Übersetzungsprojektes

[Syntess Software BV](#) (Niederlande)

[Mitaro Business Solutions](#) (Liechtenstein)

Weitere Spender

Folgend sind die Namen der Unternehmen und Personen aufgelistet, die Barmittel für die Kosten der Übersetzung ins Englische, Weiterverarbeitung des rohen übersetzten Textes und die Konvertierung des Ganzen in das DocBook 4-Format des Firebird-Projektes freigemacht haben.

| | |
|--|--|
| Integrity Software Design, Inc. (U.S.A.) | dimari GmbH (Deutschland) |
| beta Eigenheim GmbH (Deutschland) | KIMData GmbH (Deutschland) |
| Jason Wharton (U.S.A) | Trans-X (Schweden) |
| Sanchez Balcewich (Uruguay) | Cointec Ingenieros y Consultores, S.L. (Spanien) |
| Aage Johansen (Norwegen) | Mattic Software (Niederlande) |
| André Knappstein (Deutschland) | Paul F. McGuire (U.S.A.) |
| Marcus Marques da Rocha (Brasilien) | Martin Kerkhoff |
| Thomas Vedel (Dänemark) | Bulhan Bulhan (Australien) |
| Alexandre Benson Smith (Brasilien) | Guillermo Nabrink (Brasilien) |
| Pierre Voirin (Frankreich) | Heiko Tappe (Deutschland) |
| Doug Chamberlain (U.S.A.) | Craig Cox (U.S.A.) |
| OMNet, Inc. (Schweiz) | Alfred Steller (Deutschland) |
| Konrad Butz (Deutschland) | Thomas Smekal (Kanada) |
| Carlos H. Cantu (Brasilien) | XTRALOG SARL (Frankreich) |
| Laszlo Urmenyi (Brasilien) | Fernando Pimenta (Brasilien) |
| Rudolf Grauberger (Deutschland) | Thomas Steinmaurer (Austria) |
| Rene Lobsiger (Schweiz) | Hian Pin Tjioe |
| Xavier Codina | Mick Arundell (Australien) |
| Russell Belding (Neuseeland) | Anticlei Scheid (Brasilien) |
| Luca Minuti (Italien) | Mark Rotteveel (Niederlande) |
| Chris Mathews (U.S.A.) | Hannes Streicher (Deutschland) |
| Wolfgang Lemmermeyer (Deutschland) | Paolo Sciarrini (Italien) |
| Acosta Belzusarri | Daniel Motos Guerra |
| Alberto Alfonso Luna | Simeon Bodurov |
| Cees Meijer | Robert Nixon |
| Olivier Dehorter (Frankreich) | András Omacht (Ungarn) |
| Web Express | Sergio Conzalez |
| Marc Bleuwart | Gabor Boros |
| Shaymon Gracia Campos | Cserna Zsombor (Ungarn) |
| David Keith | Uwe Gerold |
| Daniele Teti (Italien) | Pedro Pablo Busto Criado |
| Istvan Szabo | Spiridon Pavlovic |
| J. L. Garcia Naranjo | A. Morales Morales |
| Helen Cullen (Neuseeland) | Francisco Ibarra Ozuna |

SQL Sprachstruktur

Diese Referenz beschreibt die von Firebird unterstützte SQL-Sprache.

Hintergrund zu Firebirds SQL-Sprache

Zu Beginn, ein paar Punkte über die Eigenschaften die im Hintergrund von Firebirds Sprache eine Rolle spielen.

SQL Bestandteile

Verschiedene *Teilmengen von SQL* gehören wiederum in verschiedene Aktivitätsbereiche. Die Teilmengen in Firebirds Sprachimplementation sind:

- Dynamic SQL (DSQL)
- Procedural SQL (PSQL)
- Embedded SQL (ESQL)
- Interactive SQL (ISQL)

Dynamic SQL macht den Hauptteil der Sprache aus, der in Abschnitt (SQL/Foundation) 2 der SQL-Spezifikation beschrieben wird. DSQL repräsentiert Statements, die von Anwendungen über die Firebird API durch die Datenbank verarbeitet werden.

Procedural SQL ergänzt Anweisungen der Dynamic SQL um lokale Variablen, Zuweisungen, Bedingungen, Schleifen und andere prozedurale Verhalten. PSQL entspricht dem 4. Teil (SQL/PSM) der SQL-Spezifikationen. Ursprünglich waren PSQL-Erweiterungen nur über persistent gespeicherte Module (Prozeduren und Trigger) verfügbar. In späteren Releases wurden diese jedoch auch in Dynamic SQL aufgenommen (vergleichen Sie hierzu auch [EXECUTE BLOCK](#)).

Embedded SQL beschreibt die Untermenge von DSQL, die von Firebird *gpre* unterstützt wird. Dies ist die Anwendung, welche es erlaubt, SQL-Konstrukte in Ihre Host-Programmiersprache (C, C++, Pascal, Cobol, etc.) einzubetten und diese in gültigen Firebird API-Aufrufen auszuführen.

Anmerkung

Nur ein Teil der in DSQL implementierten Anweisungen und Ausdrücke werden in ESQL unterstützt.

Interactive ISQL wird durch die Sprache beschrieben, die mittels Firebirds *isql* ausgeführt werden kann. Dies ist die Befehlszeilenanwendung, für den interaktiven Zugriff auf Datenbanken. Da dies eine reguläre Client-Anwendung ist, ist ihre native Sprache in DSQL verfasst. Sie nutzt außerdem einige zusätzliche Befehle, die nicht außerhalb ihrer spezifischen Umgebung gelten.

Sowohl DSQL wie auch PSQL werden vollständig in dieser Referenz behandelt. Dies gilt nicht für ESQL und ISQL, sofern nicht ausdrücklich beschrieben.

SQL-Dialekte

Der Begriff *SQL dialect* beschreibt ein spezifisches Feature der SQL-Sprache, das bei Zugriff einer Datenbank zur Verfügung steht. SQL-Dialekte können auf Datenbankebene definiert und auf Verbindungsebene spezifiziert werden. Drei Dialekte stehen zur Verfügung:

- *Dialect 1* dient ausschließlich der Abwärtskompatibilität mit sehr alten InterBase-Datenbanken bis Version 5. Dialekt 1-Datenbanken beinhalten einige Features, die sich von Dialekt 3, dem Firebird-Standard, unterscheiden.
 - Datums- und Zeitinformationen werden als DATE-Datentyp gespeichert. Ein TIMESTAMP-Datentyp ist ebenfalls verfügbar, der identisch mit dieser DATE-Implementierung ist.
 - Doppelte Anführungszeichen dürfen als Alternative für das Apostroph als Textbegrenzer verwendet werden. Dies ist gegensätzlich zum SQL-Standard—doppelte Anführungszeichen sind für den einen bestimmten Zweck sowohl in Standard SQL wie auch in Dialekt 3 reserviert. Als Textbegrenzer sollten doppelte Anführungszeichen demnach energisch vermieden werden.
 - Die Präzision für NUMERIC- und DECIMAL-Datentypen ist geringer als im Dialekt 3 und falls die Präzision einer Dezimalzahl größer als 9 Stellen sein soll, wird Firebird diese intern als Fließkommazahl speichern.
 - Der Datentyp BIGINT (64-Bit Integer) wird nicht unterstützt.
 - Bezeichner unterscheiden Groß- und Kleinschreibung und müssen immer den Regeln für Bezeichner entsprechen—vergleichen Sie den Abschnitt [Bezeichner](#).
 - Obwohl Generator-Werte als 64-Bit-Zahlen gespeichert werden, wird ein Clientaufruf von zum Beispiel `SELECT GEN_ID (MyGen, 1)` immer einen 32-Bit-Wert zurückgeben.
- *Dialect 2* ist nur in der Firebird-Clientverbindung verfügbar und kann nicht in der Datenbank definiert werden. Hintergrund ist, Entwickler beim Debugging zu unterstützen, wenn eine reguläre Datenbank von Dialekt 1 zu Dialekt 3 migriert werden soll.
- In *Dialect 3*-Datenbanken,
 - werden Zahlen (DECIMAL- und NUMERIC-Datentypen) intern als lange Fließkommazahl gespeichert, sobald die Präzision größer als 9 Stellen ist.
 - Der TIME-Datentyp ist nur für das Speichern von Tageszeiten gedacht.
 - Der DATE-Datentyp speichert nur Datumsinformationen.
 - Der BIGINT-Datentyp für 64-Bit-Integer ist verfügbar.
 - Doppelte Anführungszeichen sind reserviert als Begrenzer für irreguläre Bezeichner. Diese ermöglichen es Objektnamen zu definieren, die Groß- und Kleinschreibung unterscheiden oder die Anforderungen an reguläre Bezeichner nicht erfüllen.
 - Alle Zeichenketten müssen in einfachen Anführungszeichen begrenzt werden.

- Generatorwerte werden als 64-Bit-Ganzzahlen gespeichert.

Wichtig

Die Verwendung von Dialekt 3 wird strengstens für die Entwicklung neuer Datenbanken und Anwendungen empfohlen. Sowohl die Datenbank- als auch die Verbindungsdialekte sollte zueinander passen. Ausnahme bildet die Migration mittels Dialekt 2.

Diese Referenz beschreibt die Semantic unter SQL Dialekt 3, solange nicht anders angegeben.

Fehlerbedingungen

Jede Verarbeitung eines SQL-Statements wird erfolgreich beendet oder mit einer speziellen Fehlerbedingung abgebrochen.

Grundelemente: Statements, Klauseln, Schlüsselwörter

Das grundlegendste Konstrukt in SQL ist das *Statement*. Ein Statement definiert was das DBMS mit bestimmten Daten oder Metadaten-Objekten tun soll. Komplexere Statements bedienen sich einfacher Konstrukte—*Klauseln* und *Optionen*.

Klauseln: Eine Klausel definiert eine bestimmte Art von Direktiven in einem Statement. So bestimmt zum Beispiel die WHERE-Klausel in einem SELECT-Statement und anderen manipulativen Statements (UPDATE, DELETE) Kriterien, um Daten innerhalb einer oder mehrerer Tabellen auszuwählen, zu aktualisieren oder zu löschen. Die Klausel ORDER BY gibt an, in welcher Reihenfolge die ausgegebenen Daten —Rückgabesatz— sortiert werden sollen.

Optionen: Optionen sind die einfachsten Konstrukte und werden in Verbindung mit bestimmten Schlüsselwörtern eingesetzt. Wo Optionen zum Einsatz kommen, wird eine als Standard hinterlegt, solange nichts anderes angegeben wurde. So wird zum Beispiel das SELECT-Statement alle Datenzeilen zurückgeben, die die erforderlichen Kriterien der Abfrage erfüllen, es sei denn die Option DISTINCT schränkt diese Ausgabe auf eindeutige Zeilen ein.

Schlüsselwörter: Alle Schlüsselwörter die im SQL-Katalog enthalten sind, werden als Schlüsselwörter bezeichnet. Einige Schlüsselwörter sind *reserviert*, das heißt ihr Gebrauch als Bezeichner für Datenbankobjekte, Parameternamen oder Variablen ist in bestimmten oder gar allen Kontexten untersagt. Nichtreservierte Schlüsselwörter können als Bezeichner verwendet werden, obwohl dies nicht empfohlen wird. Von Zeit zu Zeit kann es vorkommen, dass nichtreservierte Schlüsselwörter im Zuge von Spracherweiterungen reserviert werden.

Im Beispiel wird das folgende Statement ohne Fehler ausgeführt, obwohl ABS ein Schlüsselwort und nicht reserviert ist.

```
CREATE TABLE T (ABS INT NOT NULL);
```

Andererseits wird das folgende Statement mit einem Fehler beendet, da ADD sowohl ein Schlüsselwort als auch ein reserviertes Wort ist.

```
CREATE TABLE T (ADD INT NOT NULL);
```

Bitte vergleichen Sie auch die Auflistung reservierter Wörter und Schlüsselwörter im Abschnitt [Reservierte Wörter und Schlüsselwörter](#).

Bezeichner

Alle Datenbankobjekte haben Namen, häufig auch *Bezeichner* genannt. Zwei Namensarten sind gültige Bezeichner: *reguläre* Namen, ähnlich den Variablennamen in regulären Programmiersprachen, und *begrenzte* Namen, die spezifisch für SQL sind. Um als gültig erachtet zu werden, muss jeder Bezeichnertyp konform zu gewissen Regeln sein:

Regeln für reguläre Objektbezeichner:

- Die Zeichenlänge darf 31 Zeichen nicht übersteigen
- Der Name muss mit einem unakzentuierten, 7-Bit ASCII-Zeichen beginnen. Zahlen sind nicht erlaubt. Die weiteren Zeichen dürfen aus weiteren 7-Bit ASCII-Zeichen, Zahlen, Unterstrichen oder Dollarzeichen bestehen. Keine anderen Zeichen, darunter auch Leerzeichen, dürfen Verwendung finden. Beim Namen wird nicht zwischen Groß- und Kleinschreibung unterschieden. Das heißt, der Name kann sowohl in Klein- als auch Großschreibung verwendet werden. Somit sind folgende Namen für das System gleichzusetzen:

```
fullname
FULLNAME
FuLLNaMe
FullName
```

Syntax für reguläre Namen

```
<name> ::=
  <letter> | <name><letter> | <name><digit> | <name>_ | <name>$

  <letter> ::= <upper letter> | <lower letter>

  <upper letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M |
                    N | O | P | Q | R | S | T | U | V | W | X | Y | Z

  <lower letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m |
                    n | o | p | q | r | s | t | u | v | w | x | y | z

  <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Regeln für begrenzte Objektbezeichner:

- Die Zeichenlänge darf 31 Zeichen nicht überschreiten.
- Die gesamte Zeichenkette muss in doppelte Anführungszeichen eingeschlossen werden, z.B. "anIdentifier".
- Es darf jedes Zeichen eines Latin-Zeichensatzes verwendet werden, inklusive akzentuierte Zeichen, Leerzeichen und Sonderzeichen.
- Ein Bezeichner darf ein reserviertes Wort sein.

- Begrenzte Objektbezeichner unterscheiden immer zwischen Groß- und Kleinschreibung.
- Führende Leerzeichen werden entfernt, so wie bei jeder konstanten Zeichenkette.
- Begrenzte Objektbezeichner sind nur in Dialekt 3 verfügbar. Für mehr Details, vgl. [SQL-Dialekte](#)

Syntax für begrenzte Objektbezeichner

```
<delimited name> ::= "<permitted_character>[<permitted_character> ...]"
```

Anmerkung

Ein begrenzter Bezeichner wie **"FULLNAME"** ist das Gleiche wie die regulären Bezeichner **FULLNAME**, **fullname**, **FullName**, usw.. Der Grund hierfür ist, dass Firebird alle regulären Namen in Großschreibung speichert, egal wie sie definiert oder deklariert wurden. Begrenzte Bezeichner hingegen werden immer so gespeichert, wie sie definiert wurden. Damit unterscheidet sich **"FullName"** von **FULLNAME** und beispielsweise **FullName**.

Literale

Literale werden verwendet um Daten in einem bestimmten Format zu repräsentieren. Beispiele hierfür sind:

```
integer - 0, -34, 45;
real - 0.0, -3.14, 3.23e-23;
string - 'text', 'don''t!';
binary string - x'48656C6C6F20776F726C64'
date - DATE'2018-01-19';
time - TIME'15:12:56';
timestamp - TIMESTAMP'2018-01-19 13:32:02';
null state - null
```

Wie Literale für die diversen Datentypen verwendet werden, wird im nächsten Abschnitt [Datentypen und Unterdantentypen](#) behandelt.

Operatoren und Sonderzeichen

Einige Sonderzeichen sind für die Verwendung als Operator oder Separator reserviert.

```
<special char> ::= <space> | " | % | & | ' | ( | ) | * | + | , | - |
. | / | : | ; | < | = | > | ? | [ | ] | ^ | { | }
```

Einige dieser Zeichen, allein oder in Kombination, dürfen als Operatoren (arithmetisch, Zeichenkette, logisch), als SQL Befehlsseparatoren, zum Anführen von Bezeichnern und zum Markieren von Begrenzungen von Zeichenketten oder Kommentaren verwendet werden.

Syntax für Operatoren:

```

<operator> ::=
  <string concatenation operator> |
  <arithmetic operator> |
  <comparison operator> |
  <logical operator>

<string concatenation operator> ::= {||}

<arithmetic operator> ::= * | / | + | - |

<comparison operator> ::=
  {=} | {<>} | {!=} | {~=} | {^=} |
  {>} | {<} | {>=} | {<=} | {!>} | {~>} | {^>} |
  {!<} | {~<} | {^<}

<logical operator> ::= NOT | AND | OR

```

Für weitere Details zu Operatoren, vgl. [Ausdrücke](#).

Kommentare

Kommentare können in SQL-Skripten, -Statements und PSQL-Modulen eingefügt werden. Ein Kommentar kann dabei jede Art von Text sein, die der Autor üblicherweise zum Dokumentieren seines Codes verwendet. Der Parser ignoriert Kommentartexte.

Firebird unterstützt zwei Arten von Kommentaren: **block** und **in-line**.

Syntax:

```

<comment> ::= <block comment> | <single-line comment>

<block comment> ::=
  /* <ASCII char>[<ASCII char> ...] */

<single-line comment> ::=
  -- <ASCII char>[<ASCII char> ...]<end line>

```

Block-Kommentare starten mit Zeichenpaar `/*` und enden mit dem Zeichenpaar `*/`. Text innerhalb der Block-Kommentare kann jede Länge haben und darf aus mehreren Zeilen bestehen.

In-line-Kommentare starten mit dem Zeichenpaar `--` und gelten bis zum Ende der Zeile.

Beispiel:

```

CREATE PROCEDURE P(APARAM INT)
RETURNS (B INT)
AS
BEGIN

```



```
/* This text will be ignored during the execution of the statement
   since it is a comment
*/
B = A + 1; -- In-line comment
SUSPEND;
END
```

Datentypen und Unterdatentypen

Daten unterschiedlicher Typen werden verwendet für:

- die Spaltendefinition in Datenbanktabellen mittels CREATE TABLE-Statement oder zum Ändern der Spalten mittels ALTER TABLE
- die Deklaration oder Änderung einer *Domain* unter Verwendung der Statements CREATE DOMAIN oder ALTER DOMAIN
- die Deklaration lokaler Variablen in gespeicherten Prozeduren, PSQL-Blöcken und Triggern sowie spezifizierter Parameter in gespeicherten Prozeduren
- die indirekte Spezifikation von Argumenten und Rückgabewerten bei der Deklaration externer Funktionen (UDFs—user-defined functions)
- die zur Verfügungstellung von Argumenten für die Funktion CAST() um Daten von einem Typ zu einem anderen zu konvertieren

Tabelle 3.1. Übersicht der Datentypen

| Name | Größe | Präzision & Grenzen | Beschreibung |
|-----------------------|--|--|---|
| BIGINT | 64 Bits | Von -2^{63} bis $(2^{63} - 1)$ | Nur in Dialekt 3 verfügbar |
| BLOB | unterschiedlich | Die Größe eines BLOB-Segments ist auf 64K begrenzt. Die maximale Größe eines BLOB-Feldes sind 4GB. | Ein Datentyp mit dynamisch unterschiedlicher Größe für die Ablage von großen Datenmengen, wie z.B. Bilder, Texte, Audiodaten. Die strukturelle Basiseinheit ist das Segment. Der BLOB-Untertyp definiert dessen Inhalt. |
| CHAR(n), CHARACTER(n) | n Zeichen. Größe in Bytes abhängig von der Encodierung, der Anzahl Bytes pro Zeichen | von 1 bis 32,767 Bytes | Ein Datentyp mit fester Länge. Bei Anzeige der Daten werden Leerzeichen an das Ende der Zeichenkette bis zur angegebenen Länge angefügt. Die Leerzeichen werden nicht in der Datenbank gespeichert, jedoch wiederhergestellt, um die definierte Länge bei Anzeige am Client zu erreichen. Die Leerzeichen werden nicht über das Netzwerk versendet, was den Datenverkehr reduziert. Wurde kein Zeichenlänge angegeben, wird 1 als Standardwert verwendet. |

| Name | Größe | Präzision & Grenzen | Beschreibung |
|---|---|--|--|
| DATE | 32 Bits | von 01.01.0001 AD bis 31.12.9999 AD | ISC_DATE. Nur Datum, kein Zeitelement |
| DECIMAL (<i>precision, scale</i>) | Varying (16, 32 or 64 bits) | <i>precision</i> = von 1 bis 18, legt die mindestmögliche Anzahl zu speichernder Ziffern fest; <i>scale</i> = von 0 bis 18, gibt die Anzahl der Nachkommastellen an. | Eine Fließkommazahl, die <i>scale</i> Nachkommastellen besitzt. <i>scale</i> muss kleiner oder gleich dem Wert von <i>precision</i> sein. Beispiel: DECIMAL(10,3) speichert eine Zahl im folgenden Format: ppppppp.sss |
| DOUBLE PRECISION | 64 Bits | $2.225 * 10^{-308}$ bis $1.797 * 10^{308}$ | Doppelte Präzision nach IEEE, ~15 Stellen, zuverlässige Größe hängt von der Plattform ab. |
| FLOAT | 32 bits | $1.175 * 10^{-38}$ bis $3.402 * 10^{38}$ | Einfache Präzision nach IEEE, ~7 Stellen |
| INTEGER, INT | 32 Bits | -2.147.483.648 up to 2.147.483.647 | Ganzzahlen mit Vorzeichen |
| NUMERIC (<i>precision, scale</i>) | Unterschiedlich (16, 32 oder 64 Bits) | <i>precision</i> = von 1 bis 18, legt die genaue Anzahl zu speichernder Stellen fest; <i>scale</i> = von 0 bis 18, legt die Anzahl der Nachkommastellen fest. | Eine Kommazahl mit <i>scale</i> Nachkommastellen. <i>scale</i> muss kleiner oder gleich <i>precision</i> sein. Beispiel: NUMERIC(10,3) ist eine Zahl im Format: ppppppp.sss |
| SMALLINT | 16 Bits | -32.768 bis 32.767 | Ganzzahlen mit Vorzeichen (word) |
| TIME | 32 Bits | 0:00 to 23:59:59.9999 | ISC_TIME. Tageszeit. Kann nicht zum Speichern von Zeitintervallen verwendet werden. |
| TIMESTAMP | 64 Bits (2 X 32 Bits) | Von Anfang des Tages 01.01.0001 AD bis Ende des Tages 31.12.9999 AD | Datum und Uhrzeit eines Tages |
| VARCHAR(n), CHAR VARYING, CHARACTER VARYING | n Zeichen. Größe in Bytes, abhängig von der Encodierung, der Anzahl von Bytes für ein Zeichen | von 1 bis 32,765 Bytes | Zeichenkette mit variabler Länge. Die Gesamtgröße der Zeichen darf (32KB-3) nicht übersteigen. Dies berücksichtigt auch die hinterlegte Encodierung. Die beiden hinteren Bytes speichern die deklarierte Länge. Es gibt keine Standardgröße. Das Argument <i>n</i> ist erforderlich. Führende und abschließende Leerzeichen werden |

| Name | Größe | Präzision & Grenzen | Beschreibung |
|------|-------|---------------------|--|
| | | | gespeichert und nicht abgeschnitten, außer den Leerzeichen, die hinter der definierten Länge liegen. |

Hinweis zu Daten

Beachten Sie, dass eine Zeitreihe, bestehend aus Daten der letzten Jahrhunderte, verarbeitet wird, ohne auf historische Gegebenheiten Rücksicht zu nehmen. Dennoch ist der Gregorianische Kalender komplett anwendbar.

Integer-Datentypen

Die Datentypen SMALLINT, INTEGER und BIGINT werden für Ganzzahlen verschiedener Präzisionen in Dialekt 3 verwendet. Firebird unterstützt keine vorzeichenlosen (unsigned) Integer.

SMALLINT

Der Datentyp SMALLINT dient dem Speichern von Kleinstzahlen mit einem Bereich -2^{16} bis $2^{16} - 1$, also von -32.768 bis 32.767.

SMALLINT-Beispiel:

```
CREATE DOMAIN DFLAG AS SMALLINT DEFAULT 0 NOT NULL
CHECK (VALUE=-1 OR VALUE=0 OR VALUE=1);

CREATE DOMAIN RGB_VALUE AS SMALLINT;
```

INTEGER

Daten vom Typ INTEGER werden als 4-Byte-Integer repräsentiert. Der Kurzname dieses Datentyps ist INT. Der gültige Bereich des Datentyps INTEGER reicht von -2^{32} bis $2^{32} - 1$, also von -2.147.483.648 bis 2.147.483.647.

INTEGER-Beispiel:

```
CREATE TABLE CUSTOMER (
CUST_NO INTEGER NOT NULL,
CUSTOMER VARCHAR(25) NOT NULL,
CONTACT_FIRST VARCHAR(15),
CONTACT_LAST VARCHAR(20),
```

```
CONSTRAINT INTEG_60,  
...  
PRIMARY KEY (CUST_NO) )
```

BIGINT

BIGINT ist ein SQL:99-konformer, 64-Bit langer Integer-Datentyp, der nur in Dialekt 3 zur Verfügung steht. Wenn der Client Dialekt 1 verwendet, sendet der Server den auf 32 Bit (Integer) reduzierten Generatorwert. Bei Dialekt 3 wird der Datentyp BIGINT verwendet.

Der Zahlenraum des Datentyps BIGINT liegt im Bereich zwischen -2^{63} und $2^{63} - 1$, also von -9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807.

Hexadezimaler Format für BIGINT-Zahlen: Seit Firebird 2.5 können Zahlen vom Datentyp BIGINT auch im Hexadezimalen Format geschrieben werden. Dabei können 9 bis 16 Stellen angegeben werden. Kürzere Hexadezimalzahlen werden als INTEGER interpretiert.

BIGINT-Beispiel:

```
CREATE TABLE WHOLELOTTARECORDS (  
  ID BIGINT NOT NULL PRIMARY KEY,  
  DESCRIPTION VARCHAR(32)  
) ;  
  
INSERT INTO MYBIGINTS VALUES (  
  -236453287458723,  
  328832607832,  
  22,  
  -56786237632476,  
  0X6F55A09D42, -- 478177959234  
  0X7FFFFFFFFFFFFFFF, -- 9223372036854775807  
  0XFFFFFFFFFFFFFFF, -- -1  
  0X80000000, -- -2147483648, ein INTEGER  
  0X080000000, -- 2147483648, ein BIGINT  
  0XFFFFFFFF, -- -1, ein INTEGER  
  0X0FFFFFFFF -- 4294967295, ein BIGINT  
) ;
```

Die hexadezimalen INTEGER im obigen Beispiel werden automatisch in den BIGINT-Datentyp gewandelt, bevor sie in die Tabelle eingefügt werden. Jedoch passiert dies *nachdem* der numerische Wert bestimmt wurde, wodurch 0x80000000 (8 Stellen) und 0x080000000 (9 Stellen) als unterschiedliche BIGINT-Werte gespeichert werden.

Fließkomma-Datentypen

Fließkomma-Datentypen werden nach dem IEEE 754-Standard binär gespeichert und enthalten Vorzeichen, Exponent und Mantisse. Die Genauigkeit ist dynamisch, je nach verwendetem Speicherformat, welches 4 Bytes für FLOAT oder 8 Bytes für DOUBLE PRECISION im Speicher belegt.

Angesichts der Besonderheiten bei der Speicherung von Fließkommazahlen in einer Datenbank, werden diese Datentypen nicht zum Speichern von monetären Daten empfohlen. Aus den gleichen Gründen sollte die Verwendung dieser Felder für Schlüssel und Eindeutigkeiten vermieden werden.

Für die Prüfung von Daten in Spalten mit Fließkomma-Datentypen sollten Ausdrücke auf bestimmte Bereiche, statt auf Übereinstimmungen geprüft werden.

Wenn diese Datentypen in Ausdrücken Verwendung finden, ist äußerste Sorgfalt in Bezug auf die Rundung der Ergebnisse zu legen.

FLOAT

Dieser Datentyp hat eine ungefähre Genauigkeit von 7 Stellen nach dem Komma. Um die korrekte Sicherung der Daten zu gewährleisten, verwenden Sie nur 6 Stellen.

DOUBLE PRECISION

Dieser Datentyp besitzt eine ungefähre Genauigkeit von 15 Stellen.

Festkomma-Datentypen

Festkomma-Datentypen sorgen für die Vorhersehbarkeit bei Multiplikations- und Divisionoperationen. Somit sind sie die erste Wahl zum Speichern monetärer Daten. Firebird implementiert zwei Festkomma-Datentypen: NUMERIC und DECIMAL. Gemäß dem Standard, werden beide Datentypen mit der Anzahl zu speichernder Stellen (Stellen nach dem Komma) begrenzt.

Verschiedene Behandlungen beschränken die Genauigkeit für jeden Typ: Die Genauigkeit für NUMERIC-Felder ist genau „wie deklariert“, wohingegen DECIMAL-Felder Zahlen akzeptieren, deren Genauigkeit mindestens der Deklaration entspricht.

Beispielsweise definiert NUMERIC(4, 2) eine Zahl mit insgesamt 4 Stellen und 2 Nachkommastellen; das heißt, es können bis zu zwei Stellen vor dem Komma und maximal 2 Stellen nach dem Komma verwendet werden. Wird die Zahl 3,1415 in ein Feld dieses Datentyps geschrieben, wird der Wert 3,14 gespeichert.

Die Art der Deklaration für Festkomma-Datentypen, zum Beispiel NUMERIC(p , s), ist beiden Typen gleich. Zu beachten ist, dass die Skalierung mittels des Arguments s , vielmehr die Bedeutung „Anzahl der Stellen nach dem Dezimalkomma“ hat. Das Verständnis wie Festkomma-Datentypen ihre Daten speichern und abrufen zeigt auch, warum dies so ist: beim Speichern wird die Zahl mit 10^s multipliziert (10 hoch s) und als Integer gespeichert; beim Lesen wird der Integer wieder zurückkonvertiert.

Die Art und Weise wie Festkomma-Daten im DBMS abgelegt werden, hängt von diversen Faktoren ab: Genauigkeit, Datenbankdialekt, Deklarationstyp.

Tabelle 3.2. Verfahren zum Speichern von reellen Zahlen

| Skalierung | Datentyp | Dialekt 1 | Dialekt 3 |
|------------|----------------------|------------------|-----------|
| 1 - 4 | NUMERIC | SMALLINT | SMALLINT |
| 1 - 4 | DECIMAL | INTEGER | INTEGER |
| 5 - 9 | NUMERIC ODER DECIMAL | INTEGER | INTEGER |
| 10 - 18 | NUMERIC ODER DECIMAL | DOUBLE PRECISION | BIGINT |

NUMERIC

Datenformat für die Deklaration:

NUMERIC(*p*, *s*)

Beispiel der Speicherung: In Bezug auf das obige Beispiel, wird das DBMS NUMERIC-Daten in Abhängigkeit von der angegebenen Genauigkeit (*p*) und der Skalierung (*s*) speichern. Weitere Beispiele:

| | |
|---------------------------------|-------------------------------------|
| NUMERIC(4) wird gespeichert als | SMALLINT (exakte Daten) |
| NUMERIC(4,2) | SMALLINT (Daten * 10 ²) |
| NUMERIC(10,4) (Dialekt 1) | DOUBLE PRECISION |
| (Dialekt 3) | BIGINT (Daten * 10 ⁴) |

Achtung

Beachten Sie, dass das Speicherformat von der Genauigkeit abhängt. So können Sie zum Beispiel die Spalte als NUMERIC(2,2) definieren, vorausgesetzt ihr Wertebereich liegt zwischen -0.99...0.99. Dennoch liegt der reale Wertebereich zwischen -327.68..327.67, wodurch deutlich wird, dass der NUMERIC(2,2)-Datentyp im SMALLINT-Format abgelegt wird. Gleiches gilt für NUMERIC(4,2) und NUMERIC(3,2). Das heißt, wenn Sie den Wertebereich auf -0.99...0.99 beschränken wollen, müssen Sie einen Constraint hierfür erstellen.

DECIMAL

Datenformat für die Deklaration:

DECIMAL(*p*, *s*)

Beispiel der Speicherung: Das Speicherformat in der Datenbank für DECIMAL-Felder ist ähnlich dem von NUMERIC, mit dem Unterschied, dass es leichter zu verstehen ist, mit ein paar Beispielen:

| | | |
|---------------|-----------------|------------------------------------|
| DECIMAL(4) | gespeichert als | INTEGER (exakte Daten) |
| DECIMAL(4,2) | | INTEGER (Daten * 10 ²) |
| DECIMAL(10,4) | (Dialekt 1) | DOUBLE PRECISION |
| | (Dialekt 3) | BIGINT (Daten * 10 ⁴) |

Datentypen für Datum und Zeit

Die Datentypen DATE, TIME und TIMESTAMP werden zur Arbeit mit Daten verwendet, die Daten und Zeiten beinhalten. Dialekt 3 unterstützt alle drei Typen, während Dialekt 1 nur DATE kennt. Der Datentyp DATE in Dialekt 3 ist „nur Datum“, wohingegen der Datentyp DATE in Dialekt 1 Datum und Tageszeit speichert. Somit ist dieser equivalent zu TIMESTAMP in Dialekt 3. Dialekt 1 kennt keinen „nur Datum“-Typ.

Anmerkung

In Dialekt 1 können DATE-Daten alternativ als TIMESTAMP definiert werden. Dies ist die empfohlene Anwendung für neue Definitionen in Dialekt 1-Datenbanken.

Sekundenbruchteile: Werden Sekundenbruchteile in Datums- und Zeitfeldern benötigt, speichert Firebird diese in zehntausendstel Sekunden. Wird eine geringere Genauigkeit bevorzugt, kann diese explizit als tausendstel, hundertstel oder zehntel Sekunde in Dialekt 3-Datenbanken ab ODS 11 spezifiziert werden.

Einige nützliche Informationen zur Genauigkeit von Sekundenbruchteilen:

Der Zeitanteil eines TIME- oder TIMESTAMP-Datentyps ist ein 4-Byte WORD, mit Raum für Dezimalsekunden-Genauigkeit. Die Zeitwerte werden als Dezimalmillisekunden ab Mitternacht gespeichert. Die derzeitige Genauigkeit dieser Werte, die mittels Zeit(stempel)-Funktionen oder -Variablen gelesen oder geschrieben werden ist:

- *CURRENT_TIME* nutzt standardmäßig Sekundengenauigkeit und kann bis zu Millisekundengenauigkeit definiert werden: `CURRENT_TIME (0 | 1 | 2 | 3)`
- *CURRENT_TIMESTAMP* Millisekundengenauigkeit. Genauigkeit von Sekunden bis zu Millisekunden kann mit `CURRENT_TIMESTAMP (0 | 1 | 2 | 3)` definiert werden.
- Literal *'NOW'*: Millisekundengenauigkeit
- Die Funktionen *DATEADD()* und *DATEDIFF()* unterstützen die Genauigkeit bis zu Millisekunden. Dezimalmillisekunden können definiert werden, diese werden jedoch auf die nächste Ganzzahl gerundet, bevor weitere Operationen durchgeführt werden.
- Die Funktion *EXTRACT()* gibt die Genauigkeit bis auf Dezimalmillisekunden zurück, wenn die Argumente *SECOND* und *MILLISECOND* verwendet werden.
- Für die Literale *TIME* und *TIMESTAMP* akzeptiert Firebird glücklicherweise Genauigkeiten bis zu Dezimalmillisekunden, schneidet (nicht rundet) den Zeiteil jedoch auf die nächste, untere oder gleiche Millisekunde ab. Versuchen Sie beispielsweise `SELECT TIME '14:37:54.1249' FROM rdb$database`
- werden Sie feststellen, dass die Operatoren '+' und '-' mit Dezimalmillisekunden-Genauigkeit arbeiten, jedoch nur *innerhalb* des Ausdrucks. Sobald irgendwas gespeichert oder nur von *RDB\$DATABASE* abgefragt (SELECTed) wird, wandelt sich die Genauigkeit zu Millisekunden.

Die Genauigkeit auf Basis von Dezimalmillisekunden ist selten und wird derzeit nicht in Spalten oder Variablen gespeichert. Obwohl Firebird die Werte von TIME und den Zeiteil von TIMESTAMP als Dezimalmillisekunden (10⁻⁴ Sekunden) seit Mitternacht speichert, kann die Genauigkeit von Sekunden zu Millisekunden variieren.

DATE

Der Datentyp DATE in Dialekt 3 speichert nur das Datum ohne Zeitangabe. Der verfügbare Bereich reicht vom 1. Januar 0001 bis zum 31. Dezember 9999.

In Dialekt 1 gibt es keinen „nur Datum“-Datentyp.

Tipp

In Dialekt 1, erhalten Datums-Literale ohne Zeitangabe, genauso wie 'TODAY', 'YESTERDAY' und 'TOMORROW' automatisch einen Null-Zeitteil.

Sollte es für Sie wichtig sein, aus welchem Grund auch immer, einen Zeitstempel-Literal mit einem expliziten Null-Zeitteil in Dialekt 1 zu speichern, wird die Datenbank ein Literal wie '25.12.2016 00:00:00.0000' akzeptieren. Der Wert '25.12.2016' hätte jedoch den gleichen Effekt, mit weniger Tastenschlägen!

TIME

Der Datentyp TIME ist nur in Dialekt 3 verfügbar. Er speichert die Tageszeit im Bereich von 00:00:00.0000 bis 23:59:59.9999.

Sollten Sie den Zeitteil eines DATE in Dialekt 1 benötigen, können Sie die Funktion EXTRACT verwenden.

Beispielverwendung von EXTRACT:

```
EXTRACT (HOUR FROM DATE_FIELD)
EXTRACT (MINUTE FROM DATE_FIELD)
EXTRACT (SECOND FROM DATE_FIELD)
```

TIMESTAMP

Der Datentyp TIMESTAMP ist in Dialekt 1 und 3 verfügbar. Er besteht aus zwei 32-Bit-Teilen—einem Datums- und einem Zeitteil—womit eine Struktur gebildet wird, die sowohl Datum wie auch Tageszeit aufnimmt. Der Datentyp ist identisch mit DATE in Dialekt 1.

Die Funktion EXTRACT arbeitet für TIMESTAMP genauso wie für DATE in Dialekt 1.

Operationen, die Datums- und Zeitwerte verwenden

Das Verfahren zum Speichern der Datums- und Zeitwerte macht es möglich, diese als Operanden in arithmetischen Operationen zu verwenden. Gespeichert, wird ein Datumswert als Zahl seit dem „Datum Null“—17.

November 1898— repräsentiert, ein Zeitwert als Anzahl der Sekunden (unter Berücksichtigung der Sekundenbruchteile) seit Mitternacht.

Ein Beispiel ist das Subtrahieren eines früheren Datum, einer Zeit oder eines Zeitstempels von einem späteren, was in einm Zeitintervall, in Tagen und Bruchteilen von Tagen resultiert.

Tabelle 3.3. Arithmetische Operationen für Datums- und Zeitdatentypen

| Operand 1 | Operation | Operand 2 | Ergebnis |
|-----------|-----------|----------------------|--|
| DATE | + | TIME | TIMESTAMP |
| DATE | + | Numerischer Wert n | DATE um n ganze Tage erhöht. Gebrochene Werte werden auf die nächste Ganzzahl gerundet (nicht abgeschnitten). |
| TIME | + | DATE | TIMESTAMP |
| TIME | + | Numerischer Wert n | TIME um n Sekunden erhöht. Bruchteile werden berücksichtigt. |
| TIMESTAMP | + | Numerischer Wert n | TIMESTAMP, wobei das Datum um die Anzahl der Tage und der Teil eines Tages durch die Zahl n repräsentiert wird— somit wird „+ 2.75“ das Datum um 2 Tage und 18 Stunden weiterstellen wird |
| DATE | - | DATE | Anzahl der vergangenen Tage innerhalb des Bereichs DECIMAL(9, 0) |
| DATE | - | Numerischer Wert n | DATE um n ganze Tage reduziert. Gebrochene Werte werden auf die nächste Ganzzahl gerundet (nicht abgeschnitten). |
| TIME | - | TIME | Anzahl der vergangenen Sekunden, innerhalb des Bereichs DECIMAL(9, 4) |
| TIME | - | Numerischer Wert n | TIME um n Sekunden reduziert. Bruchteile werden berücksichtigt. |
| TIMESTAMP | - | TIMESTAMP | Anzahl der Tage und der Tageszeit, innerhalb des Bereichs DECIMAL(18, 9) |
| TIMESTAMP | - | Numerischer Wert n | TIMESTAMP wobei das Datum sich auf der Anzahl der Tage und der Tageszeit beruht, die durch die Zahl n repräsentiert wird—somit wird „- 2.25“ das Datum um 2 Tage und 6 Stunden reduzieren. |

Hinweise

Der Datentyp DATE ist als TIMESTAMP in Dialekt 1 zu betrachten.

Siehe auch: [DATEADD](#), [DATEADD](#)

Zeichendatentypen

Für die Arbeit mit Zeichendaten bietet Firebird die Datentypen CHAR mit Festlänge und VARCHAR mit variabler Zeichenlänge. Die Maximalgröße der hiermit speicherbaren Daten beträgt 32.767 Bytes für CHAR und 32.765 Bytes für VARCHAR. Das Maximum der möglichen *Zeichen*, das in diese Grenzen passt, hängt vom verwendeten Zeichensatz (CHARACTER SET) ab. Die Sortiermethode COLLATE wirkt sich nicht auf die Maximalgrenze aus, wohingegen sie durchaus die maximale Größe eines Indexes auf dieser Spalte beeinflussen kann.

Wurde während der Definition eines Zeichenobjektes kein expliziter Zeichensatz festgelegt, wird der Standardzeichensatz der Datenbank verwendet. Wurde für die Datenbank kein Standardzeichensatz festgelegt, erhält das Feld den Zeichensatz NONE.

Unicode

Die meisten aktuellen Entwicklertools unterstützen Unicode, welches in Firebird mit den Zeichensätzen UTF8 und UNICODE_FSS integriert ist. UTF8 bietet Sortierungen für viele Sprachen. UNICODE_FSS ist deutlich begrenzter und wird hauptsächlich intern durch Firebird für das Speichern von Metadaten verwendet. Beachten Sie, dass ein UTF8-Zeichen bis zu 4 Bytes beanspruchen kann, wodurch die Größe von CHAR-Feldern auf 8.191 Zeichen reduziert werden kann (32.767/4).

Anmerkung

Der genaue Wert der „Bytes pro Zeichen“ hängt vom Bereich des Zeichens ab. Nicht-akzentuierte Latin-Buchstaben beanspruchen 1 Byte, kyrillische Zeichen mit WIN1251-Enkodierung beanspruchen 2 Bytes, andere Zeichenenkodierungen können bis zu 4 Bytes beanspruchen.

Der in Firebird implementierte UTF8-Zeichensatz, unterstützt die aktuellste Version des Unicode-Standards. Somit ist dieser für internationale Datenbanken empfohlen.

Client-Zeichensatz

Während der Arbeit mit Zeichenketten, ist es notwendig, den Zeichensatz des Clients zu berücksichtigen. Sollte eine Diskrepanz zwischen den Zeichensätzen der gespeicherten Daten und der Clientverbindung existieren, werden Ausgaben für Textfelder automatisch neu enkodiert. Dies gilt für Daten, die vom Client zum Server gesendet werden und anders herum. Wurde die Datenbank beispielsweise mit WIN1251 enkodiert, der Client verwendet jedoch KOI8R oder UTF8, stellt sich die Diskrepanz transparent dar.

Spezielle Zeichensätze

Zeichensatz NONE: Der Zeichensatz NONE ist ein *Spezialzeichensatz* in Firebird. Er kann so beschrieben werden, als wäre jedes Byte Teil einer Zeichenkette, die jedoch keine Angaben zur Beschreibung eines Zeichens macht: Zeichenenkodierung, Sortierung, Klein- und Großschreibung, etc. sind einfach unbekannt. Es liegt in der Verantwortung der Client-Anwendung mit den Daten umzugehen und die richtigen Mittel bereitzustellen,

um die Folge von Bytes in irgendeiner Weise interpretieren zu können, die für die Anwendung sinnvoll sind und für den Menschen lesbar.

Zeichensatz OCTETS: Daten der OCTETS-Enkodierung werden als Bytes behandelt, die nicht direkt als Zeichen interpretiert werden. OCTETS bieten einen Weg um Binärdaten zu speichern, was die Ergebnisse einiger Firebird-Funktionen sein könnten. Die Datenbank weiß nicht was mit einer Zeichenkette aus Bits in OCTETS zu tun ist, außer diese zu speichern und abzufragen. Auch hier ist wieder der Client verantwortlich für die Validierung der Daten und diese sowohl der Anwendung wie auch dem Benutzer in verständlicher Form anzuzeigen. Dies gilt auch für Ausnahmen, die durch die Enkodierung und Dekodierung verursacht werden.

COLLATION

Jeder Zeichensatz hat eine Standardsortiermethode (COLLATE). Üblicherweise stellt diese nicht mehr bereit als die Reihenfolge basierend auf einem numerischen Code der Zeichen und einer Basiszuordnung der Klein- und Großbuchstaben. Wird ein Verhalten außerhalb der Collation benötigt und eine alternative Methode für den Zeichensatz unterstützt, kann eine COLLATE *Sortier*-Klausel in der Felddefinition verwendet werden.

Eine COLLATE *Sortier*-Klausel kann auch in anderen Zusammenhängen neben der Spaltendefinition angewandt werden. Für größer-als-/kleiner-als-Vergleiche, kann sie in die WHERE-Klausel des SELECT-Statements aufgenommen werden. Wird eine speziell alphabetisch geordnete oder Groß- und Kleinschreibungsinsensitive Ausgabe benötigt, und sollte eine passende Collation existieren, dann kann die COLLATE-Klausel auch in der ORDER BY-Klausel verwendet werden.

Groß- und Kleinschreibungsinsensitive Suche

Für die Groß- und Kleinschreibungsinsensitive Suche, kann die Funktion UPPER verwendet werden, damit das Suchargument und der die gesuchte Zeichenkette in Großbuchstaben gewandelt werden, bevor der Vergleich stattfindet:

```
...
where upper(name) = upper(:flt_name)
```

Für Zeichenketten eines Zeichensatzes, der keine Groß- und Kleinschreibungsinsensitive Sortierung bereitstellt, können sie die Sortierung anwenden, um das Suchargument und die gesuchte Zeichenkette direkt miteinander zu vergleichen. Beispielsweise ist unter dem Zeichensatz WIN1251 die Sortierung PXW_CYRL Groß- und Kleinschreibungsinsensitiv. Somit gilt:

```
...
WHERE FIRST_NAME COLLATE PXW_CYRL >= :FLT_NAME
...
ORDER BY NAME COLLATE PXW_CYRL
```

Vgl. auch: [CONTAINING](#)

UTF8-Collation

Die folgende Tabelle zeigt mögliche Sortiermethoden für den UTF8-Zeichensatz.

Tabelle 3.4. Collation für den Zeichensatz UTF8

| Sortierung | Merkmale |
|---------------|--|
| UCS_BASIC | Sortierung arbeitet abhängig von der Position des Zeichens in der Tabelle (binär). Hinzugefügt in Firebird 2.0 |
| UNICODE | Sortierung arbeitet abhängig vom UCA-Algorithmus (Unicode Collation Algorithm) (alphabetisch). Hinzugefügt in Firebird 2.0 |
| UTF8 | Die Standard-Sortierung, binär, identisch zu UCS_BASIC, welche im Rahmen der SQL-Kompatibilität hinzugefügt wurde. |
| UNICODE_CI | Groß- und Kleinschreibungsinsensitive Sortierung, arbeitet ohne Groß- und Kleinschreibung zu berücksichtigen. Hinzugefügt in Firebird 2.1 |
| UNICODE_CI_AI | Groß- und Kleinschreibungsunabhängige, akzentunabhängige Sortierung, die weder Groß- und Kleinschreibung noch Akzentuierung von Zeichen berücksichtigt. Arbeitet alphabetisch. Hinzugefügt in Firebird 2.5 |

Beispiel: Ein Beispiel einer Sortierung für den UTF8-Zeichensatz ohne Berücksichtigung der Groß- und Kleinschreibung oder Akzentuierung der Zeichen (ähnlich zu COLLATE PXW_CYRL).

```
...
ORDER BY NAME COLLATE UNICODE_CI_AI
```

Zeichenindizes

In Firebird-Versionen vor 2.0 kann es zu Problemen beim Erstellen eines Index für Zeichenspalten kommen, die eine außergewöhnliche Collation nutzen: die Länge eines indizierten Feldes ist auf 252 Bytes beschränkt, sofern kein COLLATE spezifiziert wurde, andernfalls sind es 84 Bytes. Multi-Byte-Zeichensätze schränken die Indizes weiter ein.

Ab Firebird 2.0 beschränkt sich die Indexlänge auf ein Viertel der Seitengröße (page size), z.B. von 1.024 bis 4.096 Bytes. Die größtmögliche Länge einer indizierten Zeichenkette liegt bei 9 Bytes weniger als die Viertel-Seiten-Grenze.

Berechnung der maximalen Länge eines indizierten Zeichenfeldes: Die folgende Formel berechnet der maximale Länge eines indizierten Zeichenfeldes (in Zeichen):

$$\text{max_char_length} = \text{FLOOR}((\text{page_size} / 4 - 9) / N)$$

wobei N die Anzahl der Bytes pro Zeichen im Zeichensatz darstellt.

Die folgende Tabelle zeigt die Maximallänge einer indizierten Zeichenkette (in Zeichen), abhängig von der Seitengröße und Zeichensatz. Die Maximallänge wurde mittels der o.a. Formel berechnet.

Tabelle 3.5. Maximale Indexlänge nach Seitengröße und Zeichengröße

| Seitengröße (Page Size) | Maximale Länge einer indizierten Zeichenkette für ein Zeichen, Bytes / Zeichen | | | | |
|----------------------------|--|-------|-------|-------|-----|
| | 1 | 2 | 3 | 4 | 6 |
| 4.096 | 1.015 | 507 | 338 | 253 | 169 |
| 8.192 | 2.039 | 1.019 | 679 | 509 | 339 |
| 16.384 | 4.087 | 2.043 | 1.362 | 1.021 | 682 |

Anmerkung

Mit Collations („_CI“), die Groß- und Kleinschreibungsinsensitiv sind, wird ein Zeichen im **Index** nicht 4, sondern 6 Bytes, beanspruchen. Hierdurch wird die maximale Schlüssellänge für eine Seitengröße von zum Beispiel 4.096 Bytes auf 169 Zeichen begrenzt.

Vergleichen Sie auch: `CREATE DATABASE`, `COLLATION`, `SELECT`, `WHERE`, `GROUP BY`, `ORDER BY`

Zeichendatentypen im Detail

CHAR

CHAR ist ein Festlängen-Datentyp. Ist die eingegebene Anzahl der Zeichen kleiner als die definierte Länge, werden Leerzeichen zu dem Feld hinzugefügt. Grundsätzlich muss das Füllzeichen kein Leerzeichen sein: dies hängt vom Zeichensatz ab. So ist das Füllzeichen für den OCTETS-Zeichensatz beispielsweise die Null.

Der volle Name dieses Datentyps ist CHARACTER, es ist jedoch nicht erforderlich volle Namen zu verwenden und die Leute tun dies auch sehr selten.

Daten für Festlängen-Zeichen können verwendet werden, um Codes zu speichern, deren Länge standardisiert ist und die eine definierte „Breite“ besitzen. Ein Beispiel hierfür ist ein EAN13-Barcode—13 Zeiche, alle gefüllt.

Syntax für die Deklaration:

```
CHAR [(length)] [CHARACTER SET <set>] [COLLATE <name>]
```

Anmerkung

Wurde keine Länge (*length*) angegeben, wird 1 verwendet.

Eine gültige Länge (*length*) befindet sich im Bereich von 1 bis maximal so vielen Zeichen, die in 32,767 Bytes passen.

VARCHAR

VARCHAR ist der Standarddatentyp zum Speichern von Texten variabler Länge. Die Zeichen müssen in 32.765 Bytes passen. Die Speicherstruktur ist identisch mit der aktuellen Datengröße plus 2 Bytes.

Alle Zeichen, die von der Client-Anwendung an die Datenbank gesendet werden, werden als sinnvoll erachtet, was auch für führende und nachrangige Leerzeichen gilt. Jedoch werden angestellte Leerzeichen nicht gespeichert: Sie werden wiederhergestellt, sobald die Daten abgerufen werden. Die Zeichenkette wird dann bis zu der gespeicherten Länge mit dem Leerzeichen aufgefüllt.

Der volle Name dieses Datentyps ist CHARACTER VARYING. Eine weitere Variante dieses Namens wird auch mit CHAR VARYING bezeichnet.

Syntax:

```
VARCHAR (length) [CHARACTER SET <set>] [COLLATE <name>]
```

NCHAR

NCHAR ist ein Festlängen-Datentyp. Der ISO8859_1-Zeichensatz ist vordefiniert. In allen anderen Bezügen verhält sich dieser Datentyp identisch zu CHAR.

Syntax:

```
NCHAR (length)
```

Das Synonym hierfür ist NATIONAL CHAR. Für variable Längen gibt es einen ähnlichen Datentyp: NATIONAL CHARACTER VARYING.

Binärdatentypen

BLOBs (Binary Large Objects) nutzen komplexe Strukturen, um Texte und binäre Daten beliebiger Länge zu speichern. Diese Daten sind häufig sehr groß.

Syntax:

```
BLOB [SUB_TYPE <subtype>]  
      [SEGMENT SIZE <segment size>]  
      [CHARACTER SET <character set>]
```

Gekürzte Syntax:

```
BLOB (<segment size>)  
BLOB (<segment size>, <subtype>)  
BLOB (, <subtype>)
```

Segmentgröße: Das Spezifizieren von BLOB-Segmenten ist ein Rückschritt in vergangene Zeiten, als die Programme zur Verarbeitung von BLOB-Daten noch mit Hilfe des *gpre*-Pre-Compilers in C (Embedded SQL) geschrieben wurden. Heutzutage ist dies irrelevant. Die Segmentgröße für BLOB-Daten wird auf Client-Seite festgelegt und ist üblicherweise größer als die Größe der Datenseite.

BLOB Untertypen

Der optionale Parameter `SUB_TYPE` gibt die Art der zu schreibenden Felddaten an. Firebird unterstützt zwei vordefinierte Untertypen für die Datenspeicherung:

Untertyp 0: BINARY: Wurde kein Untertyp angegeben, wird von einem unspezifizierten Datentyp ausgegangen und somit als Standard `SUB_TYPE 0` verwendet. Der Alias für diesen Untertyp ist `BINARY`. Dies ist der Untertyp, der bei der Arbeit mit jeglichen Binärdaten Verwendung findet: Bilder, Audio, Textdokumente, PDFs usw.

Untertyp 1: TEXT: Untertyp 1 besitzt den Alias `TEXT`, welcher in Deklarationen und Definitionen verwendet werden kann. Zum Beispiel `BLOB SUB_TYPE TEXT`. Dies ist ein spezieller Untertyp zum Speichern von Textdaten, die zu lang für die üblichen Zeichenketten sind. Ein Zeichensatz (`CHARACTER SET`) kann definiert werden, sofern dieser vom Standard der Datenbank abweicht. Seit Firebird 2.0 ist die Angabe der `COLLATE`-Klausel ebenfalls gültig.

Benutzerdefinierte Untertypen: Weiterhin ist es möglich eigene Untertypen zu definieren, wofür der Zahlenbereich von -1 bis -32.768 reserviert ist. Selbstdefinierte Untertypen im positiven Zahlenbereich werden nicht unterstützt, da Firebird diese, von 2 aufwärts, für interne Untertypen in den Metadaten verwendet.

BLOB Specifics

Größe: Die Maximalgröße eines BLOB-Feldes ist auf 4GB begrenzt, unanhängig vom darunterliegenden 32- oder 64-Bit-Server. (Die internen Strukturen im Zusammenhang mit BLOBs nutzen ihre eigenen 4-Byte-Zähler.) Für Seitengrößen (page size) von 4 KB (4096 Bytes) ist die Maximalgröße geringer — etwas weniger als 2 GB.

Operationen und Ausdrücke: Text-BLOBs beliebiger Länge und jeder Zeichensatz—inklusive Multi-Byte—können Operanden für praktisch jede Art von Statement oder interne Funktionen sein. Die folgenden Operatoren werden vollständig unterstützt:

| | |
|---------------------|-------------------------|
| = | (Zuweisung) |
| =, <>, <, <=, >, >= | (Vergleich) |
| | (Verkettung) |
| BETWEEN, | IS [NOT] DISTINCT FROM, |
| IN, | ANY SOME, |
| ALL | |

Teilunterstützt:

- Ein Fehler wird ausgeworfen, wenn das Suchkriterium größer als 32 KB ist:

| | |
|------------------|-------|
| STARTING [WITH], | LIKE, |
| CONTAINING | |

- Aggregations-Klauseln funktionieren nicht auf Basis der eigentlichen Feldwerte, jedoch mit den BLOB-IDs. Dementsprechend gibt es ein paar Macken:

| | |
|-----------------|---|
| SELECT DISTINCT | Gibt fälschlicherweise mehrere NULL-Werte zurück, sofern diese vorhanden sind. |
| ORDER BY | — |
| GROUP BY | Verkettet die gleichen Zeichenketten, wenn sie aufeinanderfolgen, jedoch nicht, wenn sie voneinander entfernt liegen. |

BLOB-Speicherung:

- Standardmäßig wird ein regulärer Datensatz für jeden BLOB erstellt und in der Datenseite (data page), die hierfür zugewiesen wurde, gespeichert. Passt das gesamte BLOB in diese Seite, spricht man auch von einem *Level 0 BLOB*. Die Seitenzahl dieses speziellen Datensatzes wird im Tabellendatensatz gespeichert und belegt 8 Bytes.
- Passt das BLOB hingegen nicht in die Datenseite, wird der Inhalt auf separate Seiten verteilt, die exklusiv hierfür belegt werden (BLOB-Seiten). Die Seitennummern werden in den BLOB-Datensatz geschrieben. Hierbei spricht man von einem *Level 1 BLOB*.
- Falls das Array der Seitenzahlen, die die BLOB-Daten enthalten nicht in die Datenseite passen, wird diese Array auf separate BLOB-Seiten verteilt, während die Seitenzahlen dieser Seiten in den BLOB-Datensatz geschrieben werden. Hierbei spricht man von einem *Level 2 BLOB*.
- Level größer als 2 werden nicht unterstützt.

Vergleichen Sie auch: [FILTER](#), [DECLARE FILTER](#)

ARRAY Type

Die Unterstützung von Arrays im Firebird DBMS ist eine Abkehr vom traditionellen relationalen Modell. Unterstützung von Arrays im DBMS könnte es einfacher machen, Datenverarbeitungsaufgaben mit großen Mengen von ähnlichen Daten zu lösen.

Array werden in Firebird mittels speziellen BLOB-Typen gespeichert. Arrays können ein- und multidimensional sein. Sie dürfen aus beliebigen Datentypen, außer BLOB und ARRAY bestehen.

Beispiel:

```
CREATE TABLE SAMPLE_ARR (  
    ID INTEGER NOT NULL PRIMARY KEY,  
    ARR_INT INTEGER [4]);
```

Dieses Beispiel erstellt eine Tabelle mit einem Array-Feld, welches 4 Ganzzahlenelemente (Integer) enthält. Die Indizes werden von 1 bis 4 gezählt.

Festlegen expliziter Dimensionsgrenzen

Standardmäßig sind die Indizes 1-basierend—Zählung beginnt bei 1. Um die Ober- und Untergrenzen explizit festzulegen, nutzen Sie folgende Syntax:

```
[ <lower> : <upper> ]
```

Weitere Dimensionen hinzufügen

Eine neue Dimension wird nach dem Komma in der Syntax hinzugefügt. In diesem Beispiel erstellen wir eine Tabelle mit einem zweidimensionalen Array, mit der unteren Indizegrenze von 0 beginnend:

```
CREATE TABLE SAMPLE_ARR2 (  
    ID INTEGER NOT NULL PRIMARY KEY,  
    ARR_INT INTEGER [0:3, 0:3]);
```

Das DBMS bietet, im Vergleich mit anderen Sprachen oder Werkzeugen, nicht viele Möglichkeiten zur Arbeit mit Array-Inhalten. Die Datenbank `employee.fdb`, die im Unterverzeichnis `../examples/empbuild` jedes Firebird-Pakets gefunden werden kann, beinhaltet eine beispielhafte Gespeicherte Prozedur (stored procedure), die einfache Routinen im Umgang mit Arrays zeigt:

PSQL Quelltext für SHOW_LANGS, eine Prozedur zur Arbeit mit Arrays:

```
CREATE OR ALTER PROCEDURE SHOW_LANGS (  
    CODE VARCHAR(5),  
    GRADE SMALLINT,  
    CTY VARCHAR(15))  
RETURNS (LANGUAGES VARCHAR(15))  
AS  
    DECLARE VARIABLE I INTEGER;  
BEGIN  
    I = 1;  
    WHILE (I <= 5) DO  
        BEGIN  
            SELECT LANGUAGE_REQ[:I]  
            FROM JOB  
            WHERE (JOB_CODE = :CODE)  
                AND (JOB_GRADE = :GRADE)  
                AND (JOB_COUNTRY = :CTY)  
                AND (LANGUAGE_REQ IS NOT NULL))  
            INTO :LANGUAGES;  
  
            IF (LANGUAGES = '') THEN  
                /* 'NULL' anstelle von Leerzeichen ausgeben */  
                LANGUAGES = 'NULL';  
            I = I + 1;  
            SUSPEND;  
        END  
    END  
END
```

Reichen die beschriebenen Features für Ihre Aufgaben aus, können Sie darüber nachdenken, Arrays in Ihrem Projekt zu verwenden. Derzeit planen die Firebird-Entwickler keine Verbesserungen im Bereich Arrays.

Spezialdatentypen

„Spezial“-Datentypen ...

SQL_NULL-Datentyp

Der Datentyp `SQL_NULL` besitzt keinen Wert, sondern nur einen Status: `null (NULL)` oder `nicht null (NOT NULL)`. Er ist nicht verfügbar als Datentyp zur Deklaration von Tabellenfeldern, PSQL-Variablen oder als Para-

meterbeschreibung. Dieser Datentyp wurde eingeführt, um untypisierte Parameter in Ausdrücken mit dem IS NULL-Prädikat zu verwenden.

Ein Auswertungsproblem tritt auf, wenn optionale Filter verwendet werden, um Abfragen der folgenden Art zu erstellen:

```
WHERE coll = :param1 OR :param1 IS NULL
```

Nach der Verarbeitung auf API-Ebene, sieht die Abfrage folgendermaßen aus:

```
WHERE coll = ? OR ? IS NULL
```

Dies ist der Fall, wenn ein Entwickler eine SQL-Abfrage schreibt und der Auffassung ist, dass sich `:param1` wie eine *Variable* verhält, sodass er diese auf zwei Arten verwenden kann. Jedoch erhält sich die Abfrage auf der API-Ebene zwei unabhängige *Parameter*. Der Server kann den Typen des zweiten Parameters nicht ermitteln, da dieser in Verbindung mit IS NULL steht.

Der Datentyp SQL_NULL löst dieses Problem. Wannimmer die Datenbank auf das ' ? IS NULL' -Prädikat in einer Abfrage stößt, weist sie dem Parameter den Typ SQL_NULL zu, was wiederum dazu führt, dass dieser Parameter nur auf „Nullbarkeit“ geprüft werden muss und der Datentyp bzw. der Wert nicht wichtig ist.

Das folgende Beispiel demonstriert die praktische Verwendung. Es verwendet 2 benannte Parameter —sagen wir, `:size` und `:colour`—welche, im Beispiel, zwei Werte vom Bildschirmtext oder Drop-Down-Listen erhalten. Jeder dieser Parameter korrespondiert mit zwei Parametern in der Abfrage.

```
SELECT
    SH.SIZE, SH.COLOUR, SH.PRICE
FROM SHIRTS SH
WHERE (SH.SIZE = ? OR ? IS NULL)
      AND (SH.COLOUR = ? OR ? IS NULL)
```

Um zu verstehen was hier passiert, muss der Leser mit der Firebird-API sowie der Übergabe von Parametern in der XSQLVAR-Struktur vertraut sein—was unter der Haube passiert, ist nicht von Belang, sofern keine Treiber oder Anwendungen geschrieben werden, die die „nackte“ API verwenden.

Die Anwendung übergibt die parameterisierte Abfrage an den Server in der üblichen ?-Form. Paare „identischer“ Parameter können nicht in einem zusammengeführt werden, somit werden für zwei optionale Filter, beispielsweise, 4 Parameter benötigt: einer für jede ? in unserem Beispiel.

Nach dem Aufruf von `isc_dsql_describe_bind()`, wird der SQLTYPE des zweiten und vierten Parameters auf SQL_NULL gesetzt. Firebird hat keine Kenntnis ihrer speziellen Verbindung zu dem ersten bzw. dritten Parameter: hierfür ist allein die Anwendung verantwortlich.

Sobald die Werte für Größe (`size`) und Farbe (`colour`) festgelegt wurden (oder auch nicht), und die Abfrage ausgeführt wurde, wird jedes Paar der XSQLVARs folgendermaßen ausgefüllt:

Benutzer hat einen Wert angegeben

Erster Parameter (vergleiche Werte): Lege `*sqldata` auf den übergebenen Wert fest und `*sqlind` auf 0 (für NOT NULL)

Zweiter Parameter (teste auf NULL): Lege `*sqldata` auf null fest (null-Zeiger, nicht SQL NULL) und `*sqlind` auf 0 (für NOT NULL)

Benutzer hat das Feld leer gelassen

Beide Parameter: Setze sqldata auf null (null-Zeiger, nicht SQL NULL) und *sqlind auf -1 (repräsentiert NULL)

In anderen Worten: Der Parameter zum Vergleich des Wertes wird wie üblich gesetzt. Der SQL_NULL-Parameter wird genauso verwendet, mit dem Unterschied, dass sqldata über die gesamte Zeit hinweg null bleibt.

Datentyp-Konvertierungen

Wenn Sie einen Ausdruck erstellen oder eine Operation spezifizieren, sollte das Ziel sein, kompatible Datentypen als Operanden zu verwenden. Wenn die Notwendigkeit besteht, verschiedene Datentypen zu verwenden, müssen Sie einen Weg finden, diese inkompatiblen Typen vor der eigentlichen Operation zu konvertieren. Wenn Sie mit Dialekt 1 arbeiten, kann dies zum Problem werden.

Explizite Datentyp-Konvertierung

Die CAST-Funktion ermöglicht die explizite Konvertierung zwischen diversen Datentypen.

Syntax:

```
CAST ( { <value> | NULL } AS <data_type> )

<data_type> ::= sql_datatype |
              [TYPE OF] domain |
              TYPE OF COLUMN relname.colname
```

Casting zu einer Domain

Beim Ausführen eines Cast zu einer Domain, werden alle Constraints berücksichtigt, die hierfür deklariert wurden, z.B. NOT NULL- oder CHECK-Constraints. Wenn der Wert (<value>) diese Prüfung nicht besteht, schlägt der Cast fehl.

Wird TYPE OF zusätzlich angegeben —Umwandlung auf den Basistyp—, werden alle Domain-Constraints während des Cast ignoriert. Wird TYPE OF mit einem Zeichentyp (CHAR/VARCHAR) verwendet, werden der Zeichensatz und die Collation beibehalten.

Casting zu TYPE OF COLUMN

Werden Operanden auf den Datentyp einer Spalte konvertiert, kann die angegebene Spalte aus einer Tabelle oder View stammen.

Nur der Typ der Spalte selbst wird verwendet. Für Zeichentypen wird der Zeichensatz bei der Konvertierung inkludiert, jedoch nicht die Collation. Die Constraints und Vorgabewerte der Quellspalte werden nicht angewandt.

Beispiel:

```
CREATE TABLE TTT (
    S VARCHAR (40)
    CHARACTER SET UTF8 COLLATE UNICODE_CI_AI);
COMMIT;

SELECT
    CAST ('I have many friends' AS TYPE OF COLUMN TTT.S)
FROM RDB$DATABASE;
```

Mögliche konvertierungen für die CAST-Funktion

Tabelle 3.6. Konvertierungen mit CAST

| von Datentyp | zu Datentyp |
|------------------|--|
| Numerische Typen | Numerische Typen, [VAR]CHAR, BLOB |
| [VAR]CHAR | [VAR]CHAR, BLOB, Numerische Typen, DATE, TIME, TIMESTAMP |
| BLOB | [VAR]CHAR, BLOB, Numerische Typen, DATE, TIME, TIMESTAMP |
| DATE, TIME | [VAR]CHAR, BLOB, TIMESTAMP |
| TIMESTAMP | [VAR]CHAR, BLOB, DATE, TIME |

Wichtig

Beachten Sie, dass Informationsteile möglicherweise verloren gehen. Zum Beispiel geht der Zeiteil bei der Konvertierung eines TIMESTAMP zu einem DATE verloren.

Literalfomate

Für den Cast von String-Datentypen zu DATE-, TIME- oder TIMESTAMP-Datentypen, müssen Sie eines der vordefinierten Datum- bzw. Zeit-Literale (vgl. Tabelle 3.7) oder ein gültiges *Datum-Zeit-Literal*-Format verwenden:

```
<datetime_literal> ::= {
    [YYYY<p>]MM<p>DD[ <p>HH[ <p>mm[ <p>SS[ <p>NNNN]]]] |
    MM<p>DD[ <p>YYYY[ <p>HH[ <p>mm[ <p>SS[ <p>NNNN]]]] |
    DD<p>MM[ <p>YYYY[ <p>HH[ <p>mm[ <p>SS[ <p>NNNN]]]] |
    MM<p>DD[ <p>YY[ <p>HH[ <p>mm[ <p>SS[ <p>NNNN]]]] |
    DD<p>MM[ <p>YY[ <p>HH[ <p>mm[ <p>SS[ <p>NNNN]]]] |
    NOW |
    TODAY |
    TOMORROW |
    YESTERDAY
}
```

```
<date_literal> ::= {
    [YYYY<p>]MM<p>DD |
```

```

MM<p>DD[ <p>YYYY ] |
DD<p>MM[ <p>YYYY ] |
MM<p>DD[ <p>YY ] |
DD<p>MM[ <p>YY ] |
TODAY |
TOMORROW |
YESTERDAY
}

<time_literal> := HH[<p>mm[<p>SS[<p>NNNN]]]

<p> ::= whitespace | . | : | , | - | /

```

Tabelle 3.7. Formatierungsargumente für Datum- und Zeit-Literale Date and Time Literal

| Argument | Beschreibung |
|------------------|---|
| datetime_literal | Datum- und Zeit-Literal |
| time_literal | Zeit-Literal |
| date_literal | Datum-Literal |
| YYYY | vierstelliges Jahr |
| YY | zweistelliges Jahr |
| MM | Monat. Dieser kann aus 1 oder 2 Stellen bestehen (1-12 oder 01-12). Sie können auch die mit drei Buchstaben abgekürzte Form oder den vollen Monatsnamen in englisch angeben. Dies berücksichtigt keine Groß- und Kleinschreibung. |
| DD | Tag. Kann aus 1 oder 2 Stellen bestehen (1-31 oder 01-31) |
| HH | Stunde. Kann aus 1 oder 2 Stellen bestehen (0-23 oder 00-23) |
| mm | Minute. Kann aus 1 oder 2 Stellen bestehen (0-59 oder 00-59) |
| SS | Sekunde. Kann aus 1 oder 2 Stellen bestehen (0-59 oder 00-59) |
| NNNN | Zehntausendstel Sekunde. Kann aus 1 bis 4 Stellen bestehen (0-9999) |
| p | Separator, jedes gültige Zeichen. Führende und nachgestellte Leerzeichen werden ignoriert. |

Tabelle 3.8. Literale mit vordefinierten Datums- und Zeitwerten

| Literal | Beschreibung | Datentyp | |
|---------|--------------------------|-------------------|-----------|
| | | Dialekt 1 | Dialekt 3 |
| 'NOW' | Aktuelles Datum und Zeit | DATE | TIMESTAMP |
| 'TODAY' | Aktuelle Datum | DATE mit Nullzeit | DATE |

| Literal | Beschreibung | Datentyp | |
|-------------|--------------------------|-------------------|-----------|
| | | Dialekt 1 | Dialekt 3 |
| 'TOMORROW' | Heutiges Datum + 1 (Tag) | DATE mit Nullzeit | DATE |
| 'YESTERDAY' | Heutiges Datum - 1 (Tag) | DATE mit Nullzeit | DATE |

Wichtig

Die Nutzung des Jahres in vierstelliger Schreibweise —YYYY—wird dringend empfohlen, um Verwirrungen bei der Datumsberechnung und bei Aggregationen zu vermeiden.

Beispiele für die Interpretation von Datum-Literalen:

```
select
  cast('04.12.2014' as date) as d1, -- DD.MM.YYYY
  cast('04 12 2014' as date) as d2, -- MM DD YYYY
  cast('4-12-2014' as date) as d3, -- MM-DD-YYYY
  cast('04/12/2014' as date) as d4, -- MM/DD/YYYY
  cast('04,12,2014' as date) as d5, -- MM,DD,YYYY
  cast('04.12.14' as date) as d6, -- DD.MM.YY
  -- DD.MM with current year
  cast('04.12' as date) as d7,
  -- MM/DD with current year
  cast('04/12' as date) as d8,
  cast('2014/12/04' as date) as d9, -- YYYY/MM/DD
  cast('2014 12 04' as date) as d10, -- YYYY MM DD
  cast('2014.12.04' as date) as d11, -- YYYY.MM.DD
  cast('2014-12-04' as date) as d12, -- YYYY-MM-DD
  cast('4 Jan 2014' as date) as d13, -- DD MM YYYY
  cast('2014 Jan 4' as date) as dt14, -- YYYY MM DD
  cast('Jan 4, 2014' as date) as dt15, -- MM DD, YYYY
  cast('11:37' as time) as t1, -- HH:mm
  cast('11:37:12' as time) as t2, -- HH:mm:ss
  cast('11:31:12.1234' as time) as t3, -- HH:mm:ss.nnnn
  cast('11.37.12' as time) as t4, -- HH.mm.ss
  -- DD.MM.YYYY HH:mm
  cast('04.12.2014 11:37' as timestamp) as dt1,
  -- MM/DD/YYYY HH:mm:ss
  cast('04/12/2014 11:37:12' as timestamp) as dt2,
  -- DD.MM.YYYY HH:mm:ss.nnnn
  cast('04.12.2014 11:31:12.1234' as timestamp) as dt3,
  -- MM/DD/YYYY HH.mm.ss
  cast('04/12/2014 11.37.12' as timestamp) as dt4
from rdb$database
```

Kurzschreibweisen für Casts von Datums- und Zeit-Datentypen

Firebird erlaubt die Verwendung von Kurzschreibweisen („C-Stil“) für die Konvertierung von Strings zu DATE-, TIME- und TIMESTAMP-Typen.

Syntax:

```
data_type 'date_literal_string'
```

Beispiel:

```
-- 1
UPDATE PEOPLE
SET AGE CAT = 'SENIOR'
WHERE BIRTHDATE < DATE '1-Jan-1943';
-- 2
INSERT INTO APPOINTMENTS
(EMPLOYEE_ID, CLIENT_ID, APP_DATE, APP_TIME)
VALUES (973, 8804, DATE 'today' + 2, TIME '16:00');
-- 3
NEW.LASTMOD = TIMESTAMP 'now';
```

Anmerkung

Diese Kurzschreibweisen werden direkt während des Parsens ausgewertet, so als ob das Statement bereits für die Ausführung vorbereitet wäre. Dadurch hat beispielsweise `timestamp 'now'` immer den gleichen Wert, egal wie viel Zeit verstrichen ist.

Benötigen Sie hingegen eine Möglichkeit, die die Zeit bei jeder Ausführung neu ermittelt, müssen Sie die vollständige CAST-Syntax verwenden. Eine beispielhafte Verwendung eines solchen Ausdrucks in einem Trigger:

```
NEW.CHANGE_DATE = CAST('now' AS TIMESTAMP);
```

Implizite Datentyp-Konvertierung

Implizite Datenkonvertierung ist in Dialekt 3 nicht möglich — die CAST-Funktion wird fast immer benötigt, um Datentyp-Probleme zu vermeiden.

In Dialekt 1 wird in vielen Fällen ein Datentyp implizit in einen anderen gewandelt, ohne die CAST-Funktion verwenden zu müssen. Beispielsweise ist die folgende Klausel in einem SELECT-Statement in Dialekt 1 gültig:

```
WHERE DOC_DATE < '31.08.2014'
```

und der String-Typ wird implizit in den Datums-Datentyp gewandelt.

In Dialekt 1 ist das Vermischen von Ganzzahldaten und numerischen Zeichenketten grundsätzlich möglich, da der Parser versuchen wird, die Zeichenketten implizit zu konvertieren. Beispielsweise wird

```
2 + '1'
```

korrekt ausgeführt.

In Dialekt 3 wird dieser Ausdruck einen Fehler ausgeben. Somit sind Sie gezwungen, einen CAST-Ausdruck zu erstellen:

```
2 + CAST('1' AS SMALLINT)
```

Die Ausnahme bildet hier die *String-Verkettung*.

Implizite Konvertierung während der String-Verkettung

Werden mehrere Elemente miteinander verkettet, werden alle nicht-Zeichen-Daten unter der Hand zu einer Zeichenkette konvertiert, sofern möglich.

Beispiel:

```
SELECT 30||'| days hath September, April, June and November' CONCAT$
FROM RDB$DATABASE

CONCAT$
-----
30 days hath September, April, June and November
```

Benutzerdefinierte Datentypen—Domains

In Firebird wurde das Konzept „benutzerdefinierter Datentypen“ in Form von *Domains* implementiert. Beim Erstellen einer Domain wird nicht wirklich ein neuer Datentyp generiert. Eine Domain kapselt vielmehr einen bestehenden Datentyp mit diversen Attributen und macht diese „Kapsel“ für verschiedene Anwendungsbereiche in der Datenbank verfügbar. Wenn mehrere Tabellen Spalten benötigen, die die gleichen oder nahezu gleichen Eigenschaften haben sollen, macht die Verwendung von Domains Sinn.

Die Verwendung von Domains ist nicht auf die Spaltendefinition in Tabellen oder Views begrenzt. Sie können auch als Übergabe- und Rückgabeparameter sowie Variablen in PSQL-Code verwendet werden.

Domain-Eigenschaften

Eine Domain-Definition beinhaltet benötigte sowie optionale Eigenschaften. Der *Datentyp* ist ein benötigtes Attribut. Optionale Eigenschaften sind:

- ein Vorgabewert
- erlauben oder verbieten von NULL
- CHECK Constraints
- Zeichensatz (für Textdatentypen und Text-BLOB-Felder)
- Collation (für Textdatentypen)

Beispielhafte Domain-Definition:

```
CREATE DOMAIN BOOL3 AS SMALLINT
CHECK (VALUE IS NULL OR VALUE IN (0, 1));
```

vgl. auch: [Explizite Datentyp-Konvertierung](#) für die Beschreibung der Unterschiede im Mechanismus der Datenkonvertierung, wenn Domains für die TYPE OF und TYPE OF COLUMN-Modifizierer verwendet werden.

Domain-Überschreibung

Während des Festlegens der Domain-Definitionen, ist es möglich einige geerbte Eigenschaften zu überschreiben. Tabelle 3.9 fasst die Regeln hierfür zusammen.

Tabelle 3.9. Regeln zum Überschreiben von Domain-Eigenschaften in Spaltendefinitionen

| Eigenschaft | Überschreiben? | Hinweis |
|-------------------|----------------|---|
| Datentyp | Nein | |
| Vorgabewert | Ja | |
| Zeichensatz | Ja | Dies kann auch verwendet werden, um die Vorgabewerte der Datenbank für die Spalte wiederherzustellen. |
| Sortiermethode | Ja | |
| CHECK-Constraints | Ja | Zum Hinzufügen von neuen Kriterien, können Sie die korrespondierenden CHECK-Klauseln der CREATE- und ALTER-Anweisungen auf Tabellenebene verwenden. |
| NOT NULL | Nein | Häufig ist es besser Domains nullbar in ihren Definitionen zu lassen und erst beim Erstellen der Spaltendefinition über NOT NULL zu entscheiden. |

Domains erstellen und verwalten

Eine Domain wird mit der DDL-Anweisung CREATE DOMAIN erstellt.

Kurz-Syntax:

```
CREATE DOMAIN <name> [AS] <type>
[DEFAULT {<const> | <literal> | NULL | <context_var>}]
[NOT NULL] [CHECK (<condition>)]
[COLLATE collation];
```

vgl. auch: [CREATE DOMAIN](#) im Abschnitt Data Definition Language (DDL).

Ändern einer Domain

Zum Ändern der Domain-Eigenschaften, nutzen Sie die Anweisung ALTER DOMAIN. Mit dieser Anweisung können Sie

- die Domain umbenennen
- den Datentyp ändern
- den derzeitigen Vorgabewert ändern
- einen neuen Vorgabewert festlegen
- einen vorhandenen CHECK-Constraint löschen
- einen neuen CHECK-Constraint hinzufügen

Kurz-Syntax:

```
ALTER DOMAIN name
  [{TO new_name}]
  [{SET DEFAULT {literal | NULL | <context_var>} |
  DROP DEFAULT}]
  [{ADD [CONSTRAINT] CHECK (<dom_condition>) |
  DROP CONSTRAINT}]
  [{TYPE <datatype>}];
```

Wenn Sie vorhaben eine Domain zu ändern, müssen Sie die vorhandenen Abhängigkeiten berücksichtigen: wurden Tabellenspalten, Variablen, Übergabe- und/oder Rückgabeparameter mit dieser Domain in PSQL deklariert? Wenn Sie Domains voreilig ändern, ohne diese eingehend zu überprüfen, kann es passieren, dass Ihr Code anschließend nicht mehr läuft!

Wichtig

Wenn Sie Datentypen einer Domain ändern, dürfen Sie keine Konvertierungen durchführen, die zu Datenverlust führen können. Zum Beispiel sollten Sie vorab prüfen, ob das Konvertieren von VARCHAR zu INTEGER für alle Daten erfolgreich durchgeführt werden kann.

vgl. auch: [ALTER DOMAIN](#) im Abschnitt Data Definition Language (DDL).

Löschen einer Domain

Die Anweisung DROP DOMAIN löscht eine Domain aus der Datenbank, sofern diese nicht von anderen Datenbankobjekten verwendet wird.

Syntax:

```
DROP DOMAIN name
```

Wichtig

Jeder mit der Datenbank verbundene Benutzer kann eine Domain löschen.

Beispiel:

```
DROP DOMAIN Test_Domain
```

vgl. auch: **DROP DOMAIN** im Abschnitt Data Definition Language (DDL).

Allgemeine Sprachelemente

Dieser Abschnitt behandelt die Elemente, die in der SQL-Sprache als allgemeingültig betrachtet werden können—die *Ausdrücke*, die verwendet werden um Fakten aus Daten zu extrahieren, diese zu verarbeiten und die *Prädikate*, die den Wahrheitswert dieser Fakten prüfen.

Ausdrücke

SQL-Ausdrücke bieten formelle Methoden zum Auswerten, Transformieren und Vergleichen von Werten. SQL-Ausdrücke können Tabellenspalten, Variablen, Konstanten, Literale, andere Statements und Prädikate sowie andere Ausdrücke enthalten. Folgend die vollständige Liste möglicher Elemente.

Tabelle 4.1. Beschreibung der Ausdruck-Elemente

| Element | Beschreibung |
|--------------------------|--|
| Spaltenname | Kennung einer Spalte aus einer angegebenen Tabelle, die in Auswertungen oder als Suchbedingung verwendet wird. Eine Spalte des Array-Typs kann kein Element innerhalb eines Ausdrucks sein, es sei denn sie wird mit dem IS [NOT] NULL-Prädikat verwendet. |
| Array-Element | Ein Ausdruck kann einen Verweis auf ein Array-Element enthalten. |
| Arithmetische Operatoren | Die Zeichen +, -, *, / werden verwendet um Berechnungen durchzuführen. |
| Verkettungsoperator | Der Operator („Doppel-Pipe“) wird verwendet um Strings zu verketteten. |
| Logische Operatoren | Die reservierten Wörter NOT, AND sowie OR werden verwendet um einfache Suchbedingungen oder komplexere Behauptungen zu erstellen. |
| Vergleichsoperatoren | Die Zeichen =, <>, !=, ~=, ^=, <, <=, >, >=, !<, ~<, ^<, !>, ~> und ^> |
| Vergleichsprädikate | LIKE, STARTING WITH, CONTAINING, SIMILAR TO, BETWEEN, IS [NOT] NULL und IS [NOT] DISTINCT FROM |
| Existenzprädikate | Prädikate, die für die Existenzprüfung von Werten Verwendung finden. Das Prädikat IN kann sowohl innerhalb von Listen kommasetrennter Konstanten als auch mit Unterabfragen, die nur eine Spalte zurückgeben, verwendet werden. Die Prädikate EXISTS, SINGULAR, ALL, ANY und SOME können nur mit Unterabfragen verwendet werden. |
| Konstanten | Eine Zahl; oder eine Zeichenkette, die in Apostrophs eingeschlossen wird. |
| Datum-/Zeitlitterale | Ein Ausdruck, ähnlich zu Zeichenketten, eingeschlossen in Apostrophs, der als Datum, Zeit oder Zeitstempel interpretiert wird. Datumsliterale können vordefinierte Literale ('TODAY', 'NOW', etc.) oder Zeichenketten aus Buch- |

| Element | Beschreibung |
|----------------------------------|--|
| | staben oder Zahlen sein, wie zum Beispiel '25.12.2016 15:30:35', die zu einem Datum und/oder einer Zeit aufgelöst werden können. |
| Kontextvariablen | Eine intern definierte Kontextvariable |
| Lokale Variablen | Deklarierte lokale Variablen, Über- und Rückgabeparameter eines PSQL-Moduls (Stored Procedure, Trigger, unbenannter PSQL-Block in DSQL) |
| Positionale Parameter | Ein Mitglied innerhalb einer geordneten Gruppe von einem oder mehreren unbenannten Parametern, die an eine gespeicherte Prozedur oder eine vorbereitete Abfrage übergeben wurden. |
| Unterabfrage | Eine SELECT-Anweisung, die in Klammern eingeschlossen ist, die einen einzelnen (skalaren) Wert zurückgibt oder, wenn er in existenziellen Prädikaten verwendet wird, einen Satz von Werten. |
| Funktionskennung | Die Kennung einer internen oder externen Funktion in einem Funktionsausdruck |
| Type-Cast | Ein Ausdruck, der explizit Daten von einem in einen anderen Datentyp unter Verwendung der CAST-Funktion (CAST (<value> AS <datatype>)) konvertiert. Nur für Datum-/Zeit-Literale ist die Kurzschreibweise <datatype> <value> (DATE ' 25 . 12 . 2016 ') möglich. |
| Bedingter Ausdruck | Ausdrücke mit CASE und verwandten internen Funktionen |
| Klammern | Klammernpaare (...) werden verwendet, um Ausdrücke zu gruppieren. Operationen innerhalb der Klammern werden vor Operationen außerhalb von ihnen durchgeführt. Wenn eingebettete Klammern verwendet werden, werden die tiefsten eingebetteten Ausdrücke zuerst ausgewertet und dann bewegen sich die Auswertungen von innen nach außen durch die Einbettungsstufen. |
| COLLATE-Klausel | Klausel, die für CHAR- und VARCHAR-Datentypen angewendet werden kann, um die Collation für String-Vergleiche festzulegen. |
| NEXT VALUE FOR <i>Sequenz</i> | Ausdruck zum Ermitteln des nächsten Wertes eines bestimmten Generators (Sequenz). Die interne Funktion GEN_ID() tut das Gleiche. |

Konstanten

Eine Konstante ist ein Wert der direkt in einem SQL-Statement verwendet wird und weder von einem Ausdruck, einem Parameter, einem Spaltenverweis noch einer Variablen abgeleitet wird. Dies kann eine Zeichenkette oder eine Zahl sein.

Zeichenkonstanten (Literale)

Eine String-Konstante ist eine Aneinanderreihung von Zeichen, die zwischen einem Paar von Apostrophen („einfache Anführungszeichen“) eingeschlossen werden. Die größtmögliche Länge dieser Zeichenketten ist 32.767 Bytes; die maximale Anzahl der Zeichen wird durch die verwendete Zeichenkodierung bestimmt.

Anmerkung

- Doppelte Anführungszeichen *dürfen nicht* für die Kennzeichnung von Zeichenketten verwendet werden. SQL sieht hierfür einen anderen Zweck vor.
- Wird ein Apostroph innerhalb der Zeichenkette benötigt, wird dieses durch ein vorangehendes Apostroph „escaped“. Zum Beispiel 'Mother O' 'Reilly's home-made hooch'.
- Vorsicht ist geboten bei String-Längen, wenn der Wert in ein Feld des Typs VARCHAR geschrieben wird. Hierfür gilt die maximale Länge von 32.765 Bytes.

Es wird angenommen, dass der Zeichensatz einer Zeichenkonstanten der gleiche ist wie der Zeichensatz seines Bestimmungsspeichers.

Stringkonstanten in Hexadezimalnotation

Ab Firebird 2.5 aufwärts, können Stringliterals in hexadezimaler Schreibweise eingegeben werden, die sogenannten „Binary Strings“. Jedes Paar hexadezimaler Stellen definiert ein Byte der Zeichenkette. Zeichenketten die in dieser Form eingegeben werden, besitzen den Zeichensatz OCTETS als Standard. Die *Introducer-Syntax* kann auch genutzt werden um zu erzwingen, dass die Zeichenkette als ein anderer Zeichensatz interpretiert wird.

Syntax:

```
{x|X}'<hexstring>'
```

```
<hexstring> ::= eine gerade Anzahl von <hexdigit>
```

```
<hexdigit> ::= eines aus 0..9, A..F, a..f
```

Beispiele:

```
select x'4E657276656E' from rdb$database
-- liefert 4E657276656E, a 6-byte 'binary' string

select _ascii x'4E657276656E' from rdb$database
-- liefert 'Nerven' (same string, now interpreted as ASCII text)

select _iso8859_1 x'53E46765' from rdb$database
-- liefert 'Säge' (4 chars, 4 bytes)

select _utf8 x'53C3A46765' from rdb$database
-- liefert 'Säge' (4 chars, 5 bytes)
```

Hinweise

Die Client-Schnittstelle legt fest, wie Binärzeichenfolgen dem Benutzer angezeigt werden. Das *isql*-Werkzeug beispielsweise, nutzt großgeschriebene Buchstaben A-F, während FlameRobin Kleinschreibung verwendet. Andere Client-Applikationen könnten andere Konventionen bevorzugen, zum Beispiel Leerzeichen zwischen den Bytepaaren: '4E 65 72 76 65 6E'.

Mit der hexadezimalen Notation kann jeder Bytewert (einschließlich 00) an beliebiger Stelle im String eingefügt werden. Allerdings, wenn Sie diesen auf etwas anderes als OCTETS erzwingen wollen, liegt es in Ihrer Verantwortung, die Bytes in einer Sequenz zu liefern, die für den Zielzeichensatz gültig ist.

Introducer-Syntax für String-Literale

Bei Bedarf kann ein String-Literal einem Zeichensatznamen vorangestellt werden, dem ein Unterstrich „_“ vorangestellt ist. Dies wird als *Introducer-Syntax* bezeichnet. Sein Ziel ist es, die Engine darüber zu informieren, wie man den eingehenden String interpretiert und speichert.

Beispiel

```
INSERT INTO People
VALUES (_ISO8859_1 'Hans-Jörg Schäfer')
```

Zahlenkonstanten

Eine Zahlkonstante ist eine gültige Zahl in einer unterstützten Notation:

- In SQL wird der Dezimalpunkt, für Zahlen in der Standard-Dezimal-Notation, immer durch das Punkt-Zeichen dargestellt. Tausender werden nicht getrennt. Einbeziehung von Komma, Leerzeichen usw. führt zu Fehlern.
- Exponentielle Notation wird unterstützt. Zum Beispiel kann 0.0000234 auch als $2.34e-5$ geschrieben werden.
- Hexadezimal-Notation wird von Firebird 2.5 und höheren Versionen unterstützt—siehe unten.

Hexadezimale Notation für Ziffern

Von Firebird 2.5 aufwärts können ganzzahlige Werte in hexadezimaler Notation eingegeben werden. Zahlen mit 1-8 Hex-Ziffern werden als Typ INTEGER interpretiert; Zahlen mit 9-16 Hex-Ziffern als Typ BIGINT.

Syntax:

```
0{x|X}<hexdigits>

<hexdigits> ::= 1-16 als <hexdigit>
<hexdigit>  ::= eins aus 0..9, A..F, a..f
```

Beispiele:

```
select 0x6FAA0D3 from rdb$database      -- liefert 117088467
select 0x4F9 from rdb$database          -- liefert 1273
select 0x6E44F9A8 from rdb$database    -- liefert 1850014120
select 0x9E44F9A8 from rdb$database    -- liefert -1639646808 (an INTEGER)
select 0x09E44F9A8 from rdb$database   -- liefert 2655320488 (a BIGINT)
select 0x28ED678A4C987 from rdb$database -- liefert 720001751632263
select 0xFFFFFFFFFFFFFFFF from rdb$database -- liefert -1
```


Hexadezimale Wertebereiche

- Hex-Nummern im Bereich 0 .. 7FFF FFFF sind positive INTEGER mit Dezimalwerten zwischen 0 .. 2147483647. Um eine Zahl als BIGINT zu erzwingen, müssen Sie genügend Nullen voranstellen, um die Gesamtzahl der Hex-Ziffern auf neun oder mehr zu bringen. Das ändert den Typ, aber nicht den Wert.
- Hex-Nummern zwischen 8000 0000 .. FFFF FFFF erfordern etwas Aufmerksamkeit:
 - Bei der Eingabe mit acht Hex-Ziffern, wie in 0x9E44F9A8, wird ein Wert als 32-Bit-INTEGER interpretiert. Da das erste Bit (Vorzeichenbit) gesetzt ist, wird es dem negativen Dezimalbereich -2147483648 .. -1 zugeordnet.
 - Bei einer oder mehreren Nullen, die wie in 0x09E44F9A8 vorangestellt werden, wird ein Wert als 64-Bit-BIGINT im Bereich 0000 0000 8000 0000 .. 0000 0000 FFFF FFFF interpretiert. Das Zeichen-Bit ist jetzt nicht gesetzt, also wird der Dezimalwert dem positiven Bereich 2147483648 .. 4294967295 zugewiesen. So ergibt sich in diesem Bereich - und nur in diesem Bereich - anhand einer mathematisch unbedeutenden 0 ein gänzlich anderer Wert. Dies ist zu beachten.
- Hex-Zahlen zwischen 1 0000 0000 .. 7FFF FFFF FFFF FFFF sind alle positiv BIGINT.
- Hex-Zahlen zwischen 8000 0000 0000 0000 .. FFFF FFFF FFFF FFFF sind alle negativ BIGINT.
- Ein SMALLINT kann nicht in Hex geschrieben werden, streng genommen zumindest, da sogar 0x1 als INTEGER ausgewertet wird. Wenn Sie jedoch eine positive Ganzzahl innerhalb des 16-Bit-Bereichs 0x0000 (Dezimal-Null) bis 0x7FFF (Dezimalzahl 32767) schreiben, wird sie transparent in SMALLINT umgewandelt.

Es ist möglich einen negativen SMALLINT in Hex zu schreiben, wobei eine 4-Byte-Hexadezimalzahl im Bereich 0xFFFF8000 (Dezimal -32768) bis 0xFFFFFFFF (Dezimal -1) verwendet wird.

SQL-Operatoren

SQL-Operatoren umfassen Operatoren zum Vergleichen, Berechnen, Auswerten und Verknüpfen von Werten.

Vorrang der Operatoren

SQL Operatoren sind in vier Typen unterteilt. Jeder Operator-Typ hat eine *Priorität*, eine Rangfolge, die die Reihenfolge bestimmt, in der die Operatoren und die mit ihrer Hilfe erhaltenen Werte in einem Ausdruck ausgewertet werden. Je höher der Vorrang des Operator-Typs ist, desto früher wird er ausgewertet. Jeder Operator hat seine eigene Priorität innerhalb seines Typs, der die Reihenfolge bestimmt, in der sie in einem Ausdruck ausgewertet werden.

Operatoren der gleichen Rangfolge werden von links nach rechts ausgewertet. Um dieses Verhalten zu beeinflussen, können Gruppen mittels Klammern erstellt werden.

Tabelle 4.2. Vorrang der Operatortypen

| Operatortyp | Vorrang | Erläuterung |
|-------------|---------|--|
| Verkettung | 1 | Strings werden verkettet, bevor andere Operationen stattfinden |

| Operatortyp | Vorrang | Erläuterung |
|-------------|---------|---|
| Arithmetik | 2 | Arithmetische Operationen werden durchgeführt, nachdem Strings verkettet sind, aber vor Vergleichs- und logischen Operationen |
| Vergleiche | 3 | Vergleichsoperationen erfolgen nach String-Verkettung und arithmetischen Operationen, aber vor logischen Operationen |
| Logical | 4 | Logische Operatoren werden nach allen anderen Operatortypen ausgeführt |

Verkettungsoperator

Der Verkettungsoperator, zwei Pipe-Zeichen, auch „Doppel-Pipe“ — `||` — verkettet (verbindet) zwei Zeichenketten zu einer einzigen Zeichenkette. Zeichenketten können dabei Konstante Werte oder abgeleitet von einer Spalte oder einem Ausdruck sein.

Beispiel:

```
SELECT LAST_NAME || ', ' || FIRST_NAME AS FULL_NAME
FROM EMPLOYEE
```

Arithmetische Operatoren

Tabelle 4.3. Vorrang arithmetischer Operatoren

| Operator | Zweck | Vorrang |
|------------------------------|----------------|---------|
| + Zahl mit Vorzeichen | unäres Plus | 1 |
| - Zahl mit Vorzeichen | unäres Minus | 1 |
| * | Multiplikation | 2 |
| / | Division | 2 |
| + | Addition | 3 |
| - | Subtraktion | 3 |

Beispiel:

```
UPDATE T
SET A = 4 + 1/(B-C)*D
```

Anmerkung

Wenn Operatoren den gleichen Vorrang besitzen, werden diese von links nach rechts ausgewertet.

Vergleichsoperatoren**Tabelle 4.4. Prioritäten der Vergleichsoperatoren**

| Operator | Zweck | Priorität |
|----------------|-------------------------------|-----------|
| = | Ist gleich, ist identisch mit | 1 |
| <>, !=, ~=, ^= | Ist ungleich zu | 1 |
| > | Ist größer als | 1 |
| < | Ist kleiner als | 1 |
| >= | Ist größer oder gleich als | 1 |
| <= | Ist kleiner oder gleich als | 1 |
| !>, ~>, ^> | Ist nicht größer als | 1 |
| !<, ~<, ^< | Ist nicht kleiner als | 1 |

Diese Gruppe enthält außerdem die Vergleichsprädikate BETWEEN, LIKE, CONTAINING, SIMILAR TO, IS und andere.

Beispiel:

```
IF (SALARY > 1400) THEN
...
```

See also: [Andere Vergleichsprädikate.](#)

Logische Operatoren**Tabelle 4.5. Prioritäten logischer Operatoren**

| Operator | Zweck | Priorität |
|----------|--|-----------|
| NOT | Negierung eines Suchkriteriums | 1 |
| AND | Kombiniert zwei oder mehr Prädikate, wobei jedes als wahr angesehen werden muss, damit der Gesamtausdruck ebenfalls als wahr aufgelöst wird | 2 |
| OR | Kombiniert zwei oder mehr Prädikate, wobei mindestens eines als wahr angesehen werden muss, damit der Gesamtausdruck ebenfalls als wahr aufgelöst wird | 3 |

Beispiel:

```
IF (A < B OR (A > C AND A > D) AND NOT (C = D)) THEN ...
```

NEXT VALUE FOR

Verfügbar: DSQL, PSQL

NEXT VALUE FOR gibt den nächsten Wert einer Sequenz zurück. SEQUENCE ist ein SQL-konformer Begriff für Generatoren in Firebird und dessen Vorgänger, InterBase. Der Operator NEXT VALUE FOR ist äquivalent zur ursprünglichen Funktion GEN_ID (... , 1) und ist die empfohlene Syntax zum Holen des nächsten Wertes.

Syntax für NEXT VALUE FOR:

```
NEXT VALUE FOR Sequenzname
```

Beispiel:

```
NEW.CUST_ID = NEXT VALUE FOR CUSTSEQ;
```

Anmerkung

Anders als GEN_ID (... , 1) verwendet NEXT VALUE FOR keine Parameter, wodurch es nicht möglich ist den *aktuellen Wert* einer Sequenz zu ermitteln sowie eine andere Schrittweite als 1 zu nutzen. GEN_ID (... , <step value>) wird noch immer für diesen Zweck verwendet. Eine <Schrittweite> von 0 gibt den aktuellen Sequenzwert zurück.

Siehe auch: [SEQUENCE \(GENERATOR\)](#), [GEN_ID\(\)](#)

Bedingte Ausdrücke

Ein bedingter Ausdruck ist einer der verschiedene Werte zurückgibt, je nach verwendeter Bedingung. Es besteht aus einem bedingten Funktionskonstrukt, wovon Firebird mehrere unterstützt. Dieser Abschnitt beschreibt nur ein bedingtes Ausdruckskonstrukt: CASE. Alle anderen bedingten Ausdrücke sind interne Funktionen und leiten sich von CASE ab und werden in [Bedingte Funktionen](#) beschrieben.

CASE

Verfügbar: DSQL, PSQL

Das CASE-Konstrukt gibt einen einzigen Wert aus einer Reihe von Werten zurück. Zwei syntaktische Varianten werden unterstützt:

- Das *einfache* CASE, vergleichbar zu einem *CASE-Konstrukt* in Pascal oder einem *Switch* in C
- Das *gesuchte* CASE, welches wie eine Reihe aus „if ... else if ... else if“-Klauseln funktioniert.

Einfaches CASE

Syntax:

```
...
CASE <test-expr>
  WHEN <expr> THEN <result>
  [WHEN <expr> THEN <result> ...]
  [ELSE <defaultresult>]
END
...
```

Wenn diese Variante verwendet wird, wird `<test-expr>` mit `<expr> 1`, `<expr> 2` etc. verglichen, bis ein Treffer gefunden und das passende Ergebnis zurückgegeben wird. Wenn kein passender Treffer vorhanden ist, wird `<defaultresult>` aus der optionalen ELSE-Klausel zurückgegeben, andernfalls NULL.

Der Trefferwahl funktioniert identisch zum "="-Operator. Daher gilt, wenn `<test-expr>` gleich NULL ist, wird kein Treffer für `<expr>` ermittelt, nicht einmal wenn dieser zu NULL aufgelöst wird.

Das zurückgegebene Ergebnis muss kein literaler Wert sein: Es kann ein Feld oder ein Variablenname, ein Ausdruck oder NULL-Literal sein.

Beispiel:

```
SELECT
  NAME ,
  AGE ,
  CASE UPPER(SEX)
    WHEN 'M' THEN 'Male'
    WHEN 'F' THEN 'Female'
    ELSE 'Unknown'
  END GENDER ,
RELIGION
FROM PEOPLE
```

Eine Kurzform des einfachen CASE-Konstrukts wird auch in der [DECODE](#) -Funktion verwendet.

Gesuchtes CASE

Syntax:

```
CASE
  WHEN <bool_expr> THEN <result>
  [WHEN <bool_expr> THEN <result> ...]
  [ELSE <defaultresult>]
```

```
END
```

Der <bool_expr>-Ausdruck gibt ein ternäres logisches Ergebnis zurück: TRUE, FALSE oder NULL. Der erste Ausdruck, der TRUE ermittelt, wird als Ergebnis verwendet. Gibt kein Ausdruck TRUE zurück, kommt <defaultresult> aus der optionalen ELSE-Klausel zum Einsatz. Gibt kein Ausdruck TRUE zurück und gibt es keine ELSE-Klausel, ist der Rückgabewert NULL.

So wie im einfachen CASE-Konstrukt, muss das Ergebnis nicht zwangsläufig ein Literal sein: es kann ein Feld- oder Variablenname, ein zusammengesetzter Ausdruck oder NULL sein.

Beispiel:

```
CANVOTE = CASE
  WHEN AGE >= 18 THEN 'Yes'
  WHEN AGE < 18 THEN 'No'
  ELSE 'Unsure'
END
```

NULL in Ausdrücken

NULL kein Wert in SQL, sondern ein *Status* der anzeigt, dass der Wert des Elements entweder *unbekannt* (engl. unknown) ist oder nicht existiert. Es ist weder null, noch void, noch ein „leerer String“, und es verhält sich auch nicht wie ein anderer Wert.

Wenn Sie NULL in numerischen, String- oder Datums/Zeit-Ausdrücken verwenden, wird das Ergebnis immer NULL sein. Verwenden Sie NULL in logischen (Boolean) Ausdrücken, hängt das Ergebnis von der Art der Operation ab und anderen partizipierenden Werten. Wenn Sie einen Wert mit NULL vergleichen, wird das Ergebnis *unknown* sein.

Zu beachten

NULL heißt NULL, jedoch gilt in Firebird, dass das logische Ergebnis *unknown* ebenfalls durch NULL *repräsentiert* wird.

Ausdrücke die NULL zurückgeben

Ausdrücke in dieser Liste werden immer NULL zurückgeben:

```
1 + 2 + 3 + NULL
'Home ' || 'sweet ' || NULL
MyField = NULL
MyField <> NULL
NULL = NULL
not (NULL)
```

Wenn es Ihnen schwerfällt dies zu verstehen, beachten Sie, dass NULL ein Status ist, der für „unknown“ (unbekannt) steht.

NULL in logischen Ausdrücken

Es wurde bereits gezeigt, dass `not (NULL)` in `NULL` aufgeht. Dieser Effekt ist etwas komplizierter für logische AND- sowie logische OR-Operatoren:

```
NULL or false = NULL
NULL or true = true
NULL or NULL = NULL
NULL and false = false
NULL and true = NULL
NULL and NULL = NULL
```

Bis einschließlich Firebird 2.5.x existiert keine Implementierung für logische (Boolean) Datentypen—diese gibt es erst seit Firebird 3. Jedoch gibt es logische Ausdrücke (Prädikate), die `true`, `false` oder `unknown` zurückgeben können.

Beispiele:

```
(1 = NULL) or (1 <> 1) -- liefert NULL
(1 = NULL) or (1 = 1) -- liefert TRUE
(1 = NULL) or (1 = NULL) -- liefert NULL
(1 = NULL) and (1 <> 1) -- liefert FALSE
(1 = NULL) and (1 = 1) -- liefert NULL
(1 = NULL) and (1 = NULL) -- liefert NULL
```

Unterabfragen

Eine Unterabfrage ist eine spezielle Form eines Ausdrucks, die innerhalb einer anderen Abfrage eingebettet wird. Unterabfragen werden in der gleichen Weise geschrieben wie reguläre `SELECT`-Abfragen, werden jedoch von Klammern umschlossen. Unterabfrage-Ausdrücke können in folgender Art und Weise verwendet werden:

- Um eine Ausgabespalte in der `SELECT`-Liste anzugeben
- Um Werte zu holen oder als Kriterium für Suchprädikate (die `WHERE`- und `HAVING`-Klauseln)
- Um ein Set zu erstellen, das die Eltern-Abfrage verwenden kann, so als wäre dies eine reguläre Tabelle oder View. Unterabfragen wie diese erscheinen in der `FROM`-Klausel (Derived Tables) oder in einer Common Table Expression (CTE)

Korrelierte Unterabfragen

Eine Unterabfrage kann *korrelierend* sein. Sie ist korrelierend, wenn die Hauptabfrage und die Unterabfrage voneinander abhängig sind. Um einen Datensatz in der Unterabfrage zu verarbeiten, ist es notwendig einen Datensatz in der Hauptabfrage zu holen; beispielsweise hängt die Unterabfrage vollständig von der Hauptabfrage ab.

Beispiel einer korrelierenden Unterabfrage:

```
SELECT *
FROM Customers C
WHERE EXISTS
  (SELECT *
   FROM Orders O
   WHERE C.cnum = O.cnum
        AND O.odate = DATE '10.03.1990');
```

Werden Unterabfragen verwendet um Werte einer Ausgabespalte aus einer SELECT-Liste zu holen, muss die Unterabfrage ein *skalares* Ergebnis zurückliefern.

Skalare Ergebnisse

Unterabfragen, die in Suchprädikaten verwendet werden, mit Ausnahme von existenziellen und quantifizierten Prädikaten, müssen ein *skalares* Ergebnis zurückgeben; Das heißt, nicht mehr als eine Spalte von nicht mehr als einer passenden Zeile oder Aggregation. Sollte mehr zurückgegeben werden, wird es zu einem Laufzeitfehler kommen („Multiple rows in a singleton select..“).

Anmerkung

Obwohl es einen echten Fehler berichtet, kann die Nachricht etwas irreführend sein. Ein „singleton SELECT“ ist eine Abfrage, die nicht mehr als eine Zeile zurückgeben kann. Jedoch sind „singleton“ und „skalar“ nicht gleichzusetzen: nicht alle singleton SELECTs müssen zwangsläufig skalar sein; und Einspalten-SELECTs können mehrere Zeilen für existenzielle und quantifizierte Prädikate zurückgeben.

Unterabfrage-Beispiele:

1. Eine Unterabfrage als Ausgabespalte in einer SELECT-Liste:

```
SELECT
  e.first_name,
  e.last_name,
  (SELECT
    sh.new_salary
  FROM
    salary_history sh
  WHERE
    sh.emp_no = e.emp_no
  ORDER BY sh.change_date DESC ROWS 1) AS last_salary
FROM
  employee e
```

2. eine Unterabfrage in der WHERE-Klausel, um das höchste Gehalt eines Mitarbeiters zu ermitteln und hierauf zu filtern:

```
SELECT
  e.first_name,
  e.last_name,
  e.salary
FROM
```



```
employee e
WHERE
  e.salary = (
    SELECT MAX(ie.salary)
    FROM employee ie
  )
```

Prädikate

Ein Prädikat ist ein einfacher Ausdruck, der eine Behauptung aufstellt, wir nennen sie P . Wenn P zu TRUE (wahr) aufgelöst wird, ist die Behauptung erfolgreich. Wird sie zu FALSE (unwahr, falsch) oder NULL (UNKNOWN) aufgelöst, ist die Behauptung falsch. Hier gibt es einen Fallstrick: Nehmen wir an, das Prädikat P gibt FALSE zurück. In diesem Falle gilt, dass NOT(P) TRUE zurückgeben wird. Andererseits gilt, falls P NULL (unknown) zurückgibt, dann gibt NOT(P) ebenfalls NULL zurück.

In SQL können Prädikate in CHECK-Constraints auftreten, WHERE- und HAVING-Klauseln, CASE-Ausdrücken, der IIF()-Funktion und in der ON-Bedingung der JOIN-Klausel.

Behauptungen

Eine Behauptung ist ein Statement über Daten, die, wie ein Prädikat, zu TRUE, FALSE oder NULL aufgelöst werden können. Behauptungen bestehen aus einem oder mehr Prädikaten, möglicherweise mittels NOT negiert und verbunden durch AND- sowie ODER-Operatoren. Klammern können verwendet werden um Prädikate zu gruppieren und die Ausführungsreihenfolge festzulegen.

Ein Prädikat kann andere Prädikate einbetten. Die Ausführung ist nach außen gerichtet, das heißt, das innenliegende Prädikat wird zuerst ausgeführt. Jede „Ebene“ wird in ihrer Rangfolge ausgewertet bis der Wahrheitsgehalt der endgültigen Behauptung aufgelöst wird.

Vergleichs-Prädikate

Ein Vergleichsprädikat besteht aus zwei Ausdrücken, die mit einem Vergleichsoperator verbunden sind. Es existieren traditionel sechs Vergleichsoperatoren:

=, >, <, >=, <=, <>

(Für die vollständige Liste der Vergleichsoperatoren mit ihren Variantenformen siehe [Vergleichsoperatoren](#).)

Wenn eine der Seiten (links oder rechts) eines Vergleichsprädikats NULL darin hat, wird der Wert des Prädikats UNKNOWN.

Beispiele:

1. Abrufen von Informationen über Computer mit der CPU-Frequenz nicht weniger als 500 MHz und der Preis niedriger als \$800:

```
SELECT *
```

```
FROM Pc
WHERE speed >= 500 AND price < 800;
```

2. Abrufen von Informationen über alle Punktmatrixdrucker, die weniger als \$300 kosten:

```
SELECT *
FROM Printer
WHERE ptrtype = 'matrix' AND price < 300;
```

3. Die folgende Abfrage gibt keine Daten zurück, auch nicht wenn es Drucker ohne zugewiesenen Typ gibt, da ein Prädikat, das NULL mit NULL vergleicht, NULL zurückgibt:

```
SELECT *
FROM Printer
WHERE ptrtype = NULL AND price < 300;
```

Auf der anderen Seite kann *ptrtype* auf NULL getestet werden; mit dem Ergebnis, dass die kein *Vergleichstest* ist:

```
SELECT *
FROM Printer
WHERE ptrtype IS NULL AND price < 300;
```

—siehe [IS \[NOT\] NULL](#).

Hinweis zu String-Vergleichen

Werden CHAR- und VARCHAR-Felder auf Gleichheit verglichen, werden nachfolgende Leerzeichen in allen Fällen ignoriert.

Andere Vergleichsprädikate

Andere Vergleichsprädikate werden durch Schlüsselwörter gekennzeichnet.

BETWEEN

Verfügbar: DSQL, PSQL, ESQL

Syntax:

```
<value> [NOT] BETWEEN <value_1> AND <value_2>
```

Das BETWEEN-Prädikat prüft, ob ein Wert innerhalb eines angegebenen Bereichs zweier Werte liegt. (NOT BETWEEN prüft, ob dieser Wert außerhalb der beiden Grenzen liegt.)

Die Operanden des BETWEEN-Prädikates sind zwei Argumente kompatibler Datentypen. Anders als in anderen DBMS ist das BETWEEN-Prädikat nicht symmetrisch— ist der kleinere Wert nicht das erste Argument, wird

immer FALSE zurückgegeben. Die Suche ist inkludiert (die Werte beider Argumente werden in die Suche eingebunden). Anders ausgedrückt bedeutet dies, dass das BETWEEN-Prädikat auch anders geschrieben werden kann:

```
<value> >= <value_1> AND <value> <= <value_2>
```

Wird BETWEEN in Suchkriterien für DML-Abfragen verwendet, kann der Firebird-Optimizer einen Index auf der Suchspalte nutzen, sofern verfügbar.

Beispiel:

```
SELECT *
FROM EMPLOYEE
WHERE HIRE_DATE BETWEEN date '01.01.1992' AND CURRENT_DATE
```

LIKE

Verfügbar: DSQL, PSQL, ESQL

Syntax:

```
<match value> [NOT] LIKE <pattern>
  [ESCAPE <escape character>]

<match value> ::= character-type expression

<pattern> ::= search pattern

<escape character> ::= escape character
```

Das LIKE-Prädikat vergleicht zeichenbasierte Ausdrücke mit dem im zweiten Ausdruck definierten Muster. Groß- und Kleinschreibung bzw. Akzent-Sensitivität für den Vergleich wird durch die zugrunde liegende Collation bestimmt. Eine Collation kann für jeden Operanden angegeben werden, wenn erforderlich.

Wildcards

Zwei Wildcard-Zeichen sind für die Suche verfügbar:

- Das Prozentzeichen (%) berücksichtigt alle Sequenzen von null oder mehr Zeichen im getesteten Wert
- Das Unterstrichzeichen (_) berücksichtigt jedes beliebige Einzelzeichen im getesteten Wert

Wenn der getestete Wert dem Muster entspricht, unter Berücksichtigung von Wildcard-Zeichen ist das Prädikat TRUE.

Verwendung der ESCAPE-Zeichen-Option

Wenn der Such-String eines der Wildcard-Zeichen beinhaltet, kann die ESCAPE-Klausel verwendet werden, um ein Escape-Zeichen zu definieren. Das Escape-Zeichen muss dem '%' oder '_' Symbol im Suchstring vorangestellt werden, um anzuzeigen, dass das Symbol als wörtliches Zeichen interpretiert werden soll.

Beispiele für LIKE:

1. Finde die Nummern der Abteilung, deren Namen mit dem Wort „Software“ starten:

```
SELECT DEPT_NO
FROM DEPT
WHERE DEPT_NAME LIKE 'Software%';
```

Es ist möglich einen Index für das Feld DEPT_NAME zu verwenden, sofern dieser existiert.

Über LIKE und den Optimizer

Eigentlich verwendet das LIKE-Prädikat keinen Index. Wird das Prädikat jedoch in Form von LIKE 'string%' verwendet, wird dieses zum Prädikat STARTING WITH konvertiert, welches einen Index verwendet.

Somit gilt—wenn Sie nach einem Wortanfang suchen, sollten Sie das Prädikat STARTING WITH anstelle von LIKE verwenden.

2. Suche nach Mitarbeitern deren Namen aus 5 Buchstaben bestehen, die mit „Sm“ beginnen und mit „th“ enden. Das Prädikat wird wahr für die Namen wie „Smith“ und „Smyth“.

```
SELECT
    first_name
FROM
    employee
WHERE first_name LIKE 'Sm_th'
```

3. Suche nach allen Mandanten, deren Adresse den String „Rostov“ enthält:

```
SELECT *
FROM CUSTOMER
WHERE ADDRESS LIKE '%Rostov%'
```

Anmerkung

Benötigen Sie eine Suche, die Groß- und Kleinschreibung *innerhalb* einer Zeichenkette ignoriert (LIKE '%Abc%'), sollten Sie das CONTAINING-Prädikat, anstelle des LIKE-Prädikates, verwenden.

4. Suche nach Tabellen, die das Unterstrich-Zeichen im Namen besitzen. Das Zeichen „#“ wird als Escape-Zeichen definiert:

```
SELECT
    RDB$RELATION_NAME
FROM RDB$RELATIONS
WHERE RDB$RELATION_NAME LIKE '%#_%' ESCAPE '#'
```

Siehe auch: [STARTING WITH](#), [CONTAINING](#), [SIMILAR TO](#)

STARTING WITH

Verfügbar: DSQL, PSQL, ESQL

Syntax:

```
<value> [NOT] STARTING WITH <value>
```

Das Prädikat STARTING WITH sucht nach einer Zeichenkette oder einem zeichenkettenähnlichen Datentyp, die mit den Zeichen des Argumentes <value> beginnt. Die Suche unterscheidet zwischen Groß- und Kleinschreibung.

Wenn STARTING WITH als Suchkriterium in DML-Abfragen verwendet wird, nutzt der Firebird-Optimizer einen Index auf der Suchspalte, sofern vorhanden.

Beispiel: Suche nach Mitarbeitern deren Namen mit „Jo“ beginnen:

```
SELECT LAST_NAME, FIRST_NAME
FROM EMPLOYEE
WHERE LAST_NAME STARTING WITH 'Jo'
```

Siehe auch: [LIKE](#)

CONTAINING

Verfügbar: DSQL, PSQL, ESQL

Syntax:

```
<value> [NOT] CONTAINING <value>
```

Das Prädikat CONTAINING sucht innerhalb von Zeichenketten oder zeichenkettenähnlichen Datentypen nach der Buchstabenfolge, die im Argument angegeben wurde. Es kann für alphanumerische (zeichenkettenähnliche) Suchen auf Zahlen und Daten genutzt werden. Eine Suche mit CONTAINING unterscheidet nicht nach Groß- und Kleinschreibung. Wird jedoch eine akzentsensitive Collation verwendet, ist auch die Suche akzentsensitiv.

Wenn CONTAINING als Suchkriterium in DML-Abfragen verwendet wird, kann der Firebird-Optimizer einen Index der Suchspalte nutzen, sofern ein passender existiert.

Beispiele:

1. Suche nach Projekten, deren Namen die Zeichenkette „Map“ enthalten:

```
SELECT *
FROM PROJECT
WHERE PROJ_NAME CONTAINING 'Map' ;
```

Zwei Zeilen mit den Namen „AutoMap“ und „MapBrowser port“ werden zurückgegeben.

- Suche nach Änderungen in den Gehältern, die die Zahl 84 im Datum enthalten (in diesem Falle heißt dies, Änderungen im Jahr 1984):

```
SELECT *
FROM SALARY_HISTORY
WHERE CHANGE_DATE CONTAINING 84;
```

Siehe auch: [LIKE](#)

SIMILAR TO

Verfügbar: DSQL, PSQL

Syntax:

```
string-expression [NOT] SIMILAR TO <pattern> [ESCAPE <escape-char>]
<pattern> ::= an SQL regular expression
<escape-char> ::= a single character
```

SIMILAR TO findet eine Zeichenkette anhand eines Regulären Ausdruck-Musters in SQL (engl. SQL Regular Expression Pattern). Anders als in einigen anderen Sprachen muss das Muster mit der gesamten Zeichenkette übereinstimmen, um erfolgreich zu sein - die Übereinstimmung eines Teilstrings reicht nicht aus. Ist ein Operand NULL, ist auch das Ergebnis NULL. Andernfalls ist das Ergebnis TRUE oder FALSE.

Syntax: SQL Reguläre Ausdrücke

Die folgende Syntax definiert das SQL-Standardausdruckformat. Es ist eine komplette und korrekte Top-down-Definition. Es ist auch sehr formell, ziemlich lang und vermutlich perfekt geeignet, jeden zu entmutigen, der nicht schon Erfahrungen mit Regulären Ausdrücken (oder mit sehr formalen, eher langen Top-down-Definitionen) gesammelt hat. Fühlen Sie sich frei, dies zu überspringen und den nächsten Abschnitt, [Aufbau Regulärer Ausdrücke](#), zu lesen, der einen Bottom-up-Ansatz verfolgt und sich an den Rest von uns richtet.

```
<regular expression> ::= <regular term> ['|' <regular term> ...]
<regular term> ::= <regular factor> ...
<regular factor> ::= <regular primary> [<quantifier>]
<quantifier> ::= ?
                | *
                | +
                | '{' <m> [, [<n>]] '}'
<m>, <n> ::= unsigned int, mit <m> <= <n> wenn beide vorhanden
<regular primary> ::= <character>
```

```

| <character class>
| %
| (<regular expression>)

<character> ::= <escaped character>
| <non-escaped character>

<escaped character> ::= <escape-char> <special character>
| <escape-char> <escape-char>

<special character> ::= eines der Zeichen []()|^-*%_?{

<non-escaped character> ::= ein Zeichen, das nicht ein <special character> ist
und nicht gleich <escape-char> (wenn definiert)

<character class> ::= '_'
| '[' <member> ... ']'
| '[' ^ <non-member> ... ']'
| '[' <member> ... '^' <non-member> ... ']'

<member>, <non-member> ::= <character>
| <range>
| <predefined class>

<range> ::= <character>-<character>

<predefined class> ::= '[' <predefined class name> ':'

<predefined class name> ::= ALPHA | UPPER | LOWER | DIGIT
| ALNUM | SPACE | WHITESPACE

```

Aufbau Regulärer Ausdrücke

Dieser Abschnitt behandelt die Elemente und Regeln zum Aufbau Regulärer Ausdrücke in SQL.

Zeichen

Innerhalb Regulärer Ausdrücke repräsentieren die meisten Zeichen sich selbst. Die einzige Ausnahme bilden die folgenden Zeichen:

```
[ ] ( ) | ^ - + * % _ ? { }
```

...und das Escape-Zeichen, sofern definiert.

Ein Regulärer Ausdruck, der keine Sonderzeichen oder Escape-Zeichen beinhaltet, findet nur Strings, die identisch zu sich selbst sind (abhängig von der verwendeten Collation). Das heißt, es agiert wie der „="-Operator:

```

'Apple' similar to 'Apple' -- true
'Apples' similar to 'Apple' -- false
'Apple' similar to 'Apples' -- false
'APPLE' similar to 'Apple' -- abhängig von der Collation

```

Wildcards

Die bekannten SQL-Wildcards `_` und `%` finden beliebige Einzelzeichen und Strings beliebiger Länge:

```
'Birne' similar to 'B_rne'    -- true
'Birne' similar to 'B_ne'    -- false
'Birne' similar to 'B%ne'    -- true
'Birne' similar to 'Bir%ne%' -- true
'Birne' similar to 'Birr%ne' -- false
```

Beachten Sie, wie `%` auch den leeren String berücksichtigt.

Zeichenklassen

Ein Bündel von Zeichen, die in Klammern eingeschlossen sind, definiert eine Zeichenklasse. Ein Zeichen in der Zeichenfolge entspricht einer Klasse im Muster, wenn das Zeichen Mitglied der Klasse ist:

```
'Citroen' similar to 'Cit[arju]oen'    -- true
'Citroen' similar to 'Ci[tr]oen'      -- false
'Citroen' similar to 'Ci[tr][tr]oen'  -- true
```

Wie aus der zweiten Zeile ersichtlich ist, entspricht die Klasse nur einem einzigen Zeichen, nicht einer Sequenz.

Innerhalb einer Klassendefinition definieren zwei Zeichen, die durch einen Bindestrich verbunden sind, einen Bereich. Ein Bereich umfasst die beiden Endpunkte und alle Zeichen, die zwischen ihnen in der aktiven Sortierung liegen. Bereiche können überall in der Klassendefinition ohne spezielle Begrenzer platziert werden, um sie von den anderen Elementen zu trennen.

```
'Datte' similar to 'Dat[q-u]e'        -- true
'Datte' similar to 'Dat[abq-uy]e'     -- true
'Datte' similar to 'Dat[bcg-km-pwz]e' -- false
```

Vordefinierte Zeichenklassen

Die folgenden vordefinierten Zeichenklassen können auch in einer Klassendefinition verwendet werden:

`[[:ALPHA:]]`: Lateinische Buchstaben a..z und A..Z. Mit einer akzentunempfindlichen Sortierung stimmt diese Klasse auch mit akzentuierten Formen dieser Zeichen überein.

`[[:DIGIT:]]`: Dezimalziffern 0..9.

`[[:ALNUM:]]`: Gesamtheit aus `[[:ALPHA:]]` und `[[:DIGIT:]]`.

`[[:UPPER:]]`: Großgeschriebene Form der lateinischen Buchstaben A..Z. Findet auch kleingeschriebene Strings mit groß- und kleinschreibunempfindlicher Collation sowie akzentunempfindlicher Collation.

`[[:LOWER:]]`: Kleingeschriebene Form der lateinischen Buchstaben A..Z. Findet auch großgeschriebene Strings mit groß- und kleinschreibunempfindlicher Collation sowie akzentunempfindlicher Collation.

[*:SPACE:*]: Findet das Leerzeichen (ASCII 32).

[*:WHITESPACE:*]: Findet horizontalen Tabulator (ASCII 9), Zeilenvorschub (ASCII 10), vertikalen Tabulator (ASCII 11), Seitenvorschub (ASCII 12), Wagenrücklauf (ASCII 13) und Leerzeichen (ASCII 32).

Das Einbinden einer vordefinierten Klasse hat den gleichen Effekt wie das Einbinden all seiner Mitglieder. Vordefinierte Klassen sind nur in Klassendefinitionen erlaubt. Wenn Sie gegen eine vordefinierte Klasse prüfen und gegen nichts sonst, platzieren Sie ein zusätzliches Paar von Klammern um sie herum.

```
'Erdbeere' similar to 'Erd[[:ALNUM:]]eere'      -- true
'Erdbeere' similar to 'Erd[[:DIGIT:]]eere'     -- false
'Erdbeere' similar to 'Erd[a[:SPACE:]b]eere'   -- true
'Erdbeere' similar to [[:ALPHA:]]             -- false
'E'      similar to [[:ALPHA:]]                -- true
```

Wenn eine Klassendefinition mit einer eckigen Klammer beginnt, wird alles, was folgt, von der Klasse ausgeschlossen. Alle anderen Zeichen entsprechen:

```
'Framboise' similar to 'Fra[^ck-p]boise'      -- false
'Framboise' similar to 'Fr[^a][^a]boise'     -- false
'Framboise' similar to 'Fra^[[:DIGIT:]]boise' -- true
```

Wird die eckige Klammer nicht am Anfang der Reihe platziert, enthält die Klasse alles vor dieser, mit Ausnahme der Elemente die nach der Klammer vorkommen:

```
'Grapefruit' similar to 'Grap[a-m^f-i]fruit'  -- true
'Grapefruit' similar to 'Grap[abc^xyz]fruit'  -- false
'Grapefruit' similar to 'Grap[abc^de]fruit'   -- false
'Grapefruit' similar to 'Grap[abe^de]fruit'   -- false

'3' similar to '[[[:DIGIT:]]^4-8]'            -- true
'6' similar to '[[[:DIGIT:]]^4-8]'            -- false
```

Zuletzt sei noch erwähnt, dass die Wildcard-Zeichen „_“ eine eigene Zeichenklasse sind, die einem beliebigen einzelnen Zeichen entspricht.

Bezeichner

Ein Fragezeichen, direkt von einem weiteren Zeichen oder Klasse gefolgt, gibt an, dass das folgende Element gar nicht oder einmalig vorkommen darf:

```
'Hallon' similar to 'Hal?on'                  -- false
'Hallon' similar to 'Hal?lon'                 -- true
'Hallon' similar to 'Hall1?on'                -- true
'Hallon' similar to 'Hall11?on'              -- false
'Hallon' similar to 'Halx?lon'                -- true
'Hallon' similar to 'H[a-c]?llon[x-z]?'      -- true
```

Ein Sternchen, direkt von einem weiteren Zeichen oder Klasse gefolgt, gibt an, dass das folgende Element gar nicht oder mehrmals vorkommen darf:

```
'Icaque' similar to 'Ica*que'           -- true
'Icaque' similar to 'Icar*que'         -- true
'Icaque' similar to 'I[a-c]*que'      -- true
'Icaque' similar to '_*'              -- true
'Icaque' similar to '[:ALPHA:]*'      -- true
'Icaque' similar to 'Ica[xyz]*e'      -- false
```

Ein Plus-Zeichen, direkt von einem weiteren Zeichen oder Klasse gefolgt, gibt an, dass das folgende Element einmalig oder mehrmals vorkommen darf:

```
'Jujube' similar to 'Ju_+'            -- true
'Jujube' similar to 'Ju+jube'         -- true
'Jujube' similar to 'Jujuber+'       -- false
'Jujube' similar to 'J[jux]+be'      -- true
'Jujube' similar to 'J[[:DIGIT:]]+ujube' -- false
```

Folgt eine Zahl in Klammern auf ein Zeichen oder eine Klasse, muss letzteres genau so oft wie angegeben vorkommen:

```
'Kiwi' similar to 'Ki{2}wi'           -- false
'Kiwi' similar to 'K[ipw]{2}i'        -- true
'Kiwi' similar to 'K[ipw]{2}'         -- false
'Kiwi' similar to 'K[ipw]{3}'         -- true
```

Wird eine Zahl von einem Komma gefolgt, bedeutet dies, dass das Element mindestens so oft wie angegeben vorkommen muss:

```
'Limone' similar to 'Li{2,}mone'     -- false
'Limone' similar to 'Li{1,}mone'     -- true
'Limone' similar to 'Li[nezm]{2,}'    -- true
```

Wenn die Klammern zwei Zahlen enthalten, die mittels Komma getrennt sind, die zweite Zahl nicht kleiner als die erste ist, muss das Element mindestens so oft wie die erste Zahl vorkommen und maximal so oft wie in der zweiten Zahl angegeben:

```
'Mandarijn' similar to 'M[a-p]{2,5}rijn' -- true
'Mandarijn' similar to 'M[a-p]{2,3}rijn' -- false
'Mandarijn' similar to 'M[a-p]{2,3}arijn' -- true
```

Die Bezeichner `?`, `*` und `+` sind Kurzschreibweisen für `{0,1}`, `{0,}` und `{1,}`.

Oder-verknüpfte Terme

Reguläre Ausdrücke können Oder-verknüpft werden mittels `|`-Operator. Eine Gesamtübereinstimmung tritt auf, wenn die Argumentzeichenkette mit mindestens einem Term übereinstimmt.

```
'Nektarin' similar to 'Nek|tarin'     -- false
'Nektarin' similar to 'Nektarin|Persika' -- true
'Nektarin' similar to 'M_+|N_+|P_+'    -- true
```

Unterausdrücke

Ein oder mehrere Teile der regulären Ausdrücke können in Unterausdrücke gruppiert werden (auch Untermuster genannt), indem diese in runde Klammern eingeschlossen werden. Ein Unterausdruck ist ein eigener regulärer Ausdruck. Dieser kann alle erlaubten Elemente eines regulären Ausdrucks enthalten, und auch eigene Bezeichner.

```
'Orange' similar to 'O(ra|ri|ro)nge'      -- true
'Orange' similar to 'O(r[a-e])+nge'      -- true
'Orange' similar to 'O(ra){2,4}nge'      -- false
'Orange' similar to 'O(r(an|in)g|rong)?e' -- true
```

Sonderzeichen escapen

Soll eine Übereinstimmung auf Sonderzeichen innerhalb eines regulären Ausdrucks geprüft werden, muss dieses Zeichen escaped werden. Es gibt kein Standard-Escape-Zeichen; stattdessen definiert der Benutzer eines, wenn dies benötigt wird:

```
'Peer (Poire)' similar to 'P[^ ]+ \ (P[^ ]+\)' escape '\\' -- true
'Pera [Pear]' similar to 'P[^ ]+ # [P[^ ]+#]' escape '#' -- true
'Päron-Äppledryck' similar to 'P%$-Ä%' escape '$' -- true
'Pärondryck' similar to 'P%--Ä%' escape '-' -- false
```

Die letzte Zeile demonstriert, dass das Escape-Zeichen auch sich selbst escapen kann, wenn notwendig.

IS [NOT] DISTINCT FROM

Verfügbar: DSQL, PSQL

Syntax:

```
operand1 IS [NOT] DISTINCT FROM operand2
```

Zwei Operanden werden als DISTINCT angesehen, wenn sie unterschiedliche Werte besitzen oder wenn einer NULL ist und der andere nicht-NULL. Sie werden als NOT DISTINCT angesehen, wenn sie den gleichen Wert besitzen oder beide Operanden NULL sind.

Siehe auch: [IS \[NOT\] NULL](#)

IS [NOT] NULL

Verfügbar: DSQL, PSQL, ESQL

Syntax:

```
<value> IS [NOT] NULL
```

Da NULL kein Wert ist, sind diese Operatoren keine Vergleichsoperatoren. Das Prädikat IS [NOT] NULL prüft die Behauptung, dass der Ausdruck auf der linken Seite einen Wert (IS NOT NULL) oder keinen Wert hat (IS NULL).

Beispiel: Suche nach Verkäufen, die kein Versanddatum besitzen:

```
SELECT * FROM SALES
WHERE SHIP_DATE IS NULL;
```

Hinweis bezüglich des IS-Prädikates

Bis einschließlich Firebird 2.5, hat das Prädikat IS, wie andere Vergleichsprädikate, keinen Vorrang gegenüber anderer. Ab Firebird 3.0 hat dieses Prädikat Vorrang gegenüber den anderen.

Existenzprädikate

Diese Gruppe von Prädikaten beinhaltet die, die Unterabfragen nutzen um Werte für alle möglichen Arten von Behauptungen zu prüfen. Existenzprädikate werden so genannt, da sie verschiedene Methoden verwenden, um auf die *Existenz* oder *nicht-Existenz* von Behauptungen zu prüfen. Die Prädikate geben TRUE zurück, wenn die Existenz oder nicht-Existenz bestätigt wurde, andernfalls FALSE.

EXISTS

Verfügbar: DSQL, PSQL, ESQL

Syntax:

```
[NOT] EXISTS(<select_stmt>)
```

Das Prädikat EXISTS nutzt einen Unterabfrage-Ausdruck als Argument. Es gibt TRUE zurück, wenn die Unterabfrage mindestens einen Datensatz zurückgibt; andernfalls gibt es FALSE zurück.

NOT EXISTS gibt FALSE zurück, wenn die Unterabfrage mindestens eine Datenzeile zurückgibt; es gibt andernfalls TRUE zurück.

Anmerkung

Die Unterabfrage kann mehrere Spalten enthalten, oder SELECT *, da die Prüfung anhand der zurückgegebenen Datenzeilen vorgenommen wird, die die Bedingungen erfüllen.

Beispiele:

1. Finde die Mitarbeiter, die Projekte haben.

```
SELECT *
FROM employee
WHERE EXISTS(SELECT *
              FROM employee_project ep
              WHERE ep.emp_no = employee.emp_no)
```

2. Finde die Mitarbeiter, die keine Projekte haben.

```
SELECT *
FROM employee
WHERE NOT EXISTS(SELECT *
                  FROM employee_project ep
                  WHERE ep.emp_no = employee.emp_no)
```

IN

Verfügbar: DSQL, PSQL, ESQL

Syntax:

```
<value> [NOT] IN(<select_stmt> | <value_list>)
<value_list> ::= <value_1> [, <value_2> ...]
```

Das Prädikat IN prüft, ob der Wert des Ausdrucks auf der linken Seite im Wertesatz der rechten Seite vorkommt. Der Wertesatz darf nicht mehr als 1500 Elemente enthalten. Das IN-Prädikat kann mit folgender äquivalenter Form ersetzt werden:

```
(<value> = <value_1> [OR <value> = <value_2> ...])
```

Wenn das Prädikat IN als Suchbedingung in DML-Abfragen verwendet wird, kann der Firebird-Optimizer einen Index auf die Suchspalte nutzen, sofern einer vorhanden ist.

In seiner zweiten Form prüft das Prädikat IN, ob der linke Ausdruckswert im Ergebnis der Unterabfrage vorhanden ist (oder nicht vorhanden, wenn NOT IN verwendet wird).

Die Unterabfrage darf nur eine Spalte abfragen, andernfalls wird es zum Fehler „count of column list and variable list do not match“ kommen.

Abfragen, die das Prädikat IN mit einer Unterabfrage verwenden, können durch eine ähnliche Abfrage mittels des EXISTS-Prädikates ersetzt werden. Zum Beispiel folgende Abfrage:

```
SELECT
  model, speed, hd
FROM PC
WHERE
  model IN (SELECT model
            FROM product)
```

```
WHERE maker = 'A');
```

kann ersetzt werden mittels EXISTS-Prädikat

```
SELECT
  model, speed, hd
FROM PC
WHERE
  EXISTS (SELECT *
          FROM product
          WHERE maker = 'A'
          AND product.model = PC.model);
```

Jedoch gilt zu beachten, dass eine Abfrage mittels NOT IN und einer Unterabfrage nicht immer das gleiche Ergebnis zurückliefert wie sein Gegenpart mit NOT EXISTS. Dies liegt daran, dass EXISTS immer TRUE oder FALSE zurückgibt, wohingegen IN NULL in diesen beiden Fällen zurückliefert:

1. wenn der geprüfte Wert NULL ist und die IN ()-Liste nicht leer ist
 2. wenn der geprüfte Wert keinen Treffer in der IN ()-Liste enthält und mindestens ein Element NULL ist.
- Nur in diesen beiden Fällen wird IN () NULL zurückgeben, während das EXISTS-Prädikat FALSE zurückgibt ('keine passende Zeile gefunden', engl. 'no matching row found'). In einer Suche oder, zum Beispiel in einem IF (...) -Statement, bedeuten beide Ergebnisse einen „Fehler“ und es macht damit keinen Unterschied.

Aber für die gleichen Daten gibt NOT IN () NULL zurück, während NOT EXISTS TRUE zurückgibt, was das Gegenteilige Ergebnis ist.

Schauen wir uns das folgendes Beispiel an:

```
-- Suche nach Bürgern die nicht am gleichen Tag wie eine
-- berühmte New Yorker Persönlichkeit geboren wurden
SELECT P1.name AS NAME
FROM Personnel P1
WHERE P1.birthday NOT IN (SELECT C1.birthday
                        FROM Celebrities C1
                        WHERE C1.birthcity = 'New York');
```

Nehmen wir nun an, dass die Liste der New Yorker Berühmtheiten nicht leer ist und mindestens einen NULL-Geburtstag aufweist. Dann gilt für alle Bürger, die nicht am gleichen Tag mit einer Berühmtheit Geburtstag haben, dass NOT IN NULL zurückgibt, da dies genau das ist was IN tut. Die Suchbedingung wurde nicht erfüllt und die Bürger werden nicht im Ergebnis des SELECT berücksichtigt, da die Aussage falsch ist.

Bürger, die am gleichen Tag wie eine Berühmtheit Geburtstag feiern, wird NOT IN korrekterweise FALSE zurückgeben, womit diese ebenfalls aussortiert werden, und damit keine Zeile zurückgegeben wird.

Wird die Form NOT EXISTS verwendet:

```
-- Suche nach Bürgern, die nicht am gleichen Tag wie eine
-- berühmte New Yorker Persönlichkeit geboren wurden
SELECT P1.name AS NAME
FROM Personnel P1
WHERE NOT EXISTS (SELECT *
                 FROM Celebrities C1
                 WHERE C1.birthcity = 'New York')
```

```
AND C1.birthday = P1.birthday);
```

nicht-Übereinstimmungen werden im NOT EXISTS-Ergebnis TRUE erhalten und ihre Datensätze landen im Rückgabesatz.

Hinweis

Wenn es im Bereich des Möglichen liegt, dass NULL innerhalb der Suche für eine nicht-Prüfung vorhanden sein kann, sollten Sie NOT EXISTS verwenden.

Beispiele für die Verwendung:

1. Finde Mitarbeiter mit den Namen „Pete“, „Ann“ und „Roger“:

```
SELECT *
FROM EMPLOYEE
WHERE FIRST_NAME IN ('Pete', 'Ann', 'Roger');
```

2. Finde alle Computer, die deren Hersteller mit dem Buchstaben „A“ beginnt:

```
SELECT
    model, speed, hd
FROM PC
WHERE
    model IN (SELECT model
              FROM product
              WHERE maker STARTING WITH 'A');
```

Siehe auch: [EXISTS](#)

SINGULAR

Verfügbar: DSQL, PSQL, ESQL

Syntax:

```
[NOT] SINGULAR(<select_stmt>)
```

Das SINGULAR-Prädikat verwendet eine Unterabfrage als Argument und gibt True zurück, wenn diese exakt eine Datenzeile zurückgibt; andernfalls wird das Prädikat zu False aufgelöst. Die Unterabfrage kann mehrere Ausgabespalten beinhalten, da die Zeilen ja nicht wirklich ausgegeben werden. Sie werden nur auf (einzige) Existenz geprüft. Der Kürze halber, wird häufig nur 'SELECT *' verwendet. Das Prädikat SINGULAR kann nur zwei Werte zurückgeben: TRUE oder FALSE.

Beispiel: Finde die Mitarbeiter, die nur ein Projekt haben.

```
SELECT *
FROM employee
```

```
WHERE SINGULAR(SELECT *
                FROM
                 employee_project ep
                WHERE
                 ep.emp_no = employee.emp_no)
```

Quantifizierte Unterabfrage-Prädikate

Ein Quantifizierer ist ein logischer Operator, der die Anzahl der Objekte festlegt, für die diese Behauptung wahr ist. Es ist keine numerische Größe, sondern eine logische, die die Behauptung mit dem vollen Satz möglicher Objekte verbindet. Solche Prädikate basieren auf logischen universellen und existentiellen Quantifizierern, die in der formalen Logik erkannt werden.

In Unterabfrage-Ausdrücken machen es Quantifizierer-Prädikate möglich einzelne Werte mit den Ergebnissen von Unterabfragen zu vergleichen; sie besitzen die folgende gemeinsame Form:

```
<value expression> <comparison operator> <quantifier> <subquery>
```

ALL

Verfügbar: DSQL, PSQL, ESQL

Syntax:

```
<value> <op> ALL(<select_stmt>)
```

Wenn der ALL-Quantifizierer verwendet wird, ist das Prädikat TRUE, wenn jeder Wert, der von der Unterabfrage zurückgegeben wird, die Bedingung des Prädikates in der Hauptabfrage erfüllt ist.

Beispiel: Zeige nur jene Kunden an, deren Bewertungen höher sind als die Bewertung jedes Kunden in Paris.

```
SELECT c1.*
FROM Customers c1
WHERE c1.rating > ALL
      (SELECT c2.rating
       FROM Customers c2
       WHERE c2.city = 'Paris')
```

Wichtig

Wenn die Unterabfrage einen leeren Satz zurückgibt, ist das Prädikat TRUE für jeden linken Wert, unabhängig vom Operator. Dies mag widersprüchlich erscheinen, denn jeder linke Wert wird gegenüber dem rechten betrachtet als: kleiner als, größer als, gleich sowie ungleich.

Dennoch passt dies perfekt in die formale Logik: Wenn der Satz leer ist, ist das Prädikat 0 mal wahr, d.h. für jede Zeile im Satz.

ANY and SOME

Verfügbar: DSQL, PSQL, ESQL

Syntax:

```
<value> <op> {ANY | SOME} (<select_stmt>)
```

Die Quantifizierer ANY und SOME sind in ihrem Verhalten identisch. Offensichtlich sind beide im SQL-Standard vorhanden, so dass sie austauschbar verwendet werden können, um die Lesbarkeit der Operatoren zu verbessern. Wird der ANY- oder SOME-Quantifizierer verwendet, ist das Prädikat TRUE, wenn einer der zurückgegebenen Werte der Unterabfrage die Suchbedingung der Hauptabfrage erfüllt. Gibt die Unterabfrage keine Zeile zurück, wird das Prädikat automatisch als FALSE angesehen.

Beispiel: Zeige nur die Kunden, deren Bewertungen höher sind als die eines oder mehrerer Kunden in Rom.

```
SELECT *
FROM Customers
WHERE rating > ANY
  (SELECT rating
   FROM Customers
   WHERE city = 'Rome')
```

Statements der Data Definition (DDL)

DDL ist die Untermenge der SQL-Sprache von Firebird zum Festlegen von Datendefinitionen. DDL-Anweisungen werden zum Erstellen, Ändern und Löschen von Datenbankobjekten verwendet, die von Benutzern erstellt wurden. Wenn eine DDL-Anweisung committed wird, werden die Metadaten für die Objekte erstellt, geändert oder gelöscht.

DATABASE

In diesem Abschnitt wird beschrieben, wie Sie eine Datenbank erstellen, eine Verbindung zu einer vorhandenen Datenbank herstellen, die Dateistruktur einer Datenbank ändern und löschen. Außerdem wird erläutert, wie Sie eine Datenbank auf zwei verschiedene Arten sichern können und wie Sie die Datenbank in den „kopiersicheren“ Modus umwandeln können, um ein externes Backup sicher durchzuführen.

CREATE DATABASE

Benutzt für: Erstellen einer neuen Datenbank

Verfügbar in: DSQL, ESQL

Syntax:

```
CREATE {DATABASE | SCHEMA} '<filespec>'
[USER 'username' [PASSWORD 'password']]
[PAGE_SIZE [=] size]
[LENGTH [=] num [PAGE[S]]]
[SET NAMES 'charset']
[DEFAULT CHARACTER SET default_charset
 [COLLATION collation]] -- not supported in ESQL
[<sec_file> [<sec_file> ...]]
[DIFFERENCE FILE 'diff_file']; -- not supported in ESQL

<filespec> ::= [<server_spec>]{filepath | db_alias}

<server_spec> ::= servername [/{port/service}]: | \\servername\

<sec_file> ::= FILE 'filepath'
[LENGTH [=] num [PAGE[S]] [STARTING [AT [PAGE]] pagenum]
```

Tabelle 5.1. CREATE DATABASE Statement-Parameter

| Parameter | Beschreibung |
|-----------------|--|
| filespec | Dateispezifikation für primäre Datenbankdatei |
| server_spec | Spezifikation des Remoteservers im TCP / IP- oder Windows-Netzwerkstil. Enthält optional eine Portnummer oder einen Servicenamen |
| filepath | Vollständiger Pfad und Dateiname einschließlich seiner Erweiterung. Der Dateiname muss gemäß den Regeln des verwendeten Plattformatdateisystems angegeben werden. |
| db_alias | Datenbank-Alias, der zuvor in der Datei <code>aliases.conf</code> definiert wurde. |
| servername | Hostname oder IP-Adresse des Servers, auf dem die Datenbank erstellt werden soll. |
| username | Benutzername des Eigentümers der neuen Datenbank. Es kann aus bis zu 31 Zeichen bestehen. Groß- / Kleinschreibung ist vernachlässigbar. |
| password | Kennwort des Benutzers oder Datenbankeigentümers. Die maximale Länge beträgt 31 Zeichen; es werden jedoch nur die ersten 8 Zeichen berücksichtigt. Groß- / Kleinschreibung beachten. |
| size | Seitengröße der Datenbank in Bytes. Mögliche Werte sind 4096 (Standard), 8192 und 16384 |
| num | Maximale Größe des primären Datenbankdatei, oder einer sekundären, in Seiten (pages). |
| charset | Legt den Zeichensatz der Verbindung fest, die von einem Client verwendet wird, nachdem die Datenbank erfolgreich erstellt wurde. Einfache Anführungszeichen sind zu verwenden. |
| default_charset | Legt den Standardzeichensatz für String-Datentypen fest. |
| collation | Standard-Collation für den Standardzeichensatz |
| sec_file | Dateispezifikation für eine sekundäre Datei |
| pagenum | Startseitenzahl für eine sekundäre Datenbankdatei |
| diff_file | Dateipfad und -name für DIFFERENCE-Dateien (.delta-Dateien) |

Das Statement `CREATE DATABASE` erstellt eine neue Datenbank. Sie können sowohl `CREATE DATABASE` wie auch `CREATE SCHEMA` verwenden. Dies sind Synonyme füreinander.

Eine Datenbank besteht aus einer oder mehrerer Dateien. Die erste (Haupt-) Datei wird auch als die *primäre Datei* bezeichnet, folgende als *sekundäre Datei[en]*.

Multi-Datei-Datenbanken

Heutzutage gelten Multi-File-Datenbanken als rückschrittlich. Es ist sinnvoll, Datenbanken mit mehreren Dateien auf alten Dateisystemen zu verwenden, bei denen die Größe einer Datei begrenzt ist. Sie können beispielsweise keine Datei mit mehr als 4 GB auf FAT32 erstellen.

Die primäre Dateispezifikation ist der Name der Datenbankdatei und ihrer Erweiterung mit dem vollständigen Pfad zu den Regeln des verwendeten Betriebssystemplattformeisystems. Die Datenbankdatei darf zum Zeitpunkt der Datenbankerstellung nicht vorhanden sein. Wenn dies der Fall ist, erhalten Sie eine Fehlermeldung und die Datenbank wird nicht erstellt.

Wenn der vollständige Pfad zur Datenbank nicht angegeben ist, wird die Datenbank in einem der Systemverzeichnisse erstellt. Das bestimmte Verzeichnis hängt vom Betriebssystem ab. Geben Sie daher immer den absoluten Pfad an, wenn Sie entweder die Datenbank oder einen *Alias* dafür erstellen, es sei denn, Sie haben einen guten Grund, diese Situation zu bevorzugen.

Einen Datenbank-Alias verwenden Using a Database Alias

Sie können Aliasnamen anstelle des vollständigen Pfads zur primären Datenbankdatei verwenden. Aliase sind in der Datei `aliases.conf` im folgenden Format definiert:

```
alias = filepath
```

Eine Datenbank remote erstellen

Wenn Sie eine Datenbank auf einem Remoteserver erstellen, sollten Sie die Spezifikation des Remoteservers angeben. Die Spezifikation des Remoteservers hängt vom verwendeten Protokoll ab. Wenn Sie das TCP / IP-Protokoll zum Erstellen einer Datenbank verwenden, sollte die primäre Dateispezifikation wie folgt aussehen:

```
servername[/{port|service}]:{filepath | db_alias}
```

Wenn Sie das Named Pipes-Protokoll verwenden, um eine Datenbank auf einem Windows-Server zu erstellen, sollte die primäre Dateispezifikation wie folgt aussehen:

```
\\servername\{filepath | db_alias}
```

Optionale Parameter für CREATE DATABASE

Optional von USER und PASSWORD: Klauseln zur Angabe des Benutzernamens bzw. des Passworts eines vorhandenen Benutzers in der Sicherheitsdatenbank `security2.fdb`. Sie müssen den Benutzernamen und das Kennwort nicht angeben, wenn die Umgebungsvariablen `ISC_USER` und `ISC_PASSWORD` festgelegt sind. Der Benutzer, der beim Erstellen der Datenbank angegeben wird, wird ihr Eigentümer sein. Dies ist wichtig, wenn Sie Datenbank- und Objektberechtigungen berücksichtigen.

Optional PAGE_SIZE: Klausel zum Festlegen der Seitengröße der Datenbank. Diese Größe wird für die primäre Datei und alle sekundären Dateien der Datenbank festgelegt. Wenn Sie die Datenbankseitengröße unter 4.096 angeben, wird diese automatisch auf die Standardseitengröße 4.096 geändert. Andere Werte, die nicht 4.096, 8.192 oder 16.384 entsprechen, werden in den nächst kleineren unterstützten Wert geändert. Wenn die Größe der Datenbankseite nicht angegeben ist, wird der Standardwert auf 4.096 gesetzt.

Optional LENGTH: Klausel, die die maximale Größe der primären oder sekundären Datenbankdatei in Seiten angibt. Wenn eine Datenbank erstellt wird, belegen ihre primären und sekundären Dateien die Mindestanzahl an Seiten, die zum Speichern der Systemdaten erforderlich sind, unabhängig vom in der LENGTH-Klausel angegebenen Wert. Der Wert der LENGTH wirkt sich nicht auf die Größe der einzigen (oder zuletzt in einer Datei mit mehreren Dateien) Datei aus. Die Datei wird bei Bedarf automatisch vergrößert.

Optional SET NAMES: Klausel, die den Zeichensatz der Verbindung angibt, die verfügbar ist, nachdem die Datenbank erfolgreich erstellt wurde. Der Zeichensatz NONE wird standardmäßig verwendet. Beachten Sie, dass der Zeichensatz in einem Apostroph-Paar eingeschlossen sein sollte (einfache Anführungszeichen).

Optional DEFAULT CHARACTER SET: Klausel, die den Standardzeichensatz zum Erstellen von Datenstrukturen von String-Datentypen angibt. Zeichensätze werden auf Datentypen CHAR, VARCHAR und BLOB TEXT angewendet. Der Zeichensatz NONE wird standardmäßig verwendet. Es ist auch möglich, den Standardwert COLLATION für den Standardzeichensatz festzulegen, wodurch diese Sortierfolge zum Standardwert für den Standardzeichensatz wird. Der Standardwert wird für die gesamte Datenbank verwendet, es sei denn, ein alternativer Zeichensatz mit oder ohne eine angegebene Collation wird explizit für ein Feld, eine Domain, eine Variable, einen Ausdruck usw. verwendet.

STARTING AT: Klausel, die die Datenbankseitennummer angibt, bei der die nächste sekundäre Datenbankdatei gestartet werden soll. Wenn die vorherige Datei vollständig mit Daten gemäß der angegebenen Seitennummer gefüllt ist, fügt das System neue Daten zur nächsten Datenbankdatei hinzu.

Optional DIFFERENCE FILE: Klausel, die den Pfad und den Namen für das Datei-Delta angibt, das Änderungen in der Datenbankdatei speichert, nachdem es durch die Anweisung ALTER DATABASE BEGIN BACKUP auf den „kopiersicheren“ Modus gestellt wurde. Eine detaillierte Beschreibung dieser Klausel finden Sie unter ALTER DATABASE.

SET SQL DIALECT: Datenbanken werden standardmäßig in Dialekt 3 erstellt. Damit die Datenbank in SQL-Dialekt 1 erstellt wird, müssen Sie die Anweisung SET SQL DIALECT 1 aus dem Skript oder der Clientanwendung, z.B. *isql*, ausführen, noch vor der Anweisung CREATE DATABASE.

Beispiele zur Verwendung von CREATE DATABASE

1. Erstellen einer Datenbank in Windows auf der Festplatte D mit einer Seitengröße von 8.192. Der Besitzer der Datenbank ist der Benutzer *wizard*. Die Datenbank befindet sich in Dialekt 1 und verwendet als Standardzeichensatz WIN1251.

```
SET SQL DIALECT 1;
CREATE DATABASE 'D:\test.fdb'
USER 'wizard' PASSWORD 'player'
PAGE_SIZE = 8192 DEFAULT CHARACTER SET WIN1251;
```

2. Erstellen einer Datenbank im Linux-Betriebssystem mit einer Seitengröße von 4.096. Der Besitzer der Datenbank ist der Benutzer *wizard*. Die Datenbank befindet sich in Dialekt 3 und verwendet UTF8 als Standardzeichensatz, wobei UNICODE_CI_AI als Standardsortierung verwendet wird.

```
CREATE DATABASE '/home/firebird/test.fdb'
USER 'wizard' PASSWORD 'player'
DEFAULT CHARACTER SET UTF8 COLLATION UNICODE_CI_AI;
```

- Erstellen einer Datenbank auf dem entfernten Server „baseserver“ mit dem angegebenen Alias „test“, der zuvor in der Datei `aliases.conf` definiert wurde. Das TCP/IP-Protokoll wird verwendet. Der Besitzer der Datenbank wird der Benutzer `wizard` sein. Die Datenbank befindet sich in Dialekt 3 und verwendet UTF8 als Standardzeichensatz.

```
CREATE DATABASE 'baseserver:test'  
USER 'wizard' PASSWORD 'player'  
DEFAULT CHARACTER SET UTF8;
```

- Erstellen einer Datenbank in Dialekt 3 mit UTF8 als Standardzeichensatz. Die primäre Datei enthält bis zu 10.000 Seiten mit einer Seitengröße von 8.192. Sobald die primäre Datei die maximale Anzahl an Seiten erreicht hat, beginnt Firebird, Seiten der sekundären Datei `test.fdb2` zuzuweisen. Wenn diese Datei ebenfalls maximal gefüllt ist, wird `test.fdb3` zum Empfänger aller neuen Seitenzuweisungen. Als letzte Datei hat Firebird kein Seitenlimit. Neue Zuweisungen werden so lange fortgesetzt, wie es das Dateisystem zulässt, oder bis das Speichergerät keinen freien Speicher mehr hat. Wenn für diese letzte Datei ein `LENGTH`-Parameter angegeben wurde, wird dieser ignoriert.

```
SET SQL DIALECT 3;  
CREATE DATABASE 'baseserver:D:\test.fdb'  
USER 'wizard' PASSWORD 'player'  
PAGE_SIZE = 8192  
DEFAULT CHARACTER SET UTF8  
FILE 'D:\test.fdb2'  
STARTING AT PAGE 10001  
FILE 'D:\test.fdb3'  
STARTING AT PAGE 20001;
```

- Erstellen einer Datenbank in Dialekt 3 mit UTF8 als Standardzeichensatz. Die primäre Datei enthält bis zu 10.000 Seiten mit einer Seitengröße von 8.192. In Bezug auf die Dateigröße und die Verwendung von Sekundärdateien verhält sich diese Datenbank genau wie im vorherigen Beispiel.

```
SET SQL DIALECT 3;  
CREATE DATABASE 'baseserver:D:\test.fdb'  
USER 'wizard' PASSWORD 'player'  
PAGE_SIZE = 8192  
LENGTH 10000 PAGES  
DEFAULT CHARACTER SET UTF8  
FILE 'D:\test.fdb2'  
FILE 'D:\test.fdb3'  
STARTING AT PAGE 20001;
```

Siehe auch: [ALTER DATABASE](#), [DROP DATABASE](#)

ALTER DATABASE

Benutzt für: Ändern der Dateioorganisation einer Datenbank oder um diese in den „kopiersicheren“ Modus zu setzen

Verfügbar in: DSQL—beide Funktionen. ESQL—nur Dateireorganisation

Syntax:

```
ALTER {DATABASE | SCHEMA}
[<add_sec_clause> [<add_sec_clause> ...]]
[ADD DIFFERENCE FILE 'diff_file' | DROP DIFFERENCE FILE]
[ {BEGIN | END} BACKUP];

<add_sec_clause> ::= ADD <sec_file> [<sec_file> ...]

<sec_file> ::= FILE 'filepath'

ADD FILE <sec_file>

        [STARTING [AT [PAGE]] pagenum]
        [LENGTH [=] num [PAGE[S]]]
```

Anmerkung

Mehrere Dateien können mit einer ADD-Klausel hinzugefügt werden:

```
ALTER DATABASE
  ADD FILE x LENGTH 8000
  FILE y LENGTH 8000
  FILE z
```

Mehrere ADD FILE-Klauseln sind erlaubt; und eine ADD FILE-Klausel, die mehrere Dateien hinzufügt (wie im obigen Beispiel), kann mit anderen gemischt werden, die nur eine Datei hinzufügen. Die Aussage wurde in der alten *InterBase 6 Language Reference* falsch dokumentiert.

Tabelle 5.2. ALTER DATABASE Statement-Parameter

| Parameter | Beschreibung |
|----------------|---|
| add_sec_clause | Hinzufügen einer sekundären Datenbankdatei |
| sec_file | Dateispezifikation für sekundäre Datei |
| filepath | Vollständiger Pfad und Dateiname der Delta-Datei oder der sekundären Datenbankdatei |
| pagenum | Seitennummer, von der aus die sekundäre Datenbankdatei gestartet werden soll |
| num | Maximale Größe der sekundären Datei in Seiten |
| diff_file | Dateipfad und Name der .delta-Datei (Differenzdatei) |

Das Statement ALTER DATABASE kann

- sekundäre Dateien zu einer Datenbank hinzufügen
- eine Ein-Datei-Datenbank in und aus dem Modus „kopiersicher“ schalten (nur DSQL)
- festlegen und entfernen des Pfades und der Namen der Delta-Dateien für physikalische Backups (nur DSQL)

Nur **Administratoren** haben die Berechtigung das Statement ALTER DATABASE auszuführen.

Parameter für ALTER DATABASE

Die ADD FILE-Klausel: fügt eine sekundäre Datei zur Datenbank hinzu. Es ist erforderlich, den vollständigen Pfad zur Datei und den Namen der sekundären Datei anzugeben. Die Beschreibung für die sekundäre Datei ähnelt der für die Anweisung CREATE DATABASE.

Die ADD DIFFERENCE FILE-Klausel: gibt den Pfad und den Namen der Delta-Datei an, die Änderungen in der Datenbank speichert, wenn diese auf den Modus „kopiersicher“ umgestellt wird. Diese Klausel fügt tatsächlich keine Datei hinzu. Sie überschreibt nur den Standardnamen und den Standardpfad der Delta-Datei. Um die vorhandenen Einstellungen zu ändern, sollten Sie die vorher angegebene Beschreibung der .delta-Datei mit der DROP DIFFERENCE FILE-Klausel vor der Angabe der neuen Beschreibung der Deltadatei löschen. Wenn Pfad und Name der .delta-Datei nicht überschrieben werden, hat die Datei denselben Pfad und denselben Namen wie die Datenbank, jedoch mit der Dateierweiterung `.delta`.

Achtung

Wenn nur ein Dateiname angegeben ist, wird die .delta-Datei im aktuellen Verzeichnis des Servers erstellt. Unter Windows wird dies das Systemverzeichnis sein — ein sehr unkluger Speicherort für flüchtige Benutzerdateien und entgegen den Windows-Dateisystemregeln.

DROP DIFFERENCE FILE: Dies ist die Klausel, die die Beschreibung (Pfad und Name) der zuvor in der ADD DIFFERENCE FILE-Klausel angegebenen Deltadatei löscht. Die Datei wird nicht gelöscht. DROP DIFFERENCE FILE löscht den Pfad und den Namen der .delta-Datei aus dem Datenbank-Header. Beim nächsten Umschalten der Datenbank auf den „kopiersicheren“ Modus werden die Standardwerte verwendet (d.h. derselbe Pfad und Name wie die Datenbank, jedoch mit der Erweiterung .delta).

BEGIN BACKUP: Dies ist die Klausel, die die Datenbank in den „kopiersicheren“ Modus umschaltet. ALTER DATABASE friert mit dieser Klausel die Hauptdatenbankdatei ein und ermöglicht die sichere Sicherung mithilfe von Dateisystemtools, selbst wenn Benutzer verbunden sind und Operationen mit Daten ausführen. Bis der Sicherungsstatus der Datenbank auf NORMAL zurückgesetzt wird, werden alle an der Datenbank vorgenommenen Änderungen in die .delta (Differenz)-Datei geschrieben.

Wichtig

Trotz seiner Syntax startet eine Anweisung mit der Klausel BEGIN BACKUP keinen Sicherungsprozess, sondern erstellt lediglich die Bedingungen für die Ausführung einer Aufgabe, für die die Datenbankdatei nur vorübergehend schreibgeschützt sein muss.

END BACKUP: ist die Klausel, mit der die Datenbank vom „kopiersicheren“ Modus in den normalen Modus umgeschaltet wird. Eine Anweisung mit dieser Klausel fügt die .delta-Datei mit der Hauptdatenbankdatei zusammen und stellt den normalen Betrieb der Datenbank wieder her. Sobald der END BACKUP-Prozess gestartet wird, sind die Bedingungen für das Erstellen sicherer Backups mit Dateisystemtools nicht mehr vorhanden.

Warnung

Die Verwendung von BEGIN BACKUP und END BACKUP und das Kopieren der Datenbankdateien mit den Dateisystemtools ist **nicht sicher** mit Mehrdateiendatenbanken! Verwenden Sie diese Methode nur für Datenbanken mit einer einzigen Datei.

Ein sicheres Backup mit dem Dienstprogramm *gbak* ist jederzeit möglich, wenn auch nicht empfohlen, solange sich die Datenbank im Zustand LOCKED oder MERGE befindet.

Beispiele zur Verwendung von ALTER DATABASE

1. Hinzufügen einer sekundären Datei zur Datenbank. Sobald in der vorherigen primären oder sekundären Datei 30000 Seiten gefüllt sind, fügt die Firebird-Engine Daten zur sekundären Datei `test4.fdb` hinzu.

```
ALTER DATABASE
ADD FILE 'D:\test4.fdb'
STARTING AT PAGE 30001;
```

2. Pfad und Name der Delta-Datei angeben:

```
ALTER DATABASE
ADD DIFFERENCE FILE 'D:\test.diff';
```

3. Beschreibung der Delta-Datei löschen:

```
ALTER DATABASE
DROP DIFFERENCE FILE;
```

4. Wechseln der Datenbank in den „kopiersicheren“ Modus:

```
ALTER DATABASE
BEGIN BACKUP;
```

5. Umschalten der Datenbank vom „kopiersicheren“ Modus in den normalen Betriebsmodus:

```
ALTER DATABASE
END BACKUP;
```

Siehe auch: [CREATE DATABASE](#), [DROP DATABASE](#)

DROP DATABASE

Benutzt für: Löschen der Datenbank, mit der Sie gerade verbunden sind

Verfügbar in: DSQL, ESQL

Syntax:

```
DROP DATABASE
```

Die Anweisung `DROP DATABASE` löscht die aktuelle Datenbank. Bevor Sie eine Datenbank löschen, müssen Sie eine Verbindung herstellen. Die Anweisung löscht die primäre Datei, alle sekundären Dateien und alle [Schattendateien](#).

Nur **Administratoren** haben die notwendigen Rechte zum Ausführen der Anweisung DROP DATABASE.

Beispiel: Löschen der Datenbank, mit der der Client verbunden ist.

```
DROP DATABASE;
```

Siehe auch: [CREATE DATABASE](#), [ALTER DATABASE](#)

SHADOW

Ein *shadow* ist eine exakte Seite-für-Seite-Kopie einer Datenbank. Sobald ein Shadow erstellt wurde, spiegeln sich alle Änderungen in der Datenbank sofort im Shadow wider. Wenn die primäre Datenbankdatei aus irgendeinem Grund nicht verfügbar ist, wechselt das DBMS auf den Shadow.

In diesem Abschnitt wird beschrieben, wie Sie Schattendateien erstellen und löschen.

CREATE SHADOW

Benutzt für: Erstellen eines Shadows für die aktuelle Datenbank

Verfügbar in: DSQL, ESQL

Syntax:

```
CREATE SHADOW sh_num [AUTO | MANUAL] [CONDITIONAL]
'filepath' [LENGTH [=] num [PAGE[S]]]
[<secondary_file> ...];

<secondary_file> ::=
FILE 'filepath'
[STARTING [AT [PAGE]] pagenum]
[LENGTH [=] num [PAGE[S]]]
```

Tabelle 5.3. CREATE SHADOW Statement-Parameter

| Parameter | Beschreibung |
|----------------|--|
| sh_num | Schattennummer — eine positive Zahl, die den Schattensatz identifiziert |
| filepath | Der Name der Schattendatei und der Pfad dazu in Übereinstimmung mit den Regeln des Betriebssystems |
| num | Maximale Schattengröße in Seiten |
| secondary_file | Sekundäre Dateispezifikation |
| page_num | Die Nummer der Seite, auf der die sekundäre Schattendatei gestartet werden soll |

Die Anweisung `CREATE SHADOW` erstellt einen neuen Shadow. Der Schatten beginnt mit dem Duplizieren der Datenbank in dem Moment, in dem sie erstellt wird. Es ist für einen Benutzer nicht möglich, eine Verbindung zu einem Schatten herzustellen.

Wie bei einer Datenbank kann ein Shadow eine Mehrfachdatei sein. Die Anzahl und Größe der Dateien eines Schattens hängt nicht mit der Anzahl und Größe der Dateien der Datenbank, die es beschattet, zusammen.

Die Seitengröße für Schattendateien wird auf die Größe der Datenbankseite festgelegt und kann nicht geändert werden.

Wenn ein Unglück mit der ursprünglichen Datenbank auftritt, konvertiert das System den Schatten in eine Kopie der Datenbank und wechselt zu dieser. Der Schatten ist dann *nicht verfügbar*. Was als nächstes passiert, hängt von der Option `MODE` ab.

AUTO | MANUAL Modes

Wenn ein Schatten in eine Datenbank konvertiert wird, ist er nicht mehr verfügbar. Ein Schatten kann auch unverfügbar werden, weil jemand versehentlich seine Datei löscht oder der Speicherplatz, auf dem die Schatten-Dateien gespeichert sind, erschöpft ist oder selbst beschädigt ist.

- Wenn der `AUTO`-Modus ausgewählt ist (Standardwert), wird die Spiegelung automatisch beendet, alle Referenzen werden aus dem Datenbank-Header gelöscht, und die Datenbank arbeitet normal weiter.

Wenn die Option `CONDITIONAL` festgelegt wurde, versucht das System, einen neuen Schatten zu erstellen, um den verlorenen zu ersetzen. Es ist jedoch nicht immer erfolgreich und ein neuer muss möglicherweise manuell erstellt werden.

- Ist das `MANUAL`-Modus-Attribut gesetzt, wenn der Schatten nicht mehr verfügbar ist, werden alle Versuche, eine Verbindung zur Datenbank herzustellen und diese abzufragen, Fehlermeldungen erzeugen. Die Datenbank bleibt so lange unzugänglich, bis entweder der Schatten wieder verfügbar ist oder der Datenbankadministrator sie mithilfe der Anweisung `DROP SHADOW` löscht. `MANUAL` sollte ausgewählt werden, wenn kontinuierliches Shadowing wichtiger ist als der unterbrechungsfreie Betrieb der Datenbank.

Optionen für CREATE SHADOW

Optional LENGTH: Klausel, die die maximale Größe der primären oder sekundären Schattendatei in Seiten angibt. Der Wert `LENGTH` wirkt sich nicht auf die Größe der einzigen Schattendatei aus, noch auf die letzte, wenn es sich um eine Gruppe handelt. Die letzte (oder einzige) Datei wird automatisch so lange vergrößert, wie es nötig ist.

STARTING AT: Klausel, die die Schattenseitennummer angibt, bei der die nächste Schattendatei gestartet werden soll. Das System fügt neue Daten zur nächsten Schattendatei hinzu, wenn die vorherige Datei bis zur angegebenen Seitenzahl mit Daten gefüllt ist.

Nur **Administratoren** haben die notwendigen Rechte die Anweisung `CREATE SHADOW` auszuführen.

Tipp

Sie können die Größen, Namen und den Speicherort der Schattendateien überprüfen, indem Sie mit *isql* eine Verbindung zur Datenbank herstellen und den Befehl `SHOW DATABASE;`

Beispiele für die Verwendung von CREATE SHADOW:

1. Erstellen eines Schattens für die aktuelle Datenbank als „shadow number 1“:

```
CREATE SHADOW 1 'g:\data\test.shd';
```

2. Erstellen eines Mehrdatei-Schattens für die aktuelle Datenbank als „shadow number 2“:

```
CREATE SHADOW 2 'g:\data\test.sh1'  
LENGTH 8000 PAGES  
FILE 'g:\data\test.sh2';
```

Siehe auch: [CREATE DATABASE](#), [DROP SHADOW](#)

DROP SHADOW

Benutzt für: Deleting a shadow from the current database

Verfügbar in: DSQL, ESQL

Syntax:

```
DROP SHADOW sh_num
```

Tabelle 5.4. DROP SHADOW Statement-Parameter

| Parameter | Beschreibung |
|-----------|---|
| sh_num | Schattennummer — eine positive Zahl, die den Schattensatz identifiziert |

Die Anweisung DROP SHADOW löscht den angegebenen Schatten für die Datenbank, mit der eine Verbindung besteht. Wenn ein Schatten gelöscht wird, werden alle zugehörigen Dateien gelöscht und Schatten auf die angegebene *sh_num* werden beendet.

Nur [Administratoren](#) haben die notwendigen Rechte die Anweisung DROP SHADOW auszuführen.

Beispiel zum Löschen eines Schattens: Löschen von „shadow Nummer 1“.

```
DROP SHADOW 1;
```

Siehe auch: [CREATE SHADOW](#)

DOMAIN

Domain ist eine Objektart innerhalb einer relationalen Datenbank. Eine Domain wird als ein bestimmter Datentyp mit einigen Attributen erstellt. Sobald es in der Datenbank definiert wurde, kann es wiederholt verwendet werden, um Tabellenspalten, PSQL-Argumente und lokale PSQL-Variablen zu definieren. Diese Objekte erben

alle Attribute der Domain Einige Attribute können bei Bedarf überschrieben werden, wenn das neue Objekt definiert ist.

In diesem Abschnitt wird die Syntax von Anweisungen beschrieben, mit denen Domains erstellt, geändert und gelöscht werden. Eine detaillierte Beschreibung von Domains und deren Verwendung finden Sie in [Benutzerdefinierte Datentypen — Domains](#).

CREATE DOMAIN

Benutzt für: Erstellen einer neuen Domain

Verfügbar in: DSQL, ESQL

Syntax:

```
CREATE DOMAIN name [AS] <datatype>
[DEFAULT {literal | NULL | <context_var>}]
[NOT NULL] [CHECK (<dom_condition>)]
[COLLATE collation_name];

<datatype> ::=
    {SMALLINT | INTEGER | BIGINT} [<array_dim>]
  | {FLOAT | DOUBLE PRECISION} [<array_dim>]
  | {DATE | TIME | TIMESTAMP} [<array_dim>]
  | {DECIMAL | NUMERIC} [(precision [, scale])] [<array_dim>]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
    [<array_dim>] [CHARACTER SET charset_name]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING]
    [(size)] [<array_dim>]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
    [SEGMENT SIZE seglen] [CHARACTER SET charset_name]
  | BLOB [(seglen [, subtype_num])]

<array_dim> ::= [[m:]n [, [m:]n ...]]

<dom_condition> ::=
    <val> <operator> <val>
  | <val> [NOT] BETWEEN <val> AND <val>
  | <val> [NOT] IN (<val> [, <val> ...] | <select_list>)
  | <val> IS [NOT] NULL
  | <val> IS [NOT] DISTINCT FROM <val>
  | <val> [NOT] CONTAINING <val>
  | <val> [NOT] STARTING [WITH] <val>
  | <val> [NOT] LIKE <val> [ESCAPE <val>]
  | <val> [NOT] SIMILAR TO <val> [ESCAPE <val>]
  | <val> <operator> {ALL | SOME | ANY} (<select_list>)
  | [NOT] EXISTS (<select_expr>)
  | [NOT] SINGULAR (<select_expr>)
  | (<dom_condition>)
  | NOT <dom_condition>
  | <dom_condition> OR <dom_condition>
  | <dom_condition> AND <dom_condition>

<operator> ::=
    <> | != | ^= | ~= | = | < | > | <= | >= | !< | ^< | ~< | !> | ^> | ~>
```

```

<val> ::=
  VALUE
  | literal
  | <context_var>
  | <expression>
  | NULL
  | NEXT VALUE FOR genname
  | GEN_ID(genname, <val>)
  | CAST(<val> AS <datatype>)
  | (<select_one>)
  | func([<val> [, <val> ...]])
    
```

Tabelle 5.5. CREATE DOMAIN Statement-Parameter

| Parameter | Beschreibung |
|----------------|--|
| name | Domainname aus maximal 31 Zeichen |
| datatype | SQL-Datentyp |
| literal | Ein literaler Wert, der kompatibel zu <i>datatype</i> ist |
| context_var | Jede Kontextvariable, deren Typ kompatibel ist mit <i>datatype</i> |
| dom_condition | Domain-Bedingung |
| collation_name | Name einer Collation, die für <i>charset_name</i> gültig ist, sofern dieser mit <i>datatype</i> übergeben wird, oder andernfalls für den Standardzeichensatz der Datenbank |
| array_dim | Array-Dimensionen |
| m, n | INTEGER-Ganzzahlen, die den Indexbereich der Array-Dimensionen angeben |
| precision | Die Gesamtzahl der signifikanten Ziffern, die ein Wert von <i>datatype</i> aufnehmen kann (1..18) |
| scale | Die Anzahl der Stellen nach dem Dezimalpunkt (0.. <i>precision</i>) |
| size | Die maximale Anzahl einer Zeichenkette in Zeichen |
| charset_name | Der Name eines gültigen Zeichensatzes, falls sich der Zeichensatz der Domain vom Standardzeichensatz der Datenbank unterscheidet |
| subtype_num | BLOB Subtype-Nummer |
| subtype_name | BLOB-Subtyp-Mnemonikname |
| seglen | Segmentgröße (max. 65535) |
| select_one | Eine skalare SELECT-Anweisung — Auswählen einer Spalte und Zurückgeben nur eines row |
| select_list | Eine SELECT-Anweisung, die eine Spalte auswählt und null oder mehr Zeilen zurückgibt |
| select_expr | Eine SELECT-Anweisung, die eine Spalte oder mehrere Spalten auswählt und null oder mehr Zeilen zurückgibt |

| Parameter | Beschreibung |
|------------|--|
| expression | Ein Ausdruck, der auf einen Wert auflöst, der mit <i>datatype</i> kompatibel ist |
| genname | Sequenzname (Generatorname) |
| func | Interne Funktion oder UDF |

Die Anweisung CREATE DOMAIN erstellt eine neue Domain.

Jeder SQL-Datentyp kann als Domainntyp angegeben werden.

Typenspezifische Details

ARRAY Typen:

- Wenn die Domain ein Array sein soll, kann der Basistyp ein SQL-Datentyp mit Ausnahme von BLOB und ARRAY sein.
- Die Dimensionen des Arrays werden in eckigen Klammern angegeben. (Im Syntaxblock werden diese Klammern fett dargestellt, um sie von den eckigen Klammern zu unterscheiden, die optionale Syntaxelemente kennzeichnen.)
- Für jede Array-Dimension definieren eine oder zwei ganze Zahlen die untere und obere Grenze ihres Indexbereichs:
 - Standardmäßig sind Arrays 1-basiert. Die untere Grenze ist implizit und nur die obere Grenze muss angegeben werden. Eine einzelne Zahl kleiner als 1 definiert den Bereich *num* ..1 und eine Zahl größer als 1 definiert den Bereich 1.. *num* .
 - Zwei durch einen Doppelpunkt getrennte Zahlen (':') und optional ein Leerraum, der zweite ist größer als der erste, können verwendet werden, um den Bereich explizit zu definieren. Eine oder beide Grenzen können kleiner als Null sein, solange die obere Grenze größer als die untere ist.
- Wenn das Array mehrere Dimensionen hat, müssen die Bereichsdefinitionen für jede Dimension durch Kommas und ein optionales Leerzeichen getrennt werden.
- Indizes werden *nur* validiert, wenn ein Array tatsächlich existiert. Dies bedeutet, dass keine Fehlermeldungen bezüglich ungültiger Subskripte zurückgegeben werden, wenn ein bestimmtes Element nichts zurückgibt oder wenn ein Array-Feld NULL ist.

CHARACTER Typen: Sie können die CHARACTER SET-Klausel nutzen, um den Zeichensatz für die Datentypen CHAR, VARCHAR und BLOB (SUB_TYPE TEXT) zu definieren. Wird der Zeichensatz nicht angegeben, wird der in der Datenbank als DEFAULT CHARACTER SET Zeichensatz verwendet. Ist auch dieser nicht festgelegt, wird der Zeichensatz NONE als Standard für die Anlage von Domains verwendet.

Warnung

Bei Zeichensatz NONE werden Zeichendaten gespeichert und abgerufen, wie sie übermittelt wurden. Daten in einer beliebigen Codierung können zu einer Spalte auf der Grundlage einer solchen Domain hinzugefügt werden. Es ist jedoch nicht möglich, diese Daten zu einer Spalte mit einer anderen Codierung hinzuzufügen. Da zwischen Quell- und Zielcodierung keine Transkription durchgeführt wird, können Fehler auftreten.

DEFAULT-Klausel: Mit der optionalen DEFAULT-Klausel können Sie einen Standardwert für die Domain angeben. Dieser Wert wird der Tabellenspalte hinzugefügt, die diese Domain erbt, wenn die Anweisung INSERT ausgeführt wird, wenn in der DML-Anweisung kein Wert dafür angegeben ist. Lokale Variablen und Argumente in PSQL-Modulen, die auf diese Domain verweisen, werden mit dem Standardwert initialisiert. Verwenden Sie als Standardwert ein Literal eines kompatiblen Typs oder eine Kontextvariable eines kompatiblen Typs.

NOT NULL-Constraint: Spalten und Variablen basierend auf einer Domain mit der NOT NULL-Beschränkung werden daran gehindert, als NULL geschrieben zu werden, d.h. ein Wert ist *erforderlich*.

Achtung

Achten Sie beim Anlegen einer Domain darauf, keine Einschränkungen zu spezifizieren, die einander widersprechen würden. Zum Beispiel sind NOT NULL und DEFAULT NULL widersprüchlich.

CHECK-Constraint(s): Die optionale Klausel CHECK gibt Einschränkungen für die Domain an. Eine Domainbeschränkung gibt Bedingungen an, die von den Werten von Tabellenspalten oder Variablen erfüllt werden müssen, die von der Domain erben. Eine Bedingung muss in Klammern eingeschlossen werden. Eine Bedingung ist ein logischer Ausdruck (auch Prädikat genannt), der die booleschen Ergebnisse TRUE, FALSE und UNKNOWN zurückgeben kann. Eine Bedingung gilt als erfüllt, wenn das Prädikat den Wert TRUE oder „UNKNOWN“ (entspricht NULL) zurückgibt. Wenn das Prädikat FALSE zurückgibt, ist die Bedingung für die Annahme nicht erfüllt.

VALUE-Schlüsselwort: Das Schlüsselwort VALUE in einer Domainbeschränkung ersetzt die Tabellenspalte, die auf dieser Domain oder einer Variablen in einem PSQL-Modul basiert. Es enthält den Wert, der der Variablen oder der Tabellenspalte zugewiesen ist. VALUE kann überall in der CHECK-Bedingung verwendet werden, obwohl es normalerweise im linken Teil der Bedingung verwendet wird.

COLLATE: Mit der optionalen COLLATE-Klausel können Sie die Sortierreihenfolge (Collation) angeben, wenn die Domain auf einem der String-Datentypen basiert, einschließlich BLOBs mit Textsubtypen. Wenn keine Sortierreihenfolge angegeben ist, ist die Sortierreihenfolge diejenige, die für den angegebenen Zeichensatz zum Zeitpunkt der Erstellung der Domain voreingestellt ist.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann eine Domain erstellen.

Beispiele für CREATE DOMAIN

1. Erstellen einer Domain mit Werten von mehr als 1.000 und einem Standardwert von 10.000.

```
CREATE DOMAIN CUSTNO AS
INTEGER DEFAULT 10000
CHECK (VALUE > 1000);
```

2. Erstellen einer Domain, die die Werte "Ja" und "Nein" in dem Standardzeichensatz annehmen kann, der während der Erstellung der Datenbank angegeben wurde.

```
CREATE DOMAIN D_BOOLEAN AS
CHAR(3) CHECK (VALUE IN ('Yes', 'No'));
```

3. Erstellen einer Domain mit dem Zeichensatz UTF8 und der Sortierreihenfolge (Collation) UNICODE_CI_AI.

```
CREATE DOMAIN FIRSTNAME AS
VARCHAR(30) CHARACTER SET UTF8
COLLATE UNICODE_CI_AI;
```


4. Erstellen einer Domain vom Typ DATE, die NULL nicht akzeptiert und das aktuelle Datum als Standardwert verwendet.

```
CREATE DOMAIN D_DATE AS
DATE DEFAULT CURRENT_DATE
NOT NULL;
```

5. Erstellen einer Domain, die als ein Array aus zwei Elementen des Typs NUMERIC(18, 3) definiert ist. Der Start-Array-Index ist 1.

```
CREATE DOMAIN D_POINT AS
NUMERIC(18, 3) [2];
```

Anmerkung

Über einen Array-Typ definierte Domainn dürfen nur zum Definieren von Tabellenspalten verwendet werden. Sie können keine Array-Domainn verwenden, um lokale Variablen in PSQL-Modulen zu definieren.

6. Erstellen einer Domain, deren Elemente nur in der Tabelle COUNTRY definierte Ländercodes sein können.

```
CREATE DOMAIN D_COUNTRYCODE AS CHAR(3)
CHECK (EXISTS(SELECT * FROM COUNTRY
WHERE COUNTRYCODE = VALUE));
```

Anmerkung

Das Beispiel zeigt nur die Möglichkeit, Prädikate mit Abfragen in der Domainntestbedingung zu verwenden. Es wird nicht empfohlen, diesen Stil der Domain in der Praxis zu verwenden es sei denn, die Nachschlagetabelle enthält Daten, die niemals gelöscht werden.

Siehe auch: ALTER DOMAIN, DROP DOMAIN

ALTER DOMAIN

Benutzt für: Die aktuellen Attribute einer Domain ändern oder umbenennen

Verfügbar in: DSQL, ESQL

Syntax:

```
ALTER DOMAIN domain_name
[TO <new_name>]
[TYPE <datatype>]
[SET DEFAULT {literal | NULL | <context_var>} | DROP DEFAULT]
[ADD [CONSTRAINT] CHECK (<dom_condition>) | DROP CONSTRAINT]
```

```

<datatype> ::=
    {SMALLINT | INTEGER | BIGINT}
  | {FLOAT | DOUBLE PRECISION}
  | {DATE | TIME | TIMESTAMP}
  | {DECIMAL | NUMERIC} [(precision [, scale])]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
  | [CHARACTER SET charset_name]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(size)]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
  | [SEGMENT SIZE seglen] [CHARACTER SET charset_name]
  | BLOB [(seglen [, subtype_num])]

<dom_condition> ::=
    <val> <operator> <val>
  | <val> [NOT] BETWEEN <val> AND <val>
  | <val> [NOT] IN (<val> [, <val> ...] | <select_list>)
  | <val> IS [NOT] NULL
  | <val> IS [NOT] DISTINCT FROM <val>
  | <val> [NOT] CONTAINING <val>
  | <val> [NOT] STARTING [WITH] <val>
  | <val> [NOT] LIKE <val> [ESCAPE <val>]
  | <val> [NOT] SIMILAR TO <val> [ESCAPE <val>]
  | <val> <operator> {ALL | SOME | ANY} (<select_list>)
  | [NOT] EXISTS (<select_expr>)
  | [NOT] SINGULAR (<select_expr>)
  | (<dom_condition>)
  | NOT <dom_condition>
  | <dom_condition> OR <dom_condition>
  | <dom_condition> AND <dom_condition>

<operator> ::=
    <> | != | ^= | ~= | = | < | > | <= | >= | !< | ^< | ~< | !> | ^> | ~>

<val> ::=
    VALUE
  | literal
  | <context_var>
  | <expression>
  | NULL
  | NEXT VALUE FOR genname
  | GEN_ID(genname, <val>)
  | CAST(<val> AS <datatype>)
  | (<select_one>)
  | func([<val> [, <val> ...]])

```

Tabelle 5.6. ALTER DOMAIN Statement-Parameter

| Parameter | Beschreibung |
|-------------|--|
| new_name | Neuer Domainname, bestehend aus maximal 31 Zeichen |
| datatype | SQL-Datentyp |
| literal | Ein literaler Wert, der kompatibel zu <i>datatype</i> ist |
| context_var | Jede Kontextvariable, deren Typ kompatibel ist mit <i>datatype</i> |

| Parameter | Beschreibung |
|--------------|--|
| precision | Die Gesamtzahl der signifikanten Ziffern, die ein Wert des <i>Datentyps</i> aufnehmen kann (1..18) |
| scale | Die Anzahl der Stellen nach dem Dezimalkomma ($0..precision$) |
| size | Die maximale Größe einer Zeichenkette in Zeichen |
| charset_name | Der Name eines gültigen Zeichensatzes, falls sich der Zeichensatz der Domain vom Standardzeichensatz der Datenbank unterscheidet |
| subtype_num | BLOB Subtype-Nummer |
| subtype_name | BLOB-Subtyp-Mnemonikname |
| seglen | Segmentgröße (max. 65535) |
| select_one | Eine skalare SELECT-Anweisung — Auswählen einer Spalte und Zurückgeben nur eines row |
| select_list | Eine SELECT-Anweisung, die eine Spalte auswählt und null oder mehr Zeilen zurückgibt |
| select_expr | Eine SELECT-Anweisung, die eine Spalte oder mehrere Spalten auswählt und null oder mehr Zeilen zurückgibt |
| expression | Ein Ausdruck, der auf einen Wert auflöst, der mit <i>datatype</i> kompatibel ist |
| genname | Sequenzname (Generatorname) |
| func | Interne Funktion oder UDF |

Die Anweisung ALTER DOMAIN ermöglicht Änderungen an den aktuellen Attributen einer Domain einschließlich ihres Namens. Sie können beliebig viele Domainänderungen in einer ALTER DOMAIN-Anweisung vornehmen.

TO <name>: Verwenden Sie die TO-Klausel, um die Domain umzubenennen, solange keine Abhängigkeiten von der Domain vorhanden sind, z. B. Tabellenspalten, lokale Variablen oder Prozedurargumente, die darauf verweisen.

SET DEFAULT: Mit der SET DEFAULT-Klausel können Sie einen neuen Standardwert setzen. Wenn die Domain bereits einen Standardwert hat, muss sie nicht zuerst gelöscht werden, sondern wird durch die neue ersetzt.

DROP DEFAULT: Mit dieser Klausel löschen Sie einen zuvor festgelegten Standardwert und ersetzen ihn durch NULL.

ADD CONSTRAINT CHECK: Verwenden Sie die Klausel ADD CONSTRAINT CHECK, um einen CHECK-Constraint zur Domain hinzuzufügen. Existiert bereits ein CHECK-Constraint für die Domain, muss dieser zunächst gelöscht werden. Nutzen Sie dazu ein ALTER DOMAIN-Statement, das eine DROP CONSTRAINT-Klausel beinhaltet.

TYPE: Die TYPE-Klausel wird verwendet, um den Datentyp der Domain in eine andere, kompatible zu ändern. Das System verbietet jede Änderung des Typs, der zu Datenverlust führen könnte. Ein Beispiel wäre, wenn die Anzahl der Zeichen im neuen Typ kleiner als im vorhandenen Typ wäre.

Wichtig

Wenn Sie die Attribute einer Domain ändern, kann der vorhandene PSQL-Code ungültig werden. Für Informationen zur Erkennung lesen Sie bitte den Artikel [Das RDB\\$VALID_BLR Feld](#) in Anhang A.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann eine Domain ändern, sofern sie nicht durch Abhängigkeiten von Objekten verhindert wird, für die dieser Benutzer nicht über ausreichende Berechtigungen verfügt.

Was die Ausführung von ALTER DOMAIN verhindert

- Wenn die Domain als ein Array deklariert wurde, ist es nicht möglich, ihren Typ oder ihre Dimensionen zu ändern. Es kann auch kein anderer Typ in einen ARRAY-Typ geändert werden.
- In Firebird 2.5 und niedriger darf die Einschränkung NOT NULL weder für eine Domain aktiviert noch deaktiviert werden.
- Es gibt keine Möglichkeit, die Standardkollation zu ändern, ohne die Domain zu löschen und sie mit den gewünschten Attributen neu zu erstellen.

Beispiele für ALTER DOMAIN

1. Ändern des Datentyps in INTEGER und festlegen oder ändern des Standardwerts auf 2.000:

```
ALTER DOMAIN CUSTNO
  TYPE INTEGER
  SET DEFAULT 2000;
```

2. Umbenennen einer Domain.

```
ALTER DOMAIN D_BOOLEAN TO D_BOOL;
```

3. Löschen des Standardwerts und Hinzufügen einer Einschränkung für die Domain:

```
ALTER DOMAIN D_DATE
  DROP DEFAULT
  ADD CONSTRAINT CHECK (VALUE >= date '01.01.2000');
```

4. Ändern des CHECK-Constraints:

```
ALTER DOMAIN D_DATE
  DROP CONSTRAINT;

ALTER DOMAIN D_DATE
  ADD CONSTRAINT CHECK
    (VALUE BETWEEN date '01.01.1900' AND date '31.12.2100');
```

5. Ändern des Datentyps, um die zulässige Anzahl von Zeichen zu erhöhen:

```
ALTER DOMAIN FIRSTNAME
  TYPE VARCHAR(50) CHARACTER SET UTF8;
```

Siehe auch: [CREATE DOMAIN](#), [DROP DOMAIN](#)

DROP DOMAIN

Benutzt für: Eine bestehende Domain löschen

Verfügbar in: DSQL, ESQL

Syntax:

```
DROP DOMAIN domain_name
```

Die Anweisung DROP DOMAIN löscht eine Domain, die in der Datenbank vorhanden ist. Es ist nicht möglich, eine Domain zu löschen, wenn sie von Spalten der Datenbanktabellen referenziert oder in einem PSQL-Modul verwendet wird. Um eine Domain zu löschen, die verwendet wird, müssen alle Spalten in allen Tabellen, die auf die Domain verweisen, gelöscht werden und alle Verweise auf die Domain müssen aus PSQL-Modulen entfernt werden.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann eine Domain löschen.

Beispiele

Löschen der COUNTRYNAME-Domain:

```
DROP DOMAIN COUNTRYNAME;
```

Siehe auch: [CREATE DOMAIN](#), [ALTER DOMAIN](#)

TABLE

Als relationales DBMS speichert Firebird Daten in Tabellen. Eine Tabelle ist eine flache, zweidimensionale Struktur, die eine beliebige Anzahl von Zeilen enthält. Tabellenzeilen werden oft als *Datensätze* bezeichnet.

Alle Zeilen in einer Tabelle haben die gleiche Struktur und bestehen aus Spalten. Tabellenspalten werden oft als *Felder* bezeichnet. Eine Tabelle muss mindestens eine Spalte haben. Jede Spalte enthält einen einzelnen Typ von SQL-Daten.

In diesem Abschnitt wird beschrieben, wie Sie Tabellen in einer Datenbank erstellen, ändern und löschen.

CREATE TABLE

Benutzt für: Erstellen einer neuen Tabelle (Relation)

Verfügbar in: DSQL, ESQL

Syntax:

```

CREATE [GLOBAL TEMPORARY] TABLE tablename
  [EXTERNAL [FILE] '<filespec>']
  (<col_def> [, {<col_def> | <tconstraint>} ...])
  [ON COMMIT {DELETE | PRESERVE} ROWS];

<col_def> ::= <regular_col_def> | <computed_col_def>

<regular_col_def> ::=
  colname {<datatype> | domainname}
  [DEFAULT {literal | NULL | <context_var>}]
  [NOT NULL]
  [<col_constraint>]
  [COLLATE collation_name]

<computed_col_def> ::=
  colname [<datatype>]
  {COMPUTED [BY] | GENERATED ALWAYS AS} (<expression>)

<datatype> ::=
  {SMALLINT | INTEGER | BIGINT} [<array_dim>]
  | {FLOAT | DOUBLE PRECISION} [<array_dim>]
  | {DATE | TIME | TIMESTAMP} [<array_dim>]
  | {DECIMAL | NUMERIC} [(precision [, scale])] [<array_dim>]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
  [<array_dim>] [CHARACTER SET charset_name]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING]
  [(size)] [<array_dim>]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
  [SEGMENT SIZE seglen] [CHARACTER SET charset_name]
  | BLOB [(seglen [, subtype_num])]

<array_dim> ::= [[m:]n [, [m:]n ...]]

<col_constraint> ::=
  [CONSTRAINT constr_name]
  {
    PRIMARY KEY [<using_index>]
  | UNIQUE      [<using_index>]
  | REFERENCES other_table [(colname)] [<using_index>]
    [ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
    [ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
  | CHECK (<check_condition>) }

<tconstraint> ::=
  [CONSTRAINT constr_name]
  {
    PRIMARY KEY (col_list) [<using_index>]
  | UNIQUE      (col_list) [<using_index>]
  | FOREIGN KEY (col_list)
  
```

```

REFERENCES other_table [(col_list)] [<using_index>]
  [ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
  [ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
  | CHECK (<check_condition>) }"

<col_list> ::= colname [, colname ...]

<using_index> ::= USING
  [ASC[ENDING] | DESC[ENDING]] INDEX indexname

<check_condition> ::=
  <val> <operator> <val>
  | <val> [NOT] BETWEEN <val> AND <val>
  | <val> [NOT] IN (<val> [, <val> ...] | <select_list>)
  | <val> IS [NOT] NULL
  | <val> IS [NOT] DISTINCT FROM <val>
  | <val> [NOT] CONTAINING <val>
  | <val> [NOT] STARTING [WITH] <val>
  | <val> [NOT] LIKE <val> [ESCAPE <val>]
  | <val> [NOT] SIMILAR TO <val> [ESCAPE <val>]
  | <val> <operator> {ALL | SOME | ANY} (<select_list>)
  | [NOT] EXISTS (<select_expr>)
  | [NOT] SINGULAR (<select_expr>)
  | (<check_condition>)
  | NOT <check_condition>
  | <check_condition> OR <check_condition>
  | <check_condition> AND <check_condition>

<operator> ::=
<> | != | ^= | ~= | = | < | > | <= | >= | !< | ^< | ~< | !> | ^> | ~>

<val> ::=
  colname [[<array_idx> [, <array_idx> ...]]]
  | literal
  | <context_var>
  | <expression>
  | NULL
  | NEXT VALUE FOR genname
  | GEN_ID(genname, <val>)
  | CAST(<val> AS <datatype>)
  | (<select_one>)
  | func([<val> [, <val> ...]])

```

Tabelle 5.7. CREATE TABLE Statement-Parameter

| Parameter | Beschreibung |
|-----------|--|
| tablename | Name (Kennung) für die Tabelle. Sie kann aus bis zu 31 Zeichen bestehen und in der Datenbank eindeutig sein. |
| filespec | Datespezifikation (nur für externe Tabellen). Vollständiger Dateiname und Pfad, der in einfache Anführungszeichen eingeschlossen ist, unter Berücksichtigung der Regeln des lokalen Dateisystems. Die Datei muss physisch mit dem Host-Computer von Firebird verbunden sein. |
| colname | Name (Bezeichner) für eine Spalte in der Tabelle. Kann aus bis zu 31 Zeichen bestehen und in der Tabelle eindeutig sein. |

| Parameter | Beschreibung |
|-----------------|--|
| datatype | SQL-Datentyp |
| col_constraint | Spalten-Constraint |
| tconstraint | Tabellen-Constraint |
| constr_name | Der Name (Bezeichner) einer Einschränkung. Darf aus bis zu 31 Zeichen bestehen. |
| other_table | Der Name der Tabelle, auf die die Constraint verweist |
| other_col | Der Name der Spalte in <i>other_table</i> , auf die der Fremdschlüssel verweist |
| literal | Ein Literalwert, der im angegebenen Kontext zulässig ist |
| context_var | Beliebige Kontextvariable, deren Datentyp im angegebenen Kontext zulässig ist |
| check_condition | Die Bedingung, die auf eine CHECK-Einschränkung angewendet wird, die als wahr, false oder NULL aufgelöst wird. |
| collation | Collation |
| array_dim | Array-Dimensionen |
| m, n | INTEGER-Ganzzahlen die den Bereich der Array-Dimensionen angeben |
| precision | Die Gesamtzahl der signifikanten Ziffern, die ein Wert des Datentyps halten kann (1..18) |
| scale | Die Anzahl Stellen nach dem Dezimalkomma (0.. <i>precision</i>) |
| size | Die maximale Größe eines Strings in Zeichen |
| charset_name | Der Name eines gültigen Zeichensatzes, falls der Zeichensatz der Spalte vom Standardzeichensatz der Datenbank abweichen soll |
| subtype_num | BLOB-Subtype-Nummer |
| subtype_name | BLOB-Subtyp-Mnemonikname |
| seglen | Segmentgröße (max. 65535) |
| select_one | Eine skalare SELECT-Anweisung — auswählen einer Spalte und zurückgeben nur einer Zeile |
| select_list | Eine SELECT-Anweisung, die eine Spalte auswählt und null oder mehr Zeilen zurückgibt |
| select_expr | Eine SELECT-Anweisung, die eine oder mehrere Spalten auswählt und null oder mehr Zeilen zurückgibt |
| expression | Ein Ausdruck, der auf einen Wert auflöst, der im angegebenen Kontext zulässig ist |
| genname | Sequenzname (Generatorname) |
| func | Interne Funktion oder UDF |

Die Anweisung `CREATE TABLE` erstellt eine neue Tabelle. Jeder Benutzer kann sie erstellen und ihr Name muss unter den Namen aller Tabellen, Ansichten und gespeicherten Prozeduren in der Datenbank eindeutig sein.

Eine Tabelle muss mindestens eine Spalte enthalten, die nicht berechnet wird, und die Namen der Spalten müssen in der Tabelle eindeutig sein.

Eine Spalte muss entweder einen expliziten *SQL-Datentyp*, den Namen einer *Domain*, dessen Attribute für die Spalte kopiert werden oder als `COMPUTED BY`-Ausdruck (ein *berechnetes Feld*).

Eine Tabelle kann eine beliebige Anzahl von Tabelleneinschränkungen haben, einschließlich keiner.

Eine Spalte nicht nullbar machen

In Firebird sind Spalten standardmäßig nullwertig. Die optionale `NOT NULL`-Klausel gibt an, dass die Spalte `NULL` anstelle eines Wertes nicht verwenden darf.

Zeichen-Spalten

Sie können die `CHARACTER SET`-Klausel verwenden, um den Zeichensatz für die Typen `CHAR`, `VARCHAR` und `BLOB (SUB_TYPE TEXT)` anzugeben. Wenn der Zeichensatz nicht angegeben ist, wird standardmäßig der während der Erstellung der Datenbank angegebene Zeichensatz verwendet. Wurde während der Erstellung der Datenbank kein Zeichensatz angegeben, wird standardmäßig der Zeichensatz `NONE` übernommen. In diesem Fall werden Daten gespeichert und abgerufen, wie sie übermittelt wurden. Daten in einer beliebigen Kodierung können zu einer solchen Spalte hinzugefügt werden, aber es ist nicht möglich, diese Daten zu einer Spalte mit einer anderen Kodierung hinzuzufügen. Keine Transliteration wird zwischen den Quell- und Zielcodierungen, das dies zu Fehlern führen kann.

Mit der optionalen `COLLATE`-Klausel können Sie die Sortierreihenfolge für Zeichendatentypen angeben, einschließlich `BLOB SUB_TYPE TEXT`. Wenn keine Sortierreihenfolge angegeben ist, wird standardmäßig die Sortierreihenfolge angewendet, die für den angegebenen Zeichensatz beim Erstellen der Spalte standardmäßig verwendet wird.

Angabe eines DEFAULT-Wertes

Die optionale `DEFAULT`-Klausel erlaubt Ihnen, den Standardwert für eine Tabellenspalte festzulegen. Dieser Wert wird der Spalte während der Ausführung eines `INSERT`-Statements zugewiesen, sofern kein anderer Wert festgelegt wurde *und* diese Spalte von der `INSERT`-Anweisung ausgelassen wurde.

Der Standardwert kann ein Literal eines kompatiblen Typs sein, eine Kontextvariable, die mit dem Datentyp der Spalte typkompatibel ist, oder `NULL`, wenn die Spalte dies zulässt. Wenn kein Standardwert explizit angegeben ist, wird `NULL` impliziert.

Ein Ausdruck kann nicht als Standardwert verwendet werden.

Domain-basierte Spalten

Um eine Spalte zu definieren, können Sie eine zuvor definierte Domain verwenden. Wenn die Definition einer Spalte auf einer Domain basiert, enthält sie möglicherweise einen neuen Standardwert, zusätzliche `CHECK`-Einschränkungen und eine `COLLATE`-Klausel, die die in der Domain angegebenen Werte überschreibt. Die

Definition einer solchen Spalte kann zusätzliche Spaltenbeschränkungen enthalten (z. B. NOT NULL), wenn die Domain diese nicht besitzt.

Wichtig

Es ist nicht möglich, eine Domain-basierte Spalte zu definieren, die nullbar ist, wenn die Domain mit dem Attribut NOT NULL definiert wurde. Wenn Sie eine Domain haben möchten, die zum Definieren sowohl von nullbaren als auch von nicht-nullbaren Spalten und Variablen verwendet werden kann, empfiehlt es sich, die Domain auf null zu setzen und NOT NULL in der Spaltendefinition zu verwenden.

Berechnete Felder

Berechnete Felder können in der Datenbank mittels COMPUTED [BY] oder GENERATED ALWAYS AS (gemäß SQL: 2003 Standard) definiert werden. Sie meinen dasselbe. Das Beschreiben des Datentyps ist für berechnete Felder nicht erforderlich (aber möglich), da das DBMS den entsprechenden Typ als Ergebnis der Ausdrucksanalyse berechnet und speichert. Entsprechende Operationen für die in einem Ausdruck enthaltenen Datentypen müssen genau angegeben werden.

Wenn der Datentyp explizit für ein berechnetes Feld angegeben wird, wird das Berechnungsergebnis in den angegebenen Typ konvertiert. Dies bedeutet zum Beispiel, dass das Ergebnis eines numerischen Ausdrucks als String dargestellt werden kann.

In einer Abfrage, die eine COMPUTED BY-Spalte auswählt, wird der Ausdruck für jede Zeile der ausgewählten Daten ausgewertet.

Tipp

Anstelle einer berechneten Spalte ist es in manchen Fällen sinnvoll, eine reguläre Spalte zu verwenden, deren Wert in Triggern zum Hinzufügen und Aktualisieren von Daten ausgewertet wird. Es kann die Leistung des Einfügens / Aktualisierens von Datensätzen verringern, aber es wird die Leistung der Datenabfrage erhöhen.

Definieren einer ARRAY-Spalte

- Wenn die Spalte ein Array sein soll, kann der Basistyp ein beliebiger SQL-Datentyp sein, mit Ausnahme von BLOB und ARRAY.
- Die Grenzen des Arrays werden in eckigen Klammern angegeben. (Im **Syntaxblock** werden diese Klammern fett dargestellt, um sie von eckigen Klammern zu unterscheiden, die optionale Syntaxelemente kennzeichnen.)
- Für jede Array-Dimension definieren eine oder zwei ganze Zahlen die untere und obere Grenze ihres Indexbereichs:
 - Standardmäßig sind Arrays 1-basiert. Die untere Grenze ist implizit und nur die obere Grenze muss angegeben werden. Eine einzelne Zahl kleiner als 1 definiert den Bereich $num - 1$ und eine Zahl größer als 1 definiert den Bereich $1.. num$.
 - Zwei durch einen Doppelpunkt getrennte Zahlen (':') und optionaler Leerraum, der zweite ist größer als der erste, können verwendet werden, um den Bereich explizit zu definieren. Eine oder beide Grenzen können kleiner als Null sein, solange die obere Grenze größer als die untere ist.
- Wenn das Array mehrere Dimensionen hat, müssen die Bereichsdefinitionen für jede Dimension durch Kommas und optionales Leerzeichen getrennt werden.
- Indizes werden *nur* validiert, wenn ein Array tatsächlich existiert. Dies bedeutet, dass keine Fehlermeldungen bezüglich ungültiger Subskripte zurückgegeben werden, wenn ein bestimmtes Element nichts zurückgibt oder wenn ein Array-Feld NULL ist.

Constraints

Es gibt vier Constraint-Typen. Diese sind:

- Primärschlüssel (PRIMARY KEY)
- Eindeutigkeitschlüssel (UNIQUE)
- Fremdschlüssel (REFERENCES)
- CHECK-Constraint (CHECK)

Constraints können auf Spaltenebene („Spaltenbeschränkungen“) oder auf Tabellenebene („Tabellenbeschränkungen“) angegeben werden. Einschränkungen auf Tabellenebene sind erforderlich, wenn Schlüssel (Eindeutigkeitsbeschränkung, Primärschlüssel, Fremdschlüssel) über mehrere Spalten hinweg gebildet werden sollen und wenn eine CHECK-Einschränkung neben der definierten Spalte andere Spalten in der Zeile einbezieht. Die Syntax für einige Constraint-Typen kann je nachdem, ob die Constraint auf Spalten- oder Tabellenebene definiert wird, leicht unterschiedlich sein.

- Eine Spaltenbeschränkung wird während einer Spaltendefinition angegeben, nachdem alle Spaltenattribute, mit Ausnahme von COLLATION, angegeben wurden und nur die in dieser Definition angegebene Spalte enthalten ist
- Einschränkungen auf Tabellenebene werden nach allen Spaltendefinitionen angegeben. Sie sind ein flexiblerer Weg, um Einschränkungen festzulegen, da sie Einschränkungen für mehrere Spalten berücksichtigen können
- Sie können Einschränkungen auf Tabellen- und Spaltenebene im gleichen CREATE TABLE-Statement mischen

Das System erstellt automatisch den entsprechenden Index für einen Primärschlüssel (PRIMARY KEY), einen eindeutigen Schlüssel (UNIQUE) und einen Fremdschlüssel (REFERENCES ist eine Einschränkung auf Spaltenebene, FOREIGN KEY REFERENCES eine auf Tabellenebene).

Name für Constraints und ihre Indizes

Spaltenbeschränkungen und ihre Indizes werden automatisch benannt:

- Der Constraint-Name besitzt die Form INTEG_n, wobei n ein oder mehrere Ziffern repräsentiert
- Der Indexname besitzt die Form RDB\$PRIMARYn (für einen Primärschlüsselindex), RDB\$FOREIGNn (für einen Fremdschlüsselindex) oder RDB\$n (für einen Eindeutigkeitsindex). Auch hier repräsentiert n eine oder mehrere Ziffern.

Die automatische Benennung von Einschränkungen auf Tabellenebene und deren Indizes folgt dem gleichen Muster, es sei denn, die Namen werden explizit angegeben.

Benannte Constraints

Ein Constraint kann explizit benannt werden, wenn für ihre Definition die CONSTRAINT-Klausel verwendet wird. Die CONSTRAINT-Klausel ist optional für die Definition von Spaltentypen auf Spaltenebene. Sie ist jedoch obligatorisch für Tabellenstufen. Standardmäßig hat der Constraint-Index denselben Namen wie die Constraint. Wenn für den Constraint-Index ein anderer Name gewünscht wird, kann eine USING-Klausel enthalten sein.

Die USING-Klausel

Die Klausel USING erlaubt Ihnen die benutzerdefinierte Benennung des Index, der automatisch erstellt wurde und, optional, die Definition der Indexrichtung —entweder aufsteigend (Standardwert) oder absteigend.

PRIMARY KEY

Der PRIMARY KEY-Constraint wird auf einer oder mehr *Schlüsselspalten* gebildet, wobei jede Spalte mit einem NOT NULL-Constraint definiert wurde. Die Werte in den Schlüsselspalten einer Zeile müssen eindeutig sein. Eine Tabelle kann nur einen Primärschlüssel enthalten.

- Ein einspaltiger Primärschlüssel kann als Spalten- oder Tabellenebene definiert werden
- Ein mehrspaltiger Primärschlüssel muss als Einschränkung auf Tabellenebene angegeben werden.

Der UNIQUE-Constraint

Die Einschränkung UNIQUE definiert die Anforderung der Eindeutigkeit des Inhalts für die Werte in einem Schlüssel in der gesamten Tabelle. Eine Tabelle kann eine beliebige Anzahl von eindeutigen Schlüsseinschränkungen enthalten.

Wie beim Primärschlüssel kann die eindeutige Einschränkung mehrspaltig sein. Ist dies der Fall, muss es als Einschränkung auf Tabellenebene angegeben werden.

NULL in Eindeutigkeitschlüsseln

Die SQL-99-konformen Regeln für UNIQUE-Constraints erlauben einen oder mehrere NULLs in einer Spalte mit einem UNIQUE-Constraint. Dadurch ist es möglich, eine UNIQUE-Beschränkung für eine Spalte zu definieren, die nicht die Einschränkung NOT NULL hat.

Bei UNIQUE-Schlüsseln, die mehrere Spalten umfassen, ist die Logik ein wenig kompliziert:

- Mehrere Zeilen mit Null in allen Spalten des Schlüssels sind erlaubt
- Mehrere Zeilen mit Schlüsseln mit verschiedenen Kombinationen von Nullen und Nicht-Null-Werten sind erlaubt
- Mehrere Zeilen mit denselben Schlüsselspalten null und der Rest mit Werten ungleich null sind zulässig, sofern sich die Werte in mindestens einer Spalte unterscheiden
- Mehrere Zeilen mit denselben Schlüsselspalten null und der Rest mit Nicht-Nullwerten, die in jeder Spalte gleich sind, verletzen die Bedingung

Die Regeln für die Eindeutigkeit lassen sich so zusammenfassen:

Illustration:

```
RECREATE TABLE t( x int, y int, z int, unique(x,y,z));
INSERT INTO t values( NULL, 1, 1 );
INSERT INTO t values( NULL, NULL, 1 );
INSERT INTO t values( NULL, NULL, NULL );
INSERT INTO t values( NULL, NULL, NULL ); -- Erlaubt
INSERT INTO t values( NULL, NULL, 1 );    -- Nicht erlaubt
```

FOREIGN KEY

Ein Fremdschlüssel stellt sicher, dass die teilnehmenden Spalten nur Werte enthalten dürfen, die auch in der referenzierten Spalte (n) in der Mastertabelle vorhanden sind. Diese referenzierten Spalten werden oft als *Zielspalten* bezeichnet. Sie müssen der Primärschlüssel oder ein eindeutiger Schlüssel in der Zieltabelle sein. Sie müssen keine NOT NULL-Einschränkung definiert haben, obwohl sie, wenn sie der Primärschlüssel sind, natürlich diese Einschränkung haben.

Die Fremdschlüsselspalten in der referenzierenden Tabelle selbst erfordern keine NOT NULL-Einschränkung.

Ein einspaltiger Fremdschlüssel kann in der Spaltendeklaration definiert werden, wobei das Schlüsselwort REFERENCES verwendet wird:

```
... ,
ARTIFACT_ID INTEGER REFERENCES COLLECTION (ARTIFACT_ID) ,
```

Die Spalte ARTIFACT_ID im Beispiel verweist auf eine gleichnamige Spalte in der Tabelle COLLECTIONS.

Sowohl einspaltige als auch mehrspaltige Fremdschlüssel können auf der *Tabellenebene* definiert werden. Bei einem mehrspaltigen Fremdschlüssel ist die Deklaration auf Tabellenebene die einzige Option. Diese Methode ermöglicht auch die Angabe eines optionalen Namens für die Einschränkung:

```
...
CONSTRAINT FK_ARTSOURCE FOREIGN KEY(DEALER_ID, COUNTRY)
REFERENCES DEALER (DEALER_ID, COUNTRY) ,
```

Beachten Sie, dass sich die Spaltennamen in der referenzierten Tabelle („master“) möglicherweise von denen im Fremdschlüssel unterscheiden.

Anmerkung

Wenn keine Zielspalten angegeben sind, verweist der Fremdschlüssel automatisch auf den Primärschlüssel der Zieltabelle.

Fremdschlüsselaktionen

Mit den Unterklauseln ON UPDATE und ON DELETE ist es möglich, eine Aktion für die betroffenen Fremdschlüsselspalte anzugeben, wenn referenzierte Werte in der Mastertabelle geändert werden:

- NO ACTION (der Standard) - nichts wird getan
- CASCADE - Die Änderung in der Mastertabelle wird an die entsprechende(n) Zeile(n) in der untergeordneten Tabelle weitergegeben. Wenn sich ein Schlüsselwert ändert, ändert sich der entsprechende Schlüssel in den untergeordneten Datensätzen auf den neuen Wert; Wenn die Master-Zeile gelöscht wird, werden die untergeordneten Datensätze gelöscht.
- SET DEFAULT - Die Fremdschlüsselspalten in den betroffenen Zeilen werden wie bei der Definition der Fremdschlüsselbeschränkung *auf ihre Standardwerte gesetzt*.
- SET NULL - Die Fremdschlüssel-Spalten in den betroffenen Zeilen werden auf NULL gesetzt.

Die angegebene Aktion oder die Standardeinstellung NO ACTION könnte dazu führen, dass eine Fremdschlüsselspalte ungültig wird. Beispielsweise könnte er einen Wert erhalten, der in der Mastertabelle nicht vorhanden ist, oder er könnte NULL werden, während die Spalte eine NOT NULL-Einschränkung hat. Solche Bedingungen führen dazu, dass die Operation in der Master-Tabelle mit einer Fehlermeldung fehlschlägt.

Beispiel:

```
...  
CONSTRAINT FK_ORDERS_CUST  
FOREIGN KEY (CUSTOMER) REFERENCES CUSTOMERS (ID)  
ON UPDATE CASCADE ON DELETE SET NULL
```

CHECK-Constraint

Die Bedingung CHECK definiert die Bedingung, die die in diese Spalte eingefügten Werte erfüllen müssen. Eine Bedingung ist ein logischer Ausdruck (auch Prädikat genannt), der die Werte TRUE, FALSE und UNKNOWN zurückgeben kann. Eine Bedingung gilt als erfüllt, wenn das Prädikat TRUE oder den Wert UNKNOWN (entspricht NULL) zurückgibt. Wenn das Prädikat FALSE zurückgibt, wird der Wert nicht akzeptiert. Diese Bedingung wird zum Einfügen einer neuen Zeile in die Tabelle (die Anweisung INSERT) und zum Aktualisieren des vorhandenen Werts der Tabellenspalte (die Anweisung UPDATE wo eine dieser Aktionen stattfinden kann (UPDATE ODER INSERT, MERGE)).

Wichtig

Eine CHECK-Beschränkung für eine Domainbasierte Spalte ersetzt eine vorhandene CHECK-Bedingung in der Domain nicht, sondern wird zu einer Ergänzung. Die Firebird-Engine hat während der Definition keine Möglichkeit zu überprüfen, ob der zusätzliche CHECK nicht mit dem vorhandenen übereinstimmt.

CHECK-Bedingungen — ob auf Tabellen- oder Spaltenebene definiert — beziehen sich auf Tabellenspalten *durch ihren Namen*. Die Verwendung des Schlüsselworts VALUE als Platzhalter wie auch in der CHECK-Bedingung für Domains ist im Kontext der Definition von Spalteneinschränkungen nicht zulässig.

Beispiele: mit zwei Einschränkungen auf Spaltenebene und einer auf Tabellenebene:

```
CREATE TABLE PLACES (  
...  
LAT DECIMAL(9, 6) CHECK (ABS(LAT) <= 90),  
LON DECIMAL(9, 6) CHECK (ABS(LON) <= 180),  
...  
CONSTRAINT CHK_POLES CHECK (ABS(LAT) < 90 OR LON = 0)  
);
```

Global Temporary Tables (GTT)

Global Temporary Tables haben persistente Metadaten, aber ihre Inhalte sind transaktions- (Standard) oder verbindungsgebunden. Jede Transaktion oder Verbindung hat eine eigene private Instanz eines GTT, isoliert von allen anderen. Instanzen werden nur dann erstellt, wenn auf das GTT verwiesen wird. Sie werden beim Beenden der Transaktion oder beim Trennen zerstört. Die Metadaten eines GTT können mit ALTER TABLE bzw. DROP TABLE geändert oder entfernt werden.

Syntax:

```
CREATE GLOBAL TEMPORARY TABLE name
 (<column_def> [, {<column_def> | <table_constraint>} ...])
 [ON COMMIT {DELETE | PRESERVE} ROWS]
```

Hinweise zur Syntax

- ON COMMIT DELETE ROWS erstellt eine transaktionsgebundene GTT (der Standard) ON COMMIT PRESERVE ROWS eine verbindungsgebundene GTT
- Eine EXTERNAL [FILE]-Klausel ist nicht zulässig in der Definition einer GTT

Einschränkungen für GTTs

GTTs können mit allen Features und Utensilien gewöhnlicher Tabellen (Schlüssel, Referenzen, Indizes, Trigger usw.) „ausgestattet werden“, aber es gibt einige Einschränkungen:

- GTTs und reguläre Tabellen können sich nicht gegenseitig referenzieren
- Eine verbindungsgebundene („PRESERVE ROWS“) GTT kann nicht auf eine transaktionsgebundene („DELETE ROWS“) GTT verweisen
- Domain-Constraints können nicht auf GTTs verweisen
- Die Zerstörung einer GTT-Instanz am Ende ihres Lebenszyklus löst keine BEFORE / AFTER-Löschtrigger aus

Tipp

In einer vorhandenen Datenbank ist es nicht immer einfach, eine reguläre Tabelle von einer GTT oder einer GTT auf Transaktions- von einer GTT auf Verbindungsebene zu unterscheiden. Verwenden Sie diese Abfrage, um herauszufinden, welche Art von Tabelle Sie betrachten:

```
select t.rdb$type_name
       from rdb$relations r
       join rdb$types t on r.rdb$relation_type = t.rdb$type
       where t.rdb$field_name = 'RDB$RELATION_TYPE'
       and r.rdb$relation_name = 'TABLENAME'
```

Für einen Überblick über die Typen aller Beziehungen in der Datenbank:

```
select r.rdb$relation_name, t.rdb$type_name
       from rdb$relations r
       join rdb$types t on r.rdb$relation_type = t.rdb$type
       where t.rdb$field_name = 'RDB$RELATION_TYPE'
       and coalesce (r.rdb$system_flag, 0) = 0
```

Das Feld RDB\$TYPE_NAME zeigt PERSISTENT für eine reguläre Tabelle, VIEW für eine Ansicht, GLOBAL_TEMPORARY_PRESERVE für eine verbindungsgebundene GTT und GLOBAL_TEMPORARY_DELETE für eine transaktionsgebundene GTT.

Externe Tabellen

Die optionale Klausel EXTERNAL [FILE] gibt an, dass die Tabelle außerhalb der Datenbank in einer externen Textdatei mit Datensätzen fester Länge gespeichert wird. Die Spalten einer Tabelle, die in einer externen Datei gespeichert sind, können von einem beliebigen Typ außer BLOB oder ARRAY sein.

Alles, was Sie mit einer in einer externen Datei gespeicherten Tabelle tun können, ist, neue Zeilen einzufügen (INSERT) und die Daten abzufragen). Das Aktualisieren vorhandener Daten (UPDATE) und das Löschen von Zeilen (DELETE) sind nicht möglich.

Eine Datei, die als externe Tabelle definiert ist, muss sich auf einem Speichergerät befinden, das physisch auf dem Computer vorhanden ist, auf dem der Firebird-Server ausgeführt wird. Wenn der Parameter *ExternalFileAccess* in der Konfigurationsdatei *firebird.conf* auf `Restrict` lautet, muss in einem der dort aufgeführten Verzeichnisse das Argument für `Restrict` stehen. Wenn die Datei nicht schon existiert, wird Firebird diese beim ersten Zugriff erstellen.

Wichtig

Die Möglichkeit, externe Dateien für eine Tabelle zu verwenden, hängt vom Wert ab, der für den Parameter *ExternalFileAccess* in *firebird.conf* festgelegt wurde:

- Wenn der Parameter auf `None` (Standardeinstellung) eingestellt ist, wird jeder Zugriff auf eine externe Datei verweigert.
- Die Einstellung `Restrict` wird empfohlen, um den Zugriff externer Dateien auf Verzeichnisse einzuschränken, die vom Serveradministrator explizit für diesen Zweck erstellt wurden. Beispielsweise:
 - `ExternalFileAccess = Restrict externalfiles` schränkt den Zugriff auf ein Verzeichnis namens `externalfiles` direkt unterhalb des Firebird Wurzelverzeichnisses ein
 - `ExternalFileAccess = d:\databases\outfiles; e:\infiles` schränkt den Zugriff auf die zwei angegebenen Verzeichnisse des Windows-Hostservers ein. Beachten Sie, dass jeder Pfad, der eine Netzwerkzuordnung ist, nicht funktioniert. Pfade, die in einfachen oder doppelten Anführungszeichen eingeschlossen sind, funktionieren auch nicht.
- Wenn dieser Parameter auf `Full` eingestellt ist, kann auf externe Dateien im Host-Dateisystem zugegriffen werden. Es schafft eine Sicherheitslücke und wird nicht empfohlen.

Externes Dateiformat

Das „Zeilen“-Format einer externen Tabelle besteht aus fester Länge. Es gibt keine Feldbegrenzer: Sowohl Feld- als auch Zeilengrenzen werden durch die maximale Größe der Felddefinitionen in Bytes bestimmt. Dies ist sowohl beim Definieren der Struktur der externen Tabelle als auch beim Entwerfen einer Eingabedatei für eine externe Tabelle wichtig, die Daten aus einer anderen Anwendung importieren soll. Das ubiquitäre „csv“-Format ist z. B. nicht nützlich als Eingabedatei und kann nicht direkt in eine externe Datei generiert werden.

Der nützlichste Datentyp für die Spalten externer Tabellen ist der Typ CHAR mit fester Länge mit geeigneten Längen für die Daten, die sie tragen sollen. Datums- und Zahlentypen können problemlos zu und von Zeichenketten umgewandelt werden, solange die Dateien nicht von einer anderen Firebird-Datenbank gelesen werden sollen, werden die nativen Datentypen externen Anwendungen als unparsbar „alphabeti“ angezeigt.

Natürlich gibt es Möglichkeiten, typisierte Daten zu manipulieren, um Ausgabedateien von Firebird zu generieren, die als Eingabedateien für andere Anwendungen mithilfe von gespeicherten Prozeduren mit oder ohne externe Tabellen gelesen werden können. Solche Techniken liegen außerhalb des Bereichs einer Sprachreferenz. Hier finden Sie einige Richtlinien und Tipps zum Erstellen und Arbeiten mit einfachen Textdateien, da die externe Tabellenfunktion häufig als einfache Möglichkeit zum Erstellen oder Lesen von transaktionsunabhängigen Protokollen verwendet wird, die offline in einem Texteditor oder in einem Auditing untersucht werden können Anwendung.

Zeilenbegrenzer

Im Allgemeinen sind externe Dateien nützlicher, wenn Zeilen durch ein Trennzeichen in Form einer „newline“-Sequenz getrennt werden, die von Leseanwendungen auf der vorgesehenen Plattform erkannt wird. Für die meisten Kontexte unter Windows ist dies die Zwei-Byte-CRLF-Sequenz, Wagenrücklauf (ASCII-Code dezimal 13) und Zeilenvorschub (ASCII-Code dezimal 10). Auf POSIX ist LF für sich allein üblich; Bei einigen MacOSX-Anwendungen kann es sich um LFCR handeln. Es gibt verschiedene Möglichkeiten, diese Begrenzerspalte zu füllen. In unserem Beispiel unten geschieht dies mittel Before Insert-Trigger und der internen Funktion ASCII_CHAR.

Beispiel einer externen Tabelle

In unserem Beispiel definieren wir eine externe Protokolltabelle, die von einem Ausnahmehandler in einer gespeicherten Prozedur oder einem Trigger verwendet werden kann. Die externe Tabelle wird ausgewählt, da die Nachrichten aus den behandelten Ausnahmen im Protokoll beibehalten werden, selbst wenn die Transaktion, die den Prozess gestartet hat, aufgrund einer anderen, nicht behandelten Ausnahme schließlich zurückgesetzt wird. Zu Demonstrationszwecken gibt es nur zwei Datenspalten, einen Zeitstempel und eine Nachricht. Die dritte Spalte speichert den Zeilenbegrenzer:

```
CREATE TABLE ext_log
  EXTERNAL FILE 'd:\externals\log_me.txt' (
    stamp CHAR (24),
    message CHAR(100),
    crlf CHAR(2) ); -- for a Windows context
COMMIT;
```

Jetzt noch einen Trigger erstellen, der Zeitstempel und Zeilenbegrenzer festlegt, sobald eine Nachricht in die Datei geschrieben wird:

```
SET TERM ^;
CREATE TRIGGER bi_ext_log FOR ext_log
ACTIVE BEFORE INSERT
AS
BEGIN
  IF (new.stamp is NULL) then
    new.stamp = CAST (CURRENT_TIMESTAMP as CHAR(24));
    new.crlf = ASCII_CHAR(13) || ASCII_CHAR(10);
  END ^
COMMIT ^
SET TERM ;^
```

Einfügen von Datensätzen (die von einem Exception-Handler oder Shakespeare-Fan gemacht worden sein könnten):

```
insert into ext_log (message)
values('Shall I compare thee to a summer''s day?');
insert into ext_log (message)
values('Thou art more lovely and more temperate')
```

Die Ausgabe:

2015-10-07 15:19:03.4110 Shall I compare thee to a summer's day?
2015-10-07 15:19:58.7600 Thou art more lovely and more temperate

Beispiele für CREATE TABLE

1. Erstellen der Tabelle COUNTRY mit dem als Spaltenbeschränkung angegebenen Primärschlüssel.

```
CREATE TABLE COUNTRY (  
  COUNTRY COUNTRYNAME NOT NULL PRIMARY KEY,  
  CURRENCY VARCHAR(10) NOT NULL);
```

2. Erstellen der STOCK-Tabelle mit dem auf der Spaltenebene angegebenen benannten Primärschlüssel und dem auf der Tabellenebene angegebenen benannten eindeutigen Schlüssel.

```
CREATE TABLE STOCK (  
  MODEL SMALLINT NOT NULL CONSTRAINT PK_STOCK PRIMARY KEY,  
  MODELNAME CHAR(10) NOT NULL,  
  ITEMID INTEGER NOT NULL,  
  CONSTRAINT MOD_UNIQUE UNIQUE (MODELNAME, ITEMID));
```

3. Erstellen der JOB-Tabelle mit einer Primärschlüsselbeschränkung, die zwei Spalten umfasst, eine Fremdschlüsselbeschränkung für die Tabelle COUNTRY und eine CHECK-Einschränkung auf Tabellenebene. Die Tabelle enthält außerdem eine Reihe von 5 Elementen.

```
CREATE TABLE JOB (  
  JOB_CODE          JOBCODE NOT NULL,  
  JOB_GRADE         JOBGRADE NOT NULL,  
  JOB_COUNTRY       COUNTRYNAME,  
  JOB_TITLE         VARCHAR(25) NOT NULL,  
  MIN_SALARY        NUMERIC(18, 2) DEFAULT 0 NOT NULL,  
  MAX_SALARY        NUMERIC(18, 2) NOT NULL,  
  JOB_REQUIREMENT   BLOB SUB_TYPE 1,  
  LANGUAGE_REQ      VARCHAR(15) [1:5],  
  PRIMARY KEY (JOB_CODE, JOB_GRADE),  
  FOREIGN KEY (JOB_COUNTRY) REFERENCES COUNTRY (COUNTRY)  
  ON UPDATE CASCADE  
  ON DELETE SET NULL,  
  CONSTRAINT CHK_SALARY CHECK (MIN_SALARY < MAX_SALARY)  
);
```

4. Erstellen der PROJECT-Tabelle mit Primär-, Fremd- und eindeutigen Schlüsseleinschränkungen mit benutzerdefinierten Indexnamen, die mit der USING-Klausel angegeben wurden.

```
CREATE TABLE PROJECT (  
  PROJ_ID          PROJNO NOT NULL,  
  PROJ_NAME        VARCHAR(20) NOT NULL UNIQUE USING DESC INDEX IDX_PROJNAME,  
  PROJ_DESC        BLOB SUB_TYPE 1,  
  TEAM_LEADER      EMPNO,
```

```
PRODUCT      PRODTYPE,  
CONSTRAINT PK_PROJECT PRIMARY KEY (PROJ_ID) USING INDEX IDX_PROJ_ID,  
FOREIGN KEY (TEAM_LEADER) REFERENCES EMPLOYEE (EMP_NO)  
    USING INDEX IDX_LEADER  
);
```

5. Erstellen der Tabelle SALARY_HISTORY mit zwei berechneten Feldern. Der erste wird gemäß dem Standard SQL:2003 deklariert, während der zweite gemäß der traditionellen Deklaration der berechneten Felder in Firebird deklariert wird.

```
CREATE TABLE SALARY_HISTORY (  
    EMP_NO          EMPNO NOT NULL,  
    CHANGE_DATE     TIMESTAMP DEFAULT 'NOW' NOT NULL,  
    UPDATER_ID      VARCHAR(20) NOT NULL,  
    OLD_SALARY      SALARY NOT NULL,  
    PERCENT_CHANGE  DOUBLE PRECISION DEFAULT 0 NOT NULL,  
    SALARY_CHANGE   GENERATED ALWAYS AS  
        (OLD_SALARY * PERCENT_CHANGE / 100),  
    NEW_SALARY      COMPUTED BY  
        (OLD_SALARY + OLD_SALARY * PERCENT_CHANGE / 100)  
);
```

6. Erstellen einer verbindungsabhängigen Global Temporary Table.

```
CREATE GLOBAL TEMPORARY TABLE MYCONNGTT (  
    ID INTEGER NOT NULL PRIMARY KEY,  
    TXT VARCHAR(32),  
    TS TIMESTAMP DEFAULT CURRENT_TIMESTAMP)  
ON COMMIT PRESERVE ROWS;
```

7. Erstellen einer transaktionsbezogenen globalen temporären Tabelle, die einen Fremdschlüssel verwendet, um auf eine globale temporäre Tabelle mit Verbindungsbereich zu verweisen. Die ON COMMIT-Unterklausele ist optional, da DELETE ROWS die Standardeinstellung ist.

```
CREATE GLOBAL TEMPORARY TABLE MYTXGTT (  
    ID INTEGER NOT NULL PRIMARY KEY,  
    PARENT_ID INTEGER NOT NULL REFERENCES MYCONNGTT(ID),  
    TXT VARCHAR(32),  
    TS TIMESTAMP DEFAULT CURRENT_TIMESTAMP)  
ON COMMIT DELETE ROWS;
```

ALTER TABLE

Benutzt für: Ändern der Tabellenstruktur.

Verfügbar in: DSQL, ESQL

Syntax:

```

ALTER TABLE tablename
<operation> [, <operation> ...]

<operation> ::= ADD <col_def>
                ADD <tconstraint>
                DROP colname
                DROP CONSTRAINT constr_name
                ALTER [COLUMN] colname <col_mod>

<col_def> ::= <regular_col_def> | <computed_col_def>

<regular_col_def> ::=
  colname {<datatype> | domainname}
  [DEFAULT {literal | NULL | <context_var>}]
  [NOT NULL]
  [<col_constraint>]
  [COLLATE collation_name]

<computed_col_def> ::=
  colname [<datatype>]
  {COMPUTED [BY] | GENERATED ALWAYS AS} (<expression>)

<col_mod> ::= <regular_col_mod> | <computed_col_mod>

<regular_col_mod> ::=
  TO newname
  | POSITION newpos
  | TYPE {<datatype> | domainname}
  | SET DEFAULT {literal | NULL | <context_var>}
  | DROP DEFAULT

<computed_col_mod> ::=
  TO newname
  | POSITION newpos
  | [TYPE <datatype>] {COMPUTED [BY] | GENERATED ALWAYS AS} (<expression>)

<datatype> ::=
  {SMALLINT | INTEGER | BIGINT} [<array_dim>]
  | {FLOAT | DOUBLE PRECISION} [<array_dim>]
  | {DATE | TIME | TIMESTAMP} [<array_dim>]
  | {DECIMAL | NUMERIC} [(precision [, scale])] [<array_dim>]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
  | [<array_dim>] [CHARACTER SET charset_name]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING]
  | [(size)] [<array_dim>]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
  | [SEGMENT SIZE seglen] [CHARACTER SET charset_name]
  | BLOB [(seglen [, subtype_num])]

<array_dim> ::= [[m]:n [, [m]:n ...]]

<col_constraint> ::=
  [CONSTRAINT constr_name]
  {
    PRIMARY KEY [<using_index>]
    | UNIQUE      [<using_index>]
    | REFERENCES other_table [(colname)] [<using_index>]
    | [ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
  }

```

```

    [ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
    | CHECK (<check_condition>) }

<tconstraint> ::=
    [CONSTRAINT constr_name]
    {
        PRIMARY KEY (col_list) [<using_index>]
        | UNIQUE      (col_list) [<using_index>]
        | FOREIGN KEY (col_list)
            REFERENCES other_table [(col_list)] [<using_index>]
            [ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
            [ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
        | CHECK (<check_condition>) }

<col_list> ::= colname [, colname ...]

<using_index> ::= USING
[ASC[ENDING] | DESC[ENDING]] INDEX indexname

<check_condition> ::=
    <val> <operator> <val>
    | <val> [NOT] BETWEEN <val> AND <val>
    | <val> [NOT] IN (<val> [, <val> ...] | <select_list>)
    | <val> IS [NOT] NULL
    | <val> IS [NOT] DISTINCT FROM <val>
    | <val> [NOT] CONTAINING <val>
    | <val> [NOT] STARTING [WITH] <val>
    | <val> [NOT] LIKE <val> [ESCAPE <val>]
    | <val> [NOT] SIMILAR TO <val> [ESCAPE <val>]
    | <val> <operator> {ALL | SOME | ANY} (<select_list>)
    | [NOT] EXISTS (<select_expr>)
    | [NOT] SINGULAR (<select_expr>)
    | (<search_condition>)
    | NOT <search_condition>
    | <search_condition> OR <search_condition>
    | <search_condition> AND <search_condition>

<operator> ::=
    <> | != | ^= | ~= | = | < | > | <= | >= | !< | ^< | ~< | !> | ^> | ~>

<val> ::=
    colname [[<array_idx> [, <array_idx> ...]]]
    | literal
    | <context_var>
    | <expression>
    | NULL
    | NEXT VALUE FOR genname
    | GEN_ID(genname, <val>)
    | CAST(<val> AS <datatype>)
    | (<select_one>)
    | func([<val> [, <val> ...]])

```

Tabelle 5.8. ALTER TABLE Statement-Parameter

| Parameter | Beschreibung |
|-----------|--|
| tablename | Name (Kennung) für die Tabelle. Sie kann aus bis zu 31 Zeichen bestehen und in der Datenbank eindeutig sein. |

| Parameter | Beschreibung |
|------------------|--|
| operation | Eine der verfügbaren Operationen, die die Struktur der Tabelle verändert |
| colname | Name (Bezeichner) für eine Spalte in der Tabelle, max. 31 Zeichen. Muss in der Tabelle eindeutig sein. |
| newname | Neuer Name (Bezeichner) für die Spalte, max. 31 Zeichen. Muss in der Tabelle eindeutig sein. |
| newpos | Die neue Spaltenposition (eine Ganzzahl zwischen 1 und der Anzahl der Spalten in der Tabelle) |
| col_constraint | Spalten-Constraint |
| tconstraint | Tabellen-Constraint |
| constr_name | Der Name (Bezeichner) einer Einschränkung. Darf aus bis zu 31 Zeichen bestehen. |
| other_table | Der Name der Tabelle, auf die die Constraint verweist |
| literal | Ein Literalwert, der im angegebenen Kontext zulässig ist |
| context_var | Beliebige Kontextvariable, deren Datentyp im angegebenen Kontext zulässig ist |
| check_condition | Die Bedingung, die auf eine CHECK-Einschränkung angewendet wird, die als wahr, false oder NULL aufgelöst wird. |
| collation | Collation |
| array_dim | Array-Dimensionen |
| m, n | INTEGER-Ganzzahlen die den Bereich der Array-Dimensionen angeben |
| precision | Die Gesamtzahl der signifikanten Ziffern, die ein Wert des Datentyps halten kann (1..18) |
| scale | Die Anzahl Stellen nach dem Dezimalkomma (0.. <i>precision</i>) |
| size | Die maximale Größe eines Strings in Zeichen |
| charset_name | Der Name eines gültigen Zeichensatzes, falls der Zeichensatz der Spalte vom Standardzeichensatz der Datenbank abweichen soll |
| subtype_num | BLOB-Subtype-Nummer |
| subtype_name | BLOB-Subtyp-Mnemonikname |
| seglen | Segmentgröße (max. 65535) |
| select_one | Eine skalare SELECT-Anweisung — auswählen einer Spalte und zurückgeben nur einer Zeile |
| select_list | Eine SELECT-Anweisung, die eine Spalte auswählt und null oder mehr Zeilen zurückgibt |
| select_expr | Eine SELECT-Anweisung, die eine oder mehrere Spalten auswählt und null oder mehr Zeilen zurückgibt |

| Parameter | Beschreibung |
|------------|---|
| expression | Ein Ausdruck, der auf einen Wert auflöst, der im angegebenen Kontext zulässig ist |
| genname | Sequenzname (Generatorname) |
| func | Interne Funktion oder UDF |

Die ALTER TABLE-Anweisung ändert die Struktur einer vorhandenen Tabelle. Mit einer Anweisung ALTER TABLE können Sie mehrere Operationen ausführen, Spalten und Einschränkungen hinzufügen und löschen sowie Spaltenangaben ändern.

Mehrere Operationen in einer Anweisung ALTER TABLE werden durch Kommata getrennt.

Zählung von Versionsinkrementen

Einige Änderungen in der Struktur einer Tabelle erhöhen den Metadatenänderungszähler („Versionszähler“), der jeder Tabelle zugewiesen ist. Die Anzahl der Metadatenänderungen ist für jede Tabelle auf 255 beschränkt. Sobald der Zähler die Grenze von 255 erreicht hat, können Sie keine weiteren Änderungen an der Struktur der Tabelle vornehmen, ohne den Zähler zurückzusetzen.

So setzen Sie den Metadaten-Änderungszähler zurück: Sie sollten die Datenbank mithilfe des Dienstprogramms *gbak* sichern und wiederherstellen.

Die ADD-Klausel

Mit der ADD-Klausel können Sie eine neue Spalte oder eine neue Tabellenbeschränkung hinzufügen. Die Syntax zum Definieren der Spalte und die Syntax zum Definieren der Tabellenbeschränkung entsprechen denen, die für die Anweisung CREATE TABLE beschrieben wurden.

Auswirkung auf die Versionszählung:

- Jedes Mal, wenn eine neue Spalte hinzugefügt wird, wächst der Metadatenänderungszähler um eins
- Das Hinzufügen einer neuen Tabellenbeschränkung erhöht den Metadatenänderungszähler nicht

Zu beachtende Punkte

1. Seien Sie vorsichtig beim Hinzufügen einer neuen Spalte mit der Einschränkungsgruppe NOT NULL. Es kann dazu führen, dass die logische Integrität von Daten beeinträchtigt wird, wenn Sie in einer Spalte, die nicht auf Null gesetzt werden kann, Datensätze mit NULL haben. Beim Hinzufügen einer nicht-nullbaren Spalte wird empfohlen, entweder einen Standardwert dafür festzulegen oder die Spalte in vorhandenen Zeilen mit einem anderen Wert als Null zu aktualisieren.
2. Wenn eine neue CHECK-Beschränkung hinzugefügt wird, werden vorhandene Daten nicht auf Übereinstimmung geprüft. Es wird empfohlen, vorhandene Daten mit dem neuen CHECK-Ausdruck zu testen.

Die DROP-Klausel

Die Klausel DROP < Spaltenname > löscht die angegebene Spalte aus der Tabelle. Ein Versuch, eine Spalte zu löschen, schlägt fehl, wenn irgendetwas darauf verweist. Betrachten Sie die folgenden Elemente als Quellen potenzieller Abhängigkeiten:

- Spalten- oder Tabellenbeschränkungen
- Indizes
- Stored Procedures und Trigger
- Ansichten (Views)

Auswirkung auf Versionszählung: Jedes Mal, wenn eine Spalte gelöscht wird, wird der Metadatenänderungszähler der Tabelle um eins erhöht.

Die DROP CONSTRAINT-Klausel

Die DROP CONSTRAINT-Klausel löscht die angegebene Spalten- oder Tabellenebenenbeschränkung.

Eine PRIMARY KEY- oder UNIQUE-Beschränkung kann nicht gelöscht werden, wenn sie in einer anderen Tabelle durch eine FOREIGN KEY-Constraint referenziert wird. Es ist notwendig, die FOREIGN KEY-Beschränkung zu löschen, bevor Sie versuchen, die hierauf verweisenden PRIMARY KEY-Constraints oder UNIQUE-Constraints zu löschen.

Auswirkung auf Versionszähler: Das Löschen einer Spaltenbeschränkung oder einer Tabellenbeschränkung erhöht den Metadatenänderungszähler nicht.

Die ALTER [COLUMN]-Klausel

Mit der ALTER [COLUMN]-Klausel können Attribute bestehender Spalten geändert werden, ohne dass die Spalte gelöscht oder neu hinzugefügt werden muss. Erlaubte Änderungen sind:

- Ändern des Namens (wirkt sich nicht auf den Metadatenversionszähler aus)
- Ändern des Datentyps (erhöht den Metadatenversionszähler um eins)
- Ändern der Spaltenposition in der Spaltenliste der Tabelle (wirkt sich nicht auf den Metadatenversionszähler aus)
- Löschen des Standardspaltenwert (wirkt sich nicht auf den Metadatenversionszähler aus)
- Festlegen des Standardspaltenwertes oder den vorhandenen Standardwert ändern (hat keinen Einfluss auf den Metadatenversionszähler)
- Ändern des Typs und des Ausdrucks für eine berechnete Spalte (wirkt sich nicht auf den Metadatenversionszähler aus)

Umbenennen einer Spalte: das TO-Schlüsselwort

Das TO-Schlüsselwort mit einer neuen Kennung benennt eine vorhandene Spalte um. Die Tabelle darf keine existierende Spalte haben, die den gleichen Bezeichner hat.

Es ist nicht möglich, den Namen einer Spalte zu ändern, die in einer Einschränkung enthalten ist: PRIMARY KEY, UNIQUE-Schlüssel, FOREIGN KEY, Spaltenbeschränkung oder der CHECK-Constraint einer Tabelle.

Das Umbenennen einer Spalte ist ebenfalls nicht zulässig, wenn die Spalte in einem Trigger, einer gespeicherten Prozedur oder einer Ansicht (View) verwendet wird.

Ändern des Datentyps einer Spalte: Das TYPE-Schlüsselwort

Das Schlüsselwort TYPE ändert den Datentyp einer vorhandenen Spalte in einen anderen, zulässigen Typ. Eine Typänderung, die zu einem Datenverlust führen kann, wird nicht zugelassen. Beispielsweise kann die Anzahl der Zeichen im neuen Typ für eine CHAR- oder VARCHAR-Spalte nicht kleiner sein als die bestehende Spezifikation dafür.

Wenn die Spalte als Array deklariert wurde, ist keine Änderung des Typs oder der Anzahl der Dimensionen zulässig.

Der Datentyp einer Spalte, die an einem Fremdschlüssel, einem Primärschlüssel oder einer eindeutigen Einschränkung beteiligt ist, kann überhaupt nicht geändert werden.

Ändern der Position einer Spalte: Das POSITION-Schlüsselwort

Das POSITION-Schlüsselwort ändert die Position einer vorhandenen Spalte im fiktiven „von links nach rechts“-Layout des Datensatzes.

Die Nummerierung der Spaltenpositionen beginnt bei 1.

- Wenn eine Position kleiner als 1 angegeben ist, wird eine Fehlermeldung zurückgegeben
- Wenn eine Positionsnummer größer als die Anzahl der Spalten in der Tabelle ist, wird die neue Position automatisch an die Anzahl der Spalten angepasst.

Die DROP DEFAULT- und SET DEFAULT-Klauseln

Die optionale DROP DEFAULT-Klausel löscht den Standardwert für die Spalte, wenn sie zuvor durch eine CREATE TABLE- oder ALTER TABLE-Anweisung dort gesetzt wurde.

- Wenn die Spalte auf einer Domäne mit einem Standardwert basiert, wird der Standardwert auf die Standard-domain zurückgesetzt
- Ein Ausführungsfehler wird ausgelöst, wenn versucht wird, den Standardwert einer Spalte zu löschen, die keinen Standardwert hat oder deren Standardwert Domain-basiert ist

Die optionale SET DEFAULT-Klausel setzt einen Standardwert für die Spalte. Wenn die Spalte bereits einen Standardwert hat, wird sie durch die neue ersetzt. Der Standardwert, der auf eine Spalte angewendet wird, überschreibt immer einen von einer Domain geerbten Wert.

Die COMPUTED [BY]- oder GENERATED ALWAYS AS-Klauseln

Der Datentyp und der Ausdruck, die einer berechneten Spalte zugrunde liegen, können mit der ALTER TABLE ALTER [COLUMN]-Anweisung angepasst werden. Die Umwandlung einer regulären Spalte in eine berechnete und umgekehrt ist nicht zulässig.

Attribute, die nicht geändert werden können

Folgende Änderungen werden nicht unterstützt:

- Aktivieren oder Deaktivieren der Einschränkung NOT NULL für eine Spalte

- Ändern der Standardkollation für eine Zeichentypspalte

Nur der Tabelleneigentümer und [Administratoren](#) haben die Berechtigung für ALTER TABLE.

Beispiele für die Verwendung von ALTER TABLE

1. Hinzufügen der Spalte CAPITAL zur Tabelle COUNTRY.

```
ALTER TABLE COUNTRY
ADD CAPITAL VARCHAR(25);
```

2. Hinzufügen der CAPITAL-Spalte mit der UNIQUE-Einschränkung und Löschen der CURRENCY-Spalte.

```
ALTER TABLE COUNTRY
  ADD CAPITAL VARCHAR(25) NOT NULL UNIQUE,
  DROP CURRENCY;
```

3. Hinzufügen der CHK_SALARY-Prüfbedingung und eines Fremdschlüssels zur JOB-Tabelle.

```
ALTER TABLE JOB
ADD CONSTRAINT CHK_SALARY CHECK (MIN_SALARY < MAX_SALARY),
ADD FOREIGN KEY (JOB_COUNTRY) REFERENCES COUNTRY (COUNTRY);
```

4. Festlegen des Standardwerts für das Feld MODEL, Ändern des Typs der ITEMID-Spalte und umbenennen der Spalte MODELNAME.

```
ALTER TABLE STOCK
ALTER COLUMN MODEL SET DEFAULT 1,
ALTER COLUMN ITEMID TYPE BIGINT,
ALTER COLUMN MODELNAME TO NAME;
```

5. Ändern der berechneten Spalten NEW_SALARY und SALARY_CHANGE.

```
ALTER TABLE SALARY_HISTORY
ALTER NEW_SALARY GENERATED ALWAYS AS
  (OLD_SALARY + OLD_SALARY * PERCENT_CHANGE / 100),
ALTER SALARY_CHANGE COMPUTED BY
  (OLD_SALARY * PERCENT_CHANGE / 100);
```

Siehe auch: [CREATE TABLE](#), [DROP TABLE](#), [CREATE DOMAIN](#)

DROP TABLE

Benutzt für: Löschen einer Tabelle

Verfügbar in: DSQL, ESQL

Syntax:

```
drop table tablename;
```

Tabelle 5.9. DROP TABLE Statement-Parameter

| Parameter | Beschreibung |
|-----------|----------------------------|
| tablename | Name (Kennung) der Tabelle |

Die Anweisung DROP TABLE löscht eine vorhandene Tabelle. Wenn die Tabelle Abhängigkeiten hat, schlägt die Anweisung DROP TABLE mit einem Ausführungsfehler fehl.

Wenn eine Tabelle gelöscht wird, werden auch alle Trigger für ihre Ereignisse und Indizes, die für ihre Felder erstellt wurden, gelöscht.

Nur der Tabelleneigentümer und [Administratoren](#) haben die Berechtigung, DROP TABLE zu verwenden.

Beispiel: Die Tabelle COUNTRY löschen.

```
DROP TABLE COUNTRY;
```

Siehe auch: [CREATE TABLE](#), [ALTER TABLE](#), [RECREATE TABLE](#)

RECREATE TABLE

Benutzt für: Erstellen einer neuen Tabelle (Relation) oder Wiederherstellen einer bestehenden

Verfügbar in: DSQL

Syntax:

```
RECREATE [GLOBAL TEMPORARY] TABLE tablename
  [EXTERNAL [FILE] '<filespec>']
  (<col_def> [, {<col_def> | <tconstraint>} ...])
  [ON COMMIT {DELETE | PRESERVE} ROWS]
```

Vergleichen Sie den [CREATE TABLE- Abschnitt](#) für die vollständige Syntax für CREATE TABLE und beachten Sie die Beschreibungen zum Definieren von Tabellen, Spalten und Constraints.

RECREATE TABLE erstellt eine Tabelle neu oder erneut. Wenn bereits eine Tabelle mit diesem Namen vorhanden ist, versucht die Anweisung RECREATE TABLE, diese zu löschen und eine neue zu erstellen. Bestehende Abhängigkeiten verhindern die Ausführung der Anweisung.

Beispiel: Erstellen oder Wiederherstellen der Tabelle COUNTRY.

```
RECREATE TABLE COUNTRY (
  COUNTRY COUNTRYNAME NOT NULL PRIMARY KEY,
  CURRENCY VARCHAR(10) NOT NULL);
```

Siehe auch: [CREATE TABLE](#), [DROP TABLE](#)

INDEX

Ein Index ist ein Datenbankobjekt, das für eine schnellere Datenabfrage aus einer Tabelle oder zur Beschleunigung der Sortierung der Abfrage verwendet wird. Indizes werden auch verwendet, um die referenziellen Integritätsbedingungen PRIMARY KEY, FOREIGN KEY und UNIQUE sicherzustellen.

In diesem Abschnitt wird beschrieben, wie Sie Indizes erstellen, aktivieren und deaktivieren, löschen und Statistiken sammeln (Selektivität neu berechnen).

CREATE INDEX

Benutzt für: Einen Index für eine Tabelle erstellen

Verfügbar in: DSQL, ESQL

Syntax:

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]
INDEX indexname ON tablename
{(col [, col ...]) | COMPUTED BY (<expression>)};
```

Tabelle 5.10. CREATE INDEX Statement-Parameter

| Parameter | Beschreibung |
|------------|---|
| indexname | Indexname. Es kann aus bis zu 31 Zeichen bestehen |
| tablename | Der Name der Tabelle, für die der Index erstellt werden soll |
| col | Name einer Spalte in der Tabelle. Spalten der Typen BLOB sowie ARRAY und berechnete Felder können nicht in einem Index verwendet werden |
| expression | Der Ausdruck, der die Werte für einen berechneten Index berechnet, auch bekannt als „Ausdrucksindex“ |

Die Anweisung CREATE INDEX erstellt einen Index für eine Tabelle, mit der das Suchen, Sortieren und Gruppieren beschleunigt werden kann. Indizes werden automatisch beim Definieren von Constraints wie Primärschlüssel, Fremdschlüssel oder eindeutigen Constraints erstellt.

Ein Index kann auf den Inhalt von Spalten eines beliebigen Datentyps mit Ausnahme von BLOB und Arrays aufgebaut werden. Der Name (Bezeichner) eines Index muss unter allen Indexnamen eindeutig sein.

Schlüssel-Indizes

Wenn ein Primärschlüssel, ein Fremdschlüssel oder eine eindeutige Einschränkung zu einer Tabelle oder Spalte hinzugefügt wird, wird automatisch ein Index mit demselben Namen erstellt, ohne explizite Anweisung vom Designer. Beispielsweise wird der PK_COUNTRY-Index automatisch erstellt, wenn Sie die folgende Anweisung ausführen und übergeben:

```
ALTER TABLE COUNTRY ADD CONSTRAINT PK_COUNTRY  
PRIMARY KEY (ID);
```

Unique-Indizes

Wenn Sie in der Indexerstellungsanweisung das Schlüsselwort UNIQUE angeben, wird ein Index erstellt, in dem die Eindeutigkeit in der gesamten Tabelle durchgesetzt wird. Der Index wird als „eindeutiger Index“ bezeichnet. Ein eindeutiger Index ist keine Einschränkung.

Eindeutige Indizes dürfen keine doppelten Schlüsselwerte (oder Duplikatschlüsselwertkombinationen im Falle von *berechnet* Indizes oder multi-column oder multi-segment) enthalten. Duplizierte NULLs sind gemäß dem SQL: 99-Standard sowohl in Indizes mit einem einzelnen Segment als auch mit mehreren Segmenten zulässig.

Index-Sortierung

Alle Indizes in Firebird sind unidirektional. Ein Index kann vom niedrigsten Wert zum höchsten (aufsteigend) oder vom höchsten zum niedrigsten (absteigend) erstellt werden. Die Schlüsselwörter ASC [ENDING] und DESC [ENDING] werden verwendet, um die Richtung des Index anzugeben. Die Standardindexreihenfolge ist ASC [ENDING]. Es ist durchaus sinnvoll, sowohl einen auf- als auch einen absteigenden Index für dieselbe Spalte oder denselben Schlüsselsatz zu definieren.

Tipp

Ein absteigender Index kann für eine Spalte nützlich sein, die Suchanfragen auf den hohen Werten unterzogen wird („neuestes“, Maximum usw.)

Berechnete (Ausdrucks-) Indizes

Beim Erstellen eines Index können Sie die COMPUTED BY -Klausel verwenden, um anstelle einer oder mehrerer Spalten einen Ausdruck anzugeben. Berechnete Indizes werden in Abfragen verwendet, bei denen die Bedingung in einer WHERE, ORDER BY oder GROUP BY-Klausel exakt dem Ausdruck in der Indexdefinition entspricht. Der Ausdruck in einem berechneten Index kann mehrere Spalten in der Tabelle enthalten.

Anmerkung

Sie können tatsächlich einen berechneten Index für ein berechnetes Feld erstellen, der Index wird jedoch nie verwendet.

Indexgrenzen

Bestimmte Beschränkungen gelten für Indizes.

Die maximale Länge eines Schlüssels in einem Index ist auf $\frac{1}{4}$ der Seitengröße.

Maximale Anzahl Indizes pro Tabelle

Die Anzahl der Indizes, die für jede Tabelle untergebracht werden können, ist begrenzt. Das tatsächliche Maximum für eine bestimmte Tabelle hängt von der Seitengröße und der Anzahl der Spalten in den Indizes ab.

Tabelle 5.11. Maximale Anzahl Indizes pro Tabelle

| Seitengröße | Anzahl der Indizes in Abhängigkeit von der Spaltenanzahl | | |
|-------------|--|-------------|-------------|
| | einspaltig | zweispaltig | dreispaltig |
| 4096 | 203 | 145 | 113 |
| 8192 | 408 | 291 | 227 |
| 16384 | 818 | 584 | 454 |

Zeichenindexgrenzen

Die maximale Länge der indizierten Zeichenfolgen beträgt 9 Byte weniger als die maximale Schlüssellänge. Die maximale indexierbare Zeichenfolgenlänge hängt von der Seitengröße und dem Zeichensatz ab.

Tabelle 5.12. Maximale indizierbare (VAR)CHAR-Länge

| Seitengröße | Maximale indexierbare Zeichenfolgenlänge nach Zeichensatz | | | |
|-------------|---|-----------------|-----------------|-----------------|
| | 1 Byte/Zeichen | 2 Bytes/Zeichen | 3 Bytes/Zeichen | 4 Bytes/Zeichen |
| 4096 | 1015 | 507 | 338 | 253 |
| 8192 | 2039 | 1019 | 679 | 509 |
| 16384 | 4087 | 2043 | 1362 | 1021 |

Nur der Tabelleneigentümer und [Administratoren](#) besitzen die notwendigen Rechte für die Verwendung von CREATE INDEX.

Beispiel für die Verwendung von CREATE INDEX

1. Erstellen eines Index für die UPDATER_ID in der Tabelle SALARY_HISTORY

```
CREATE INDEX IDX_UPDATER
  ON SALARY_HISTORY (UPDATER_ID);
```

- Erstellen eines Index mit Schlüsseln in absteigender Reihenfolge für die CHANGE_DATE-Spalte in der Tabelle SALARY_HISTORY.

```
CREATE DESCENDING INDEX IDX_CHANGE
  ON SALARY_HISTORY (CHANGE_DATE);
```

- Erstellen eines Multisegment-Index für die Spalten ORDER_STATUS sowie PAID in der Tabelle SALES

```
CREATE INDEX IDX_SALESTAT
  ON SALES (ORDER_STATUS, PAID);
```

- Erstellen eines Index, der keine doppelten Werte für die Spalte NAME in der Tabelle COUNTRY zulässt

```
CREATE UNIQUE INDEX UNQ_COUNTRY_NAME
  ON COUNTRY (NAME);
```

- Erstellen eines berechneten Index für die PERSONS-Tabelle

```
CREATE INDEX IDX_NAME_UPPER ON PERSONS
  COMPUTED BY (UPPER (NAME));
```

Ein solcher Index kann für eine Groß- / Kleinschreibungs-sensitive Suche verwendet werden:

```
SELECT *
  FROM PERSONS
  WHERE UPPER(NAME) STARTING WITH UPPER('Iv');
```

Siehe auch: [ALTER INDEX](#), [DROP INDEX](#)

ALTER INDEX

Benutzt für: Aktivieren oder Deaktivieren eines Indexes; einen Index neu aufbauen

Verfügbar in: DSQL, ESQL

Syntax:

```
ALTER INDEX indexname {ACTIVE | INACTIVE};
```

Tabelle 5.13. ALTER INDEX Statement-Parameter

| Parameter | Beschreibung |
|-----------|--------------|
| indexname | Indexname |

Die ALTER INDEX-Anweisung aktiviert oder deaktiviert einen Index. In dieser Anweisung gibt es keine Möglichkeit, Attribute des Index zu ändern.

- Mit der Option INACTIVE wird der Index vom aktiven in den inaktiven Zustand geschaltet. Der Effekt ähnelt der Anweisung DROP INDEX, mit der Ausnahme, dass die Indexdefinition in der Datenbank verbleibt. Das Ändern eines Beschränkungsindex in den inaktiven Zustand ist nicht zulässig.

Ein aktiver Index kann deaktiviert werden, wenn keine Abfragen mit diesem Index vorhanden sind. Andernfalls wird ein „Objekt in Verwendung“-Fehler zurückgegeben.

Die Aktivierung eines inaktiven Index ist ebenfalls sicher. Wenn jedoch aktive Transaktionen die Tabelle ändern, schlägt die Transaktion mit der Anweisung ALTER INDEX fehl, wenn das Attribut NOWAIT vorhanden ist. Wenn sich die Transaktion im Modus WAIT befindet, wartet sie auf den Abschluss der gleichzeitigen Transaktionen.

Auf der anderen Seite der Münze werden andere Transaktionen, die diese Tabelle modifizieren den Index nach einem COMMIT neu erstellen oder fehlschlagen je nach Status der WAIT / NO WAIT-Attribute. Die Situation ist genau dieselbe für CREATE INDEX.

Wie sinnvoll ist dies?

Es kann sinnvoll sein, einen Index in den inaktiven Zustand zu wechseln, während Sie einen großen Stapel von Datensätzen in der Tabelle, in der sich der Index befindet, einfügen, aktualisieren oder löschen.

- Wenn sich der Index im Status INAKTIV befindet, wird es mit der Option ACTIVE in den aktiven Status umgeschaltet, und das System erstellt den Index neu.

Wie sinnvoll ist dies?

Auch wenn der Index *active* ist, wenn ALTER INDEX ... ACTIVE ausgeführt wird, wird der Index neu erstellt. Die Wiederherstellung von Indizes kann eine nützliche Haushaltshilfe sein, gelegentlich auch für die Indizes einer großen Tabelle in einer Datenbank, die häufige Neuaufnahmen Aktualisierungen oder Löschungen aufweist, aber selten wiederhergestellt wird.

Verwendung von ALTER INDEX in einem Constraint-Index

Das Ändern des Erzwingungsindex für eine PRIMARY KEY, FOREIGN KEY- oder UNIQUE-Einschränkung auf INACTIVE ist nicht zulässig. der oder UNIQUE ist nicht zulässig. Jedoch funktioniert ALTER INDEX ... ACTIVE genauso gut wie andere als Indexwiederherstellungstool.

Nur der Tabelleneigentümer und [Administratoren](#) haben die Berechtigungen für die Anweisung ALTER INDEX.

Beispiele für ALTER INDEX:

1. Deaktivieren des IDX_UPDATER-Index

```
ALTER INDEX IDX_UPDATER INACTIVE;
```

2. Den IDX_UPDATER-Index in den aktiven Zustand zurückschalten und neu erstellen

```
ALTER INDEX IDX_UPDATER ACTIVE;
```

Siehe auch: [CREATE INDEX](#), [DROP INDEX](#), [SET STATISTICS](#)

DROP INDEX

Benutzt für: Deleting an index

Verfügbar in: DSQL, ESQL

Syntax:

```
DROP INDEX indexname;
```

Tabelle 5.14. DROP INDEX Statement-Parameter

| Parameter | Beschreibung |
|-----------|--------------|
| indexname | Indexname |

Das Statement DROP INDEX löscht den angegebenen Index aus der Datenbank.

Anmerkung

Ein Constraint-Index kann nicht mittels DROP INDEX gelöscht werden. Constraint-Indizes werden während des Ausführens des Befehls ALTER TABLE ... DROP CONSTRAINT ... gelöscht.

Nur die Tabelleneigentümer und [Administratoren](#) besitzen die Berechtigungen die Anweisung DROP INDEX auszuführen.

DROP INDEX Example: Löschen des Index IDX_UPDATER

```
DROP INDEX IDX_UPDATER;
```

Siehe auch: [CREATE INDEX](#), [ALTER INDEX](#)

SET STATISTICS

Benutzt für: Neuberechnung der Selektivität eines Index

Verfügbar in: DSQL, ESQL

Syntax:

```
SET STATISTICS INDEX indexname
```

Tabelle 5.15. SET STATISTICS Statement-Parameter

| Parameter | Beschreibung |
|-----------|--------------|
| indexname | Indexname |

Die Anweisung SET STATISTICS berechnet die Selektivität des angegebenen Index neu.

Index-Selektivität

Die Selektivität eines Index ergibt sich aus der Auswertung der Anzahl der Zeilen, die bei einer Suche für jeden Indexwert ausgewählt werden können. Ein eindeutiger Index hat die maximale Selektivität, da es nicht möglich ist, mehr als eine Zeile für jeden Wert eines Indexschlüssels auszuwählen, wenn dieser verwendet wird. Die Selektivität eines Index auf dem neuesten Stand zu halten ist wichtig für die Wahl des Optimierers bei der Suche nach dem optimalen Abfrageplan.

Indexstatistiken in Firebird werden nicht automatisch als Reaktion auf große Stapel von Neuaufnahmen, Aktualisierungen oder Löschungen neu berechnet. Es kann vorteilhaft sein, die Selektivität eines Index nach solchen Operationen neu zu berechnen, da die Selektivität dazu neigt, zu veralten.

Anmerkung

Die Anweisungen CREATE INDEX und ALTER INDEX ACTIVE speichern beide Indexstatistiken, die vollständig dem Inhalt des (neu) erstellten Index entsprechen.

Die Selektivität eines Index kann vom Besitzer der Tabelle oder einem [Administrator](#) neu berechnet werden. Es kann unter gleichzeitiger Belastung ohne Korruptionsrisiko durchgeführt werden. Beachten Sie jedoch, dass die neu berechnete Statistik bei gleichzeitiger Auslastung veraltet sein kann, sobald SET STATISTICS beendet ist.

Beispiele für die Verwendung von SET STATISTICS: Neuberechnung der Selektivität des Index IDX_UPDATER

```
SET STATISTICS INDEX IDX_UPDATER;
```

Siehe auch: [CREATE INDEX](#), [ALTER INDEX](#)

VIEW

Eine Sicht (VIEW) ist eine virtuelle Tabelle, bei der es sich um eine gespeicherte und benannte SELECT-Abfrage zum Abrufen von Daten beliebiger Komplexität handelt. Daten können aus einer oder mehreren Tabellen, aus anderen Ansichten und aus ausgewählten gespeicherten Prozeduren abgerufen werden.

Im Gegensatz zu regulären Tabellen in relationalen Datenbanken ist eine Sicht kein unabhängiger Datensatz, der in der Datenbank gespeichert ist. Das Ergebnis wird dynamisch als Datensatz erstellt, wenn die Ansicht ausgewählt wird.

Die Metadaten einer Sicht sind für den Prozess verfügbar, der den Binärcode für gespeicherte Prozeduren und Trigger generiert, so als wären es konkrete Tabellen, in denen persistente Daten gespeichert werden.

CREATE VIEW

Benutzt für: Erstellen einer View

Verfügbar in: DSQL

Syntax:

```
CREATE VIEW viewname [<full_column_list>]
AS <select_statement>
[WITH CHECK OPTION];

<full_column_list> ::= (colname [, colname ...])
```

Tabelle 5.16. CREATE VIEW Statement-Parameter

| Parameter | Beschreibung |
|------------------|--|
| viewname | View-Name, maximal 31 Zeichen |
| select_statement | SELECT-Statement |
| full_column_list | Die Liste der Spalten in der View |
| colname | Spaltenname der View. Doppelte Spaltennamen sind nicht zulässig. |

Die Anweisung CREATE VIEW erstellt eine neue View. Der Bezeichner (Name) einer View muss unter den Namen aller Ansichten, Tabellen und gespeicherten Prozeduren in der Datenbank eindeutig sein.

Dem Namen der neuen Sicht kann die Liste der Spaltennamen folgen, die beim Aufrufen der View an den Aufrufer zurückgegeben werden sollen. Namen in der Liste müssen nicht mit den Namen der Spalten in den Basistabellen verknüpft sein, von denen sie abgeleitet werden.

Wenn die Sichtspaltenliste ausgelassen wird, verwendet das System die Spaltennamen und / oder Aliase aus der Anweisung SELECT. Wenn doppelte Namen oder nicht von Alias-Ausdrücken abgeleitete Spalten das Erstellen einer gültigen Liste unmöglich machen, schlägt die Erstellung der View mit einem Fehler fehl.

Die Anzahl der Spalten in der Liste der Ansichten muss genau der Anzahl der Spalten in der Auswahlliste der zugrunde liegenden SELECT-Anweisung in der Sichtdefinition entsprechen.

Zusätzliche Punkte

- Wenn die vollständige Liste der Spalten angegeben ist, ist es nicht sinnvoll, Aliase in der Anweisung SELECT anzugeben, da die Namen in der Spaltenliste diese überschreiben
- Die Spaltenliste ist optional, wenn alle Spalten in SELECT explizit benannt sind und in der Auswahlliste eindeutig sind

Aktualisierbare Views

Eine View kann aktualisierbar oder schreibgeschützt sein. Wenn eine Sicht aktualisierbar ist, können die beim Aufruf dieser Sicht abgerufenen Daten durch die DML-Anweisungen INSERT, UPDATE, DELETE, UPDATE OR INSERT oder MERGE. Änderungen, die in einer aktualisierbaren View vorgenommen wurden, werden auf die zugrunde liegenden Tabelle (n) angewendet.

Eine Nur-Lese-Ansicht kann mit Hilfe von Triggern aktualisiert werden. Nachdem Trigger in einer Sicht definiert wurden, werden Änderungen, die an sie gesendet wurden, niemals automatisch in die zugrunde liegende Tabelle geschrieben, selbst wenn die Ansicht von vornherein aktualisiert werden konnte. Es liegt in der Verantwortung des Programmierers, sicherzustellen, dass die Trigger die Basistabellen nach Bedarf aktualisieren (oder löschen oder einfügen).

Eine View wird automatisch aktualisiert, wenn alle der folgenden Bedingungen erfüllt sind:

- Die SELECT-Anweisung fragt nur eine Tabelle oder eine aktualisierbare Ansicht ab
- Die Anweisung SELECT ruft keine gespeicherten Prozeduren auf
- jede Spalte der Basistabelle (oder Basissicht), die in der Sichtdefinition nicht vorhanden ist, wird durch eine der folgenden Bedingungen abgedeckt:
 - sie ist nullbar
 - sie hat einen nicht-NULL Standardwert
 - sie hat einen Trigger, der einen zulässigen Wert liefert
- die Anweisung SELECT enthält keine Felder, die von Unterabfragen oder anderen Ausdrücken abgeleitet wurden
- die SELECT-Anweisung enthält keine durch Aggregatfunktionen definierten Felder wie MIN, MAX, AVG, SUM, COUNT, LIST
- Die SELECT-Anweisung enthält keine der Klauseln ORDER BY oder GROUP BY
- Das SELECT-Statement enthält weder das Schlüsselwort DISTINCT noch zeilenbeschränkende Schlüsselwörter wie ROWS, FIRST, SKIP

WITH CHECK OPTION

Die optionale Klausel WITH CHECK OPTION erfordert eine aktualisierbare Sicht, um zu überprüfen, ob neue oder aktualisierte Daten die in der WHERE-Klausel der SELECT. Jeder Versuch, einen neuen Datensatz einzufügen oder einen bestehenden zu aktualisieren, wird überprüft, ob der neue oder aktualisierte Datensatz den

Kriterien aus WHERE entspricht. Wenn die Überprüfung fehlschlägt, wird die Operation nicht ausgeführt und eine entsprechende Fehlermeldung wird zurückgegeben.

WITH CHECK OPTION kann nur in einer CREATE VIEW-Anweisung angegeben werden, in der eine WHERE-Klausel vorhanden ist, um die Ausgabe der SELECT-Anweisung einzuschränken. Eine Fehlermeldung wird ansonsten zurückgegeben.

Bitte beachten:

Wenn WITH CHECK OPTION verwendet wird, prüft das Modul die Eingabe für die WHERE-Klausel, bevor etwas an die Basisrelation übergeben wird. Wenn die Überprüfung der Eingabe fehlschlägt, werden daher keine Standardklauseln oder Trigger der Basisrelation, welche möglicherweise zur Korrektur der Eingaben erstellt wurden, ausgeführt.

Außerdem werden Sichtfelder, die in der Anweisung INSERT ausgelassen wurden, unabhängig von ihrer Anwesenheit oder Abwesenheit in der WHERE-Klausel als NULLs an die Basisrelation übergeben. Infolgedessen werden Basistabellenstandards, die für solche Felder definiert sind, nicht angewendet. Auslöser dagegen werden wie erwartet feuern und arbeiten.

Für Views, die WITH CHECK OPTION nicht enthalten, werden Felder, die in der Anweisung INSERT fehlen, überhaupt nicht an die Basisrelation übergeben.

Eigentümer einer View

Der Ersteller einer Sicht wird zu seinem Besitzer.

Um eine Sicht zu erstellen, benötigt ein Benutzer ohne Administratorrechte mindestens SELECT-Zugriff auf die zugrunde liegenden Tabelle(n) und / oder Ansichten und die Berechtigung EXECUTE auf abfragbare gespeicherte Prozeduren.

Um Einfügungen, Aktualisierungen und Löschungen durch die Sicht zu ermöglichen, muss der Ersteller / Eigentümer auch die entsprechenden Berechtigungen INSERT, UPDATE und DELETE auf die zugrunde liegenden Objekt(e) besitzen.

Das Freigeben anderer Benutzerprivilegien für die Sicht ist nur möglich, wenn der Sichtbesitzer selbst diese Berechtigungen für die zugrunde liegenden Objekte WITH GRANT OPTION besitzt. Dies ist immer dann der Fall, wenn der View-Besitzer auch Eigentümer der zugrunde liegenden Objekte ist.

Beispiele zum Erstellen von Views:

1. Creating view returning the JOB_CODE and JOB_TITLE columns only for those jobs where MAX_SALARY is less than \$15,000.

```
CREATE VIEW ENTRY_LEVEL_JOBS AS
SELECT JOB_CODE, JOB_TITLE
FROM JOB
WHERE MAX_SALARY < 15000;
```

2. Erstellen einer Ansicht, die die Spalten JOB_CODE und JOB_TITLE nur für diejenigen Jobs zurückgibt, bei denen MAX_SALARY weniger als 15.000 USD beträgt. Immer wenn ein neuer Datensatz eingefügt oder ein vorhandener Datensatz aktualisiert wird, wird der Wert MAX_SALARY < 15000 Zustand wird überprüft. Wenn die Bedingung nicht wahr ist, wird die Operation zum Einfügen / Aktualisieren zurückgewiesen.

```
CREATE VIEW ENTRY_LEVEL_JOBS AS
SELECT JOB_CODE, JOB_TITLE
FROM JOB
WHERE MAX_SALARY < 15000
WITH CHECK OPTION;
```

3. Erstellen einer Ansicht mit einer expliziten Spaltenliste.

```
CREATE VIEW PRICE_WITH_MARKUP (
    CODE_PRICE,
    COST,
    COST_WITH_MARKUP
) AS
SELECT
    CODE_PRICE,
    COST,
    COST * 1.1
FROM PRICE;
```

4. Erstellen einer View mit Hilfe von Aliasnamen für Felder in der SELECT-Anweisung (das gleiche Ergebnis wie in Beispiel 3).

```
CREATE VIEW PRICE_WITH_MARKUP AS
SELECT
    CODE_PRICE,
    COST,
    COST * 1.1 AS COST_WITH_MARKUP
FROM PRICE;
```

5. Erstellen einer schreibgeschützten Ansicht basierend auf zwei Tabellen und einer gespeicherten Prozedur.

```
CREATE VIEW GOODS_PRICE AS
SELECT
    goods.name AS goodsname,
    price.cost AS cost,
    b.quantity AS quantity
FROM
    goods
    JOIN price ON goods.code_goods = price.code_goods
    LEFT JOIN sp_get_balance(goods.code_goods) b ON 1 = 1;
```

Siehe auch: [ALTER VIEW](#), [CREATE OR ALTER VIEW](#), [RECREATE VIEW](#), [DROP VIEW](#)

ALTER VIEW

Benutzt für: Ändern einer existierenden View

Verfügbar in: DSQL

Syntax:

```
ALTER VIEW viewname [<full_column_list>]
AS <select_statement>
[WITH CHECK OPTION];

<full_column_list> ::= (colname [, colname ...])
```

Tabelle 5.17. ALTER VIEW Statement-Parameter

| Parameter | Beschreibung |
|------------------|---|
| viewname | Name einer existierenden View |
| select_statement | SELECT-Statement |
| full_column_list | Die Liste der Spalten in der View |
| colname | Spaltenname der View Doppelte Spaltennamen sind nicht zulässig. |

Verwenden Sie die Anweisung ALTER VIEW, um die Definition einer vorhandenen View zu ändern. Berechtigungen für Ansichten bleiben erhalten und Abhängigkeiten sind nicht betroffen.

Die Syntax der Anweisung ALTER VIEW entspricht vollständig der von CREATE VIEW.

Achtung

Seien Sie vorsichtig, wenn Sie die Anzahl der Spalten in einer Ansicht ändern. Vorhandener Anwendungscode und PSQL-Module, die auf die Sicht zugreifen, können ungültig werden. Informationen zur Erkennung dieser Art von Problemen in gespeicherten Prozeduren und Triggern finden Sie unter *Das RDB\$VALID_BLR-Feld* im Anhang.

Nur der View-Eigentümer und [Administratoren](#) besitzen die notwendigen Berechtigungen zum Ausführen von ALTER VIEW.

Beispiele zur Verwendung von ALTER VIEW: Ansicht ändern PRICE_WITH_MARKUP

```
ALTER VIEW PRICE_WITH_MARKUP (
  CODE_PRICE,
  COST,
  COST_WITH_MARKUP
) AS
SELECT
  CODE_PRICE,
  COST,
  COST * 1.15
FROM PRICE;
```

Siehe auch: [CREATE VIEW](#), [CREATE OR ALTER VIEW](#), [RECREATE VIEW](#)

CREATE OR ALTER VIEW

Benutzt für: Creating a new view or altering an existing view.

Verfügbar in: DSQL

Syntax:

```
CREATE OR ALTER VIEW viewname [<full_column_list>]
AS <select_statement>
[WITH CHECK OPTION];

<full_column_list> ::= (colname [, colname ...])
```

Tabelle 5.18. CREATE OR ALTER VIEW Statement-Parameter

| Parameter | Beschreibung |
|------------------|---|
| viewname | Name einer View, die vorhanden oder nicht vorhanden ist |
| select_statement | SELECT-Statement |
| full_column_list | Die Liste der Spalten in der View |
| colname | Spaltenname der View. Duplikate sind nicht zulässig. |

Verwenden Sie die Anweisung CREATE OR ALTER VIEW, um die Definition einer vorhandenen Ansicht zu ändern oder sie zu erstellen, falls sie nicht existiert. Berechtigungen für eine vorhandene Ansicht bleiben erhalten und Abhängigkeiten werden nicht beeinflusst.

Die Syntax der Anweisung CREATE OR ALTER VIEW entspricht vollständig der von CREATE VIEW.

Beispiel: Erstellen Sie die Ansicht PRICE_WITH_MARKUP der neuen Ansicht oder ändern Sie sie, falls diese bereits vorhanden ist:

```
CREATE OR ALTER VIEW PRICE_WITH_MARKUP (
  CODE_PRICE,
  COST,
  COST_WITH_MARKUP
) AS
SELECT
  CODE_PRICE,
  COST,
  COST * 1.15
FROM PRICE;
```

Siehe auch: [CREATE VIEW](#), [ALTER VIEW](#), [RECREATE VIEW](#)

DROP VIEW

Benutzt für: Löschen (dropping) einer View

Verfügbar in: DSQL

Syntax:

```
DROP VIEW viewname;
```

Tabelle 5.19. DROP VIEW Statement-Parameter

| Parameter | Beschreibung |
|-----------|--------------|
| viewname | View-Name |

Die Anweisung DROP VIEW löscht eine existierende View. Die Anweisung wird fehlschlagen wenn die View Abhängigkeiten besitzt.

Nur der Eigentümer der View und [Administratoren](#) besitzen die notwendigen Berechtigungen zum Ausführen von DROP VIEW.

Beispiel: Löschen der View PRICE_WITH_MARKUP.

```
DROP VIEW PRICE_WITH_MARKUP;
```

Siehe auch: [CREATE VIEW](#), [RECREATE VIEW](#), [CREATE OR ALTER VIEW](#)

RECREATE VIEW

Benutzt für: Erstellen einer neuen Ansicht oder Wiederherstellen einer vorhandenen View

Verfügbar in: DSQL

Syntax:

```
RECREATE VIEW viewname [<full_column_list>]
AS <select_statement>
[WITH CHECK OPTION];

<full_column_list> ::= (colname [, colname ...])
```

Tabelle 5.20. RECREATE VIEW Statement-Parameter

| Parameter | Beschreibung |
|-----------|-------------------------------|
| viewname | View-Name, maximal 31 Zeichen |

| Parameter | Beschreibung |
|------------------|--|
| select_statement | SELECT-Statement |
| full_column_list | Die Liste der Spalten in der View |
| colname | Spaltenname der View. Duplikate sind nicht zulässig. |

Erstellt oder erstellt eine Ansicht neu. Wenn bereits eine Ansicht mit diesem Namen vorhanden ist, versucht die Engine, sie vor dem Erstellen der neuen Instanz zu löschen. Wenn die vorhandene Sicht aufgrund von Abhängigkeiten oder unzureichenden Rechten nicht gelöscht werden kann, schlägt RECREATE VIEW mit einem Fehler fehl.

Beispiel: Die neue View PRICE_WITH_MARKUP erstellen oder neu erstellen, falls diese bereits vorhanden ist.

```
RECREATE VIEW PRICE_WITH_MARKUP (
  CODE_PRICE,
  COST,
  COST_WITH_MARKUP
) AS
SELECT
  CODE_PRICE,
  COST,
  COST * 1.15
FROM PRICE;
```

Siehe auch: [CREATE VIEW](#), [DROP VIEW](#), [CREATE OR ALTER VIEW](#)

TRIGGER

Ein Trigger ist ein spezieller Typ einer gespeicherten Prozedur, der nicht direkt aufgerufen wird, sondern ausgeführt wird, wenn ein bestimmtes Ereignis in der zugeordneten Tabelle oder Sicht (View) auftritt. Ein Trigger ist spezifisch für eine und nur eine Relation (Tabelle oder View) und eine Phase im Timing des Ereignisses (*BEFORE* oder *AFTER*). Es kann angegeben werden, dass dieser für ein bestimmtes Ereignis (Einfügen, Aktualisieren, Löschen) oder für eine Kombination von zwei oder drei dieser Ereignisse ausgeführt wird.

Eine andere Form eines Triggers - bekannt als ein „Datenbanktrigger“ - kann spezifiziert werden, um in Verbindung mit dem Start oder dem Ende einer Benutzersitzung (Verbindung) oder einer Benutzertransaktion zu ausgelöst zu werden.

CREATE TRIGGER

Benutzt für: Erstellen eines neuen Triggers

Verfügbar in: DSQL, ESQL

Syntax:

```

CREATE TRIGGER trigname {
  <relation_trigger_legacy> |
  <relation_trigger_sql2003> |
  <database_trigger> }
AS
  [<declarations>]
BEGIN
  [<PSQL_statements>]
END

<relation_trigger_legacy> ::=
  FOR {tablename | viewname}
  [ACTIVE | INACTIVE]
  {BEFORE | AFTER} <mutation_list>
  [POSITION number]

<relation_trigger_sql2003> ::=
  [ACTIVE | INACTIVE]
  {BEFORE | AFTER} <mutation_list>
  [POSITION number]
  ON {tablename | viewname}

<database_trigger> ::=
  [ACTIVE | INACTIVE] ON db_event [POSITION number]

<mutation_list> ::=
  <mutation> [OR <mutation> [OR <mutation>]]

<mutation> ::= { INSERT | UPDATE | DELETE }

<db_event> ::= {
  CONNECT |
  DISCONNECT |
  TRANSACTION START |
  TRANSACTION COMMIT |
  TRANSACTION ROLLBACK
}

<declarations> ::= {<declare_var> | <declare_cursor>};
  [{<declare_var> | <declare_cursor>}; ...]

```

Tabelle 5.21. CREATE TRIGGER Statement-Parameter

| Parameter | Beschreibung |
|--------------------------|---|
| trigname | Triggernamen bestehend aus bis zu 31 Zeichen. Dieser muss unter allen Triggernamen in der Datenbank eindeutig sein. |
| relation_trigger_legacy | Legacy-Stil der Trigger-Deklaration für einen Relation-Trigger |
| relation_trigger_sql2003 | Relation-Trigger-Deklaration gemäß dem SQL: 2003-Standard |
| database_trigger | Datenbank-Triggerdeklaration |
| tablename | Name der Tabelle, der der Relationstrigger zugeordnet ist |
| viewname | Name der Sicht, der der Relationstrigger zugeordnet ist |

| Parameter | Beschreibung |
|-----------------|---|
| mutation_list | Liste von Relationsereignissen (Tabelle Ansicht) |
| number | Position des Triggers in der Zündreihenfolge. Von 0 bis 32.767 |
| db_event | Verbindungs- oder Transaktionsereignis |
| declarations | Abschnitt zum Deklarieren von lokalen Variablen und benannten Cursors |
| declare_var | Lokale Variablendeklarieren |
| declare_cursor | Benannte Cursor-Deklaration |
| PSQL_statements | Anweisungen in der Programmiersprache von Firebird (PSQL) |

Die Anweisung CREATE TRIGGER dient zum Erstellen eines neuen Triggers. Ein Trigger kann entweder für ein Ereignis *Relation (Table | View)* (oder eine Kombination von Ereignissen) oder für ein *Datenbankereignis* erstellt werden.

CREATE TRIGGER, zusammen mit den zugehörigen Assoziaten ALTER TRIGGER, CREATE ODER ALTER TRIGGER und RECREATE TRIGGER, ist eine *zusammengesetzte Anweisung*, bestehend aus einem Header und einem Rumpf. Der Header spezifiziert den Namen des Triggers, den Namen der Relation (für einen Relations-Trigger), die Phase des Triggers und das Ereignis, für das er gilt. Der Körper besteht aus optionalen Deklarationen von lokalen Variablen und benannten Cursors, gefolgt von einer oder mehreren Anweisungen oder Anweisungsblöcken, die alle in einem äußeren Block eingeschlossen sind, der mit dem Schlüsselwort BEGIN beginnt und mit dem Schlüsselwort END endet. Deklarationen und eingebettete Anweisungen werden mit Semikolons (;) abgeschlossen.

Der Name des Triggers muss unter allen Triggernamen eindeutig sein.

Statement-Terminatoren

Einige SQL-Anweisungseditoren - insbesondere das mit Firebird mitgelieferte Dienstprogramm *isql* und möglicherweise einige Editoren von Drittanbietern - verwenden eine interne Konvention, die erfordert, dass alle Anweisungen mit einem Semikolon abgeschlossen werden. Dies führt bei der Codierung in diesen Umgebungen zu einem Konflikt mit der PSQL-Syntax. Wenn Sie mit diesem Problem und seiner Lösung nicht vertraut sind, lesen Sie bitte die Details im Kapitel PSQL im Abschnitt [Umschalten des Terminators in isql](#).

Relations-Trigger (auf Tabellen oder Views)

Relation-Trigger werden jedes Mal auf der Zeilen- (Datensatz-) Ebene ausgeführt, wenn sich das Zeilenbild ändert. Ein Trigger kann entweder ACTIVE oder INACTIVE sein. Nur aktive Trigger werden ausgeführt. Trigger werden standardmäßig mit ACTIVE erstellt.

Formen der Deklaration

Firebird unterstützt zwei Arten der Deklaration für Relationstrigger:

- Die ursprüngliche Legacy-Syntax

- Das standardmäßige SQL: 2003-Formular (empfohlen)

Das standardmäßige SQL: 2003-Formular ist das empfohlene Format.

Ein Relationstrigger spezifiziert unter anderem < eine *Phase* und ein oder mehrere *Ereignisse*.

Phase

Die Phase betrifft das Timing des Triggers in Bezug auf das Zustandswechselereignis in der Datenzeile:

- Ein BEFORE-Trigger wird ausgelöst, bevor die angegebene Datenbankoperation (Einfügen, Aktualisieren oder Löschen) ausgeführt wird.
- Ein AFTER-Trigger wird ausgeführt, nachdem die Datenbankoperation abgeschlossen wurde.

Multi-Aktions-Trigger

Eine Relationstriggerdefinition gibt mindestens eine der DML-Operationen INSERT, UPDATE und DELETE an, um ein oder mehrere Ereignisse anzuzeigen, auf die der Trigger ausgelöst werden soll. Wenn mehrere Operationen angegeben werden, müssen sie durch das Schlüsselwort OR getrennt werden. Keine Operation darf mehrmals auftreten.

Innerhalb des Anweisungsblocks werden die Booleschen Kontextvariablen **INSERTING**, **UPDATING** und **DELETING** verwendet, um die Art der derzeit ausgeführten Operation zu prüfen.

Ausführungsreihenfolge von Triggern

Das Schlüsselwort POSITION erlaubt es, eine optionale Ausführungsreihenfolge („firing order“) für eine Reihe von Triggern anzugeben, die dieselbe Phase und dasselbe Ereignis wie ihr Ziel haben. Die Standardposition ist 0. Wenn keine Positionen angegeben werden oder wenn mehrere Trigger eine einzelne Positionsnummer haben, werden die Trigger in der alphabetischen Reihenfolge ihrer Namen ausgeführt.

Variablendeklarationen

Der optionale Deklarationsabschnitt unter dem Schlüsselwort AS im Header des Triggers dient zum Definieren von Variablen und benannten Cursoren, die lokal zum Trigger gehören. Weitere Informationen finden Sie unter **DECLARE VARIABLE** und **DECLARE CURSOR** im Kapitel **Prozedurales SQL**.

Der Trigger-Body

Die lokalen Deklarationen (falls vorhanden) sind der letzte Teil des Headerabschnitts eines Triggers. Der Trigger-Body folgt, wobei ein oder mehrere Blöcke von PSQL-Anweisungen in einer Struktur eingeschlossen sind, die mit dem Schlüsselwort BEGIN beginnt und mit dem Schlüsselwort END endet.

Nur der Eigentümer der Sicht oder Tabelle und **Administratoren** haben die Berechtigung, CREATE TRIGGER zu verwenden.

Beispiele für CREATE TRIGGER für Tabellen und Views:

1. Erstellung eines Triggers in „Legacy“-Form. Wird vor dem Einfügen eines neuen Datensatzes in die Tabelle CUSTOMER ausgelöst.

```
CREATE TRIGGER SET_CUST_NO FOR CUSTOMER
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.CUST_NO IS NULL) THEN
    NEW.CUST_NO = GEN_ID(CUST_NO_GEN, 1);
  END IF;
END
```

2. Erstellen eines Triggers in SQL:2003-konformer Variante, der vor dem Einfügen eines neuen Datensatzes in die Tabelle CUSTOMER ausgelöst wird.

```
CREATE TRIGGER set_cust_no
ACTIVE BEFORE INSERT POSITION 0 ON customer
AS
BEGIN
  IF (NEW.cust_no IS NULL) THEN
    NEW.cust_no = GEN_ID(cust_no_gen, 1);
  END IF;
END
```

3. Einen Trigger erstellen, der nach dem Einfügen, Aktualisieren oder Löschen eines Datensatzes in der Tabelle CUSTOMER ausgeführt wird.

```
CREATE TRIGGER TR_CUST_LOG
ACTIVE AFTER INSERT OR UPDATE OR DELETE POSITION 10
ON CUSTOMER
AS
BEGIN
  INSERT INTO CHANGE_LOG (LOG_ID,
                        ID_TABLE,
                        TABLE_NAME,
                        MUTATION)
  VALUES (NEXT VALUE FOR SEQ_CHANGE_LOG,
          OLD.CUST_NO,
          'CUSTOMER',
          CASE
            WHEN INSERTING THEN 'INSERT'
            WHEN UPDATING  THEN 'UPDATE'
            WHEN DELETING  THEN 'DELETE'
          END);
END
```

Datenbank-Trigger

Trigger können definiert werden, um auf „Datenbankereignisse“ zu reagieren. Dies können verschiedene Ereignisse sein. Diese können auf den Umfang einer Sitzung (Verbindung) hinweg agieren, aber auch über die Umgebung einer Transaktion wirken:

- CONNECT
- DISCONNECT

- TRANSACTION START
- TRANSACTION COMMIT
- TRANSACTION ROLLBACK

Ausführung von Datenbanktriggern und Fehlerbehandlung

CONNECT- und DISCONNECT-Trigger werden in einer speziell für diesen Zweck erstellten Transaktion ausgeführt. Läuft alles glatt, wird die Transaktion committed. Nicht abgefangene Ausnahmen bewirken, dass die Transaktion zurückgesetzt wird, und

- für einen CONNECT-Trigger wird die Verbindung unterbrochen und die Ausnahme wird an den Client zurückgegeben
- für einen DISCONNECT-Trigger werden Ausnahmen nicht gemeldet. Die Verbindung ist wie beabsichtigt unterbrochen

TRANSACTION-Trigger werden innerhalb der Transaktion ausgeführt, deren Start, Commit oder Rollback sie hervorruft. Die Aktion, die nach einer nicht abgefangenen Ausnahme ausgeführt wird, hängt vom Ereignis ab:

- In einem TRANSACTION START-Trigger wird die Ausnahme an den Client gemeldet und die Transaktion wird zurückgesetzt
- In einem TRANSACTION COMMIT-Trigger wird die Ausnahme gemeldet, die bisherigen Aktionen des Triggers werden rückgängig gemacht und das Commit abgebrochen
- In einem TRANSACTION ROLLBACK-Trigger wird die Ausnahme nicht gemeldet und die Transaktion wird wie beabsichtigt zurückgesetzt.

Fallen

Offensichtlich gibt es keine direkte Möglichkeit zu wissen, ob ein DISCONNECT- oder TRANSACTION ROLLBACK-Trigger eine Ausnahme verursacht hat. Daraus folgt auch, dass die Verbindung zur Datenbank nicht zustande kommen kann, wenn ein CONNECT-Trigger eine Ausnahme verursacht und eine Transaktion nicht gestartet werden kann, wenn auch ein TRANSACTION START-Trigger einen auslöst. Beide Phänomene sperrt Sie effektiv aus Ihrer Datenbank aus, bis Sie mit unterdrückten Datenbanktriggern zurückkehren und den fehlerhaften Code beheben.

Triggerunterdrückung

Einige Firebird-Befehlszeilentools wurden mit Switches ausgestattet, mit denen ein Administrator die automatische Auslösung von Datenbanktriggern unterdrücken kann. Bisher sind sie:

```
gbak -nodbtriggers
isql -nodbtriggers
nbackup -T
```

Zwei-Phasen Commit

In einem zweiphasigen Commit-Szenario löst ein TRANSACTION COMMIT-Trigger bereits in der Vorbereitungsphase aus und nicht erst beim Commit.

Einige Vorbehalte

1. Die Verwendung der Anweisung `IN AUTONOMOUS TRANSACTION DO` in den Datenbanktriggern für Transaktionen (`TRANSACTION START`, `TRANSACTION ROLLBACK`, `TRANSACTION COMMIT`) kann dazu führen, dass die autonome Transaktion eine Endlosschleife generiert
2. Die Ereignistrigger `DISCONNECT` und `TRANSACTION ROLLBACK` werden nicht ausgeführt, wenn Clients über Überwachungstabellen getrennt werden (`DELETE FROM MON$ATTACHMENTS`)

Nur der Datenbankbesitzer und [Administratoren](#) haben die Berechtigung zum Erstellen von Datenbanktriggern.

Beispiele für `CREATE TRIGGER` für „Database Triggers“:

1. Einen Trigger für das Ereignis erstellen, bei dem eine Verbindung zur Datenbank hergestellt wird, in der Benutzer protokolliert werden, die sich am System anmelden. Der Trigger wird als inaktiv erstellt.

```
CREATE TRIGGER tr_log_connect
INACTIVE ON CONNECT POSITION 0
AS
BEGIN
    INSERT INTO LOG_CONNECT (ID,
                            USERNAME,
                            ATIME)
    VALUES (NEXT VALUE FOR SEQ_LOG_CONNECT,
            CURRENT_USER,
            CURRENT_TIMESTAMP);
END
```

2. Einen Trigger für das Ereignis der Verbindung mit der Datenbank erstellen, das es keinem Benutzer, außer `SYSDBA`, erlaubt, sich außerhalb der Geschäftszeiten anzumelden.

```
CREATE EXCEPTION E_INCORRECT_WORKTIME 'The working day has not started yet.';

CREATE TRIGGER TR_LIMIT_WORKTIME ACTIVE
ON CONNECT POSITION 1
AS
BEGIN
    IF ((CURRENT_USER <> 'SYSDBA') AND
        NOT (CURRENT_TIME BETWEEN time '9:00' AND time '17:00')) THEN
        EXCEPTION E_INCORRECT_WORKTIME;
END
```

Siehe auch: [ALTER TRIGGER](#), [CREATE OR ALTER TRIGGER](#), [RECREATE TRIGGER](#), [DROP TRIGGER](#)

ALTER TRIGGER

Benutzt für: Ändern und Deaktivieren eines vorhandenen Triggers

Verfügbar in: `DSQL`, `ESQL`

Syntax:

```
ALTER TRIGGER trigname
[ACTIVE | INACTIVE]
[ {BEFORE | AFTER} <mutation_list> | ON db_event]
[POSITION number]
[
  AS
  [<declarations>]
  BEGIN
    [<PSQL_statements>]
  END
]

<mutation_list> ::=
  <mutation> [OR <mutation> [OR <mutation>]]

<mutation> ::= { INSERT | UPDATE | DELETE }

<db_event> ::= {
  CONNECT |
  DISCONNECT |
  TRANSACTION START |
  TRANSACTION COMMIT |
  TRANSACTION ROLLBACK
}

<declarations> ::= {<declare_var> | <declare_cursor>};
[ {<declare_var> | <declare_cursor>}; ...]
```

Tabelle 5.22. ALTER TRIGGER Statement-Parameter

| Parameter | Beschreibung |
|-----------------|--|
| trigname | Name eines vorhandenen Triggers |
| mutation_list | Liste von Relation-Ereignissen (Tabelle Ansicht) |
| number | Position des Triggers in der Zündreihenfolge. Von 0 bis 32.767 |
| declarations | Abschnitt zum Deklarieren von lokalen Variablen und benannter Cursor |
| declare_var | Lokale Variablendeklaration |
| declare_cursor | Benannte Cursor-Deklaration |
| PSQL_statements | Anweisungen in der Programmiersprache von Firebird (PSQL) |

Die Anweisung ALTER TRIGGER erlaubt bestimmte Änderungen am Header und am Rumpf eines Triggers.

Zulässige Änderungen an Triggern

- Status (ACTIVE | INACTIVE)

- Phase (BEFORE | AFTER)
- Veranstaltungen; aber Relationstriggerereignisse können nicht in Datenbanktriggerereignisse geändert werden, und umgekehrt
- Position innerhalb der Ausführungsreihenfolge
- Änderungen am Code im Trigger-Body

Wenn ein Element nicht angegeben wurde, bleibt es unverändert.

Zur Erinnerungen

Das BEFORE-Schlüsselwort weist darauf hin, dass der Trigger ausgeführt wird, bevor das zugehörige Ereignis eintritt. Das Schlüsselwort AFTER weist darauf hin, dass es nach dem Ereignis ausgeführt wird.

Mehr als ein Beziehungsereignis - INSERT, UPDATE, DELETE - kann mit einem einzigen Trigger abgedeckt werden. Die Ereignisse sollten mit dem Schlüsselwort OR getrennt werden. Kein Ereignis sollte mehr als einmal erwähnt werden.

Das Schlüsselwort POSITION erlaubt es, eine optionale Ausführungsreihenfolge („firing order“) für eine Reihe von Triggern anzugeben, die dieselbe Phase und dasselbe Ereignis wie ihr Ziel haben. Die Standardposition ist 0. Wenn keine Positionen angegeben werden oder wenn mehrere Trigger eine einzelne Positionsnummer haben, werden die Trigger in der alphabetischen Reihenfolge ihrer Namen ausgeführt.

Administratoren und folgende Benutzer haben die Berechtigung für die Ausführung von ALTER TRIGGER:

- Für Relations-Trigger der Besitzer des Tisches
- Für Datenbank-Trigger der Eigentümer der Datenbank

Beispiele mit ALTER TRIGGER:

1. Deaktivieren des Triggers set_cust_no (Umschalten in den inaktiven Status)

```
ALTER TRIGGER set_cust_no INACTIVE;
```

2. Ändern der Ausführungsreihenfolge des Triggers set_cust_no.

```
ALTER TRIGGER set_cust_no POSITION 14;
```

3. Den Trigger TR_CUST_LOG in den inaktiven Status schalten und die Liste der Ereignisse ändern.

```
ALTER TRIGGER TR_CUST_LOG  
INACTIVE AFTER INSERT OR UPDATE;
```

4. Den tr_log_connect-Trigger in den aktiven Status schalten und seine Position und seinen Körper ändern.

```
ALTER TRIGGER tr_log_connect  
ACTIVE POSITION 1
```

```

AS
BEGIN
  INSERT INTO LOG_CONNECT ( ID,
                           USERNAME,
                           ROLENAME,
                           ATIME )
  VALUES (NEXT VALUE FOR SEQ_LOG_CONNECT,
          CURRENT_USER,
          CURRENT_ROLE,
          CURRENT_TIMESTAMP);
END

```

Siehe auch: [CREATE TRIGGER](#), [CREATE OR ALTER TRIGGER](#), [RECREATE TRIGGER](#), [DROP TRIGGER](#)

CREATE OR ALTER TRIGGER

Benutzt für: Erstellen eines neuen Triggers oder Ändern eines vorhandenen Triggers

Verfügbar in: DSQL

Syntax:

```

CREATE OR ALTER TRIGGER trigname {
  <relation_trigger_legacy> |
  <relation_trigger_sql2003> |
  <database_trigger> }
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END

```

Für das vollständige Detail der Syntax siehe [CREATE TRIGGER](#).

Die Anweisung `CREATE OR ALTER TRIGGER` erstellt einen neuen Trigger, falls er nicht existiert. Andernfalls ändert und kompiliert er es erneut, wobei die Privilegien intakt und die Abhängigkeiten unberührt bleiben.

Beispiel mit CREATE OR ALTER TRIGGER: Erstellen eines neuen Triggers, falls er nicht existiert oder anpassen, falls vorhanden.

```

CREATE OR ALTER TRIGGER set_cust_no
ACTIVE BEFORE INSERT POSITION 0 ON customer
AS
BEGIN
  IF (NEW.cust_no IS NULL) THEN
    NEW.cust_no = GEN_ID(cust_no_gen, 1);
END

```

Siehe auch: [CREATE TRIGGER](#), [ALTER TRIGGER](#), [RECREATE TRIGGER](#)

DROP TRIGGER

Benutzt für: Einen vorhandenen Trigger löschen

Verfügbar in: DSQL, ESQL

Syntax:

```
DROP TRIGGER trigname
```

Tabelle 5.23. DROP TRIGGER Statement-Parameter

| Parameter | Beschreibung |
|-----------|--------------|
| trigname | Triggenname |

Die Anweisung DROP TRIGGER löscht einen vorhandenen Trigger.

[Administrators](#) und die folgenden Benutzer besitzen die Berechtigung, die Anweisung DROP TRIGGER auszuführen:

- Für Relations-Trigger, der Eigentümer der Tabelle
- Für Datenbank-Trigger, der Eigentümer der Datenbank

Beispiele für DROP TRIGGER: Löschen des Triggers set_cust_no.

```
DROP TRIGGER set_cust_no;
```

Siehe auch: [CREATE TRIGGER](#), [RECREATE TRIGGER](#)

RECREATE TRIGGER

Benutzt für: Erstellen eines neuen Triggers oder Neuerstellung eines vorhandenen

Verfügbar in: DSQL

Syntax:

```
RECREATE TRIGGER trigname {
  <relation_trigger_legacy> |
  <relation_trigger_sql2003> |
  <database_trigger> }
```

```
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END
```

Für die detaillierte Syntax, vergleichen Sie [CREATE TRIGGER](#).

Die Anweisung `RECREATE TRIGGER` erzeugt einen neuen Trigger, wenn kein Trigger mit dem angegebenen Namen existiert. Andernfalls versucht die Anweisung `RECREATE TRIGGER`, den vorhandenen Trigger zu löschen und einen neuen zu erstellen. Die Operation schlägt beim `COMMIT` fehl, wenn Triggerabhängigkeiten vorliegen.

Warnung

Beachten Sie, dass Abhängigkeitsfehler erst in der Phase `COMMIT` dieser Operation erkannt werden.

Beispiele für RECREATE TRIGGER: Erstellen oder Neuerstellung des Triggers `set_cust_no`.

```
RECREATE TRIGGER set_cust_no
ACTIVE BEFORE INSERT POSITION 0 ON customer
AS
BEGIN
  IF (NEW.cust_no IS NULL) THEN
    NEW.cust_no = GEN_ID(cust_no_gen, 1);
  END
```

Siehe auch: [CREATE TRIGGER](#), [DROP TRIGGER](#), [CREATE OR ALTER TRIGGER](#)

PROCEDURE

Eine gespeicherte Prozedur (Stored Procedure) ist ein Softwaremodul, das von einem Client, einer anderen Prozedur, einem ausführbaren Block oder einem Trigger aufgerufen werden kann. Gespeicherte Prozeduren, ausführbare Blöcke und Trigger werden in prozeduralem SQL (PSQL) geschrieben. Die meisten SQL-Anweisungen sind auch in PSQL verfügbar, manchmal mit Einschränkungen oder Erweiterungen. Zu den bemerkenswerten Ausnahmen zählen DDL- und Transaktionskontrollanweisungen.

Gespeicherte Prozeduren können viele Eingabe- und Ausgabeparameter haben.

CREATE PROCEDURE

Benutzt für: Erstellen einer neuen gespeicherten Prozedur

Verfügbar in: DSQL, ESQL

Syntax:

```

CREATE PROCEDURE procname
[(

```

Tabelle 5.24. CREATE PROCEDURE Statement-Parameter

| Parameter | Beschreibung |
|----------------|---|
| procname | Der Name der gespeicherten Prozedur besteht aus bis zu 31 Zeichen. Muss für alle Tabellen-, View- und Prozedurnamen in der Datenbank eindeutig sein |
| inparam | Beschreibung der Eingabeparameter |
| outparam | Beschreibung der Ausgangsparameter |
| declarations | Abschnitt zum Deklarieren von lokalen Variablen und benannten Cursors |
| declare_var | Lokale Variablendeklaration |
| declare_cursor | Benannte Cursor-Deklaration |

| Parameter | Beschreibung |
|-----------------|--|
| PSQL_statements | Prozedurale SQL-Anweisungen |
| literal | Ein Literalwert, der mit dem Datentyp des Parameters zuweisungskompatibel ist |
| context_var | Jede Kontextvariable, deren Typ mit dem Datentyp des Parameters kompatibel ist |
| paramname | Der Name eines Eingabe- oder Ausgabeparameters der Prozedur. Dieser kann aus bis zu 31 Zeichen bestehen. Der Name des Parameters muss unter den Eingabe- und Ausgabeparametern der Prozedur und ihren lokalen Variablen eindeutig sein |
| datatype | SQL-Datentyp |
| collation | Sortierfolge |
| domain | Domain-Name |
| rel | Tabellen- oder View-Name |
| col | Spaltenname einer Tabelle oder View |
| precision | Die Gesamtanzahl der signifikanten Stellen, die der Parameter halten kann (1..18) |
| scale | Die Anzahl der Stellen nach dem Dezimalpunkt (0.. <i>precision</i>) |
| size | Die maximale Größe eines Zeichenfolgentypparameters oder einer Variablen in Zeichen |
| charset | Zeichensatz eines String-Typ-Parameters oder einer Variablen |
| subtype_num | Subtyp-Nummer eines BLOB |
| subtype_name | Mnemonsischer Name eines BLOB-Subtyps |
| seglen | Segmentgröße (max. 65535) |

Die Anweisung CREATE PROCEDURE erstellt eine neue gespeicherte Prozedur. Der Name der Prozedur muss unter den Namen aller gespeicherten Prozeduren, Tabellen und Ansichten in der Datenbank eindeutig sein.

CREATE PROCEDURE ist eine *zusammengesetzte Anweisung*, bestehend aus einem Header und einem Body. Der Header gibt den Namen der Prozedur an und deklariert Eingabeparameter und ggf. die Ausgabeparameter, die von der Prozedur zurückgegeben werden sollen.

Der Prozedurhauptteil besteht aus Deklarationen für alle lokalen Variablen und benannten Cursors, die von der Prozedur verwendet werden, gefolgt von einer oder mehreren Anweisungen oder Anweisungsblöcken, die alle in einem äußeren Block eingeschlossen sind, der mit dem Schlüsselwort BEGIN beginnt und endet mit dem Schlüsselwort END endet. Deklarationen und eingebettete Anweisungen werden mit Semikolon (;) abgeschlossen.

Statement-Terminatoren

Einige SQL-Anweisungseditoren — insbesondere das mit Firebird mitgelieferte Dienstprogramm *isql* und möglicherweise einige Editoren von Drittanbietern — verwenden eine interne Konvention, die erfordert, dass alle

Anweisungen mit einem Semikolon abgeschlossen werden. Dies führt bei der Codierung in diesen Umgebungen zu einem Konflikt mit der PSQL-Syntax. Wenn Sie mit diesem Problem und seiner Lösung nicht vertraut sind, lesen Sie bitte die Details im Kapitel PSQL im Abschnitt [Umschalten des Terminators in *isql*](#).

Parameter

Jeder Parameter hat einen Datentyp, der dafür angegeben ist. Die Einschränkung NOT NULL kann auch für jeden beliebigen Parameter angegeben werden, um zu verhindern, dass NULL übergeben oder zugewiesen wird.

Mit der Klausel COLLATE kann eine Sortierfolge für Parameter vom Typ String festgelegt werden.

Eingabeparameter:

Eingabeparameter werden nach dem Namen der Prozedur in Klammern angezeigt. Sie werden als Werte an die Prozedur übergeben, d.h. alles, was sie innerhalb der Prozedur ändert, hat keine Auswirkungen auf die Parameter im aufrufenden Programm.

Eingabeparameter können Standardwerte haben. Diejenigen, für die Werte angegeben sind, müssen sich am Ende der Parameterliste befinden.

Ausgabeparameter:

Die optionale Klausel RETURNS dient zur Angabe einer eingeklammerten Liste von Ausgabeparametern für die gespeicherte Prozedur.

Verwendung von Domains in Deklarationen

Ein Domainname kann als Typ eines Parameters angegeben werden. Der Parameter erbt alle Domainattribute. Wenn ein Standardwert für den Parameter angegeben wird, überschreibt dieser den in der Domänendefinition angegebenen Standardwert.

Wenn die Klausel TYPE OF vor dem Domänennamen hinzugefügt wird, wird nur der Datentyp der Domain verwendet: Alle anderen Attribute der Domain — NOT NULL-Einschränkung, CHECK-Bedingung, Standardwert — werden weder geprüft noch verwendet. Wenn die Domain jedoch aus einem Texttyp besteht, werden immer ihre Zeichensatz und die Sortierreihenfolge verwendet.

Verwendung des Spaltentyps in Deklarationen

Eingabe- und Ausgabeparameter können auch über den Datentyp von Spalten in vorhandenen Tabellen und Ansichten deklariert werden. Die Klausel TYPE OF COLUMN wird dafür verwendet, wobei *relationname.columnname* als Argument angegeben wird.

Wenn TYPE OF COLUMN verwendet wird, erbt der Parameter nur den Datentyp, bei Zeichentypen den Zeichensatz und die Sortierreihenfolge. Die Constraints und der Standardwert der Spalte werden ignoriert.

Bugwarnung für Versionen vor Firebird 3:

Für Eingabeparameter wird die Sortierung, die mit dem Typ der Spalte geliefert wird, in Vergleichen ignoriert (z. B. Gleichheitstests). Bei lokalen Variablen variiert das Verhalten.

Der Bug wurde für Firebird 3 behoben.

Variablen- und Cursor-Deklarationen

Der optionale Deklarationsabschnitt, der zuletzt im Headerabschnitt der Prozedurdefinition aufgeführt ist, definiert lokale Variablen für die Prozedur und ihre benannten Cursor. Lokale Variablendeklarationen folgen denselben Regeln wie Parameter bezüglich der Spezifikation des Datentyps. Bitte entnehmen Sie Details den Abschnitten [PSQL chapter for DECLARE VARIABLE](#) und [DECLARE CURSOR](#).

Auf den Headerabschnitt folgt der Prozedurhauptteil, der aus einer oder mehreren PSQL-Anweisungen besteht, die zwischen den äußeren Schlüsselwörtern BEGIN und END eingeschlossen sind. Mehrere BEGIN ... END-Blöcke von beendeten Anweisungen können in den Prozedurtext eingebettet werden.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann eine neue gespeicherte Prozedur erstellen. Der Benutzer, der eine gespeicherte Prozedur erstellt, wird zu seinem Besitzer.

Beispiele: Erstellen einer gespeicherten Prozedur, die einen Datensatz in die BREED-Tabelle einfügt und den Code des eingefügten Datensatzes zurückgibt:

```
CREATE PROCEDURE ADD_BREED (
  NAME_D_BREEDNAME, /* Domain attributes are inherited */
  NAME_EN TYPE OF D_BREEDNAME, /* Only the domain type is inherited */
  SHORTNAME TYPE OF COLUMN BREED.SHORTNAME,
  /* The table column type is inherited */
  REMARK VARCHAR(120) CHARACTER SET WIN1251 COLLATE PXW_CYRL,
  CODE_ANIMAL INT NOT NULL DEFAULT 1
)
RETURNS (
  CODE_BREED INT
)
AS
BEGIN
  INSERT INTO BREED (
    CODE_ANIMAL, NAME, NAME_EN, SHORTNAME, REMARK)
  VALUES (
    :CODE_ANIMAL, :NAME, :NAME_EN, :SHORTNAME, :REMARK)
  RETURNING CODE_BREED INTO CODE_BREED;
END
```

Erstellen einer wählbaren gespeicherten Prozedur, die Daten für Adressetiketten generiert (aus employee.fdb):

```
CREATE PROCEDURE mail_label (cust_no INTEGER)
RETURNS (line1 CHAR(40), line2 CHAR(40), line3 CHAR(40),
  line4 CHAR(40), line5 CHAR(40), line6 CHAR(40))
AS
  DECLARE VARIABLE customer VARCHAR(25);
  DECLARE VARIABLE first_name VARCHAR(15);
  DECLARE VARIABLE last_name VARCHAR(20);
  DECLARE VARIABLE addr1 VARCHAR(30);
  DECLARE VARIABLE addr2 VARCHAR(30);
  DECLARE VARIABLE city VARCHAR(25);
  DECLARE VARIABLE state VARCHAR(15);
  DECLARE VARIABLE country VARCHAR(15);
  DECLARE VARIABLE postcode VARCHAR(12);
```

```

DECLARE VARIABLE cnt INTEGER;
BEGIN
line1 = '';
line2 = '';
line3 = '';
line4 = '';
line5 = '';
line6 = '';

SELECT customer, contact_first, contact_last, address_line1,
address_line2, city, state_province, country, postal_code
FROM CUSTOMER
WHERE cust_no = :cust_no
INTO :customer, :first_name, :last_name, :addr1, :addr2,
:city, :state, :country, :postcode;

IF (customer IS NOT NULL) THEN
line1 = customer;
IF (first_name IS NOT NULL) THEN
line2 = first_name || ' ' || last_name;
ELSE
line2 = last_name;
IF (addr1 IS NOT NULL) THEN
line3 = addr1;
IF (addr2 IS NOT NULL) THEN
line4 = addr2;

IF (country = 'USA') THEN
BEGIN
IF (city IS NOT NULL) THEN
line5 = city || ', ' || state || ' ' || postcode;
ELSE
line5 = state || ' ' || postcode;
END
ELSE
BEGIN
IF (city IS NOT NULL) THEN
line5 = city || ', ' || state;
ELSE
line5 = state;
line6 = country || ' ' || postcode;
END

SUSPEND; -- the statement that sends an output row to the buffer
-- and makes the procedure "selectable"
END

```

Siehe auch: [CREATE OR ALTER PROCEDURE](#), [ALTER PROCEDURE](#), [RECREATE PROCEDURE](#), [DROP PROCEDURE](#)

ALTER PROCEDURE

Benutzt für: Ändern einer vorhandenen gespeicherten Prozedur

Verfügbar in: DSQL, ESQL

Syntax:

```

ALTER PROCEDURE procname
[(

```

Tabelle 5.25. ALTER PROCEDURE Statement-Parameter

| Parameter | Beschreibung |
|-----------------|---|
| procname | Der Name einer existierenden gespeicherten Prozedur |
| inparam | Beschreibung der Eingabeparameter |
| outparam | Beschreibung der Ausgangsparameter |
| declarations | Abschnitt zum Deklarieren von lokalen Variablen und benannten Cursors |
| declare_var | Lokale Variablendeklaration |
| declare_cursor | Benannte Cursor-Deklaration |
| PSQL_statements | Prozedurale SQL-Anweisungen |

| Parameter | Beschreibung |
|--------------|--|
| literal | Ein Literalwert, der mit dem Datentyp des Parameters zuweisungskompatibel ist |
| context_var | Jede Kontextvariable, deren Typ mit dem Datentyp des Parameters kompatibel ist |
| paramname | Der Name eines Eingabe- oder Ausgabeparameters der Prozedur. Dieser kann aus bis zu 31 Zeichen bestehen. Der Name des Parameters muss unter den Eingabe- und Ausgabeparametern der Prozedur und ihren lokalen Variablen eindeutig sein |
| datatype | SQL-Datentyp |
| collation | Sortierfolge |
| domain | Domain-Name |
| rel | Tabellen- oder View-Name |
| col | Spaltenname einer Tabelle oder View |
| precision | Die Gesamtanzahl der signifikanten Stellen, die der Parameter halten kann (1..18) |
| scale | Die Anzahl der Stellen nach dem Dezimalpunkt (<i>0..precision</i>) |
| size | Die maximale Größe eines Zeichenfolgentypparameters oder einer Variablen in Zeichen |
| charset | Zeichensatz eines String-Typ-Parameters oder einer Variablen |
| subtype_num | Subtyp-Nummer eines BLOB |
| subtype_name | Mnemonic Name eines BLOB-Subtyps |
| seglen | Segmentgröße (max. 65535) |

Die Anweisung ALTER PROCEDURE erlaubt folgende Änderungen der Definition für gespeicherte Prozeduren:

- Satz und die Eigenschaften von Eingangs- und Ausgangsparametern
- lokale Variablen
- Code im Hauptteil der gespeicherten Prozedur

Nach der Ausführung von ALTER PROCEDURE bleiben vorhandene Berechtigungen erhalten und Abhängigkeiten werden nicht beeinflusst.

Achtung

Achten Sie darauf, die Anzahl und Art der Eingabe- und Ausgabeparameter in gespeicherten Prozeduren zu ändern. Bestehender Anwendungscode und Prozeduren und Trigger, die ihn aufrufen, könnten ungültig werden, weil die neue Beschreibung der Parameter mit dem alten Aufrufformat nicht kompatibel ist. Informationen zum Beheben einer solchen Situation finden Sie im Artikel [Das Feld RDB\\$VALID_BLR](#) im Anhang.

Der Prozedureigentümer und [Administratoren](#) besitzen die Recht zum Ausführen von ALTER PROCEDURE.

Beispiele für ALTER PROCEDURE: Ändern der gespeicherten Prozedur GET_EMP_PROJ

```
ALTER PROCEDURE GET_EMP_PROJ (
    EMP_NO SMALLINT)
RETURNS (
    PROJ_ID VARCHAR(20))
AS
BEGIN
    FOR SELECT
        PROJ_ID
    FROM
        EMPLOYEE_PROJECT
    WHERE
        EMP_NO = :emp_no
    INTO :proj_id
    DO
        SUSPEND;
END
```

Siehe auch: [CREATE PROCEDURE](#), [CREATE OR ALTER PROCEDURE](#), [RECREATE PROCEDURE](#), [DROP PROCEDURE](#)

CREATE OR ALTER PROCEDURE

Benutzt für: Erstellen einer neuen gespeicherten Prozedur oder Ändern einer vorhandenen

Verfügbar in: DSQL

Syntax:

```
CREATE OR ALTER PROCEDURE procname
[(<inparam> [, <inparam> ...])]
[RETURNS (<outparam> [, <outparam> ...])]
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END
```

Das vollständige Syntaxdetail finden Sie unter [CREATE DATABASE](#) .

Die Anweisung CREATE OR ALTER PROCEDURE erstellt eine neue gespeicherte Prozedur oder ändert eine vorhandene Prozedur. Wenn die gespeicherte Prozedur nicht vorhanden ist, wird sie durch das transparente Aufrufen einer Anweisung CREATE PROCEDURE erstellt. Wenn die Prozedur bereits vorhanden ist, wird sie geändert und kompiliert, ohne die vorhandenen Berechtigungen und Abhängigkeiten zu beeinträchtigen.

Beispiel: Creating or altering the GET_EMP_PROJ procedure.

```
CREATE OR ALTER PROCEDURE GET_EMP_PROJ (
    EMP_NO SMALLINT)
RETURNS (
    PROJ_ID VARCHAR(20))
AS
BEGIN
    FOR SELECT
```

```

        PROJ_ID
    FROM
        EMPLOYEE_PROJECT
    WHERE
        EMP_NO = :emp_no
    INTO :proj_id
DO
    SUSPEND;
END

```

Siehe auch: [CREATE PROCEDURE](#), [ALTER PROCEDURE](#), [RECREATE PROCEDURE](#)

DROP PROCEDURE

Benutzt für: Löschen einer gespeicherten Prozedur

Verfügbar in: DSQL, ESQL

Syntax:

```
DROP PROCEDURE procname
```

Tabelle 5.26. DROP PROCEDURE Statement-Parameter

| Parameter | Beschreibung |
|-----------|---|
| procname | Name einer vorhandenen gespeicherten Prozedur |

Die Anweisung DROP PROCEDURE löscht eine vorhandene gespeicherte Prozedur. Wenn die gespeicherte Prozedur Abhängigkeiten aufweist, schlägt der Löschversuch fehl und der entsprechende Fehler wird ausgelöst.

Der Prozedureigentümer und [Administratoren](#) haben die Berechtigung, DROP PROCEDURE zu verwenden.

Beispiel: Löschen der gespeicherten Prozedur GET_EMP_PROJ

```
DROP PROCEDURE GET_EMP_PROJ;
```

Siehe auch: [CREATE PROCEDURE](#), [RECREATE PROCEDURE](#)

RECREATE PROCEDURE

Benutzt für: Eine neue gespeicherte Prozedur erstellen oder eine vorhandene wiederherstellen

Verfügbar in: DSQL

Syntax:

```
RECREATE PROCEDURE procname
```

```
[(<inparam> [, <inparam> ...]]]
[RETURNS (<outparam> [, <outparam> ...])]
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END
```

Das vollständige Syntaxdetail finden Sie unter [CREATE PROCEDURE](#) .

Die Anweisung RECREATE PROCEDURE erstellt eine neue gespeicherte Prozedur oder erstellt eine vorhandene Prozedur neu. Wenn es bereits eine Prozedur mit diesem Namen gibt, versucht die Engine diese zu löschen und eine neue zu erstellen. Das Wiederherstellen einer vorhandenen Prozedur schlägt bei der Anforderung COMMIT fehl, wenn die Prozedur Abhängigkeiten aufweist.

Warnung

Beachten Sie, dass Abhängigkeitsfehler erst in der Phase COMMIT dieser Operation erkannt werden.

Nachdem eine Prozedur erfolgreich neu erstellt wurde, werden Berechtigungen zum Ausführen der gespeicherten Prozedur und die Berechtigungen der gespeicherten Prozedur selbst gelöscht.

Beispiel: Erstellen der neuen gespeicherten Prozedur GET_EMP_PROJ oder Wiederherstellen der vorhandenen gespeicherten Prozedur GET_EMP_PROJ.

```
RECREATE PROCEDURE GET_EMP_PROJ (
    EMP_NO SMALLINT)
RETURNS (
    PROJ_ID VARCHAR(20))
AS
BEGIN
    FOR SELECT
        PROJ_ID
    FROM
        EMPLOYEE_PROJECT
    WHERE
        EMP_NO = :emp_no
    INTO :proj_id
DO
    SUSPEND;
END
```

Siehe auch: [CREATE PROCEDURE](#), [DROP PROCEDURE](#), [CREATE OR ALTER PROCEDURE](#)

EXTERNAL FUNCTION

ÜBERPRÜFUNGSSTATUS

Alle Abschnitte von diesem Punkt bis zum Ende des Kapitels warten auf eine technische und redaktionelle Überprüfung.

Externe Funktionen, die auch als „benutzerdefinierte Funktionen“ (UDFs) bezeichnet werden, sind Programme, die in einer externen Programmiersprache geschrieben und in dynamisch geladenen Bibliotheken gespeichert werden. Sobald sie in einer Datenbank deklariert sind, werden sie in dynamischen und prozeduralen Anweisungen verfügbar, als wären sie intern in der SQL-Sprache implementiert.

Externe Funktionen erweitern die Möglichkeiten zur Datenverarbeitung mit SQL erheblich. Um eine Funktion für eine Datenbank verfügbar zu machen, wird sie mit der Anweisung `DECLARE EXTERNAL FUNCTION` deklariert.

Die Bibliothek, die eine Funktion enthält, wird geladen, wenn eine darin enthaltene Funktion aufgerufen wird.

Anmerkung

Externe Funktionen können in mehr als einem Bibliotheks- oder „-Modul“ enthalten sein, wie es in der Syntax erwähnt wird.

DECLARE EXTERNAL FUNCTION

Benutzt für: Deklarieren einer benutzerdefinierten Funktion (UDF) zur Datenbank

Verfügbar in: DSQL, ESQL

Syntax:

```

DECLARE EXTERNAL FUNCTION funcname
[<arg_type_decl> [, <arg_type_decl> ...]]
RETURNS {
  sqltype [BY {DESCRIPTOR | VALUE}] |
  CSTRING(length) |
  PARAMETER param_num }
[FREE_IT]
ENTRY_POINT 'entry_point' MODULE_NAME 'library_name';

<arg_type_decl> ::=
  sqltype [{BY DESCRIPTOR} | NULL] |
  CSTRING(length) [NULL]
    
```

Tabelle 5.27. DECLARE EXTERNAL FUNCTION Statement-Parameter

| Parameter | Beschreibung |
|--------------|---|
| funcname | Funktionsname in der Datenbank. Es kann aus bis zu 31 Zeichen bestehen. Es sollte unter allen internen und externen Funktionsnamen in der Datenbank eindeutig sein und nicht mit dem Namen übereinstimmen, der aus der UDF-Bibliothek über <code>ENTRY_POINT</code> . |
| entry_point | Der exportierte Name der Funktion |
| library_name | Der Name des Moduls (<code>MODULE_NAME</code> , aus dem die Funktion exportiert wird. Dies ist der Name der Datei ohne die „.dll“ oder „.so“-Dateierweiterung. |
| sqltype | SQL-Datentyp. Es kann kein Array oder Array-Element sein |

| Parameter | Beschreibung |
|-----------|---|
| length | Die maximale Länge einer nullterminierten Zeichenfolge, angegeben in Bytes |
| param_num | Die Nummer des Eingabeparameters, von 1 in der Liste der Eingabeparameter in der Deklaration nummeriert, beschreibt den Datentyp, der von der Funktion zurückgegeben wird |

Die Anweisung `DECLARE EXTERNAL FUNCTION` stellt eine benutzerdefinierte Funktion in der Datenbank zur Verfügung. UDF-Deklarationen müssen in *jeder Datenbank* vorgenommen werden, die sie verwenden soll. Es gibt keine Notwendigkeit, UDFs zu deklarieren, die niemals verwendet werden.

Der Name der externen Funktion muss unter allen Funktionsnamen eindeutig sein. Es kann sich vom exportierten Namen der Funktion unterscheiden, wie im `ENTRY_POINT`-Argument angegeben.

DECLARE EXTERNAL FUNCTION-Eingabeparameter

Die Eingabeparameter der Funktion folgen dem Namen der Funktion und sind durch Kommas getrennt. Für jeden Parameter ist ein SQL-Datentyp angegeben. Arrays können nicht als Funktionsparameter verwendet werden. Neben den SQL-Typen steht der `CSTRING`-Typ für die Angabe einer nullterminierten Zeichenfolge mit einer maximalen Länge von `LENGTH` Bytes zur Verfügung.

Standardmäßig werden Eingabeparameter *durch Referenz* übergeben. Die `BY DESCRIPTOR`-Klausel kann stattdessen angegeben werden, wenn der Eingabeparameter durch den Deskriptor übergeben wird. Das Übergeben eines Parameters nach Deskriptor ermöglicht die Verarbeitung von `NULLs`.

Klauseln und Schlüsselwörter

RETURNS-Klausel: (Erforderlich) gibt den von der Funktion zurückgegebenen Ausgabeparameter an. Eine Funktion ist skalar: Sie gibt nur einen Parameter zurück. Der Ausgabeparameter kann von einem beliebigen SQL-Typ (außer einem Array oder einem Array-Element) oder einer nullterminierten Zeichenfolge (`CSTRING`) sein. Der Ausgabeparameter kann durch Referenz (Standard), Deskriptor oder Wert übergeben werden. Wenn die Klausel `BY DESCRIPTOR` angegeben ist, wird der Ausgabeparameter von Deskriptor übergeben. Wenn die `BY VALUE`-Klausel angegeben ist, wird der Ausgabeparameter über den Wert übergeben.

PARAMETER-Schlüsselwort: gibt an, dass die Funktion den Wert aus dem Parameter an Stelle `param_num` zurückgibt. Es ist notwendig, wenn Sie einen Wert des Datentyps `BLOB` zurückgeben müssen.

FREE_IT Schlüsselwort: bedeutet, dass der Speicher, der zum Speichern des Rückgabewerts zugewiesen wurde, freigegeben wird, nachdem die Funktion ausgeführt wurde. Sie wird nur verwendet, wenn der Speicher in der UDF dynamisch zugewiesen wurde. In einer solchen UDF muss der Speicher mit Hilfe der Funktion `ib_util_malloc` aus dem `ib_util`-Modul zugewiesen werden, was die Kompatibilität mit den im Firebird-Code verwendeten Funktionen und Code der ausgelieferten UDF-Module, zum Zuweisen und Freigeben von Speicher.

ENTRY_POINT-Klausel: gibt den Namen des Einstiegspunkts (den Namen der importierten Funktion) an, der aus dem Modul exportiert wird.

MODULE_NAME-Klausel: Definiert den Namen des Moduls, in dem sich die exportierte Funktion befindet. Der Link zum Modul sollte nicht der vollständige Pfad und die Erweiterung der Datei sein, wenn dies vermieden werden kann. Wenn sich das Modul am Standardspeicherort (im Unterverzeichnis `../UDF` des Firebird-Ser-

verstammes) oder an einem in `firebird.conf` explizit konfigurierten Speicherort befindet, ist es einfacher, die Datenbank zwischen verschiedene Plattformen. Der Parameter `UDFAccess` in der Datei `firebird.conf` ermöglicht die Konfiguration von Zugriffsbeschränkungen für externe Funktionsmodule.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann eine externe Funktion (UDF) deklarieren.

Beispiele zur Verwendung von `DECLARE EXTERNAL FUNCTION`:

1. Deklarieren der externen `AddDate`-Funktion im `fbudf`-Modul. Die Ein- und Ausgabeparameter werden als Referenz übergeben.

```
DECLARE EXTERNAL FUNCTION addDay
TIMESTAMP, INT
RETURNS TIMESTAMP
ENTRY_POINT 'addDay' MODULE_NAME 'fbudf';
```

2. Deklaration der externen `Invl`-Funktion im `fbudf`-Modul. Die Ein- und Ausgabeparameter werden vom Deskriptor übergeben.

```
DECLARE EXTERNAL FUNCTION invl
INT BY DESCRIPTOR, INT BY DESCRIPTOR
RETURNS INT BY DESCRIPTOR
ENTRY_POINT 'idNvl' MODULE_NAME 'fbudf';
```

3. Declaring the `isLeapYear` external function located in the `fbudf` module. The input parameter is passed by reference, while the output parameter is passed by value.

```
DECLARE EXTERNAL FUNCTION isLeapYear
TIMESTAMP
RETURNS INT BY VALUE
ENTRY_POINT 'isLeapYear' MODULE_NAME 'fbudf';
```

4. Deklaration der externen Funktion `i64Truncate` im `fbudf`-Modul. Die Ein- und Ausgabeparameter werden vom Deskriptor übergeben. Der zweite Parameter der Funktion wird als Rückgabewert verwendet.

```
DECLARE EXTERNAL FUNCTION i64Truncate
NUMERIC(18) BY DESCRIPTOR, NUMERIC(18) BY DESCRIPTOR
RETURNS PARAMETER 2
ENTRY_POINT 'fbtruncate' MODULE_NAME 'fbudf';
```

Siehe auch: [ALTER EXTERNAL FUNCTION](#), [DROP EXTERNAL FUNCTION](#)

ALTER EXTERNAL FUNCTION

Benutzt für: Ändern des Eintrittspunkts und / oder des Modulnamens für eine benutzerdefinierte Funktion (UDF)

Verfügbar in: DSQL

Syntax:

```
ALTER EXTERNAL FUNCTION funcname
[ENTRY_POINT 'new_entry_point']
[MODULE_NAME 'new_library_name'];
```

Tabelle 5.28. ALTER EXTERNAL FUNCTION Statement-Parameter

| Parameter | Beschreibung |
|------------------|---|
| funcname | Funktionsname in der Datenbank |
| new_entry_point | Der neue exportierte Name der Funktion |
| new_library_name | Der neue Name des Moduls (MODULE_NAME, aus dem die Funktion exportiert wird, ist der Name der Datei ohne die „.dll“- oder „.so“-Dateierweiterung. |

Die Anweisung ALTER EXTERNAL FUNCTION ändert den Eintrittspunkt und / oder den Modulnamen für eine benutzerdefinierte Funktion (UDF). Vorhandene Abhängigkeiten bleiben erhalten, nachdem die Anweisung mit der Änderung [s] ausgeführt wurde.

Die ENTRY_POINT-Klausel: dient zur Angabe des neuen Eintrittspunktes (der Name der Funktion, die aus dem Modul exportiert wird).

Die MODULE_NAME-Klausel: Gibt den neuen Namen des Moduls an, in dem sich die exportierte Funktion befindet.

Jeder an die Datenbank angeschlossene Benutzer kann den Eintrittspunkt und den Modulnamen ändern.

Beispiele für die Verwendung von ALTER EXTERNAL FUNCTION:

1. Ändern des Einstiegspunkts für eine externe Funktion

```
ALTER EXTERNAL FUNCTION invl ENTRY_POINT 'intNvl';
```

2. Ändern des Modulnamens für eine externe Funktion

```
ALTER EXTERNAL FUNCTION invl MODULE_NAME 'fbudf2';
```

Siehe auch: [DECLARE EXTERNAL FUNCTION](#), [DROP EXTERNAL FUNCTION](#)

DROP EXTERNAL FUNCTION

Benutzt für: Entfernen einer benutzerdefinierten Funktion (UDF) aus einer Datenbank

Verfügbar in: DSQL, ESQL

Syntax:

```
DROP EXTERNAL FUNCTION funcname
```

Tabelle 5.29. DROP EXTERNAL FUNCTION Statement-Parameter

| Parameter | Beschreibung |
|-----------|--------------------------------|
| funcname | Funktionsname in der Datenbank |

Die Anweisung `DROP EXTERNAL FUNCTION` löscht die Deklaration einer benutzerdefinierten Funktion aus der Datenbank. Wenn es Abhängigkeiten von der externen Funktion gibt, schlägt die Anweisung fehl und der entsprechende Fehler wird ausgelöst.

Jeder mit der Datenbank verbundene Benutzer kann die Deklaration einer internen Funktion löschen.

Beispiel der Verwendung von `DROP EXTERNAL FUNCTION`: Löschen der Deklaration der `addDay`-Funktion.

```
DROP EXTERNAL FUNCTION addDay;
```

Siehe auch: [DECLARE EXTERNAL FUNCTION](#)

FILTER

Ein `BLOB FILTER`-Filter ist ein Datenbankobjekt, das eigentlich ein spezieller Typ einer externen Funktion ist, mit dem alleinigen Zweck, ein `BLOB`-Objekt in einem Format zu verwenden und es zu konvertieren zu einem `BLOB`-Objekt in einem anderen Format. Die Formate der `BLOB`-Objekte werden mit benutzerdefinierten `BLOB`-Subtypen angegeben.

Externe Funktionen zum Konvertieren von `BLOB`-Typen werden in dynamischen Bibliotheken gespeichert und bei Bedarf geladen.

Weitere Informationen zu `BLOB`-Subtypen finden Sie unter [Binäre Datentypen](#).

DECLARE FILTER

Benutzt für: Deklarieren eines `BLOB`-Filters zur Datenbank

Verfügbar in: DSQL, ESQL

Syntax:

```
DECLARE FILTER filtername
INPUT_TYPE <sub_type> OUTPUT_TYPE <sub_type>
ENTRY_POINT 'function_name' MODULE_NAME 'library_name';

<sub_type> ::= number | <mnemonic>
```

```
<mnemonic> ::= binary | text | blr | acl | ranges
              | summary | format | transaction_description
              | external_file_description | user_defined
```

Tabelle 5.30. DECLARE FILTER Statement-Parameter

| Parameter | Beschreibung |
|---------------|--|
| filtername | Name des Filters in der Datenbank. Es kann aus bis zu 31 Zeichen bestehen. Es muss nicht derselbe Name sein, wie der aus der Filterbibliothek via ENTRY_POINT geholt wird. |
| sub_type | BLOB-Subtype |
| number | BLOB SUB_TYPE-Nummer (muss negativ sein) |
| mnemonic | BLOB SUB_TYPE mnemonischer Name |
| function_name | Der exportierte Name (Einstiegspunkt) der Funktion |
| library_name | Der Name des Moduls, in dem sich der Filter befindet |
| user_defined | Benutzerdefinierter BLOB SUB_TYPE mnemonischer Name |

Mit der Anweisung DECLARE FILTER wird ein BLOB-Filter für die Datenbank verfügbar. Der Name des BLOB-Filters muss unter den Namen von BLOB-Filtern eindeutig sein.

Spezifizieren der Subtypen

Die Subtypen können als Untertypnummer oder als Subtyp-Mnemonikname angegeben werden. Benutzerdefinierte Subtypen müssen durch negative Zahlen (von -1 bis -32.768) dargestellt werden. Ein Versuch, mehr als einen BLOB-Filter mit derselben Kombination der Ein- und Ausgabetypen zu deklarieren, schlägt mit einem Fehler fehl.

INPUT_TYPE: Klausel, die den BLOB-Subtyp des zu konvertierenden Objekts definiert

OUTPUT_TYPE: Klausel definiert den BLOB-Subtyp des zu erstellenden Objekts.

Anmerkung

Mnemonische Namen können für benutzerdefinierte BLOB-Subtypen definiert und manuell in die Systemtabelle RDB\$TYPES eingefügt werden.

```
INSERT INTO RDB$TYPES (RDB$FIELD_NAME, RDB$TYPE, RDB$TYPE_NAME)
VALUES ('RDB$FIELD_SUB_TYPE', -33, 'MIDI');
```

Nachdem die Transaktion bestätigt wurde, können die mnemonischen Namen in Deklarationen verwendet werden, wenn Sie neue Filter erstellen.

Der Wert der Spalte RDB\$FIELD_NAME muss immer 'RDB\$FIELD_SUB_TYPE' sein. Bei mnemonischen Namen in Großbuchstaben können sie bei der Definition eines Filters case-insensitiv und ohne Anführungszeichen verwendet werden.

Warning: Ab Firebird 3 sind die Systemtabellen nicht mehr von Benutzern beschreibbar.

Parameters

ENTRY_POINT: Klausel, die den Namen des Einstiegspunkts (den Namen der importierten Funktion) im Modul definiert.

MODULE_NAME: Die Klausel definiert den Namen des Moduls, in dem sich die exportierte Funktion befindet. Standardmäßig müssen sich die Module im UDF-Ordner des Stammverzeichnisses auf dem Server befinden. Der Parameter *UDFAccess* in `firebird.conf` ermöglicht das Bearbeiten von Zugriffsbeschränkungen für Filterbibliotheken.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann einen BLOB-Filter deklarieren.

Beispiele:

1. Erstellen eines BLOB-Filters mit Subtypnummern.

```
DECLARE FILTER DESC_FILTER
INPUT_TYPE 1
OUTPUT_TYPE -4
ENTRY_POINT 'desc_filter'
MODULE_NAME 'FILTERLIB';
```

2. Erstellen eines BLOB-Filters mit Untertyp-Mnemoniknamen.

```
DECLARE FILTER FUNNEL
INPUT_TYPE blr OUTPUT_TYPE text
ENTRY_POINT 'blr2asc' MODULE_NAME 'myfilterlib';
```

Siehe auch: [DROP FILTER](#)

DROP FILTER

Benutzt für: Entfernen einer BLOB-Filterdeklaration aus der Datenbank

Verfügbar in: DSQL, ESQL

Syntax:

```
DROP FILTER filtername;
```

Tabelle 5.31. DROP FILTER Statement-Parameter

| Parameter | Beschreibung |
|------------|-----------------------------|
| filtername | Filtername in der Datenbank |

Die Anweisung `DROP FILTER` entfernt die Deklaration eines BLOB-Filters aus der Datenbank. Wenn Sie einen BLOB-Filter aus einer Datenbank entfernen, kann er für diese Datenbank nicht mehr verwendet werden. Die dynamische Bibliothek, in der sich die Konvertierungsfunktion befindet, bleibt erhalten und das Entfernen aus einer Datenbank hat keine Auswirkungen auf andere Datenbanken, in denen derselbe BLOB-Filter noch deklariert ist.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann einen BLOB-Filter löschen.

Beispiel: Löschen eines BLOB-Filters.

```
DROP FILTER DESC_FILTER;
```

Siehe auch: [DECLARE FILTER](#)

SEQUENCE (GENERATOR)

Eine Sequenz oder ein Generator ist ein Datenbankobjekt, das verwendet wird, um eindeutige Zahlenwerte zu erhalten, um eine Reihe zu füllen. „Sequenz“ ist der SQL-konforme Begriff für das gleiche Ding, das in Firebird traditionell als „Generator“ bekannt ist. Beide Begriffe sind in Firebird implementiert. Für beide Terme ist eine Syntax implementiert.

Sequenzen (oder Generatoren) werden immer als 64-Bit-Ganzzahlen gespeichert, unabhängig vom SQL-Dialekt der Datenbank.

Achtung

Wenn ein Client mit Dialekt 1 verbunden ist, sendet der Server Sequenzwerte als 32-Bit-Ganzzahlen an ihn. Das Übergeben eines Sequenzwerts an ein 32-Bit-Feld oder eine Variable führt nicht zu Fehlern, solange der aktuelle Wert der Sequenz die Grenzen einer 32-Bit-Zahl nicht überschreitet. Sobald jedoch der Sequenzwert diese Grenze überschreitet, erzeugt die Datenbank in Dialekt 3 einen Fehler. Eine Datenbank in Dialekt 1 wird weiterhin die Werte beschneiden, was die Einzigartigkeit der Serie beeinträchtigt.

In diesem Abschnitt wird beschrieben, wie Sequenzen erstellt, festgelegt und gelöscht werden.

CREATE SEQUENCE

Benutzt für: Erstellen einer neuen SEQUENCE (GENERATOR)

Verfügbar in: DSQL, ESQL

Syntax:

```
CREATE {SEQUENCE | GENERATOR} seq_name
```

Tabelle 5.32. CREATE SEQUENCE | CREATE GENERATOR Statement-Parameter

| Parameter | Beschreibung |
|-----------|---|
| seq_name | Name der Sequenz (Generator). Diese kann aus bis zu 31 Zeichen bestehen |

Die Anweisungen CREATE SEQUENCE und CREATE GENERATOR sind Synonyme — beide erzeugen eine neue Sequenz. Jede kann verwendet werden, jedoch wird CREATE SEQUENCE empfohlen, sofern die normkonforme Metadatenverwaltung wichtig ist.

Wenn eine Sequenz erstellt wird, wird ihr Wert auf 0 gesetzt. Jedes Mal, wenn der `NEXT VALUE FOR seq_name` wird der Wert um 1 erhöht. Die `GEN_ID(seq_name, <step>)`-Funktion kann stattdessen aufgerufen werden, um die Reihe durch eine andere Ganzzahl zu erhöhen oder zu reduzieren.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann eine Sequenz (Generator) erstellen.

Beispiele:

1. Erstellen der EMP_NO_GEN-Sequenz mittels CREATE SEQUENCE.

```
CREATE SEQUENCE EMP_NO_GEN;
```

2. Erstellen der EMP_NO_GEN-Sequenz mittels CREATE GENERATOR.

```
CREATE GENERATOR EMP_NO_GEN;
```

Siehe auch: ALTER SEQUENCE, SET GENERATOR, DROP SEQUENCE (GENERATOR), NEXT VALUE FOR, GEN_ID() function

ALTER SEQUENCE

Benutzt für: Festlegen des Werts einer Sequenz oder eines Generators auf einen bestimmten Wert

Verfügbar in: DSQL

Syntax:

```
ALTER SEQUENCE seq_name RESTART WITH new_val
```

Tabelle 5.33. ALTER SEQUENCE Statement-Parameter

| Parameter | Beschreibung |
|-----------|------------------------------|
| seq_name | Name der Sequenz (Generator) |

| Parameter | Beschreibung |
|-----------|--|
| new_val | Neuer Sequenzwert (Generatorwert). Eine 64-Bit-Ganzzahl von -2^{63} bis $2^{63}-1$. |

Mit der Anweisung ALTER SEQUENCE wird der aktuelle Wert einer Sequenz oder eines Generators auf den angegebenen Wert gesetzt.

Warnung

Die falsche Verwendung der ALTER SEQUENCE-Anweisung (Ändern des aktuellen Werts der Sequenz oder des Generators) kann die logische Integrität von Daten beeinträchtigen.

Jeder an die Datenbank angeschlossene Benutzer kann den Sequenzwert (Generator) festlegen.

Beispiele:

1. Festlegen des Werts der EMP_NO_GEN-Sequenz auf 145.

```
ALTER SEQUENCE EMP_NO_GEN RESTART WITH 145;
```

2. Das gleiche Prozedere unter Verwendung von SET GENERATOR:

```
SET GENERATOR EMP_NO_GEN TO 145;
```

Siehe auch: SET GENERATOR, CREATE SEQUENCE (GENERATOR), DROP SEQUENCE (GENERATOR), NEXT VALUE FOR, GEN_ID() function

SET GENERATOR

Benutzt für: Festlegen des Werts einer Sequenz oder eines Generators auf einen bestimmten Wert

Verfügbar in: DSQL, ESQL

Syntax:

```
SET GENERATOR seq_name TO new_val
```

Tabelle 5.34. SET GENERATOR Statement-Parameter

| Parameter | Beschreibung |
|-----------|--|
| seq_name | Name des Generators (Sequenz) |
| new_val | Neuer Sequenzwert (Generatorwert). Eine 64-Bit-Ganzzahl von -2^{63} bis $2^{63}-1$. |

Mit der Anweisung SET GENERATOR wird der aktuelle Wert einer Sequenz oder eines Generators auf den angegebenen Wert gesetzt.

Anmerkung

Obwohl SET GENERATOR als veraltet gilt, wird es aus Gründen der Abwärtskompatibilität beibehalten. Die standardkonforme Anweisung ALTER SEQUENCE ist aktuell und wird empfohlen.

Jeder an die Datenbank angeschlossene Benutzer kann den Sequenzwert (Generator) festlegen.

Beispiele:

1. Einstellen des Werts der EMP_NO_GEN-Sequenz auf 145:

```
SET GENERATOR EMP_NO_GEN TO 145;
```

2. Das gleiche Prozedere unter Verwendung von ALTER SEQUENCE:

```
ALTER SEQUENCE EMP_NO_GEN RESTART WITH 145;
```

Siehe auch: ALTER SEQUENCE, CREATE SEQUENCE (GENERATOR)

DROP SEQUENCE

Benutzt für: Löschen von SEQUENCE (GENERATOR)

Verfügbar in: DSQL, ESQL

Syntax:

```
DROP {SEQUENCE | GENERATOR} seq_name
```

Tabelle 5.35. DROP SEQUENCE | DROP GENERATOR Statement-Parameter

| Parameter | Beschreibung |
|-----------|--|
| seq_name | Name der Sequenz (Generator). Kann aus bis zu 31 Zeichen bestehen. |

Die Anweisungen DROP SEQUENCE und DROP GENERATOR sind gleichwertig: Beide löschen eine vorhandene Sequenz (Generator). Beide sind gültig, jedoch wird DROP SEQUENCE empfohlen.

Die Anweisungen schlagen fehl, wenn die Sequenz (Generator) Abhängigkeiten hat.

Jeder an die Datenbank angeschlossene Benutzer kann eine Sequenz (Generator) löschen.

Beispiele: Löschen der EMP_NO_GEN-Sequenz:

```
DROP SEQUENCE EMP_NO_GEN;
```

Siehe auch: [CREATE SEQUENCE \(GENERATOR, ALTER SEQUENCE, SET GENERATOR](#)

EXCEPTION

In diesem Abschnitt wird beschrieben, wie *benutzerdefinierte Ausnahmen* zur Verwendung in Fehlerbehandlungsroutinen in PSQL-Modulen erstellt, geändert und gelöscht werden.

CREATE EXCEPTION

Benutzt für: Erstellen einer neuen Ausnahme für die Verwendung in PSQL-Modulen

Verfügbar in: DSQL, ESQL

Syntax:

```
CREATE EXCEPTION exception_name 'message'
```

Tabelle 5.36. CREATE EXCEPTION Statement-Parameter

| Parameter | Beschreibung |
|----------------|---|
| exception_name | Name der Ausnahme. Die maximale Länge beträgt 31 Zeichen |
| message | Standardfehlermeldung. Die maximale Länge beträgt 1.021 Zeichen |

Die Anweisung CREATE EXCEPTION erstellt eine neue Ausnahme zur Verwendung in PSQL-Modulen. Wenn eine Ausnahme desselben Namens existiert, schlägt die Anweisung mit einer entsprechenden Fehlermeldung fehl.

Der Name der Ausnahme ist eine Standardkennung. In einer Dialekt 3-Datenbank kann sie in doppelte Anführungszeichen eingeschlossen werden, um Groß- und Kleinschreibung zu berücksichtigen und bei Bedarf Zeichen zu verwenden, die in regulären Bezeichnern nicht gültig sind. Weitere Informationen finden Sie unter [Bezeichner](#).

Die Standardnachricht wird im Zeichensatz NONE gespeichert, d.h. in Zeichen eines Einbytezeichensatzes. Der Text kann im PSQL-Code überschrieben werden, wenn die Ausnahme ausgelöst wird.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann eine Ausnahme erstellen.

Beispiele:

1. Erstellen einer Ausnahme mit dem Namen E_LARGE_VALUE:

```
CREATE EXCEPTION E_LARGE_VALUE
    'The value is out of range';
```

- Erstellen einer Ausnahme mit dem Namen ERROR_REFIN_RATE:

```
CREATE EXCEPTION ERROR_REFIN_RATE
    'Error detected in the spread of discount rates';
```

Tipps

Die Zusammenfassung von CREATE EXCEPTION-Anweisungen in Systemaktualisierungsskripts vereinfacht die Arbeit mit ihnen und dokumentiert sie. Ein System von Präfixen zum Benennen und Kategorisieren von Gruppen von Ausnahmen wird empfohlen.

Benutzerdefinierte Ausnahmen werden in der Systemtabelle gespeichert `RDB$EXCEPTIONS`.

Siehe auch: ALTER EXCEPTION, CREATE OR ALTER EXCEPTION, DROP EXCEPTION, RECREATE EXCEPTION

ALTER EXCEPTION

Benutzt für: Ändern der Nachricht, die von einer benutzerdefinierten Ausnahme zurückgegeben wird

Verfügbar in: DSQL, ESQL

Syntax:

```
ALTER EXCEPTION exception_name 'message'
```

Tabelle 5.37. ALTER EXCEPTION Statement-Parameter

| Parameter | Beschreibung |
|----------------|--|
| exception_name | Name der Ausnahme |
| message | Neue Standardfehlermeldung. Die maximale Länge beträgt 1.021 Zeichen |

Die Anweisung ALTER EXCEPTION kann jederzeit verwendet werden, um den Standardtext der Nachricht zu ändern. Jeder Benutzer, der mit der Datenbank verbunden ist, kann eine Ausnahmemeldung ändern.

Beispiele:

- Ändern der Standardnachricht für die Ausnahme E_LARGE_VALUE:

```
ALTER EXCEPTION E_LARGE_VALUE
    'The value exceeds the prescribed limit of 32,765 bytes';
```

2. Ändern der Standardnachricht für die Ausnahme ERROR_REFIN_RATE:

```
ALTER EXCEPTION ERROR_REFIN_RATE 'Rate is outside the allowed range';
```

Siehe auch: [CREATE EXCEPTION](#), [CREATE OR ALTER EXCEPTION](#), [DROP EXCEPTION](#), [RECREATE EXCEPTION](#)

CREATE OR ALTER EXCEPTION

Benutzt für: Ändern der Nachricht, die von einer benutzerdefinierten Ausnahme zurückgegeben wird, wenn die Ausnahme existiert; andernfalls erstellen Sie eine neue Ausnahme

Verfügbar in: DSQL

Syntax:

```
CREATE OR ALTER EXCEPTION exception_name 'message'
```

Tabelle 5.38. CREATE OR ALTER EXCEPTION Statement-Parameter

| Parameter | Beschreibung |
|----------------|--|
| exception_name | Name der Ausnahme |
| message | Fehlermeldung. Die maximale Länge ist auf 1.021 Zeichen begrenzt |

Die Anweisung CREATE ODER ALTER EXCEPTION wird verwendet, um die angegebene Ausnahme zu erstellen, falls sie nicht existiert, oder um den Text der von ihr zurückgegebenen Fehlermeldung zu ändern. Wenn eine vorhandene Ausnahme durch diese Anweisung geändert wird, bleiben vorhandene Abhängigkeiten erhalten.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann diese Anweisung verwenden, um eine Ausnahme zu erstellen oder den bereits vorhandenen Text zu ändern.

Beispiel: Nachricht für die Ausnahme ändern E_LARGE_VALUE:

```
CREATE OR ALTER EXCEPTION E_LARGE_VALUE
    'The value is higher than the permitted range 0 to 32,765';
```

Siehe auch: [CREATE EXCEPTION](#), [ALTER EXCEPTION](#), [RECREATE EXCEPTION](#)

DROP EXCEPTION

Benutzt für: Löschen einer benutzerdefinierten Ausnahme

Verfügbar in: DSQL, ESQL

Syntax:

```
DROP EXCEPTION exception_name
```

Tabelle 5.39. DROP EXCEPTION Statement-Parameter

| Parameter | Beschreibung |
|----------------|-------------------|
| exception_name | Name der Ausnahme |

Die Anweisung DROP EXCEPTION wird zum Löschen einer Ausnahme verwendet. Alle Abhängigkeiten von der Ausnahme führen dazu, dass die Anweisung fehlschlägt und nicht gelöscht wird.

Wenn eine Ausnahme nur in gespeicherten Prozeduren verwendet wird, kann sie jederzeit gelöscht werden. Wenn es in einem Auslöser verwendet wird, kann es nicht gelöscht werden.

Bei der Planung, eine Ausnahme zu löschen, sollten alle Verweise darauf aus dem Code der gespeicherten Prozeduren entfernt werden, um zu vermeiden, dass die Abwesenheit Fehler verursacht.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann eine Ausnahme löschen.

Beispiele:

1. Ausnahme löschen ERROR_REFIN_RATE:

```
DROP EXCEPTION ERROR_REFIN_RATE;
```

2. Ausnahme löschen E_LARGE_VALUE:

```
DELETE EXCEPTION E_LARGE_VALUE;
```

Siehe auch: [CREATE EXCEPTION](#), [RECREATE EXCEPTION](#)

RECREATE EXCEPTION

Benutzt für: Eine neue benutzerdefinierte Ausnahme oder eine bestehende erstellen

Verfügbar in: DSQL

Syntax:

```
RECREATE EXCEPTION exception_name 'message'
```

Tabelle 5.40. RECREATE EXCEPTION Statement-Parameter

| Parameter | Beschreibung |
|----------------|--|
| exception_name | Name der Ausnahme. Die maximale Länge beträgt 31 Zeichen |
| message | Fehlermeldung. Die maximale Länge ist auf 1.021 Zeichen begrenzt |

Die Anweisung RECREATE EXCEPTION erstellt eine neue Ausnahme für die Verwendung in PSQL-Modulen. Wenn bereits eine Exception mit demselben Namen existiert, versucht die Anweisung RECREATE EXCEPTION, diese zu löschen und eine neue zu erstellen. Wenn es Abhängigkeiten zur bestehenden Ausnahme gibt, schlägt die versuchte Löschung fehl und RECREATE EXCEPTION wird nicht ausgeführt.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann eine Ausnahme erstellen.

Beispiel: Wiederherstellen der Ausnahme E_LARGE_VALUE:

```
RECREATE EXCEPTION E_LARGE_VALUE
    'The value exceeds its limit';
```

Siehe auch: [CREATE EXCEPTION](#), [DROP EXCEPTION](#), [CREATE OR ALTER EXCEPTION](#)

COLLATION

CREATE COLLATION

Benutzt für: Erstellen einer neuen Sortierung für einen unterstützten Zeichensatz, der für die Datenbank verfügbar ist

Verfügbar in: DSQL

Syntax:

```
CREATE COLLATION collname
FOR charset
[FROM basecoll | FROM EXTERNAL ('extname')]
[NO PAD | PAD SPACE]
[CASE [IN]SENSITIVE]
[ACCENT [IN]SENSITIVE]
['<specific-attributes>'];

<specific-attributes> ::= <attribute> [; <attribute> ...]

<attribute> ::= attrname=attrvalue
```

Tabelle 5.41. CREATE COLLATION Statement-Parameter

| Parameter | Beschreibung |
|-----------|---|
| collname | Der Name, der für die neue Collation verwendet werden soll. Die maximale Länge beträgt 31 Zeichen |
| charset | Ein in der Datenbank vorhandener Zeichensatz |
| basecoll | Eine bereits in der Datenbank vorhandene Collation |
| extname | Der in der Datei <code>.conf</code> verwendete Collation-Name |

Die Anweisung `CREATE COLLATION` „erzeugt“ nichts: Sie dient dazu, eine Datenbank-Collation bekannt zu machen. Die Collation muss bereits auf dem System vorhanden sein, normalerweise in einer Bibliotheksdatei und muss ordnungsgemäß in einer `.conf`-Datei im Unterverzeichnis `intl` der Firebird-Installation registriert sein.

Die Collation kann alternativ auf einer basieren, die bereits in der Datenbank vorhanden ist.

Wie die Engine die Collation erkennt

Wenn keine `FROM`-Klausel vorhanden ist, scannt Firebird die `.conf`-Dateien im `intl`-Unterverzeichnis nach einer Collation mit dem Namen, der als Objekt mit `CREATE COLLATION` gesetzt wurde. Anders ausgedrückt, das Weglassen der `FROM basecoll`-Klausel entspricht der Angabe von `FROM EXTERNAL ('collname')`.

Bei der Angabe von `'extname'` muss die Groß- / Kleinschreibung beachtet werden und sie muss genau mit dem Collations-Namen in der Datei `.conf` übereinstimmen. Bei den Parametern `collname`, `zeichensatz` und `basecoll` wird zwischen Groß- und Kleinschreibung unterschieden, sofern sie nicht in doppelten Anführungszeichen eingeschlossen sind.

Spezifische Attribute

Die verfügbaren spezifischen Attribute sind in der folgenden Tabelle aufgeführt. Nicht alle spezifischen Attribute gelten für jede Collation auch wenn sie nicht durch einen Fehler verursacht werden.

Wichtig

Spezifische Attribute unterscheiden Groß- und Kleinschreibung.

In der Tabelle zeigt „1 bpc“ an, dass ein Attribut für Collationen von Zeichensätzen mit 1 Byte pro Zeichen (sogenannte enge Zeichensätze) gültig ist. „UNI“ steht für „UNICODE Collationen“.

Tabelle 5.42. Spezifische Collations-Attribute

| Attribute | Wert | Gültig für | Kommentar |
|----------------------|------|------------|--|
| DISABLE-COMPRESSIONS | 0, 1 | 1 bpc | Deaktiviert Komprimierungen (a.k.a. Kontraktionen). Komprimierungen bewirken, dass bestimmte Zeichenfolgen als atomare Einheiten sortiert werden, z.B. spanisch <code>c + h</code> als einzelner Buchstabe <code>ch</code> |

| Attribute | Wert | Gültig für | Kommentar |
|--------------------|------------------|------------|--|
| DISABLE-EXPANSIONS | 0, 1 | 1 bpc | Deaktiviert Erweiterungen. Erweiterungen bewirken, dass bestimmte Zeichen (z. B. Ligaturen oder umlautete Vokale) als Zeichenfolgen behandelt und entsprechend sortiert werden |
| ICU-VERSION | default oder H.U | UNI | Gibt die zu verwendende ICU-Bibliotheksversion an. Gültige Werte sind diejenigen, die im anwendbaren Element <code><intl_module></code> in <code>intl / fbintl.conf</code> . Format: entweder das Stringliteral „default“ oder eine Hauptversion + Unterversion wie „3.0“ (beide nicht angegeben). |
| LOCALE | xx_YY | UNI | Gibt das Sortierungskriterium an. Benötigt eine vollständige Version von ICU-Bibliotheken. Format: eine Gebietsschema-Zeichenfolge wie „du_NL“ (nicht angegeben) |
| MULTI-LEVEL | 0, 1 | 1 bpc | Verwendet mehr als eine Sortierebene |
| NUMERIC-SORT | 0, 1 | UNI | Behandelt zusammenhängende Gruppen von Dezimalziffern in der Zeichenfolge als atomare Einheiten und sortiert sie numerisch. (Dies ist auch als natürliche Sortierung bekannt) |
| SPECIALS-FIRST | 0, 1 | 1 bpc | Ordnet Sonderzeichen (Leerzeichen, Symbole etc.) vor alphanumerischen Zeichen |

Tipp

Deklariieren Sie die gespeicherte Prozedur `sp_register_character_set` (name, max_bytes_per_character), die Sie in `misc / intl.sql` / im Firebird-Installationsverzeichnis.

Hinweis: Damit dies funktioniert, muss der Zeichensatz auf dem System vorhanden und in einer `.conf` im Unterverzeichnis `intl` liegen.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann `CREATE COLLATION` verwenden, um eine neue Collation hinzuzufügen.

Beispiele für die Verwendung von `CREATE COLLATION`:

1. Erstellen einer Collation mit dem in der Datei `fbintl.conf` enthaltenen Namen (Groß- und Kleinschreibung beachten).

```
CREATE COLLATION ISO8859_1_UNICODE FOR ISO8859_1;
```

2. Erstellen einer Collation, mithilfe eines speziellen (benutzerdefinierten) Namens (der „externe“ Name muss vollständig mit dem Namen in der Datei `fbintl.conf` übereinstimmen).

```
CREATE COLLATION LAT_UNI
FOR ISO8859_1
FROM EXTERNAL ( 'ISO8859_1_UNICODE' );
```

- Erstellen einer Collation, die Groß- und Kleinschreibung nicht berücksichtigt und auf einer bereits in der Datenbank vorhandenen basiert.

```
CREATE COLLATION ES_ES_NOPAD_CI
FOR ISO8859_1
FROM ES_ES
NO PAD
CASE INSENSITIVE;
```

- Erstellen einer Collation, die Groß- und Kleinschreibung nicht berücksichtigt und auf einer bereits in der Datenbank vorhandenen basiert. Angabe spezifischer Attribute.

```
CREATE COLLATION ES_ES_CI_COMPR
FOR ISO8859_1
FROM ES_ES
CASE INSENSITIVE
'DISABLE-COMPRESSIONS=0';
```

- Erstellen einer Collation, die Groß- und Kleinschreibung nicht berücksichtigt durch Verwendung von Zahlen (die sogenannte natürliche Collation).

```
CREATE COLLATION nums_coll FOR UTF8
FROM UNICODE
CASE INSENSITIVE 'NUMERIC-SORT=1';

CREATE DOMAIN dm_nums AS varchar(20)
CHARACTER SET UTF8 COLLATE nums_coll; -- original (manufacturer) numbers

CREATE TABLE wares(id int primary key, articul dm_nums ...);
```

Siehe auch: [DROP COLLATION](#)

DROP COLLATION

Benutzt für: Löschen einer Collation aus der Datenbank

Verfügbar in: DSQL

Syntax:

```
DROP COLLATION collname
```

Tabelle 5.43. DROP COLLATION Statement-Parameter

| Parameter | Beschreibung |
|-----------|--------------------|
| collname | Name der Collation |

Die Anweisung DROP COLLATION entfernt die angegebene Collation aus der Datenbank, falls vorhanden. Ein Fehler wird ausgelöst, wenn die angegebene Collation nicht vorhanden ist.

Tipp

Deklarieren und führen Sie die gespeicherte Prozedur `sp_unregister_character_set (name)` aus dem Unterverzeichnis `misc / intl.sql` der Firebird-Installation aus, wenn Sie einen ganzen Zeichensatz mit allen Collationen aus der Datenbank entfernen möchten.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann DROP COLLATION verwenden, um eine Collation zu entfernen.

Beispiele für die Verwendung von DROP COLLATION: Löschen der ES_ES_NOPAD_CI-Collation.

```
DROP COLLATION ES_ES_NOPAD_CI;
```

Siehe auch: [CREATE COLLATION](#)

CHARACTER SET

ALTER CHARACTER SET

Benutzt für: Festlegen der Standardsortierung für einen Zeichensatz

Verfügbar in: DSQL

Syntax:

```
ALTER CHARACTER SET charset
SET DEFAULT COLLATION collation;
```

Tabelle 5.44. ALTER CHARACTER SET Statement-Parameter

| Parameter | Beschreibung |
|-----------|--------------------|
| charset | Zeichensatzkennung |

| Parameter | Beschreibung |
|-----------|-------------------------|
| collation | Der Name der Sortierung |

Die Anweisung ALTER CHARACTER SET ändert die Standardsortierung für den angegebenen Zeichensatz. Dies wirkt sich auf die zukünftige Verwendung des Zeichensatzes aus, mit Ausnahme der Fälle, in denen die COLLATE-Klausel explizit überschrieben wird. In diesem Fall bleibt die Sortierreihenfolge bestehender Domainn, Spalten und PSQL-Variablen nach der Änderung der Standardsortierung des zugrunde liegenden Zeichensatzes erhalten.

Hinweise

Wenn Sie die Standardsortierung für den Datenbankzeichensatz ändern (die beim Erstellen der Datenbank definiert wurde), wird die Standardsortierung für die Datenbank geändert.

Wenn Sie die Standardsortierung für den Zeichensatz ändern, der während der Verbindung angegeben wurde, werden Stringkonstanten gemäß dem neuen Sortierungswert interpretiert, außer in den Fällen, in denen der Zeichensatz und / oder die Collation überschrieben wurde.

Beispiel zur Verwendung: Festlegen der Standard-UNICODE_CI_AI-Sortierung für die UTF8-Codierung.

```
ALTER CHARACTER SET UTF8
    SET DEFAULT COLLATION UNICODE_CI_AI ;
```

ROLE

Eine Rolle ist ein Datenbankobjekt, das eine Reihe [SQL-Berechtigungen](#) paketiert. Rollen implementieren das Konzept der Zugriffskontrolle auf Gruppenebene. Der Rolle werden mehrere Berechtigungen erteilt, und diese Rolle kann einem oder mehreren Benutzern gewährt oder widerrufen werden.

Ein Benutzer, dem eine Rolle zugewiesen wurde, muss diese Rolle in seinen Anmeldeinformationen bereitstellen, um die zugehörigen Berechtigungen auszuüben. Alle anderen Berechtigungen, die dem Benutzer gewährt werden, sind von seiner Anmeldung mit der Rolle nicht betroffen. Die gleichzeitige Anmeldung mit mehreren Rollen wird nicht unterstützt.

In diesem Abschnitt werden die Aufgaben zum Erstellen und Löschen von Rollen besprochen.

CREATE ROLE

Benutzt für: Erstellen eines neuen ROLE-Objektes

Verfügbar in: DSQL, ESQL

Syntax:

```
CREATE ROLE rolename;
```

Tabelle 5.45. CREATE ROLE Statement-Parameter

| Parameter | Beschreibung |
|-----------|---|
| rolename | Rollenname. Die maximale Länge beträgt 31 Zeichen |

Die Anweisung CREATE ROLE erstellt ein neues Rollenobjekt, dem später ein oder mehrere Berechtigungen erteilt werden können. Der Name einer Rolle muss unter den Namen der Rollen in der aktuellen Datenbank eindeutig sein.

Warnung

Es ist ratsam, den Namen einer Rolle auch unter den Benutzernamen eindeutig zu machen. Das System verhindert nicht die Erstellung einer Rolle, deren Name mit einem vorhandenen Benutzernamen kollidiert. Wenn dies der Fall ist, kann der Benutzer keine Verbindung zur Datenbank herstellen.

Jeder Benutzer, der mit der Datenbank verbunden ist, kann eine Rolle erstellen. Der Benutzer, der eine Rolle erstellt, wird zu seinem Besitzer.

Beispiel: Erstellen einer Rolle SELLERS:

```
CREATE ROLE SELLERS;
```

Siehe auch: [DROP ROLE](#), [GRANT](#), [REVOKE](#)

ALTER ROLE

ALTER ROLE hat keinen Platz im create-alter-drop-Paradigma für Datenbankobjekte, da eine Rolle keine Attribute besitzt, die geändert werden können. Sein tatsächlicher Effekt besteht darin, ein Attribut der Datenbank zu ändern: Firebird verwendet es, um die Fähigkeit von Windows Administratoren zu aktivieren und zu deaktivieren, beim Anmelden automatisch [Administratorrechte](#) zu erhalten.

Diese Prozedur trifft nur auf eine Rolle zu: Die systemgenerierte Rolle RDB\$ADMIN, die in jeder Datenbank von ODS 11.2 oder höher vorhanden ist. Bei der Aktivierung dieser Funktion sind mehrere Faktoren beteiligt.

Für weitere Details, siehe [AUTO ADMIN MAPPING](#) im Kapitel *Sicherheit*.

DROP ROLE

Benutzt für: Löschen einer Rolle

Verfügbar in: DSQL, ESQL

Syntax:

```
DROP ROLE rolename;
```

Die Anweisung DROP ROLE löscht eine vorhandene Rolle. Es braucht nur ein einziges Argument, den Namen der Rolle. Sobald die Rolle gelöscht wurde, wird der gesamte Berechtigungssatz von allen Benutzern und Objekten, denen die Rolle gewährt wurde, widerrufen.

Eine Rolle kann vom Eigentümer gelöscht oder von einem [Administrator](#).

Beispiel: Löschen der Rolle SELLERS:

```
DROP ROLE SELLERS;
```

Siehe auch: [CREATE ROLE](#), [GRANT](#), [REVOKE](#)

COMMENTS

Datenbankobjekte und eine Datenbank selbst können Kommentare enthalten. Es ist ein bequemer Mechanismus zur Dokumentation der Entwicklung und Pflege einer Datenbank. Kommentare, die mit COMMENT ON erstellt wurden, überstehen eine *gbak*-Sicherung und -Wiederherstellung.

COMMENT ON

Benutzt für: Documentation von Metadaten

Verfügbar in: DSQL

Syntax:

```
COMMENT ON <object> IS {'sometext' | NULL}

<object> ::=
    DATABASE
    | <basic-type> objectname
    | COLUMN relationname.fieldname
    | PARAMETER procname.paramname

<basic-type> ::=
    CHARACTER SET |
    COLLATION |
    DOMAIN |
    EXCEPTION |
    EXTERNAL FUNCTION |
    FILTER |
    GENERATOR |
    INDEX |
    PROCEDURE |
```

```
ROLE |
SEQUENCE |
TABLE |
TRIGGER |
VIEW
```

Tabelle 5.46. COMMENT ON Statement-Parameter

| Parameter | Beschreibung |
|--------------|--------------------------------------|
| sometext | Kommentartext |
| basic-type | Metadatenobjekttyp |
| objectname | Name des Metadatenobjekts |
| relationname | Name der Tabelle oder View |
| procname | Name der Stored Procedure |
| paramname | Name des Stored Procedure Parameters |

Die Anweisung COMMENT ON fügt Kommentare zu Datenbankobjekten (Metadaten) hinzu. Kommentare werden in Textfeldern des Typs BLOB in der Spalte RDB\$DESCRIPTION der entsprechenden Systemtabellen gespeichert. Clientanwendungen können Kommentare aus diesen Feldern anzeigen.

Anmerkung

Wenn Sie einen leeren Kommentar (") hinzufügen, wird dieser als NULL in der Datenbank gespeichert.

Der Eigentümer der Tabelle oder Prozedur und [Administratoren](#) haben die notwendigen Berechtigungen die Anweisung COMMENT ON zu verwenden.

Beispiele zur Verwendung von COMMENT ON:

1. Einen Kommentar für die aktuelle Datenbank hinzufügen

```
COMMENT ON DATABASE IS 'It is a test ('my.fdb') database';
```

2. Einen Kommentar für die METALS-Tabelle hinzufügen

```
COMMENT ON TABLE METALS IS 'Metal directory';
```

3. Hinzufügen eines Kommentars zum Feld ISALLOY in der METALS-Tabelle

```
COMMENT ON COLUMN METALS.ISALLOY IS '0 = fine metal, 1 = alloy';
```

4. Einen Kommentar für einen Parameter hinzufügen

```
COMMENT ON PARAMETER ADD_EMP_PROJ. EMP_NO IS 'Employee ID';
```


Statements der Data Manipulation Language (DML)

ÜBERPRÜFUNGSSTATUS

Alle Abschnitte von diesem Punkt bis zum Ende des Kapitels warten auf technische und redaktionelle Überprüfung.

DML — Datenbearbeitungssprache — ist die Teilmenge von SQL, die von Anwendungen und prozeduralen Modulen zum Extrahieren und Ändern von Daten verwendet wird. Die Extraktion zum Lesen sowohl roher als auch manipulierter Daten wird mit der Anweisung `SELECT` erreicht. `INSERT` dient zum Hinzufügen neuer Daten und `DELETE` dient zum Löschen von Daten, die nicht mehr benötigt werden. `UPDATE`, `MERGE` und `UPDATE OR INSERT` ändern alle Daten auf verschiedene Arten.

SELECT

Verwendet für: Retrieving data

Verfügbar in: DSQL, ESQL, PSQL

Globale Syntax:

```
SELECT
[WITH [RECURSIVE] <cte> [, <cte> ...]]
SELECT
  [FIRST m] [SKIP n]
  [DISTINCT | ALL] <columns>
FROM
  source [[AS] alias]
  [<joins>]
[WHERE <condition>]
[GROUP BY <grouping-list>]
[HAVING <aggregate-condition>]]
[PLAN <plan-expr>]
[UNION [DISTINCT | ALL] <other-select>]
[ORDER BY <ordering-list>]
[ROWS m [TO n]]
[FOR UPDATE [OF <columns>]]
[WITH LOCK]
[INTO <variables>]

<variables> ::= [:]varname [, [:]varname ...]
```

Beschreibung

Die Anweisung SELECT ruft Daten aus der Datenbank ab und übergibt sie an die Anwendung oder die umschließende SQL-Anweisung. Daten werden in null oder mehr *Zeilen* zurückgegeben, die jeweils eine oder mehrere *Spalten* oder *Felder* enthalten. Die Summe der zurückgegebenen Zeilen ist die *Ergebnismenge* der Anweisung.

Die einzigen obligatorischen Teile der Anweisung SELECT sind:

- Das Schlüsselwort SELECT gefolgt von einer Spaltenliste. Dieser Teil spezifiziert *was* Sie abrufen möchten.
- Das Schlüsselwort FROM gefolgt von einem auswählbaren Objekt. Dies sagt der Engine *von wo* Sie Daten erhalten möchten.

In der einfachsten Form ruft SELECT eine Anzahl von Spalten aus einer einzelnen Tabelle oder Sicht ab, wie folgt:

```
select id, name, address
from contacts
```

Oder, um alle Spalten abzurufen:

```
select * from sales
```

In der Praxis werden die abgerufenen Zeilen oft durch eine Klausel WHERE begrenzt. Die Ergebnismenge kann nach einer ORDER BY-Klausel sortiert werden. FIRST, SKIP oder ROWS können die Anzahl der Ausgabezeilen eingrenzen. Die Spaltenliste kann alle Arten von Ausdrücken anstelle von nur Spaltennamen enthalten, und die Quelle muss keine Tabelle oder Sicht sein; sie kann auch eine abgeleitete Tabelle, ein allgemeiner Tabellenausdruck (CTE) oder eine auswählbare gespeicherte Prozedur (SP) sein. Mehrere Quellen können in einer JOIN kombiniert werden, und mehrere Ergebnismengen können in einer UNION kombiniert werden.

In den folgenden Abschnitten werden die verfügbaren SELECT Unterklauseln und ihre Verwendung im Detail erläutert.

FIRST, SKIP

Verwendet für: Abrufen eines Teiles von Zeilen aus einer geordneten Menge

Verfügbar in: DSQL, PSQL

Syntax:

```
SELECT
  [FIRST <m>] [SKIP <n>]
  FROM ...
  ...

<m>, <n> ::= integer-literal | query-parameter | (integer-expression)
```

Tabelle 6.1. Argumente für die FIRST- und SKIP-Klauseln

| Argument | Beschreibung |
|-----------------|--|
| integer literal | Ganzzahliges Literal |
| query parameter | Abfrageparameter-Platzhalter. ? in DSQL und :paramname in PSQL |

| Argument | Beschreibung |
|--------------------|---|
| integer-expression | Ausdruck, der einen Ganzzahlwert zurückgibt |

FIRST und SKIP sind keine Standardsyntax

FIRST und SKIP sind Firebird-spezifische, nicht-SQL-konforme Schlüsselwörter. Es wird dringend empfohlen die [ROWS](#)-Syntax zu verwenden.

Beschreibung

FIRST begrenzt die Ausgabe der Abfrage auf die ersten m Zeilen. SKIP überspringt die ersten n Zeilen, bevor mit der Ausgabe begonnen wird.

FIRST und SKIP sind optional. Bei Verwendung in „FIRST m SKIP n “ werden die obersten n Zeilen der Ausgabe verworfen und die ersten m Zeilen des Rests der Menge zurückgegeben.

Eigenschaften von FIRST und SKIP

- Jedes Argument für FIRST und SKIP, das kein Integer-Literal oder ein SQL-Parameter ist, muss in Klammern stehen. Dies bedeutet, dass ein Unterabfrageausdruck in *zwei* Klammern eingeschlossen sein muss.
- SKIP 0 ist erlaubt, jedoch vollkommen sinnlos.
- FIRST 0 ist ebenfalls erlaubt und gibt eine leere Ergebnismenge zurück.
- Negative Werte für SKIP und/oder FIRST resultieren in einem Fehler.
- Wenn eine SKIP hinter dem Ende des Datensatzes landet, wird ein leerer Satz zurückgegeben.
- Wenn die Anzahl der Zeilen im Datensatz (oder der Rest nach einem SKIP) kleiner als der Wert des m -Arguments für FIRST ist, wird die kleinere Anzahl von Zeilen zurückgegeben. Dies sind gültige Ergebnisse, keine Fehler.

Achtung

Wenn Sie FIRST in Unterabfragen verwenden, tritt ein Fehler auf. Diese Abfrage

```
DELETE FROM MYTABLE
WHERE ID IN (SELECT FIRST 10 ID FROM MYTABLE)
```

löscht **ALLE** Datensätze aus der Tabelle. Die Unterabfrage ruft jedesmal 10 Zeilen ab, löscht sie und die Operation wird wiederholt, bis die Tabelle leer ist. Beachten Sie dies! Oder, besser, verwenden Sie die Klausel [ROWS](#) in der DELETE-Anweisung.

Beispiele

Die folgende Abfrage gibt die ersten 10 Namen aus der People-Tabelle zurück:

```
select first 10 id, name from People
order by name asc
```

Die folgende Abfrage gibt alles zurück *but* the first 10 names:

```
select skip 10 id, name from People
order by name asc
```

Und dieser gibt die letzten 10 Zeilen zurück. Beachten Sie die doppelten Klammern:

```
select skip ((select count(*) - 10 from People))
id, name from People
order by name asc
```

Diese Abfrage gibt die Zeilen 81 bis 100 der People-Tabelle zurück:

```
select first 20 skip 80 id, name from People
order by name asc
```

Siehe auch: [ROWS](#)

Die SELECT-Spaltenliste

Die Spaltenliste enthält einen oder mehrere durch Kommas getrennte Wertausdrücke. Jeder Ausdruck liefert einen Wert für eine Ausgabespalte. Alternativ kann * („select star“) verwendet werden, um für alle Spalten in einer Beziehung zu stehen (d. H. Für eine Tabelle, eine Ansicht oder eine auswählbare gespeicherte Prozedur).

Syntax:

```
SELECT
  [...]
  [DISTINCT | ALL] <output-column> [, <output-column> ...]
  [...]
  FROM ...

<output-column> ::= [qualifier.]*
                   | <value-expression> [COLLATE collation] [[AS] alias]

<value-expression> ::= [qualifier.]table-column
                      | [qualifier.]view-column
                      | [qualifier.]selectable-SP-outparm
                      | constant
                      | context-variable
                      | function-call
                      | single-value-subselect
                      | CASE-construct
                      | „jeder andere Ausdruck, der einen einzelnen Wert eines Fir
                        oder NULL zurückgibt
                      | "

qualifier ::= ein Beziehungsname oder ein Alias
collation ::= ein gültiger Collations-Name (nur für Zeichentypspalten)
```

Tabelle 6.2. Argumente der SELECT-Spaltenliste

| Argument | Beschreibung |
|-----------|--|
| qualifier | Name der Beziehung (Sicht, gespeicherte Prozedur, abgeleitete Tabelle); oder ein Alias dafür |

| Argument | Beschreibung |
|-------------------------|---|
| collation | Nur für zeichenartige Spalten: Ein Collations-Name der für den Zeichensatz der Daten existiert und gültig ist |
| alias | Spalten- oder Feldalias |
| table-column | Name einer Tabellenspalte |
| view-column | Name einer Ansichtsspalte |
| selectable-SP-outparm | Deklariertes Name eines Ausgabeparameters einer auswählbaren gespeicherten Prozedur |
| constant | Eine Konstante |
| context-variable | Kontextvariable |
| function-call | Skalarer oder Aggregatfunktionsaufrufausdruck |
| single-value-subselect | Eine Unterabfrage, die einen Skalarwert zurückgibt (Singleton) |
| CASE-construct | CASE-Konstrukt, das Bedingungen für einen Rückgabewert definiert |
| other-single-value-expr | Jeder andere Ausdruck, der einen einzelnen Wert eines Firebird-Datentyps zurückgibt; oder NULL |

Beschreibung

Es ist immer gültig, einen Spaltennamen zu qualifizieren (oder „*“). Dies geschieht mit dem Namen oder Alias der Tabelle, Ansicht oder abfragbaren gespeicherten Prozedur, gefolgt von einem Punkt, z.B. `relationname.columnname`, `relationname.*`, `alias.columnname`, `alias.*`. Qualifizierend ist **required**, wenn der Spaltenname in mehr als einer Relation auftritt, die an einem Join beteiligt ist. Qualifizierendes „*“ ist immer obligatorisch, wenn es nicht das einzige Element in der Spaltenliste ist.

Wichtig

Aliase verschleiern den ursprünglichen Beziehungsnamen: Sobald eine Tabelle, eine Sicht oder eine Prozedur mit einem Alias versehen wurde, kann nur der Alias als Qualifikationsmerkmal für die gesamte Abfrage verwendet werden. Der Beziehungsname selbst ist nicht mehr verfügbar.

Der Spaltenliste kann optional eines der Schlüsselwörter DISTINCT or ALL vorangestellt werden:

- DISTINCT filtert alle doppelten Zeilen aus. Das heißt, wenn zwei oder mehr Zeilen die gleichen Werte in jeder entsprechenden Spalte haben, ist nur einer von ihnen in der Ergebnismenge enthalten
- ALL ist der Standard: es gibt alle Zeilen zurück, einschließlich Duplikate. ALL wird selten verwendet; Es wird für die Einhaltung des SQL-Standards unterstützt.

Eine Klausel COLLATE ändert das Erscheinungsbild der Spalte als solche nicht. Wenn die angegebene Sortierung jedoch die Groß- / Kleinschreibung der Spalte ändert, kann dies folgende Auswirkungen haben:

- Die Reihenfolge, wenn eine Klausel ORDER BY ebenfalls vorhanden ist und diese Spalte umfasst
- Gruppierung, wenn die Spalte Teil einer Klausel GROUP BY ist
- Die abgerufenen Zeilen (und damit die Gesamtzahl der Zeilen in der Ergebnismenge), wenn DISTINCT verwendet wird

Beispiele für SELECT-Abfragen mit verschiedenen Arten von Spaltenlisten

Ein einfaches SELECT, das nur Spaltennamen verwendet:

```
select cust_id, cust_name, phone
  from customers
 where city = 'London'
```

Eine Abfrage mit einem Verkettungsausdruck und einem Funktionsaufruf in der Spaltenliste:

```
select 'Mr./Mrs. ' || lastname, street, zip, upper(city)
  from contacts
 where date_last_purchase(id) = current_date
```

Eine Abfrage mit zwei Unterabfragen

```
select p.fullname,
       (select name from classes c where c.id = p.class) as class,
       (select name from mentors m where m.id = p.mentor) as mentor
  from pupils p
```

Die folgende Abfrage führt dasselbe wie das vorherige mit Joins statt Unterabfragen durch:

```
select p.fullname,
       c.name as class,
       m.name as mentor
  from pupils p
 join classes c on c.id = p.class
 join mentors m on m.id = p.mentor
```

Diese Abfrage verwendet ein CASE-Konstrukt, um die korrekte Anrede zu ermitteln, z.B. für das Senden von E-Mails an eine Person:

```
select case upper(sex)
       when 'F' then 'Mrs.'
       when 'M' then 'Mr.'
       else ''
       end as title,
  lastname,
  address
  from employees
```

Abfrage einer auswählbaren gespeicherten Prozedur:

```
select * from interesting_transactions(2010, 3, 'S')
 order by amount
```

Auswahl aus Spalten einer abgeleiteten Tabelle. Eine abgeleitete Tabelle ist eine eingeklammerte SELECT-Anweisung, deren Ergebnismenge in einer einschließenden Abfrage so verwendet wird, als wäre sie eine reguläre Tabelle oder Sicht. Die abgeleitete Tabelle ist hier fett dargestellt:

```
select fieldcount,
       count(relation) as num_tables
  from (select r.rdb$relation_name as relation,
             count(*) as fieldcount
        from rdb$relations r)
```

```

        join rdb$relation_fields rf
        on rf.rdb$relation_name = r.rdb$relation_name
    group by relation)
group by fieldcount
    
```

Die Zeit durch eine Kontextvariable abfragen (CURRENT_TIME):

```
select current_time from rdb$database
```

Für diejenigen, die mit RDB\$DATABASE nicht vertraut sind: Dies ist eine Systemtabelle, die in allen Firebird-Datenbanken vorhanden ist und nur genau eine Zeile enthält. Obwohl es für diesen Zweck nicht erstellt wurde, ist es unter Firebird-Programmierern Standard geworden, diese Tabelle abzufragen, wenn Sie „aus nichts“ abfragen möchten, d.h. wenn Sie Daten benötigen, die nicht an eine Tabelle oder Ansicht gebunden sind, diese aber über Ausdrücke in den Ausgabespalten abgeleitet werden können. Ein anderes Beispiel ist:

```
select power(12, 2) as twelve_squared, power(12, 3) as twelve_cubed
from rdb$database
```

Zum Schluss ein Beispiel, in dem Sie aussagekräftige Informationen aus RDB\$DATABASE selbst ermitteln:

```
select rdb$character_set_name from rdb$database
```

Wie Sie vielleicht schon vermutet haben, erhalten Sie den Standardzeichensatz der Datenbank.

Siehe auch: [Skalarfunktionen](#), [Aggregatfunktionen](#), [Kontextvariablen](#), [CASE](#), [Unterabfragen](#)

Die FROM-Klausel

Die Klausel FROM gibt die Quelle (n) an, aus der die Daten abgerufen werden sollen. In seiner einfachsten Form ist dies nur eine einzelne Tabelle oder Ansicht. Die Quelle kann jedoch auch eine auswählbare gespeicherte Prozedur, eine abgeleitete Tabelle oder ein allgemeiner Tabellenausdruck sein. Mehrere Quellen können mit verschiedenen Arten von Joins kombiniert werden.

Dieser Abschnitt konzentriert sich auf Single-Source-Selects. [Joins](#) werden in einem der folgenden Abschnitte behandelt.

Syntax:

```

SELECT
    ...
FROM <source>
[<joins>]
[...]

<source> ::= {table
              | view
              | selectable-stored-procedure [(args)]
              | <derived-table>
              | <common-table-expression>}
          [[AS] alias]

<derived-table> ::= (select-statement) [[AS] alias]
                  [(<column-aliases>)]
    
```

```

<common-table-expression>
    ::= WITH [RECURSIVE] <cte-def> [, <cte-def> ...]
       select-statement

<cte-def>
    ::= name [( <column-aliases> )] AS (select-statement)

<column-aliases>
    ::= column-alias [, column-alias ...]
    
```

Tabelle 6.3. Argumente der FROM-Klausel

| Argument | Beschreibung |
|-----------------------------|---|
| table | Name einer Tabelle |
| view | Name einer Ansicht |
| selectable-stored-procedure | Name einer auswählbaren gespeicherten Prozedur |
| args | Auswählbare Argumente für gespeicherte Prozeduren |
| derived table | Abgeleiteter Tabellenabfrageausdruck |
| cte-def | Definition des gemeinsamen Tabellenausdrucks (Common Table Expression, CTE), einschließlich eines „ad hoc“-Namens |
| select-statement | Beliebige SELECT-Anweisung |
| column-aliases | Alias für eine Spalte in einer Relation, CTE oder abgeleitete Tabelle |
| name | Der „ad hoc“-Name für eine CTE |
| alias | Der Alias einer Datenquelle (Tabelle, View, Prozedur, CTE, abgeleitete Tabelle) |

Abfragen einer Tabelle oder Ansicht mit FROM

Bei der Auswahl aus einer einzelnen Tabelle oder Sicht muss die FROM-Klausel nichts mehr als den Namen enthalten. Ein Alias kann nützlich oder sogar notwendig sein, wenn es Unterabfragen gibt, die auf die Haupt-Select-Anweisung verweisen (wie sie es sooft tun — Unterabfragen wie diese werden auch *korrelierte Unterabfragen* genannt).

Beispiele

```

select id, name, sex, age from actors
  where state = 'Ohio'
    
```

```

select * from birds
  where type = 'flightless'
  order by family, genus, species
    
```

```

select firstname,
       middlename,
       lastname,
    
```



```
    date_of_birth,  
    (select name from schools s where p.school = s.id) schoolname  
from pupils p  
where year_started = '2012'  
order by schoolname, date_of_birth
```

Mischen Sie niemals Spaltennamen mit Spaltenaliasnamen!

Wenn Sie einen Alias für eine Tabelle oder eine Sicht angeben, müssen Sie diesen Alias anstelle des Tabellennamens immer verwenden, wenn Sie die Spalten der Relation abfragen (und wo auch immer Sie auf Spalten verweisen, z. B. ORDER BY, GROUP BY) und WHERE-Klauseln.

Richtige Verwendung:

```
SELECT PEARS  
FROM FRUIT
```

```
SELECT FRUIT.PEARS  
FROM FRUIT
```

```
SELECT PEARS  
FROM FRUIT F
```

```
SELECT F.PEARS  
FROM FRUIT F
```

Falsche Verwendung:

```
SELECT FRUIT.PEARS  
FROM FRUIT F
```

Abfragen einer gespeicherten Prozedur mit FROM

Eine *auswählbare gespeicherte Prozedur* ist eine Prozedur, die:

- enthält mindestens einen Ausgabeparameter und
- das Schlüsselwort `SUSPEND` verwendet, damit der Aufrufer die Ausgabezeilen nacheinander abrufen kann, genau so wie bei der Auswahl aus einer Tabelle oder Ansicht.

Die Ausgabeparameter einer auswählbaren gespeicherten Prozedur entsprechen den Spalten einer regulären Tabelle.

Die Abfrage aus einer gespeicherten Prozedur ohne Eingabeparameter entspricht der Abfrage aus einer Tabelle oder Ansicht:

```
select * from suspicious_transactions  
where assignee = 'John'
```

Alle erforderlichen Eingabeparameter müssen nach dem in Klammern angegebenen Prozedurnamen angegeben werden:

```
select name, az, alt from visible_stars('Brugge', current_date, '22:30')  
where alt >= 20  
order by az, alt
```

Werte für optionale Parameter (d.h. Parameter, für die Standardwerte definiert wurden) können weggelassen oder bereitgestellt werden. Wenn Sie diese jedoch nur teilweise angeben, müssen die Parameter, die Sie weglassen, alle am Ende stehen.

Angenommen, die Prozedur `visible_stars` aus dem vorherigen Beispiel hat zwei optionale Parameter: `min_magn` (numerisch (3,1)) und `spectral_class` (varchar (12)). Die folgenden Abfragen sind alle gültig:

```
select name, az, alt from visible_stars('Brugge', current_date, '22:30')
select name, az, alt from visible_stars('Brugge', current_date, '22:30', 4.0)
select name, az, alt from visible_stars('Brugge', current_date, '22:30', 4.0, 'G')
```

Diese jedoch nicht, da es ein „Loch“ in der Parameterliste gibt:

```
select name, az, alt from visible_stars('Brugge', current_date, '22:30', 'G')
```

Ein Alias für eine auswählbare gespeicherte Prozedur wird *nach* der Parameterliste angegeben:

```
select number,
       (select name from contestants c where c.number = gw.number)
from get_winners('#34517', 'AMS') gw
```

Wenn Sie auf einen Ausgabeparameter („column“) verweisen, indem Sie ihn mit dem vollständigen Prozedurnamen qualifizieren, sollte die Parameterliste weggelassen werden:

```
select number,
       (select name from contestants c where c.number = get_winners.number)
from get_winners('#34517', 'AMS')
```

Siehe auch: [Gespeicherte Prozeduren, CREATE PROCEDURE](#)

Abfragen aus einer abgeleiteten Tabelle mittels FROM

Eine abgeleitete Tabelle ist eine gültige SELECT-Anweisung, die in Klammern eingeschlossen ist, optional gefolgt von einem Tabellenalias und / oder Spaltenaliasnamen. Die Ergebnismenge der Anweisung fungiert als virtuelle Tabelle, die die umschließende Anweisung abfragen kann.

Syntax:

```
(select-query)
  [[AS] derived-table-alias]
  [(<derived-column-aliases>)]

<derived-column-aliases> := column-alias [, column-alias ...]
```

Die von diesem „SELECT FROM(SELECT FROM ...)“-Stil der Anweisung zurückgegebene Datenmenge ist eine virtuelle Tabelle, die innerhalb der umschließenden Anweisung abgefragt werden kann, als wäre sie eine normale Tabelle oder Ansicht.

Beispiel mit einer abgeleiteten Tabelle

Die abgeleitete Tabelle in der folgenden Abfrage gibt die Liste der Tabellennamen in der Datenbank und die Anzahl der Spalten in jeder Datenbank zurück. Eine „Drill-Down“-Abfrage für die abgeleitete Tabelle gibt die Anzahl der Felder und die Anzahl der Tabellen mit jeder Feldanzahl zurück:

```
SELECT
    FIELD COUNT,
    COUNT(RELATION) AS NUM_TABLES
FROM (SELECT
        R.RDB$RELATION_NAME RELATION,
        COUNT(*) AS FIELD COUNT
    FROM RDB$RELATIONS R
        JOIN RDB$RELATION_FIELDS RF
        ON RF.RDB$RELATION_NAME = R.RDB$RELATION_NAME
    GROUP BY RELATION)
GROUP BY FIELD COUNT
```

Ein triviales Beispiel, das demonstriert, wie der Alias einer abgeleiteten Tabelle und die Liste der Spaltenalias (beide optional) verwendet werden können:

```
SELECT
    DBINFO.DESCR, DBINFO.DEF_CHARSET
FROM (SELECT *
    FROM RDB$DATABASE) DBINFO
    (DESCR, REL_ID, SEC_CLASS, DEF_CHARSET)
```

Mehr über abgeleitete Tabellen

Abgeleitete Tabellen können

- verschachtelt werden
- Unions sein und in Unions verwendet werden
- Aggregatfunktionen, Unterabfragen und Joins enthalten
- in Aggregatfunktionen, Unterabfragen und Joins verwendet werden
- Aufrufe an abfragbare gespeicherte Prozeduren oder Abfragen auf diese sein
- WHERE-, ORDER BY- und GROUP BY-Klauseln, FIRST, SKIP- oder ROWS-Direktiven, usw enthalten.

Weiter gilt:

- Jede Spalte in einer abgeleiteten Tabelle muss einen Namen haben. Wenn sie keinen Namen hat, z. B. wenn es sich um einen Konstanten- oder einen Laufzeitausdruck handelt, sollte ihr ein Alias zugewiesen werden, entweder auf reguläre Weise oder durch Einfügen in die Liste der Spaltenalias in der Spezifikation der abgeleiteten Tabelle.
 - *Die Liste der Spaltenalias ist optional, aber falls vorhanden, muss sie einen Alias für jede Spalte in der abgeleiteten Tabelle enthalten*
- Der Optimierer kann abgeleitete Tabellen sehr effektiv verarbeiten. Wenn eine abgeleitete Tabelle jedoch in einem Inner Join enthalten ist und eine Unterabfrage enthält, kann der Optimierer keine Join-Reihenfolge verwenden.

Ein nützlicheres Beispiel

Angenommen, wir haben eine Tabelle COEFFS, die die Koeffizienten einer Anzahl von quadratischen Gleichungen enthält, die wir lösen müssen. Diese wurde folgendermaßen definiert:

```
create table coeffs (
  a double precision not null,
  b double precision not null,
  c double precision not null,
  constraint chk_a_not_zero check (a <> 0)
)
```

Abhängig von den Werten für a , b und c kann jede Gleichung null, eine oder zwei Lösungen haben. Es ist möglich, diese Lösungen mit einer einstufigen Abfrage für die Tabelle COEFFS zu finden, aber der Code sieht ziemlich unordentlich aus und mehrere Werte (wie die Diskriminante) müssen mehrmals pro Zeile berechnet werden. Eine abgeleitete Tabelle kann dabei helfen, die Dinge sauber zu halten:

```
select
  iif (D >= 0, (-b - sqrt(D)) / denom, null) sol_1,
  iif (D > 0, (-b + sqrt(D)) / denom, null) sol_2
from
  (select b, b*b - 4*a*c, 2*a from coeffs) (b, D, denom)
```

Wenn wir die Koeffizienten neben den Lösungen anzeigen möchten (was keine schlechte Idee ist), können wir die Abfrage folgendermaßen ändern:

```
select
  a, b, c,
  iif (D >= 0, (-b - sqrt(D)) / denom, null) sol_1,
  iif (D > 0, (-b + sqrt(D)) / denom, null) sol_2
from
  (select a, b, c, b*b - 4*a*c as D, 2*a as denom
   from coeffs)
```

Beachten Sie, dass, während die erste Abfrage eine Liste mit Spaltenaliasen für die abgeleitete Tabelle verwendet, nutzt die zweite Abfrage intern hinzugefügte Alias, wo diese benötigt werden. Beide Methoden funktionieren, solange jede Spalte einen Namen hat.

Abfragen einer CTE mittels FROM

Ein allgemeiner Tabellenausdruck (Common Table Expression) oder CTE ist eine komplexere Variante der abgeleiteten Tabelle, aber auch leistungsfähiger. Eine Präambel, beginnend mit dem Schlüsselwort WITH, definiert einen oder mehrere benannte CTE mit jeweils einer optionalen Spalten-Alias-Liste. Die Hauptabfrage, die der Präambel folgt, kann dann auf diese CTE wie normale Tabellen oder Ansichten zugreifen. Sobald die Hauptabfrage ausgeführt wurde, werden die CTEs nicht mehr betrachtet.

Für eine vollständige Beschreibung der CTEs, beachten Sie bitte den Abschnitt *Common Table Expressions* („WITH ... AS ... SELECT“).

Das folgende ist eine andere Variante unseres abgeleiteten Tabellenbeispiels als CTE:

```
with vars (b, D, denom) as (
  select b, b*b - 4*a*c, 2*a from coeffs
)
select
  iif (D >= 0, (-b - sqrt(D)) / denom, null) sol_1,
  iif (D > 0, (-b + sqrt(D)) / denom, null) sol_2
from vars
```

Abgesehen von der Tatsache, dass die Berechnungen, die zuerst gemacht werden müssen, jetzt am Anfang stehen, ist dies keine große Verbesserung gegenüber der abgeleiteten Tabellenversion. Aber wir können jetzt auch die doppelte Berechnung von \sqrt{D} für jede Zeile eliminieren:

```
with vars (b, D, denom) as (
  select b, b*b - 4*a*c, 2*a from coeffs
),
vars2 (b, D, denom, sqrtD) as (
  select b, D, denom, iif (D >= 0, sqrt(D), null) from vars
)
select
  iif (D >= 0, (-b - sqrtD) / denom, null) sol_1,
  iif (D > 0, (-b + sqrtD) / denom, null) sol_2
from vars2
```

Der Code ist jetzt etwas komplizierter, könnte aber effizienter ausgeführt werden (abhängig davon, was mehr Zeit benötigt: die Ausführung der Funktion SQRT oder die Übergabe der Werte von b , D und $denom$ durch eine weitere CTE). Übrigens hätten wir das Gleiche mit abgeleiteten Tabellen tun können, aber das würde Verschachtelung bedeuten.

Siehe auch: *Common Table Expressions („WITH ... AS ... SELECT“)*.

Joins

Joins kombinieren Daten aus zwei Quellen zu einem einzelnen Satz. Dies wird durch einen Zeile-für-Zeilen-Vergleich durchgeführt und beinhaltet üblicherweise eine *Join-Bedingung*, um festzulegen welche Zeilen zusammengeführt werden sollen und im Ergebnisdatensatz erscheinen sollen. Es gibt unterschiedliche Arten (INNER, OUTER) und Klassen (qualifiziert, natürlich, etc.), jede mit eigener Syntax und Regeln.

Da Joins verkettet werden können, können die an einem Join beteiligten Datensätze selbst verbundene Sets sein.

Syntax:

```
SELECT
  ...
FROM <source>
[<joins>]
[...]
```

<source> ::= {table
| view
| selectable-stored-procedure [(args)]
| derived-table
| common-table-expression}
[[AS] alias]

<joins> ::= <join> [<join> ...]

<join> ::= [<join-type>] JOIN <source> <join-condition>
| NATURAL [<join-type>] JOIN <source>
| {CROSS JOIN | ,} <source>

<join-type> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]

<join-condition> ::= ON condition | USING (column-list)

Tabelle 6.4. Argumente für JOIN-Klauseln

| Argument | Beschreibung |
|-----------------------------|--|
| table | Name einer Tabelle |
| view | Name einer Ansicht |
| selectable-stored-procedure | Name einer auswählbaren gespeicherten Prozedur |
| args | Wählbare gespeicherte Prozedur-Eingangsparameter |
| derived-table | Referenz, namentlich, auf eine abgeleitete Tabelle |
| common-table-expression | Verweis auf einen gemeinsamen Tabellenausdruck (CTE) |
| alias | Ein Alias für eine Datenquelle (Tabelle, View, Prozedur, CTE, abgeleitete Tabelle) |
| condition | Join-Bedingung (Kriterium) |
| column-list | Die Liste der Spalten, die für einen Equi-Join verwendet werden |

Inner vs. Outer Joins

Ein Join kombiniert immer Datenzeilen aus zwei Mengen (normalerweise als die linke Menge und die rechte Menge bezeichnet). Standardmäßig werden nur Zeilen in die Ergebnismenge aufgenommen, die die Join-Bedingung erfüllen (d.h. wenn bei der Join-Bedingung mindestens eine Zeile in der anderen Gruppe übereinstimmt). Dieser Standardtyp von Join wird als *Inner Join* bezeichnet. Angenommen, wir haben die folgenden zwei Tabellen:

Tabelle A:

| ID | S |
|-----|----------------|
| 87 | Just some text |
| 235 | Silence |

Tabelle B:

| CODE | X |
|------|---------|
| -23 | 56.7735 |
| 87 | 416.0 |

Wenn wir diese Tabellen wie folgt verbinden:

```
select *
  from A
 join B on A.id = B.code
```

then the result set will be:

| ID | S | CODE | X |
|----|----------------|------|-------|
| 87 | Just some text | 87 | 416.0 |

Die erste Zeile von A wurde mit der zweiten Zeile von B verbunden, weil sie zusammen die Bedingung „A.id = B.code“ erfüllten. Die anderen Zeilen aus den Quellentabellen haben keine Übereinstimmung in der entgegengesetzten Menge und sind daher nicht in der Verknüpfung enthalten. Denken Sie daran, dies ist ein INNER Join. Wir können diese Tatsache explizit machen, indem wir schreiben:

```
select *
  from A
 inner join B on A.id = B.code
```

Da jedoch

`<datenbank>INNER</datenbank>`

die Standardeinstellung ist, wird dies selten durchgeführt.

Es ist durchaus möglich, dass eine Zeile im linken Satz mit mehreren Zeilen vom rechten Satz übereinstimmt oder umgekehrt. In diesem Fall sind alle diese Kombinationen enthalten und wir können Ergebnisse erhalten wie:

| ID | S | CODE | X |
|-----|------------------|------|---------|
| 87 | Just some text | 87 | 416.0 |
| 87 | Just some text | 87 | -1.0 |
| -23 | Don't know | -23 | 56.7735 |
| -23 | Still don't know | -23 | 56.7735 |
| -23 | I give up | -23 | 56.7735 |

Manchmal möchten (oder brauchen) *alle* die Zeilen einer oder beider Quellen in der verbundenen Menge erscheinen, unabhängig davon, ob sie mit einem Datensatz in der anderen Quelle übereinstimmen. An dieser Stelle kommen Outer Joins ins Spiel. Ein Outer Join LEFT enthält alle Datensätze aus dem linken Satz, aber nur übereinstimmende Datensätze aus dem richtigen Satz. In einem RIGHT Outer Join ist es umgekehrt. FULL Outer Joins umfassen alle Datensätze aus beiden Sets. In allen äußeren Joins sind die „Löcher“ (die Stellen, an denen ein eingeschlossener Quelldatensatz keine Übereinstimmung in der anderen Menge hat) mit NULL gefüllt.

Um einen Outer Join zu erstellen, müssen Sie LEFT, RIGHT oder FULL angeben, optional vom Schlüsselwort OUTER gefolgt.

Im Folgenden sind die Ergebnisse der verschiedenen äußeren Joins aufgeführt, wenn sie auf unsere ursprünglichen Tabellen A und B angewendet werden:

```
select *
  from A
```

```
left [outer] join B on A.id = B.code
```

| ID | S | CODE | X |
|-----|----------------|--------|--------|
| 87 | Just some text | 87 | 416.0 |
| 235 | Silence | <null> | <null> |

```
select *
  from A
 right [outer] join B on A.id = B.code
```

| ID | S | CODE | X |
|--------|----------------|------|---------|
| <null> | <null> | -23 | 56.7735 |
| 87 | Just some text | 87 | 416.0 |

```
select *
  from A
 full [outer] join B on A.id = B.code
```

| ID | S | CODE | X |
|--------|----------------|--------|---------|
| <null> | <null> | -23 | 56.7735 |
| 87 | Just some text | 87 | 416.0 |
| 235 | Silence | <null> | <null> |

Qualifizierte Joins

Qualifizierte Joins geben Bedingungen für das Kombinieren von Zeilen an. Dies geschieht entweder explizit in einer ON-Klausel oder implizit in einer USING-Klausel.

Syntax:

```
<qualified-join> ::= [<join-type>] JOIN <source> <join-condition>
<join-type>      ::= INNER | {LEFT | RIGHT | FULL} [OUTER]
<join-condition> ::= ON condition | USING (column-list)
```

Joins mit expliziter Bedingung

Die meisten qualifizierten Joins haben eine Klausel ON mit einer expliziten Bedingung, bei der es sich um einen beliebigen gültigen booleschen Ausdruck handeln kann, der jedoch normalerweise einen Vergleich zwischen den beiden beteiligten Quellen beinhaltet.

Häufig ist die Bedingung ein Gleichheitstest (oder eine Anzahl von AND-verknüpften Gleichheitstests) unter Verwendung des Operators „=“. Joins wie diese heißen *Equi-Joins*. (Die Beispiele im Abschnitt über innere und äußere Verknüpfung waren *Equi-Joins*.)

Beispiele für Joins mit expliziter Bedingung:

```
/* Wählen Sie alle Detroit-Kunden aus, die 2013 einen
   Kauf getätigt haben, zusammen mit den Kaufdetails: */
select * from customers c
  join sales s on s.cust_id = c.id
 where c.city = 'Detroit' and s.year = 2013
```

```
/* Dasselbe wie oben, aber auch nicht kaufende Kunden: */
select * from customers c
  left join sales s on s.cust_id = c.id
 where c.city = 'Detroit' and s.year = 2013
```

```
/* Wähle für jeden Mann die Frauen aus, die größer sind als er.
   Männer, für die keine solche Frau existiert, sind nicht enthalten. */
select m.fullname as man, f.fullname as woman
 from males m
  join females f on f.height > m.height
```

```
/* Wähle alle Schüler mit ihrer Klasse und ihrem Mentor aus.
   Schüler ohne Mentor sind ebenfalls enthalten. Schüler ohne
   Klasse sind nicht enthalten. */
select p.firstname, p.middlename, p.lastname,
       c.name, m.name
 from pupils p
  join classes c on c.id = p.class
  left join mentors m on m.id = p.mentor
```

Joins für benannte Spalten

Equi-Joins vergleichen häufig Spalten, die in beiden Tabellen denselben Namen haben. Wenn dies der Fall ist, können wir auch den zweiten Typ von qualifiziertem Join verwenden: die *Joins für benannte Spalten*.

Anmerkung

Joins für benannte Spalten werden in Dialekt 1 nicht unterstützt.

Joins für benannte Spalten besitzen eine USING-Klausel, welche nur die Spaltennamen enthält. Anstelle dieser Variante:

```
select * from flotsam f
  join jetsam j
 on f.sea = j.sea
 and f.ship = j.ship
```

können wir auch diese schreiben:

```
select * from flotsam
  join jetsam using (sea, ship)
```

welche deutlich kürzer ist. Der Ergebnissatz ist etwas anders. The result set is a little—zumindest bei der Verwendung von „SELECT *“:

- Der Join mit expliziter Bedingung—mit der ON-Klausel— wird jede der Spalten SEA und SHIP zweimal enthalten: einmal für Tabelle FLOTSAM und einmal für Tabelle JETSAM. Offensichtlich werden sie die gleichen Werte haben.
- Der Join für benannte Spalten—mit der USING-Klausel—enthält diese Spalten nur einmal.

Wenn Sie alle Spalten in der Ergebnismenge der benannten Spalten verknüpfen möchten, richten Sie Ihre Abfrage wie folgt ein:

```
select f.*, j.*
   from flotsam f
   join jetsam j using (sea, ship)
```

Dadurch erhalten Sie genau das gleiche Ergebnis wie beim Join der expliziten Bedingung.

Für einen Join mit benannter Spalte vom Typ OUTER gibt es eine zusätzliche Wendung, wenn „SELECT *“ oder ein nicht qualifizierter Spaltenname aus der USING-Liste verwendet wird:

Wenn eine Zeile aus einer Quellgruppe keine Übereinstimmung in der anderen enthält, muss sie dennoch aufgrund der LEFT-, RIGHT- oder FULL-Direktive enthalten sein. Die zusammengeführte Spalte im zusammgeführten Satz erhält den Wert nicht-NULL. Das ist soweit gut, aber jetzt können Sie nicht sagen, ob dieser Wert aus der linken, rechten oder beiden Mengen stammt. Dies kann besonders trügerisch sein, wenn der Wert von der rechten Seite stammt, weil „*“ immer kombinierte Spalten im linken Teil zeigt—selbst im Falle eines RIGHT Join.

Ob dies ein Problem ist oder nicht, hängt von der Situation ab. Wenn ja, benutzen Sie die „a.*-, b.*“-Ansatz wie oben gezeigt, mit a und b als Namen oder Alias der beiden Quellen. Oder noch besser, vermeiden Sie „*“ insgesamt in Ihren seriösen Abfragen und qualifizieren Sie alle Spaltennamen in verbundenen Mengen. Dies hat den zusätzlichen Vorteil, dass Sie gezwungen sind, darüber nachzudenken, welche Daten Sie abrufen möchten und woher.

Es liegt in Ihrer Verantwortung sicherzustellen, dass die Spaltennamen in der USING-Liste kompatible Typen zwischen den beiden Quellen sind. Wenn die Typen kompatibel aber nicht gleich sind, konvertiert die Engine sie in den Typ mit dem breitesten Wertebereich, bevor sie die Werte vergleicht. Dies ist auch der Datentyp der zusammengeführten Spalte, die in der Ergebnismenge angezeigt wird, wenn „SELECT *“ oder der nicht qualifizierte Spaltenname verwendet wird. Qualifizierte Spalten behalten ihren ursprünglichen Datentyp immer bei.

Natürliche Joins

Greift man die Idee der benannten Spalten auf und geht noch einen Schritt weiter, führt ein *natürlicher Join* einen automatischen Equi-Join für alle Spalten durch, die in der linken und rechten Tabelle den gleichen Namen haben. Die Datentypen dieser Spalten müssen kompatibel sein.

Anmerkung

Natürliche Joins werden in Dialekt 1-Datenbanken nicht unterstützt.

Syntax:

```
<natural-join> ::= NATURAL [<join-type>] JOIN <source>
<join-type>   ::= INNER | {LEFT | RIGHT | FULL} [OUTER]
```

Gegeben sind diese beiden Tabellen

```
create table TA (  
  a bigint,  
  s varchar(12),  
  ins_date date  
)
```

```
create table TB (  
  a bigint,  
  descr varchar(12),  
  x float,  
  ins_date date  
)
```

Ein natürlicher Join auf TA und TB würde die Spalten *a* und *ins_date* einbeziehen, und die folgenden zwei Anweisungen würden die gleiche Wirkung haben:

```
select * from TA  
  natural join TB
```

```
select * from TA  
  join TB using (a, ins_date)
```

Wie alle Joins sind natürliche Joins standardmäßig innere Joins, die Sie jedoch durch Angabe von LEFT, RIGHT oder FULL vor dem Schlüsselwort JOIN in äußere Joins umwandeln können.

Vorsicht: Wenn in den beiden Quellbeziehungen keine Spalten mit demselben Namen vorhanden sind, wird ein CROSS JOIN ausgeführt. Wir kommen in einer Minute zu dieser Art von Join.

Eine Anmerkung zur Gleichheit

Wichtig

Diese Notiz über Gleichheits- und Ungleichheitsoperatoren gilt überall in der Firebird SQL-Sprache, nicht nur unter JOIN-Bedingungen.

Der Operator „="“, welcher explizit für diverse bedingte Joins und implizit in Joins mit benannten Spalten und natürlichen Joins verwendet wird, vergleicht nur Werte mit Werten. Nach dem SQL-Standard gilt, dass NULL kein Wert ist und somit zwei NULLen weder identisch noch unidentisch zueinander sind. Wenn Sie NULL benötigen, um in einem Join übereinzustimmen, verwenden Sie den Operator IS NOT DISTINCT FROM. Dieser Operator gibt "true" zurück, wenn die Operanden denselben Wert haben *oder* wenn sie beide NULL sind.

```
select *  
  from A join B  
  on A.id is not distinct from B.code
```

In den —extrem seltenen— Fällen, in denen Sie im Join auf die *in*-Gleichheit prüfen möchten, verwenden Sie IS DISTINCT FROM, nicht „<>“, falls Sie NULL von anderen Werten unterscheiden müssen und zwei NULLen als gleich betrachtet werden sollen:

```
select *  
  from A join B  
  on A.id is distinct from B.code
```

Cross Joins

Ein Cross Join erzeugt das vollständige Set-Produkt der beiden Datenquellen. Dies bedeutet, dass jede Zeile in der linken Quelle mit jeder Zeile in der rechten Quelle übereinstimmt.

Syntax:

```
<cross-join> ::= {CROSS JOIN | ,} <source>
```

Bitte beachten Sie, dass die Kommasyntax veraltet ist! Es wird nur unterstützt, um Legacy-Code zu erhalten, und wird möglicherweise in einer zukünftigen Version verschwinden.

Das Zusammenführen von zwei Sätzen ist gleichbedeutend damit, dass sie sich einer Tautologie anschließen (eine Bedingung, die immer wahr ist). Die folgenden beiden Aussagen haben den gleichen Effekt:

```
select * from TA
  cross join TB
```

```
select * from TA
  join TB on 1 = 1
```

Cross Joins sind innere Joins, da sie nur übereinstimmende Datensätze enthalten - dies ergibt sich daraus, dass *jeder*-Eintrag übereinstimmt! Ein äußerer Cross Join würde, falls vorhanden, dem Ergebnis nichts hinzufügen, weil die äußeren Joins keine übereinstimmenden Datensätze sind und diese nicht in Cross Joins existieren.

Cross Joins sind selten nützlich, außer wenn Sie alle möglichen Kombinationen von zwei oder mehr Variablen auflisten möchten. Angenommen, Sie verkaufen ein Produkt in verschiedenen Größen, Farben und Materialien. Wenn diese Variablen jeweils in einer eigenen Tabelle aufgeführt sind, gibt diese Abfrage alle Kombinationen zurück:

```
select m.name, s.size, c.name
  from materials m
  cross join sizes s
  cross join colors c
```

Mehrdeutige Feldnamen in Joins

Firebird weist unqualifizierte Feldnamen in einer Abfrage zurück, wenn diese Feldnamen in mehr als einem Datensatz vorhanden sind, der an einem Join beteiligt ist. Dies gilt sogar für innere Equi-Joins, bei denen der Feldname in der ON-Klausel so aussieht:

```
select a, b, c
  from TA
  join TB on TA.a = TB.a
```

Es gibt eine Ausnahme zu dieser Regel: Bei Joins mit benannten Spalten und natürlichen Joins kann der nicht qualifizierte Feldname einer Spalte, die am Matching-Prozess teilnimmt, legal verwendet werden und verweist auf die zusammengeführte Spalte mit demselben Namen. Für Joins mit benannten Spalten sind dies die in der USING-Klausel aufgelisteten Spalten. Bei natürlichen Joins sind dies die Spalten, die in beiden Relationen denselben Namen haben. Aber bitte beachten Sie noch einmal, dass insbesondere in Outer Joins der reine *Spaltenname* nicht immer dasselbe ist wie *links.Spaltenname* oder *rechts.Spaltenname*. Die Typen kön-

nen sich unterscheiden, und eine der qualifizierten Spalten kann NULL sein, während die andere nicht. In diesem Fall kann der Wert in der zusammengeführten, nicht qualifizierten Spalte die Tatsache maskieren, dass einer der Quellwerte nicht vorhanden ist.

Joins mit gespeicherten Prozeduren

Wenn eine Verknüpfung mit einer gespeicherten Prozedur durchgeführt wird, die nicht über Eingabeparameter mit anderen Datenströmen korreliert, gibt es keine Kuriositäten. Wenn es Korrelationen *gibt*, zeigt sich eine unangenehme Eigenart. Das Problem ist, dass der Optimierer sich selbst jede Möglichkeit nimmt, die Beziehungen der Eingabeparameter der Prozedur zu den Feldern in den anderen Datenströmen zu bestimmen:

```
SELECT *
FROM MY_TAB
JOIN MY_PROC(MY_TAB.F) ON 1 = 1
```

Hier wird die Prozedur ausgeführt, bevor ein einzelner Datensatz aus der Tabelle MY_TAB abgerufen wurde. Der Fehler `isc_no_cur_rec_error` (*kein aktueller Datensatz für die Abrufoperation*) wird ausgelöst, wodurch die Ausführung unterbrochen wird.

Die Lösung besteht darin, eine Syntax zu verwenden, die die Join-Reihenfolge *explizit* angibt:

```
SELECT *
FROM MY_TAB
LEFT JOIN MY_PROC(MY_TAB.F) ON 1 = 1
```

Dies erzwingt, dass die Tabelle vor der Prozedur gelesen wird und alles ordnungsgemäß funktioniert.

Tipp

Diese Eigenart wurde als Fehler im Optimierer erkannt und wird in der nächsten Version von Firebird behoben.

Die WHERE-Klausel

Die WHERE-Klausel dient dazu, die zurückgegebenen Zeilen auf diejenigen zu beschränken, an denen der Aufrufer interessiert ist. Die Bedingung nach dem Schlüsselwort WHERE kann so einfach sein wie „Anzahl = 3“ oder ein mehrschichtiger, geschachtelter Ausdruck, der Unterabfragen, Prädikate, Funktionsaufrufe, mathematische und logische Operatoren, Kontextvariablen und mehr enthält.

Die Bedingung in der WHERE-Klausel wird häufig als die *Suchbedingung*, der *Suchausdruck* oder einfach die *Suche* bezeichnet.

In DSQL und ESQL kann der Suchausdruck Parameter enthalten. Dies ist nützlich, wenn eine Abfrage mehrmals mit unterschiedlichen Eingabewerten wiederholt werden muss. In der SQL-Zeichenfolge, die an den Server übergeben wird, werden Fragezeichen als Platzhalter für die Parameter verwendet. Sie heißen *Positionsparameter*, weil sie nur durch ihre Position in der Zeichenfolge voneinander getrennt werden können. Konnektivitätsbibliotheken unterstützen oft *benannte Parameter* der Form `:id`, `:amount`, `:a` usw. Diese sind benutzerfreundlicher; Die Bibliothek sorgt dafür, dass die benannten Parameter in Positionsparameter übersetzt werden, bevor die Anweisung an den Server übergeben wird.

Die Suchbedingung kann auch lokale (PSQL) oder Host (ESQL)-Variablenamen enthalten, denen ein Doppelpunkt vorangestellt ist.

Syntax:

```
SELECT ...
  FROM ...
  [...]
 WHERE <search-condition>
  [...]

<search-condition> ::= Ein boolescher Ausdruck, der
                    TRUE, FALSE oder möglicherweise UNKNOWN (NULL)
                    zurückgibt
```

Nur die Zeilen, für die die Suchbedingung TRUE ergibt, sind in der Ergebnismenge enthalten. Seien Sie vorsichtig mit möglichen NULL-Ergebnissen: Wenn Sie einen NULL-Ausdruck mit NOT negieren, ist das Ergebnis immer NULL und die Zeile wird nicht berücksichtigt. Dies wird in einem der folgenden Beispiele demonstriert.

Beispiele

```
select genus, species from mammals
  where family = 'Felidae'
  order by genus
```

```
select * from persons
  where birthyear in (1880, 1881)
     or birthyear between 1891 and 1898
```

```
select name, street, borough, phone
  from schools s
  where exists (select * from pupils p where p.school = s.id)
  order by borough, street
```

```
select * from employees
  where salary >= 10000 and position <> 'Manager'
```

```
select name from wrestlers
  where region = 'Europe'
     and weight > all (select weight from shot_putters
                      where region = 'Africa')
```

```
select id, name from players
  where team_id = (select id from teams where name = 'Buffaloes')
```

```
select sum (population) from towns
  where name like '%dam'
     and province containing 'land'
```

```
select password from usertable
  where username = current_user
```

Das folgende Beispiel zeigt, was passieren kann, wenn die Suchbedingung auf NULL ausgewertet wird.

Angenommen, Sie haben eine Tabelle mit den Namen einiger Kinder und der Anzahl der Murmeln, die sie besitzen. Zu einem bestimmten Zeitpunkt enthält die Tabelle diese Daten:

| CHILD | MARBLES |
|----------|---------|
| Anita | 23 |
| Bob E. | 12 |
| Chris | <null> |
| Deirdre | 1 |
| Eve | 17 |
| Fritz | 0 |
| Gerry | 21 |
| Hadassah | <null> |
| Isaac | 6 |

Zuerst beachten Sie bitte den Unterschied zwischen NULL und 0: Fritz ist *bekannt* dafür überhaupt keine Murmeln zu haben, Chris' und Hadassahs Murmelanzahlen unbekannt.

Nun, wenn Sie diese SQL-Anweisung ausgeben:

```
select list(child) from marbletable where marbles > 10
```

Sie werden die Namen Anita, Bob E., Eve und Gerry bekommen. Diese Kinder haben alle mehr als 10 Murmeln.

Wenn Sie den Ausdruck negieren:

```
select list(child) from marbletable where not marbles > 10
```

Deirdre, Fritz und Isaac sind an der Reihe. Chris und Hadassah sind nicht enthalten, weil nicht *bekannt* ist, dass sie zehn oder weniger Murmeln besitzen. Sollten Sie diese letzte Abfrage ändern in:

```
select list(child) from marbletable where marbles <= 10
```

wird das Ergebnis immer noch dasselbe sein, weil der Ausdruck `NULL <= 10` nun UNKNOWN ergibt. Das ist nicht dasselbe wie TRUE, also sind Chris und Hadassah nicht aufgelistet. Wenn Sie sie mit den „armen“-Kindern anzeigen möchten, ändern Sie die Abfrage in:

```
select list(child) from marbletable where marbles <= 10 or marbles is null
```

Jetzt wird die Suchbedingung für Chris und Hadassah wahr, weil „`marbles is null`“ gibt in diesem Fall offensichtlich TRUE zurück. Tatsächlich kann die Suchbedingung jetzt für niemanden

<konstant>NULL</konstant>
sein.

Zuletzt zwei Beispiele für SELECT-Abfragen mit Parametern in der Suche. Es hängt von der Anwendung ab, wie Sie Abfrageparameter definieren sollten und selbst wenn es überhaupt möglich ist. Beachten Sie, dass Abfragen wie diese nicht sofort ausgeführt werden können: Sie müssen zuerst *vorbereitet* (prepared) sein. Nachdem eine parametrisierte Abfrage vorbereitet wurde, kann der Benutzer (oder der Aufrufcode) Werte für die Parameter bereitstellen und sie mehrmals ausführen lassen, wobei vor jedem Aufruf neue Werte eingegeben werden. Wie die Werte eingegeben werden und die Ausführung gestartet wird, ist Sache der Anwendung. In einer GUI-

Umgebung gibt der Benutzer die Parameterwerte in der Regel in ein oder mehrere Textfelder ein und klickt dann auf die Schaltfläche „Ausführen“ oder „Aktualisieren“.

```
select name, address, phone from stores
where city = ? and class = ?
```

```
select * from pants
where model = :model and size = :size and color = :col
```

Die letzte Abfrage kann nicht direkt an die Engine übergeben werden. Die Anwendung muss sie zuerst in das andere Format konvertieren und benannte Parameter den Positionsparametern zuordnen.

Die GROUP BY-Klausel

GROUP BY führt Ausgangszeilen mit derselben Kombination von Werten in der Elementliste in eine einzelne Zeile zusammen. Aggregatfunktionen in der Auswahlliste werden für jede Gruppe einzeln und nicht für das gesamte Dataset angewendet.

Wenn die Auswahlliste nur Aggregatspalten oder allgemeiner Spalten enthält, deren Werte nicht von einzelnen Zeilen in der zugrunde liegenden Menge abhängen, ist GROUP BY optional. Wenn sie weggelassen wird, besteht die endgültige Ergebnismenge aus einer einzelnen Zeile (vorausgesetzt, dass mindestens eine aggregierte Spalte vorhanden ist).

Wenn die Auswahlliste sowohl Aggregatspalten als auch Spalten enthält, deren Werte je Zeile variieren können, wird die Klausel GROUP BY obligatorisch.

Syntax:

```
SELECT ... FROM ...
  GROUP BY <grouping-item> [, <grouping-item> ...]
  [HAVING <grouped-row-condition>]
  ...

<grouping-item> ::= <non-aggr-select-item>
                  | <non-aggr-expression>

<non-aggr-select-item> ::= column-copy
                          | column-alias
                          | column-position
```

Tabelle 6.5. Argumente der GROUP BY-Klausel

| Argument | Beschreibung |
|---------------------|--|
| non-aggr-expression | Jeder nicht aggregierende Ausdruck, der nicht in der SELECT-Liste enthalten ist, d.h. nicht ausgewählte Spalten aus der Quellenmenge oder Ausdrücke, die überhaupt nicht von den Daten in der Menge abhängen |
| column-copy | Eine Literalkopie aus der SELECT-Liste eines Ausdrucks, der keine Aggregatfunktion enthält |
| column-alias | Der Alias aus der SELECT-Liste eines Ausdrucks (Spalte), der keine Aggregatfunktion enthält |

| Argument | Beschreibung |
|-----------------|--|
| column-position | Die Positionsnummer in der SELECT-Liste eines Ausdrucks (Spalte), der keine Aggregatfunktion enthält |

Eine allgemeine Faustregel besagt, dass jedes nicht aggregierte Element in der SELECT-Liste ebenfalls in der GROUP BY-Liste enthalten sein muss. Sie können dies auf drei Arten tun:

1. Indem der Gegenstand wörtlich aus der Auswahlliste kopiert wird, z.B. „class“ oder „'D:' || upper(doccode)“.
2. Durch Angabe des Spaltenalias, falls vorhanden.
3. Durch Angabe der Spaltenposition als Ganzzahl *literal* zwischen 1 und der Anzahl der Spalten. Ganzzahlwerte, die sich aus Ausdrücken oder Parametersubstitutionen ergeben, sind einfach unveränderlich und werden als solche in der Gruppierung verwendet. Sie werden jedoch keinen Effekt haben, da ihr Wert für jede Zeile gleich ist.

Anmerkung

Wenn Sie nach einer Spaltenposition gruppieren, wird der Ausdruck an dieser Position intern aus der Auswahlliste kopiert. Wenn es sich um eine Unterabfrage handelt, wird diese Unterabfrage in der Gruppierungsphase erneut ausgeführt. Das bedeutet, dass die Gruppierung nach der Spaltenposition, anstatt den Unterabfrageausdruck in der Gruppierungsklausel zu duplizieren, Tastenanschläge und Bytes speichert, dies ist jedoch keine Möglichkeit, Verarbeitungszyklen zu speichern!

Zusätzlich zu den erforderlichen Elementen kann die Gruppierungsliste auch Folgendes enthalten:

- Spalten aus der Quelltable, die nicht in der Auswahlliste enthalten sind, oder Nicht-Aggregat-Ausdrücke, die auf solchen Spalten basieren. Das Hinzufügen solcher Spalten kann die Gruppen weiter unterteilen. Da diese Spalten jedoch nicht in der Auswahlliste enthalten sind, können Sie nicht feststellen, welche aggregierte Zeile mit welchem Wert in der Spalte übereinstimmt. Wenn Sie also an diesen Informationen interessiert sind, fügen Sie auch die Spalte oder den Ausdruck in die Auswahlliste—ein, die Sie wieder zur Regel führt: „Jede Nicht-Aggregat-Spalte in der Auswahlliste muss ebenfalls in der Gruppierungsliste sein“.
- Ausdrücke, die nicht von den Daten in dem zugrunde liegenden Satz abhängen, z. Konstanten, Kontextvariablen, einwertige nicht-korrelierte Subselects usw. Dies wird nur der Vollständigkeit halber erwähnt, da das Hinzufügen solcher Elemente völlig sinnlos ist: Sie beeinflussen die Gruppierung überhaupt nicht. „Harmlose, aber nutzlose“ Elemente wie diese können auch in der Auswahlliste erscheinen, ohne in die Gruppierungsliste kopiert zu werden.

Beispiele

Wenn die Auswahlliste nur Aggregatspalten enthält, ist GROUP BY nicht obligatorisch:

```
select count(*), avg(age) from students
where sex = 'M'
```

Dies wird eine einzelne Zeile zurückgeben, die die Anzahl der männlichen Studenten und deren Durchschnittsalter auflistet. Das Hinzufügen von Ausdrücken, die nicht von Werten in einzelnen Zeilen der Tabelle STUDENTS abhängen, ändert das nicht:

```
select count(*), avg(age), current_date from students
where sex = 'M'
```

Die Zeile wird jetzt eine zusätzliche Spalte haben, die das aktuelle Datum anzeigt, aber ansonsten hat sich nichts Grundlegendes geändert. Eine Klausel GROUP BY ist weiterhin nicht erforderlich.

In beiden obigen Beispielen ist dies jedoch *erlaubt*. Dies ist absolut gültig:

```
select count(*), avg(age) from students
  where sex = 'M'
  group by class
```

und gibt eine Reihe für jede Klasse zurück, in der sich Jungen befinden, die die Anzahl der Jungen und ihr Durchschnittsalter in dieser bestimmten Klasse auflistet. (Wenn Sie auch das Feld `current_date` beibehalten, wird dieser Wert in jeder Zeile wiederholt, was nicht besonders aufregend ist.)

Die obige Abfrage hat jedoch einen großen Nachteil: Sie gibt Ihnen Informationen über die verschiedenen Klassen, aber Sie erfahren nicht, welche Zeile für welche Klasse gilt. Um diese zusätzlichen Informationen zu erhalten, muss die nicht aggregierte Spalte CLASS zur Auswahlliste hinzugefügt werden:

```
select class, count(*), avg(age) from students
  where sex = 'M'
  group by class
```

Jetzt haben wir eine nützliche Abfrage. Beachten Sie, dass durch das Hinzufügen der Spalte CLASS auch die Klausel GROUP BY obligatorisch wird. Wir können diese Klausel nicht mehr löschen, es sei denn, wir entfernen auch CLASS aus der Spaltenliste.

Die Ausgabe unserer letzten Abfrage könnte etwa so aussehen:

| CLASS | COUNT | AVG |
|-------|-------|------|
| 2A | 12 | 13.5 |
| 2B | 9 | 13.9 |
| 3A | 11 | 14.6 |
| 3B | 12 | 14.4 |
| ... | ... | ... |

Die Überschriften „COUNT“ und „AVG“ sind nicht sehr informativ. In einem einfachen Fall wie diesem, könnten Sie damit durchkommen, aber im Allgemeinen sollten Sie aggregierten Spalten einen aussagekräftigen Namen geben, indem wir je einen Alias nutzen:

```
select class,
  count(*) as num_boys,
  avg(age) as boys_avg_age
  from students
  where sex = 'M'
  group by class
```

Wie Sie aus der formalen Syntax der Spaltenliste entnehmen können, ist das Schlüsselwort AS optional.

Wenn Sie weitere nicht aggregierte (oder besser: zeilenabhängige) Spalten hinzufügen, müssen Sie sie auch der Klausel GROUP BY hinzufügen. Zum Beispiel möchten Sie vielleicht die oben genannten Informationen auch für Mädchen sehen; und Sie möchten vielleicht auch zwischen Internats- und Tagesschülern unterscheiden:

```
select class,
```

```

sex,
boarding_type,
count(*) as number,
avg(age) as avg_age
from students
group by class, sex, boarding_type

```

Dies kann zu folgendem Ergebnis führen:

| CLASS | SEX | BOARDING_TYPE | NUMBER | AVG_AGE |
|-------|-----|---------------|--------|---------|
| 2A | F | BOARDING | 9 | 13.3 |
| 2A | F | DAY | 6 | 13.5 |
| 2A | M | BOARDING | 7 | 13.6 |
| 2A | M | DAY | 5 | 13.4 |
| 2B | F | BOARDING | 11 | 13.7 |
| 2B | F | DAY | 5 | 13.7 |
| 2B | M | BOARDING | 6 | 13.8 |
| ... | ... | ... | ... | ... |

Jede Zeile in der Ergebnismenge entspricht einer bestimmten Kombination der Variablen class, sex und boarding type. Die zusammengefassten Ergebnisse—Anzahl und durchschnittliches Alter—sind für jede dieser eher spezifischen Gruppen einzeln angegeben. In einer Abfrage wie dieser sehen Sie keine Gesamtzahl für Jungen als Ganzes oder Tagesschüler als Ganzes. Das ist der Nachteil: Je mehr Nicht-Aggregat-Spalten Sie hinzufügen, desto mehr können Sie sehr spezifische Gruppen bestimmen, aber desto mehr verlieren Sie auch das allgemeine Bild aus den Augen. Natürlich können Sie die „größeren“ Aggregate auch über separate Abfragen erhalten.

HAVING

Genau wie eine WHERE-Klausel die Zeilen in einer Datenmenge auf solche begrenzt, die die Suchbedingung erfüllen, so beschränkt die Unterklasse HAVING die aggregierten Zeilen in einer gruppierten Gruppe. HAVING ist optional und kann nur in Verbindung mit GROUP BY verwendet werden.

Die Bedingung(en) in der HAVING-Klausel können sich beziehen auf:

- Jede aggregierte Spalte in der Auswahlliste. Dies ist die am häufigsten verwendete Alternative.
- Jeder aggregierte Ausdruck, der nicht in der Auswahlliste enthalten ist, aber im Kontext der Abfrage zulässig ist. Dies ist manchmal auch nützlich.
- Eine beliebige Spalte in der Liste GROUP BY. Obwohl dies legal ist, ist es effizienter, diese nicht aggregierten Daten zu einem früheren Zeitpunkt zu filtern: in der Klausel WHERE.
- Ein beliebiger Ausdruck, dessen Wert nicht vom Inhalt des Datensets abhängt (wie eine Konstante oder eine Kontextvariable). Das ist zwar stichhaltig, aber völlig sinnlos, weil es entweder die gesamte Menge unterdrückt oder sie unberührt lässt, basierend auf Bedingungen, die nichts mit der Menge selbst zu tun haben.

Eine HAVING-Klausel kann *nicht* enthalten:

- Nicht aggregierte Spaltenausdrücke, die nicht in der GROUP BY-Liste enthalten sind.
- Spaltenpositionen. Eine Ganzzahl in der HAVING-Klausel ist nur eine Ganzzahl.
- Spaltenalias - nicht einmal wenn sie in der GROUP BY-Klausel vorkommen!

Beispiele

Aufbauend auf unseren früheren Beispielen könnte dies verwendet werden, um kleine Gruppen von Schülern zu überspringen:

```
select class,
       count(*) as num_boys,
       avg(age) as boys_avg_age
from students
where sex = 'M'
group by class
having count(*) >= 5
```

So wählen Sie nur Gruppen mit einem Mindestalter aus:

```
select class,
       count(*) as num_boys,
       avg(age) as boys_avg_age
from students
where sex = 'M'
group by class
having max(age) - min(age) > 1.2
```

Beachten Sie, dass Sie, wenn Sie wirklich an diesen Informationen interessiert sind, diese normalerweise einschließen würden mittels `min(age)` und `max(age) - min(age)` – oder dem Ausdruck „`max(age) - min(age)`“ – auch in der Select-Liste!

Um nur die 3. Klassen einzuschließen:

```
select class,
       count(*) as num_boys,
       avg(age) as boys_avg_age
from students
where sex = 'M'
group by class
having class starting with '3'
```

Besser wäre es, diese Bedingung in die WHERE-Klausel zu verschieben:

```
select class,
       count(*) as num_boys,
       avg(age) as boys_avg_age
from students
where sex = 'M' and class starting with '3'
group by class
```

Die PLAN-Klausel

Die PLAN-Klausel ermöglicht es dem Benutzer, einen Datenabrufplan einzureichen, wodurch der Plan überschrieben wird, den der Optimierer automatisch erstellt hätte.

Syntax:

```

PLAN <plan-expr>

<plan-expr> ::= (<plan-item> [, <plan-item> ...])
              | <sorted-item>
              | <joined-item>
              | <merged-item>

<sorted-item> ::= SORT (<plan-item>)

<joined-item> ::= JOIN (<plan-item>, <plan-item> [, <plan-item> ...])

<merged-item> ::= [SORT] MERGE (<sorted-item>, <sorted-item> [, <sorted-item> ...])

<plan-item> ::= <basic-item> | <plan-expr>

<basic-item> ::= <relation>
               {NATURAL
                | INDEX (<indexlist>)
                | ORDER index [INDEX (<indexlist>)]}

<relation> ::= table
             | view [table]

<indexlist> ::= index [, index ...]
    
```

Tabelle 6.6. Argumente der PLAN-Klausel

| Argument | Beschreibung |
|----------|------------------------------|
| table | Tabellenname oder sein Alias |
| view | Ansichtname |
| index | Indexname |

Jedes Mal, wenn ein Benutzer eine Abfrage an die Firebird-Engine sendet, berechnet der Optimierer eine Datenabrufstrategie. Die meisten Firebird Clients können diesen Abrufplan für den Benutzer sichtbar machen. In Firebirds eigenem isql-Dienstprogramm geschieht dies mit dem Befehl SET PLAN ON. Wenn Sie Abfragepläne analysieren und keine Abfragen ausführen, zeigt SET PLANONLY ON den Plan an, ohne die Abfrage auszuführen.

In den meisten Situationen können Sie darauf vertrauen, dass Firebird den optimalen Abfrageplan für Sie auswählt. Wenn Sie jedoch komplizierte Abfragen haben, die nicht leistungsfähig sind, lohnt es sich möglicherweise, den Plan zu prüfen und zu prüfen, ob Sie ihn verbessern können.

Einfache Pläne

Die einfachsten Pläne bestehen nur aus einem Beziehungsnamen gefolgt von einer Abrufmethode. Z. B. für eine unsortierte Ein-Tabellen-Auswahl ohne eine WHERE-Klausel:

```

select * from students
plan (students natural)
    
```

Wenn eine WHERE- oder eine HAVING-Klausel vorhanden ist, können Sie den Index angeben, der zum Auffinden von Übereinstimmungen verwendet werden soll:

```
select * from students
  where class = '3C'
  plan (students index (ix_stud_class))
```

Die Anweisung INDEX wird auch für Join-Bedingungen verwendet (etwas später diskutiert). Es kann eine Liste von Indizes enthalten, die durch Kommata getrennt sind.

ORDER gibt den Index zum Sortieren des Satzes an, wenn eine Klausel ORDER BY oder GROUP BY vorhanden ist:

```
select * from students
  plan (students order pk_students)
  order by id
```

ORDER und INDEX können kombiniert werden:

```
select * from students
  where class >= '3'
  plan (students order pk_students index (ix_stud_class))
  order by id
```

Es ist völlig in Ordnung, wenn ORDER und INDEX denselben Index angeben:

```
select * from students
  where class >= '3'
  plan (students order ix_stud_class index (ix_stud_class))
  order by class
```

Wenn Sie einen Sortiersatz verwenden möchten, wenn kein verwendbarer Index verfügbar ist (oder wenn Sie die Verwendung des Index unterdrücken möchten), lassen Sie ORDER aus und stellen Sie dem Planausdruck SORT voran:

```
select * from students
  plan sort (students natural)
  order by name
```

Oder wenn ein Index für die Suche verwendet wird:

```
select * from students
  where class >= '3'
  plan sort (students index (ix_stud_class))
  order by name
```

Beachten Sie, dass sich SORT im Gegensatz zu ORDER außerhalb der Klammern befindet. Dies spiegelt die Tatsache wider, dass die Datenzeilen ungeordnet abgerufen und anschließend von der Engine sortiert werden.

Geben Sie bei der Auswahl aus einer Ansicht die Ansicht und die betreffende Tabelle an. Zum Beispiel, wenn Sie eine Ansicht FRESHMEN haben, die nur die Erstsemester auswählt:

```
select * from freshmen
  plan (freshmen students natural)
```

Oder zum Beispiel:

```
select * from freshmen
```

```
where id > 10
plan sort (freshmen students index (pk_students))
order by name desc
```

Wichtig

Wenn eine Tabelle oder Sicht mit einem Alias versehen wurde, muss der Alias und nicht der ursprüngliche Name in der Klausel PLAN verwendet werden.

Zusammengesetzte Pläne

Wenn ein Join erstellt wird, können Sie den Index angeben, der für den Abgleich verwendet werden soll. Sie müssen auch die Anweisung JOIN für die beiden Streams im Plan verwenden:

```
select s.id, s.name, s.class, c.mentor
from students s
join classes c on c.name = s.class
plan join (s natural, c index (pk_classes))
```

Derselbe Join, sortiert nach einer indizierten Spalte:

```
select s.id, s.name, s.class, c.mentor
from students s
join classes c on c.name = s.class
plan join (s order pk_students, c index (pk_classes))
order by s.id
```

Und auf einer nicht indizierten Spalte:

```
select s.id, s.name, s.class, c.mentor
from students s
join classes c on c.name = s.class
plan sort (join (s natural, c index (pk_classes)))
order by s.name
```

Mit einer Suche hinzugefügt:

```
select s.id, s.name, s.class, c.mentor
from students s
join classes c on c.name = s.class
where s.class <= '2'
plan sort (join (s index (fk_student_class), c index (pk_classes)))
order by s.name
```

Als linker Outer Join:

```
select s.id, s.name, s.class, c.mentor
from classes c
left join students s on c.name = s.class
where s.class <= '2'
plan sort (join (c natural, s index (fk_student_class)))
order by s.name
```

Wenn für die Join-Kriterien kein Index verfügbar ist (oder wenn Sie ihn nicht verwenden möchten), muss der Plan zuerst beide Streams in ihren Join-Spalten sortieren und dann zusammenführen. Dies wird mit der Anweisung SORT (die wir bereits erreicht haben) und MERGE anstelle von JOIN erreicht:

```
select * from students s
  join classes c on c.cookie = s.cookie
  plan merge (sort (c natural), sort (s natural))
```

Durch das Hinzufügen einer ORDER BY-Klausel muss das Ergebnis der Zusammenführung ebenfalls sortiert werden:

```
select * from students s
  join classes c on c.cookie = s.cookie
  plan sort (merge (sort (c natural), sort (s natural)))
  order by c.name, s.id
```

Schließlich fügen wir eine Suchbedingung für zwei indexierbare Spalten der Tabelle STUDENTS hinzu:

```
select * from students s
  join classes c on c.cookie = s.cookie
  where s.id < 10 and s.class <= '2'
  plan sort (merge (sort (c natural),
                   sort (s index (pk_students, fk_student_class))))
  order by c.name, s.id
```

Wie aus der formalen Syntaxdefinition hervorgeht, können JOINS und MERGES im Plan mehr als zwei Datenströme kombinieren. Außerdem kann jeder Planausdruck als Planposten in einem umfassenden Plan verwendet werden. Dies bedeutet, dass Pläne bestimmter komplizierter Abfragen verschiedene Verschachtelungsebenen haben können.

Schließlich können Sie anstelle von MERGE auch SORT MERGE schreiben. Da dies absolut keinen Unterschied macht und zu Verwechslungen mit „real“ SORT-Direktiven führen kann (diejenigen, die etwas *tun* machen einen Unterschied), ist es wahrscheinlich am besten zu bleiben zu einfach MERGE.

Warnung

Gelegentlich akzeptiert der Optimierer einen Plan und folgt ihm dann nicht, obwohl er ihn nicht als ungültig zurückweist. Ein solches Beispiel war

```
MERGE (unsortierter Stream, unsortierter Stream)
```

Es ist ratsam, einen solchen Plan als „veraltet“ zu behandeln .

UNION

Ein UNION verkettet zwei oder mehr Datasets und erhöht so die Anzahl der Zeilen, nicht aber die Anzahl der Spalten. Datasets, die an einer UNION teilnehmen, müssen die gleiche Anzahl von Spalten haben, und Spalten an entsprechenden Positionen müssen vom selben Typ sein. Abgesehen davon können sie völlig unabhängig sein.

Standardmäßig unterdrückt eine Union doppelte Zeilen. UNION ALL zeigt alle Zeilen einschließlich aller Duplikate an. Das optionale Schlüsselwort DISTINCT macht das Standardverhalten explizit.

Syntax:

```
<union> ::= <individual-select>
          UNION [DISTINCT | ALL]
          <individual-select>
          [UNION [DISTINCT | ALL]
```



```

        <individual-select>
        ...]
        [<union-wide-clauses>]

<individual-select> ::= SELECT
        [TRANSACTION name]
        [FIRST <m>] [SKIP <n>]
        [DISTINCT | ALL] <columns>
        [INTO <host-varlist>]
        FROM source [[AS] alias]
        [<joins>]
        [WHERE <condition>]
        [GROUP BY <grouping-list>]
        [HAVING <aggregate-condition>]]
        [PLAN <plan-expr>]

<union-wide-clauses> ::= [ORDER BY <ordering-list>]
        [ROWS m [TO n]]
        [FOR UPDATE [OF <columns>]]
        [WITH LOCK]
        [INTO <PSQL-varlist>]
    
```

Unions ermitteln ihre Spaltennamen aus der *ersten* Abfrage. Wenn Sie einen Alias für Vereinigungsspalten verwenden möchten, tun Sie dies in der Spaltenliste des obersten SELECT. Aliasnamen in anderen teilnehmenden Selects sind zulässig und können sogar nützlich sein, werden jedoch nicht auf Unionsebene weitergegeben.

Wenn eine Union eine ORDER BY-Klausel hat, sind die einzigen zulässigen Sortierelemente Integerlitterale, die 1-basierte Spaltenpositionen angeben, optional gefolgt von einem ASC/DESC und/oder einer NULLS FIRST/LAST-Direktive. Dies bedeutet auch, dass Sie eine Union nicht nach etwas sortieren können, die keine Spalte in der Union ist. (Sie können jedoch eine abgeleitete Tabelle einfügen, die Ihnen alle üblichen Sortieroptionen zurückgibt.)

Unions sind in Unterabfragen jeglicher Art erlaubt und können selbst Unterabfragen enthalten. Sie können auch Joins enthalten und an einem Join teilnehmen, wenn sie in eine abgeleitete Tabelle eingebunden werden.

Beispiele

Diese Abfrage präsentiert Informationen aus verschiedenen Musiksammlungen in einem Datensatz mithilfe von Unionen:

```

select id, title, artist, length, 'CD' as medium
  from cds
union
select id, title, artist, length, 'LP'
  from records
union
select id, title, artist, length, 'MC'
  from cassettes
order by 3, 2 -- artist, title
    
```

Wenn id, title, artist und length die einzigen Felder in den involvierten Tabellen sind, kann die Abfrage auch so geschrieben werden:

```

select c.*, 'CD' as medium
  from cds c
union
select r.*, 'LP'
  from records r
    
```

```
union
select c.*, 'MC'
  from cassettes c
order by 3, 2 -- artist, title
```

Das Qualifizieren der „Sternchen“ ist hier notwendig, da sie nicht das einzige Element in der Spaltenliste sind. Beachten Sie, dass die Aliase von „c“ in der ersten und dritten Auswahl nicht miteinander in Konflikt stehen: ihre Gültigkeitsbereiche sind nicht unionsweit, sondern gelten nur für ihre jeweiligen Auswahlabfragen.

Die nächste Abfrage ruft Namen und Telefonnummern von Übersetzern und Korrektoren ab. Übersetzer, die auch als Korrekturleser arbeiten, werden nur einmal im Ergebnis angezeigt, sofern ihre Telefonnummer in beiden Tabellen identisch ist. Das gleiche Ergebnis kann ohne DISTINCT erzielt werden. Mit ALL würden diese Personen zweimal angezeigt.

```
select name, phone from translators
union distinct
select name, telephone from proofreaders
```

Ein UNION innerhalb einer Unterabfrage:

```
select name, phone, hourly_rate from clowns
where hourly_rate < all
  (select hourly_rate from jugglers
   union
   select hourly_rate from acrobats)
order by hourly_rate
```

ORDER BY

Wenn eine SELECT-Anweisung ausgeführt wird, ist die Ergebnismenge in keiner Weise sortiert. Es kommt häufig vor, dass Zeilen chronologisch sortiert angezeigt werden, weil sie in derselben Reihenfolge zurückgegeben werden, in der sie von INSERT-Anweisungen zur Tabelle hinzugefügt wurden. Um eine Sortierreihenfolge für die Mengenspezifikation anzugeben, wird eine ORDER BY-Klausel verwendet.

Syntax:

```
SELECT ... FROM ...
...
ORDER BY <ordering-item> [, <ordering-item> ...]

<ordering-item> ::=
  {col-name | col-alias | col-position | expression}
  [COLLATE collation-name]
  [ASC[ENDING] | DESC[ENDING]]
  [NULLS {FIRST|LAST}]
```

Tabelle 6.7. Argumente für die ORDER BY-Klausel

| Argument | Beschreibung |
|-----------|---------------------------|
| col-name | Vollständiger Spaltenname |
| col-alias | Spaltenalias |

| Argument | Beschreibung |
|----------------|---|
| col-position | Spaltenposition in der SELECT-Liste |
| expression | Jeder Ausdruck |
| collation-name | Collations-Name (Sortierreihenfolge für String-Typen) |

Beschreibung

Die ORDER BY-Klausel besteht aus einer durch Komma getrennten Liste der Spalten, auf denen der Ergebnissatz sortiert werden soll. Die Sortierreihenfolge kann durch den Namen der Spalte—angegeben werden, jedoch nur, wenn die Spalte zuvor in der Spaltenliste SELECT nicht mit einem Alias versehen war. Der Alias muss verwendet werden, wenn er dort verwendet wurde. Die Ordnungsnummer der Spalte, des Alias, der der Spalte in der SELECT-Liste mit Hilfe des Schlüsselworts AS zugewiesen wurde, oder die Nummer der Spalte in der Liste SELECT können uneingeschränkt verwendet werden.

Die drei Arten, die Spalten für die Sortierreihenfolge auszudrücken, können in der gleichen ORDER BY-Klausel gemischt werden. Zum Beispiel kann eine Spalte in der Liste durch ihren Namen spezifiziert werden und eine andere Spalte kann durch ihre Nummer spezifiziert werden.

Anmerkung

Wenn Sie die Spaltenposition verwenden, um die Sortierreihenfolge für eine Abfrage des SELECT * -Stils anzugeben, erweitert der Server das Sternchen auf die vollständige Spaltenliste, um die Spalten für die Sortierung zu bestimmen. Es wird jedoch als „schlampige Praxis“ angesehen, um auf diese Weise geordnete Mengen zu entwerfen.

Sortierrichtung

Das Schlüsselwort ASCENDING, normalerweise abgekürzt als ASC, gibt eine Sortierrichtung vom niedrigsten zum höchsten an. ASCENDING ist die Standardsortierrichtung.

Das Schlüsselwort DESCENDING, normalerweise abgekürzt als DESC, gibt eine Sortierrichtung vom höchsten zum niedrigsten an.

Angaben der aufsteigenden Reihenfolge für eine Spalte und der absteigenden Reihenfolge für eine andere Spalte ist zulässig.

Collations-Reihenfolge

Das Schlüsselwort COLLATE gibt die Sortierreihenfolge für eine Zeichenfolgespalte an, wenn Sie eine Sortierung benötigen, die sich von der normalen Sortierung für diese Spalte unterscheidet. Die normale Sortierreihenfolge ist entweder die Standardreihenfolge für den Datenbankzeichensatz oder eine, die explizit in der Definition der Spalte festgelegt wurde.

NULLen positionieren

Das Schlüsselwort NULLS gibt an, wo NULL in der betroffenen Spalte in der Sortierung stehen wird: NULLS FIRST platziert die Zeilen mit der NULL-Spalte *oberhalb* der Zeilen mit den Spaltenwerten; NULLS LAST platziert diese Zeilen *hinter* den Spaltenwerten.

NULLS FIRST ist der Standard.

Sortieren von UNION-Sätzen

Die einzelnen Abfragen, die zu einer UNION beitragen, können keine ORDER BY-Klausel verwenden. Die einzige Option besteht darin, die gesamte Ausgabe mit einer ORDER BY-Klausel am Ende der gesamten Abfrage zu sortieren.

Das einfachste—und in einigen Fällen die einzige—Methode zum Angeben der Sortierreihenfolge ist die Ordinalspaltenposition. Es ist jedoch auch zulässig, die Spaltennamen oder Aliase *nur* aus der ersten beitragenden Abfrage zu verwenden.

Die Anweisungen ASC / DESC und / oder NULLS sind für diese globale Gruppe verfügbar.

Wenn eine diskrete Reihenfolge innerhalb der beitragenden Menge erforderlich ist, kann die Verwendung von abgeleiteten Tabellen oder allgemeinen Tabellenausdrücken für diese Mengen eine Lösung sein.

Beispiele

Sortierung der Ergebnismenge in aufsteigender Reihenfolge, Sortierung nach den Spalten RDB\$CHARACTER_SET_ID, RDB\$COLLATION_ID der Tabelle DB\$COLLATIONS:

```
SELECT
  RDB$CHARACTER_SET_ID AS CHARSET_ID,
  RDB$COLLATION_ID AS COLL_ID,
  RDB$COLLATION_NAME AS NAME
FROM RDB$COLLATIONS
ORDER BY RDB$CHARACTER_SET_ID, RDB$COLLATION_ID
```

Das Gleiche, aber Sortieren nach den Spaltenaliasnamen:

```
SELECT
  RDB$CHARACTER_SET_ID AS CHARSET_ID,
  RDB$COLLATION_ID AS COLL_ID,
  RDB$COLLATION_NAME AS NAME
FROM RDB$COLLATIONS
ORDER BY CHARSET_ID, COLL_ID
```

Sortieren der Ausgabedaten nach den Spaltenpositionsnummern:

```
SELECT
  RDB$CHARACTER_SET_ID AS CHARSET_ID,
  RDB$COLLATION_ID AS COLL_ID,
  RDB$COLLATION_NAME AS NAME
FROM RDB$COLLATIONS
ORDER BY 1, 2
```

Sortierung einer SELECT *-Abfrage nach Positionsnummern möglich, aber *hässlich* und nicht empfohlen:

```
SELECT *
```

```
FROM RDB$COLLATIONS
ORDER BY 3, 2
```

Sortierung nach der zweiten Spalte in der BOOKS-Tabelle:

```
SELECT
    BOOKS.*,
    FILMS.DIRECTOR
FROM BOOKS, FILMS
ORDER BY 2
```

Achtung

Ausdrücke, deren Berechnungsergebnisse nicht negative ganze Zahlen sind, werden als Spaltenpositionsnummern interpretiert und verursachen eine Ausnahme, wenn sie außerhalb des Bereichs von 1 bis zur Anzahl der Spalten liegen.

Beispiel:

```
SELECT
    X, Y, NOTE
FROM PAIRS
ORDER BY X+Y DESC
```

- Die von einer Funktion oder Prozedur zurückgegebene Zahl ist nicht vorhersehbar, unabhängig davon, ob die Sortierreihenfolge durch den Ausdruck selbst oder durch die Spaltennummer definiert ist.
- Nur nicht negative ganze Zahlen werden als Spaltennummern interpretiert
- Eine Ganzzahl, die durch einmalige Auswertung eines Ausdrucks oder durch Parametersubstitution erhalten wurde, wird als Konstante gespeichert, da dieser Wert für alle Zeilen gilt.

Beispiele, Fortsetzung

Sortierung in absteigender Reihenfolge nach den Werten der Spalte PROCESS_TIME, wobei NULL am Anfang der Menge steht:

```
SELECT *
FROM MSG
ORDER BY PROCESS_TIME DESC NULLS FIRST
```

Sortieren der Menge, die von einer UNION von zwei Abfragen erhalten wurde. Die Ergebnisse werden in absteigender Reihenfolge für die Werte in der zweiten Spalte sortiert, wobei NULLen am Ende der Menge stehen; und in aufsteigender Reihenfolge für die Werte der ersten Spalte mit NULLen am Anfang.

```
SELECT
    DOC_NUMBER, DOC_DATE
FROM PAYORDER
UNION ALL
SELECT
```

```
DOC_NUMBER, DOC_DATE
FROM BUDGORDER
ORDER BY 2 DESC NULLS LAST, 1 ASC NULLS FIRST
```

ROWS

Verwendet für: Abrufen eines Stücks von Zeilen aus einer geordneten Menge

Verfügbar in: DSQL, PSQL

Syntax:

```
SELECT <columns> FROM ...
  [WHERE ...]
  [ORDER BY ...]
  ROWS <m> [TO <n>]
```

Tabelle 6.8. Argumente für die ROWS-Klausel

| Argument | Beschreibung |
|----------|------------------------------|
| m, n | Beliebiger Integer-Ausdrücke |

Beschreibung: Begrenzt die Anzahl der Zeilen, die von der Anweisung SELECT an eine angegebene Zahl oder einen angegebenen Bereich zurückgegeben werden.

Die Klauseln FIRST und SKIP haben die gleiche Aufgabe wie ROWS sind jedoch nicht SQL-konform. Die Verwendung von ROWS ist daher im neuen Code vorzuziehen. Im Gegensatz zu FIRST und SKIP akzeptieren die Klauseln ROWS und TO beliebige Integerausdrücke als Argumente ohne Klammern. Natürlich können Klammern für verschachtelte Auswertungen innerhalb des Ausdrucks noch immer benötigt werden und eine Unterabfrage muss immer in Klammern eingeschlossen sein.

Wichtig

- Nummerierung der Zeilen in der Zwischenmenge—die Gesamtmenge, die auf der Festplatte zwischengespeichert wird, bevor die „Scheibe“ extrahiert wird—beginnt bei 1.
- Sowohl FIRST / SKIP als auch ROWS können ohne ORDER BY-Klausel verwendet werden, obwohl dies selten sinnvoll ist, es sei denn Sie möchten nur einen kurzen Blick auf die Tabellendaten werfen und es ist nicht wichtig, dass Zeilen in zufälliger Reihenfolge stehen. Zu diesem Zweck würde eine Abfrage wie „SELECT * FROM TABLE1 ROWS 20“ die ersten 20 Zeilen anstelle einer ganzen Tabelle, die ziemlich groß sein könnte, zurückgeben.

Der Aufruf von ROWS m gibt die ersten m Zeilen des angegebenen Satzes zurück.

Merkmale der Verwendung von ROWS m ohne eine TO-Klausel:

- Wenn m größer als die Gesamtzahl der Datensätze im Zwischendatensatz ist, wird die gesamte Menge zurückgegeben
- Wenn $m = 0$, wird ein leerer Satz zurückgegeben
- Wenn $m < 0$, wird der SELECT-Aufruf in einem Fehler enden

Der Aufruf von ROWS m TO n gibt die Zeilen aus dem Satz zurück, beginnend mit Zeile m und endend nach Zeile n —inklusive Satz.

Merkmale der Verwendung von ROWS m mit einer TO-Klausel:

- Wenn m größer ist als die Gesamtzahl der Zeilen in der Zwischengruppe und $n < = m$, wird eine leere Menge zurückgegeben
- Ist m nicht größer als n und n größer als die Gesamtzahl der Zeilen in der Zwischengruppe, ist die Ergebnismenge beschränkt auf Zeilen beginnend mit m bis zum Ende des Satzes
- Wenn $m < 1$ und $n < 1$, schlägt der SELECT-Anweisungsaufwurf mit einem Fehler fehl
- Wenn $n = m - 1$ ist, wird eine leere Menge zurückgegeben
- Wenn $n < m - 1$, schlägt der Aufruf der Anweisung SELECT mit einem Fehler fehl

Verwenden einer TO-Klausel ohne eine ROWS-Klausel:

Während ROWS die nicht standardmäßige Syntax FIRST und SKIP ersetzt, gibt es eine Situation, in der die Standardsyntax nicht das gleiche Verhalten bietet: Mit Angabe von SKIP n wird der gesamte Zwischensatz ohne die ersten n -Reihen zurückgegeben. Die ROWS ... TO-Syntax benötigt ein wenig Hilfe, um dies zu erreichen.

Bei der Syntax ROWS benötigen Sie eine ROWS-Klausel *in Verbindung mit* der TO-Klausel. Anschließend machen Sie das zweite Argument (n) größer als die Größe des Zwischendatensatzes. Dies wird erreicht, indem ein Ausdruck für n erstellt wird, der eine Unterabfrage verwendet, um die Anzahl der Zeilen in der Zwischengruppe abzurufen, und 1 dazu addiert.

Das Mischen von ROWS und FIRST / SKIP

Die Syntax ROWS kann nicht mit der Syntax FIRST / SKIP im selben SELECT-Ausdruck gemischt werden. Die Verwendung der verschiedenen Syntaxen in verschiedenen Unterabfragen in derselben Anweisung ist jedoch zulässig.

ROWS-Syntax in UNION-Abfragen

Wenn ROWS in einer UNION-Abfrage verwendet wird, wird die ROWS-Direktive auf UNION-Satzes angewendet und muss hinter dem letzten SELECT-Statement stehen.

Wenn es erforderlich ist, die Teilmengen zu begrenzen, die von einer oder mehreren SELECT-Anweisungen innerhalb von UNION zurückgegeben werden, gibt es eine Reihe von Optionen:

1. Verwenden Sie die FIRST / SKIP-Syntax in diesen SELECT-Anweisungen — bedenken Sie, dass eine ordering-Klausel (ORDER BY) nicht lokal auf die einzelnen Abfragen angewendet werden kann, aber nur für den kombinierten Ausgang.
2. Konvertieren Sie die Abfragen in abgeleitete Tabellen mit ihren eigenen ROWS-Klauseln.

Beispiele

Die folgenden Beispiele schreiben die [Beispiele](#) des Abschnitts über FIRST und SKIP, [weiter oben in diesem Kapitel](#), neu.

Gib die ersten zehn Namen aus der Ausgabe einer sortierten Abfrage in der Tabelle PEOPLE aus:

```
SELECT id, name
FROM People
```

```
ORDER BY name ASC
ROWS 1 TO 10
```

oder äquivalent dazu:

```
SELECT id, name
FROM People
ORDER BY name ASC
ROWS 10
```

Gib alle Datensätze aus der PEOPLE-Tabelle mit Ausnahme der ersten 10 Namen zurück:

```
SELECT id, name
FROM People
ORDER BY name ASC
ROWS 11 TO (SELECT COUNT(*) FROM People)
```

Und diese Abfrage gibt die letzten 10 Datensätze zurück (achten Sie auf die Klammern):

```
SELECT id, name
FROM People
ORDER BY name ASC
ROWS (SELECT COUNT(*) - 9 FROM People)
TO (SELECT COUNT(*) FROM People)
```

Diese gibt die Zeilen 81-100 aus der Tabelle PEOPLE zurück:

```
SELECT id, name
FROM People
ORDER BY name ASC
ROWS 81 TO 100
```

Anmerkung

ROWS kann außerdem in Kombination mit den Anweisungen UPDATE und DELETE verwendet werden.

FOR UPDATE [OF]

Syntax:

```
SELECT ... FROM single_table
  [WHERE ...]
  [FOR UPDATE [OF ...]]
```

FOR UPDATE tut nicht, was es vorgibt. Der einzige Effekt ist derzeit, den Pre-Fetch-Puffer zu deaktivieren.

Tipp

Dies wird sich wahrscheinlich in Zukunft ändern: Es ist geplant, mit FOR UPDATE markierte Cursor zu validieren, wenn sie wirklich aktualisierbar sind und positionierte Aktualisierungen und Löschungen für Cursor ablehnen, die als nicht aktualisierbar eingestuft werden.

Die OF-Unterklausele tut rein gar nichts.

WITH LOCK

Verfügbar in: DSQL, PSQL

Verwendet für: Begrenzte pessimistische Sperrung

Beschreibung: WITH LOCK bietet eine begrenzte explizite pessimistische Sperrfunktion für die vorsichtige Verwendung unter Bedingungen, bei denen für den betroffenen Zeilensatz Folgendes gilt:

- a. extrem klein (idealerweise ein Singleton), *und*
- b. genau gesteuert durch den Anwendungscode.

Dies ist nur für Experten!

Die Notwendigkeit einer pessimistischen Sperre in Firebird ist in der Tat sehr selten und sollte gut verstanden werden, bevor die Verwendung dieser Erweiterung in Betracht gezogen wird.

Es ist wichtig, die Auswirkungen der Transaktionsisolation und anderer Transaktionsattribute zu verstehen, bevor Sie das explizite Sperren in Ihrer Anwendung implementieren.

Syntax:

```
SELECT ... FROM single_table
  [WHERE ...]
  [FOR UPDATE [OF ...]]
WITH LOCK
```

Wenn die WITH LOCK-Klausel erfolgreich ist, sichert sie eine Sperre für die ausgewählten Zeilen und verhindert, dass eine andere Transaktion Schreibzugriff auf eine dieser Zeilen oder deren abhängige Elemente erhält, bis die Transaktion endet.

WITH LOCK kann nur mit einer SELECT-Anweisung der obersten Ebene verwendet werden. Es steht *nicht* zur Verfügung:

- in einer Unterabfrage-Spezifikation
- für Join-Sätze
- mit dem DISTINCT-Operator, einer GROUP BY-Klausel oder einer anderen Aggregat-Operation
- mit einer Ansicht
- mit der Ausgabe einer abfragbaren gespeicherten Prozedur
- mit einer externen Tabelle
- mit einer UNION-Abfrage

Da die Engine wiederum berücksichtigt, dass jeder Datensatz unter eine explizite Sperranweisung fällt, gibt sie entweder die derzeit festgeschriebene Datensatzversion (zum Zeitpunkt als die Anweisung gesendet wurde) zurück, unabhängig vom Datenbankstatus, oder eine Ausnahme.

Das Warteverhalten und die Konfliktmeldung hängen von den im TPB-Block angegebenen Transaktionsparametern ab:

Tabelle 6.9. Wie TPB-Einstellungen das explizite Sperren beeinflussen

| TPB-Modus | Verhalten |
|--|---|
| isc_tpb_consistency | Explizite Sperren werden von impliziten oder expliziten Sperren auf Tabellenebene außer Kraft gesetzt und ignoriert. |
| isc_tpb_concurrency + isc_tpb_nowait | Wenn ein Datensatz von einer Transaktion geändert wird, die festgeschrieben wurde, seit die Transaktion versucht hat, die explizite Sperre zu starten, oder eine aktive Transaktion eine Änderung dieses Datensatzes durchgeführt hat, wird sofort eine Aktualisierungskonfliktausnahme ausgelöst. |
| isc_tpb_concurrency + isc_tpb_wait | Wenn der Datensatz von einer Transaktion geändert wird, die seit dem Versuch der Ausführung der expliziten Sperre festgeschrieben wurde, wird sofort eine Ausnahme für den Aktualisierungskonflikt ausgelöst. Wenn eine aktive Transaktion den Besitz dieses Datensatzes innehat (durch explizites Sperren oder durch eine normale optimistische Schreibsperre), wartet die Transaktion, die die explizite Sperre verursacht, auf das Ergebnis der blockierenden Transaktion und, wenn sie beendet ist, versucht sie, die Sperre zu wieder aufzuheben. Wenn die blockierende Transaktion eine geänderte Version dieses Datensatzes erstellt hat, wird eine Ausnahme für den Aktualisierungskonflikt ausgelöst. |
| isc_tpb_read_committed + isc_tpb_nowait | Wenn es eine aktive Transaktion gibt, die den Besitz für diesen Datensatz innehat (durch explizites Sperren oder normale Aktualisierung), wird sofort eine Aktualisierungskonfliktausnahme ausgelöst. |
| isc_tpb_read_committed + isc_tpb_wait | Wenn es eine aktive Transaktion gibt, die den Besitz dieses Datensatzes innehat (durch explizites Sperren oder durch eine normale optimistische Schreibsperre), wartet die Transaktion, die die explizite Sperre verursacht, auf das Ergebnis der Blockierungstransaktion und wenn sie beendet ist, versucht sie die Sperre zu wieder aufzuheben. Aktualisierungskonfliktausnahmen können niemals durch eine explizite Sperranweisung in diesem TPB-Modus ausgelöst werden. |

Verwendung einer FOR UPDATE-Klausel

Wenn die Unterklausel FOR UPDATE der Unterklausel WITH LOCK vorangestellt ist, werden gepufferte Abrufe unterdrückt. Somit wird die Sperre zum Zeitpunkt des Abrufens auf jede Zeile einzeln angewendet. Es wird dann möglich, dass eine Sperre, die bei Anforderung erfolgreich zu sein scheint, trotzdem *anschließend fehlschlägt*, wenn versucht wird, eine Zeile abzurufen, die in der Zwischenzeit durch eine andere Transaktion gesperrt wurde.

Tipp

Als eine Alternative kann es in Ihren Zugriffskomponenten möglich sein, die Größe des Abrufpuffers auf 1 zu setzen. Dies würde es Ihnen ermöglichen, die aktuell gesperrte Zeile zu verarbeiten, bevor die nächste geholt und gesperrt wird, oder Fehler zu behandeln, ohne ihre Transaktion zurückzurollen.

OF <Spaltennamen>

Diese optionale Unterklausele macht überhaupt nichts.

Siehe auch: FOR UPDATE [OF]

Wie die Engine mit WITH LOCK umgeht

Wenn eine UPDATE-Anweisung versucht, auf einen Datensatz zuzugreifen, der durch eine andere Transaktion gesperrt ist, löst sie abhängig vom TPB-Modus entweder eine Aktualisierungskonfliktausnahme aus oder wartet auf den Abschluss der Sperrtransaktion. Das Verhalten der Engine ist hier so, als wäre dieser Datensatz bereits durch die Sperrtransaktion modifiziert worden.

Bei Konflikten mit pessimistischen Sperren werden keine speziellen gscodes zurückgegeben.

Die Engine garantiert, dass alle von einer expliziten Lock-Anweisung zurückgegebenen Datensätze tatsächlich gesperrt sind und die in der WHERE-Klausel angegebenen Suchbedingungen *erfüllen*, solange die Suchbedingungen nicht von anderen Tabellen, Joins, Unterabfragen usw. abhängen. Außerdem wird garantiert, dass Zeilen, die die Suchbedingungen nicht erfüllen, nicht von der Anweisung gesperrt werden. Sie kann *nicht* garantieren, dass es keine Zeilen gibt, die zwar die Suchbedingungen erfüllen, aber nicht gesperrt sind.

Anmerkung

Diese Situation kann auftreten, wenn andere parallele Transaktionen ihre Änderungen im Verlauf der Ausführung der Sperranweisung festschreiben.

Die Engine sperrt Zeilen zum Abrufzeitpunkt. Dies hat wichtige Konsequenzen, wenn Sie mehrere Zeilen gleichzeitig sperren. Bei vielen Zugriffsmethoden für Firebird-Datenbanken wird die Ausgabe standardmäßig in Paketen mit einigen hundert Zeilen abgerufen („gepufferte Abrufe“). Die meisten Datenzugriffskomponenten können die Zeilen, die im zuletzt abgerufenen Paket enthalten sind, nicht anzeigen, wenn ein Fehler aufgetreten ist.

Fallstricke mit WITH LOCK

- Durch das Zurücksetzen eines impliziten oder expliziten Sicherungspunkts werden Datensatzsperren freigegeben, die unter diesem Sicherungspunkt ausgeführt wurden, jedoch keine wartenden Transaktionen. Anwendungen sollten nicht von diesem Verhalten abhängig sein, da sie sich in Zukunft ändern können.
- Während explizite Sperren zum Verhindern und / oder Behandeln ungewöhnlicher Fehler beim Updatekonflikt verwendet werden können, steigt die Anzahl der Deadlockfehler, wenn Sie Ihre Sperrstrategie nicht sorgfältig planen und streng steuern.
- Die meisten Anwendungen benötigen keine expliziten Sperren. Die Hauptzwecke expliziter Sperren sind (1) die teure Behandlung von Fehlern bei der Aktualisierung von Konflikten in stark ausgelasteten Anwendungen zu verhindern und (2) die Integrität von Objekten zu erhalten, die einer relationalen Datenbank in einer Clusterumgebung zugeordnet sind. Wenn Ihre explizite Sperrung nicht in eine dieser beiden Kategorien fällt, ist dies die falsche Vorgehensweise in Firebird.
- Explizites Sperren ist eine erweiterte Funktion. Missbrauchen Sie sie nicht! Während Lösungen für diese Art von Problemen sehr wichtig für Websites sein können, die Tausende von gleichzeitigen Schreibzugriffen behandeln, oder für ERP / CRM-Systeme, die in großen Unternehmen arbeiten, müssen die meisten Anwendungsprogramme unter solchen Bedingungen nicht arbeiten.

Beispiele, die explizites Sperren verwenden

i. Einfach:

```
SELECT * FROM DOCUMENT WHERE ID=? WITH LOCK
```

ii. Mehrere Zeilen, eins-zu-eins-Verarbeitung mit SQL-Cursor:

```
SELECT * FROM DOCUMENT WHERE PARENT_ID=?  
FOR UPDATE WITH LOCK
```

INTO

Verwendet für: Übergabe von SELECT-Ausgaben an Variablen

Verfügbar in: PSQL

Im PSQL-Code (Trigger, gespeicherte Prozeduren und ausführbare Blöcke) können die Ergebnisse einer SELECT-Anweisung Zeile für Zeile in lokale Variablen geladen werden. Es ist oft die einzige Möglichkeit, etwas mit den zurückgegebenen Werten zu tun. Die Anzahl, Reihenfolge und Typen der Variablen müssen mit den Spalten in der Ausgabezeile übereinstimmen.

Eine „reine“ SELECT-Anweisung kann nur in PSQL verwendet werden, wenn sie höchstens eine Zeile zurückgibt, d.h. wenn es ein *Singleton* ist. Bei mehrzeiligen Auswahlmöglichkeiten bietet PSQL das **FOR SELECT**-Schleifenkonstrukt, das später im PSQL-Kapitel erläutert wird. PSQL unterstützt auch die Anweisung **DECLARE CURSOR**, die einen benannten Cursor an eine Anweisung SELECT bindet. Der Cursor kann dann verwendet werden, um die Ergebnismenge zu durchlaufen.

Syntax: In PSQL wird die INTO-Klausel am Ende des SELECT-Statements platziert.

```
SELECT [...] <column-list>  
FROM ...  
[...]  
[INTO <variable-list>]  
  
<variable-list> ::= [:]psqlvar [, [:]psqlvar ...]
```

Anmerkung

Das Doppelpunkt-Präfix vor lokalen Variablennamen in PSQL ist optional.

Beispiele

Einige aggregierte Werte auswählen und an zuvor deklarierte Variablen `min_amt`, `avg_amt` und `max_amt` übergeben:

```
select min(amount), avg(cast(amount as float)), max(amount)  
from orders  
where artno = 372218  
into min_amt, avg_amt, max_amt;
```

Anmerkung

Die CAST dient dazu, den Durchschnitt zu einer reellen Zahl zu machen; Sonst würde amount vermutlich ein Integer-Feld sein, SQL-Regeln würden es auf die nächste niedrigere Ganzzahl abschneiden.

Ein PSQL-Trigger der zwei Werte als BLOB zurückliefert (Verwendung der LIST()-Funktion) und diese mittels INTO einem dritten Feld zuweist:

```
select list(name, ', ')
from persons p
where p.id in (new.father, new.mother)
into new.parentnames;
```

Common Table Expressions („WITH ... AS ... SELECT“)

Verfügbar in: DSQL, PSQL

Ein allgemeiner Tabellenausdruck oder CTE kann als virtuelle Tabelle oder Ansicht beschrieben werden, die in einer Präambel einer Hauptabfrage definiert ist und nach der Ausführung der Hauptabfrage den Gültigkeitsbereich verlässt. Die Hauptabfrage kann auf alle CTEs verweisen, die in der Präambel definiert sind, als wären sie reguläre Tabellen oder Sichten. CTEs kann rekursiv sein, d.h. sich selbst referenzieren, aber sie können nicht verschachtelt sein.

Syntax:

```
<cte-construct> ::= <cte-defs>
                  <main-query>

<cte-defs>      ::= WITH [RECURSIVE] <cte> [, <cte> ...]

<cte>           ::= name [( <column-list> )] AS ( <cte-stmt> )

<column-list>  ::= column-alias [, column-alias ...]
```

Tabelle 6.10. Argumente für Common Table Expressions

| Argument | Beschreibung |
|--------------|--|
| cte-stmt | Jede SELECT-Anweisung, einschließlich UNION |
| main-query | Die Hauptanweisung SELECT, die sich auf die in der Präambel definierten CTEs beziehen kann |
| name | Alias für einen Tabellenausdruck |
| column-alias | Alias für eine Spalte in einem Tabellenausdruck |

Beispiel:

```
with dept_year_budget as (
  select fiscal_year,
         dept_no,
         sum(projected_budget) as budget
  from proj_dept_budget
```

```

    group by fiscal_year, dept_no
)
select d.dept_no,
       d.department,
       dyb_2008.budget as budget_08,
       dyb_2009.budget as budget_09
from department d
     left join dept_year_budget dyb_2008
           on d.dept_no = dyb_2008.dept_no
           and dyb_2008.fiscal_year = 2008
     left join dept_year_budget dyb_2009
           on d.dept_no = dyb_2009.dept_no
           and dyb_2009.fiscal_year = 2009
where exists (
  select * from proj_dept_budget b
  where d.dept_no = b.dept_no
)

```

CTE-Hinweise

- Eine CTE-Definition kann ein beliebiges SELECT-Statement sein, sofern es keine „WITH...“-Präampel besitzt (keine Verschachtelung).
- CTEs für die gleiche Hauptabfrage definiert sind, können sich gegenseitig referenzieren, es sollte jedoch darauf geachtet werden, Schleifen zu vermeiden.
- CTEs kann von überall in der Hauptabfrage referenziert werden.
- Jede CTE kann in der Hauptabfrage mehrmals referenziert werden, wobei bei Bedarf verschiedene Aliase verwendet werden.
- Ein in Klammern eingeschlossenes CTE-Konstrukt kann als Unterabfrage in SELECT-Statements verwendet werden, aber auch in UPDATES, MERGES etc.
- In PSQL, werden CTEs auch in FOR-Schleifen verwendet:

```

for
  with my_rivers as (select * from rivers where owner = 'me')
  select name, length from my_rivers into :rname, :rlen
do
begin
  ..
end

```

Wichtig

Wenn eine CTE deklariert ist, muss sie später verwendet werden: Andernfalls erhalten Sie einen Fehler wie diesen: 'CTE "AAA" is not used in query'.

Rekursive CTEs

Eine rekursive (selbstverweisende) CTE ist eine UNION, die mindestens ein nicht-rekursives Member haben muss, das als *Anker* bezeichnet wird. Die nicht rekursiven Member müssen vor den rekursiven Members platziert

werden. Rekursive Member sind untereinander und mit ihrem nicht rekursiven Nachbarn durch UNION ALL-Operatoren verknüpft. Die Verbindungen zwischen nicht-rekursiven Mitgliedern können beliebiger Art sein.

Für rekursive CTEs muss das Schlüsselwort RECURSIVE unmittelbar nach WITH vorhanden sein. Jedes rekursive Union-Member darf sich nur einmal selbst referenzieren, und zwar in einer FROM-Klausel.

Ein großer Vorteil von rekursiven CTEs ist, dass sie viel weniger Speicher und CPU-Zyklen als eine äquivalente rekursive gespeicherte Prozedur verwenden.

Ausführungsmuster

Das Ausführungsmuster einer rekursiven CTE lautet wie folgt:

- Die Engine beginnt mit der Ausführung von einem nicht rekursiven Element.
- Für jede ausgewertete Zeile beginnt die Ausführung jedes rekursiven Elements nacheinander, wobei die aktuellen Werte aus der äußeren Reihe als Parameter verwendet werden.
- Wenn die aktuell ausgeführte Instanz eines rekursiven Members keine Zeilen erzeugt, führt die Ausführung eine Schleife zurück und ruft die nächste Zeile aus der äußeren Ergebnismenge ab.

Beispiele für CTEs:

```
WITH RECURSIVE DEPT_YEAR_BUDGET AS (  
  SELECT  
    FISCAL_YEAR,  
    DEPT_NO,  
    SUM(PROJECTED_BUDGET) BUDGET  
  FROM PROJ_DEPT_BUDGET  
  GROUP BY FISCAL_YEAR, DEPT_NO  
) ,  
DEPT_TREE AS (  
  SELECT  
    DEPT_NO ,  
    HEAD_DEPT ,  
    DEPARTMENT ,  
    CAST(' ' AS VARCHAR(255)) AS INDENT  
  FROM DEPARTMENT  
  WHERE HEAD_DEPT IS NULL  
  UNION ALL  
  SELECT  
    D.DEPT_NO ,  
    D.HEAD_DEPT ,  
    D.DEPARTMENT ,  
    H.INDENT || ' ' ,  
  FROM DEPARTMENT D  
    JOIN DEPT_TREE H ON H.HEAD_DEPT = D.DEPT_NO  
)  
SELECT  
  D.DEPT_NO ,  
  D.INDENT || D.DEPARTMENT DEPARTMENT ,  
  DYB_2008.BUDGET AS BUDGET_08 ,  
  DYB_2009.BUDGET AS BUDGET_09  
FROM DEPT_TREE D  
  LEFT JOIN DEPT_YEAR_BUDGET DYB_2008 ON  
  (D.DEPT_NO = DYB_2008.DEPT_NO) AND
```

```
(DYB_2008.FISCAL_YEAR = 2008)
LEFT JOIN DEPT_YEAR_BUDGET DYB_2009 ON
(D.DEPT_NO = DYB_2009.DEPT_NO) AND
(DYB_2009.FISCAL_YEAR = 2009)
```

Das nächste Beispiel gibt den Stammbaum eines Pferdes zurück. Der Hauptunterschied besteht darin, dass Rekursion gleichzeitig in zwei Zweigen des Stammbaums auftritt.

```
WITH RECURSIVE PEDIGREE (
  CODE_HORSE,
  CODE_FATHER,
  CODE_MOTHER,
  NAME,
  MARK,
  DEPTH)
AS (SELECT
  HORSE.CODE_HORSE,
  HORSE.CODE_FATHER,
  HORSE.CODE_MOTHER,
  HORSE.NAME,
  CAST(' ' AS VARCHAR(80)),
  0
FROM
  HORSE
WHERE
  HORSE.CODE_HORSE = :CODE_HORSE
UNION ALL
SELECT
  HORSE.CODE_HORSE,
  HORSE.CODE_FATHER,
  HORSE.CODE_MOTHER,
  HORSE.NAME,
  'F' || PEDIGREE.MARK,
  PEDIGREE.DEPTH + 1
FROM
  HORSE
  JOIN PEDIGREE
  ON HORSE.CODE_HORSE = PEDIGREE.CODE_FATHER
WHERE
  PEDIGREE.DEPTH < :MAX_DEPTH
UNION ALL
SELECT
  HORSE.CODE_HORSE,
  HORSE.CODE_FATHER,
  HORSE.CODE_MOTHER,
  HORSE.NAME,
  'M' || PEDIGREE.MARK,
  PEDIGREE.DEPTH + 1
FROM
  HORSE
  JOIN PEDIGREE
  ON HORSE.CODE_HORSE = PEDIGREE.CODE_MOTHER
WHERE
  PEDIGREE.DEPTH < :MAX_DEPTH
)
SELECT
  CODE_HORSE,
```



```

NAME ,
MARK ,
DEPTH
FROM
  PEDIGREE

```

Hinweise zu rekursiven CTEs:

- Aggregate (DISTINCT, GROUP BY, HAVING) und Aggregatfunktionen (SUM, COUNT, MAX etc) sind in rekursiven Union-Membren nicht erlaubt.
- Eine rekursive Referenz kann nicht an einem Outer Join teilnehmen.
- Die maximale Rekursionstiefe beträgt 1024.

INSERT

Verwendet für: Einfügen von Datenzeilen in eine Tabelle

Verfügbar in: DSQL, ESQL, PSQL

Syntax:

```

INSERT INTO target
{DEFAULT VALUES | [(column_list)] <value_source>}
[RETURNING <returning_list> [INTO <variables>]]

<column_list> ::= colname [, colname ...]

<value_source> ::= VALUES (<value_list>) | <select_stmt>

<value_list> ::= value [, value ...]

<returning_list> ::= ret_value [, ret_value ...]
<variables> ::= [:]varname [, [:]varname ...]

```

Tabelle 6.11. Argumente für die Parameter des INSERT-Statements

| Argument | Beschreibung |
|-----------|--|
| target | Der Name der Tabelle oder Sicht, zu der eine neue Zeile oder ein Stapel von Zeilen hinzugefügt werden soll |
| colname | Spalte in der Tabelle oder in der Ansicht |
| value | Ein Ausdruck, dessen Wert zum Einfügen in die Tabelle verwendet wird |
| ret_value | Der Ausdruck, der in der RETURNING-Klausel zurückgegeben werden soll |
| varname | Name einer lokalen PSQL-Variablen |

Beschreibung: Das INSERT-Statement wird verwendet um Zeilen zu einer Tabelle hinzuzufügen. Alternativ können Zeilen auch in eine oder mehrere Tabellen eingefügt werden, die als Basis für eine View dienen:

- Wenn die Spaltenwerte in einer VALUES-Klausel angegeben werden, wird genau eine Zeile eingefügt
- Die Werte können stattdessen durch einen Ausdruck SELECT bereitgestellt werden, in welchem Fall null bis viele Zeilen eingefügt werden können
- Mit der Klausel DEFAULT VALUES werden überhaupt keine Werte bereitgestellt und genau eine Zeile eingefügt.

Beschränkungen

- Für Spalten, die an die `NEW.column_list`-Kontextvariablen in Triggern zurückgegeben werden, darf kein Doppelpunkt (, :“) vorangestellt sein
- Keine Spalte darf mehr als einmal in der Spaltenliste vorkommen.

ACHTUNG :: 'BEFORE INSERT'-Trigger

Berücksichtigen Sie unabhängig von der zum Einfügen von Zeilen verwendeten Methode alle Spalten in der Zieltabelle oder -sicht, die von BEFORE INSERT-Triggern gefüllt werden, z. B. Primärschlüssel und Suchfelder ohne Beachtung der Groß- / Kleinschreibung. Diese Spalten sollten sowohl von der `column_list` als auch von der VALUES-Liste ausgeschlossen werden, wenn die Trigger die `NEW.column_name` auf NULL prüfen.

INSERT ... VALUES

Die VALUES-Liste muss für jede Spalte in der Spaltenliste einen Wert in derselben Reihenfolge und mit dem richtigen Typ angeben. Die Spaltenliste muss nicht jede Spalte im Ziel angeben, aber wenn die Spaltenliste nicht vorhanden ist, benötigt die Engine für jede Spalte in der Tabelle oder Ansicht einen Wert (berechnete Spalten ausgeschlossen).

Anmerkung

Introducer-Syntax bietet eine Möglichkeit, den Zeichensatz eines Werts zu identifizieren, der eine Zeichenfolgenkonstante (Literal) ist. Die Introducer-Syntax funktioniert nur mit Literalstrings: Sie kann nicht auf Stringvariablen, Parameter, Spaltenreferenzen oder Werte, die Ausdrücke sind, angewendet werden.

Beispiele:

```
INSERT INTO cars (make, model, year)
VALUES ('Ford', 'T', 1908);

INSERT INTO cars
VALUES ('Ford', 'T', 1908, 'USA', 850);

-- notice the '_' prefix (introducer syntax)
INSERT INTO People
VALUES (_ISO8859_1 'Hans-Jörg Schäfer')
```

INSERT ... SELECT

Für diese Einfügemethode müssen die Ausgabespalten der Anweisung SELECT für jede Zielspalte in der Spaltenliste einen Wert in derselben Reihenfolge und vom richtigen Typ bereitstellen.

Literale Werte, Kontextvariablen oder Ausdrücke des kompatiblen Typs können für jede Spalte in der Quellzeile verwendet werden. In diesem Fall sind eine Quellspaltenliste und eine entsprechende VALUES-Liste erforderlich.

Wenn die Spaltenliste abwesend ist — wie es ist, wenn SELECT * für den Quellausdruck — verwendet wird, muss die *column_list* die Namen jeder Spalte in der Zieltabelle oder Sicht enthalten (berechnete Spalten ausgeschlossen).

Beispiele:

```
INSERT INTO cars (make, model, year)
  SELECT make, model, year
  FROM new_cars;

INSERT INTO cars
  SELECT * FROM new_cars;

INSERT INTO Members (number, name)
  SELECT number, name FROM NewMembers
  WHERE Accepted = 1
UNION ALL
  SELECT number, name FROM SuspendedMembers
  WHERE Vindicated = 1

INSERT INTO numbers(num)
  WITH RECURSIVE r(n) as (
    SELECT 1 FROM rdb$database
    UNION ALL
    SELECT n+1 FROM r WHERE n < 100
  )
  SELECT n FROM r
```

Natürlich müssen die Spaltennamen in der Quelltable nicht mit denen in der Zieltabelle übereinstimmen. Jede Art von SELECT-Anweisung ist zulässig, solange ihre Ausgabespalten exakt mit den Einfügespalten in Anzahl, Reihenfolge und Typ übereinstimmen. Typen müssen nicht exakt gleich sein, aber sie müssen zuweisungskompatibel sein.

Das Problem mit dem „instabilen Cursor“

In Firebird muss bis zu dieser Version ein Implementierungsfehler geachtet werden, der diese Art von Einfügungen betrifft, wenn das Ziel darin besteht, Zeilen in derselben Tabelle zu duplizieren. Beispielsweise

```
INSERT INTO T
  SELECT * FROM T
```

liebevoll als die „unendliche Einfügeschleife“ bezeichnet, wählt fortlaufend Zeilen aus und fügt sie immer wieder ein, bis das System keinen Speicherplatz mehr hat.

Dies ist eine Eigenart, die alle datenverändernden DML-Operationen mit einer Vielzahl von Effekten beeinflusst. Dies geschieht, weil DML-Anweisungen in den Ausführungsebenen implizite Cursor zum Ausführen der Operationen verwenden. Mit unserem einfachen Beispiel funktioniert die Ausführung folgendermaßen:

```
FOR SELECT <values> FROM T INTO <tmp_vars>
DO
  INSERT INTO T VALUES (<tmp_vars>)
```

Die Implementierung führt zu einem Verhalten, das nicht mit den SQL-Standards übereinstimmt. Zukünftige Versionen von Firebird werden dem Standard entsprechen.

INSERT ... DEFAULT VALUES

Die DEFAULT VALUES-Klausel erlaubt das Einfügen eines Datensatzes ohne die Angabe von Werten, weder direkt oder durch ein SELECT-Statement. Dies ist nur möglich, wenn jede NOT NULL- oder CHECK-basierte Spalte in der Tabelle entweder einen gültigen Standardwert deklariert hat oder die Werte über einen BEFORE INSERT-Trigger erhält. Darüber hinaus dürfen Trigger, die erforderliche Feldwerte bereitstellen, nicht von dem Vorhandensein von Eingabewerten abhängen.

Beispiel:

```
INSERT INTO journal
  DEFAULT VALUES
RETURNING entry_id
```

Die RETURNING-Klausel

Eine INSERT-Anweisung, die *höchstens eine Zeile* hinzufügt, kann optional eine Klausel RETURNING enthalten, um Werte aus der eingefügten Zeile zurückzugeben. Die Klausel, falls vorhanden, muss nicht alle Einfügungsspalten enthalten und kann auch andere Spalten oder Ausdrücke enthalten. Die zurückgegebenen Werte spiegeln alle Änderungen wider, die möglicherweise in den BEFORE INSERT-Triggern vorgenommen wurden.

ACHTUNG :: Mehrfache INSERTs

In DSQL gibt eine Anweisung mit RETURNING immer nur eine Zeile zurück. Wenn die RETURNING-Klausel angegeben ist und mehr als eine Zeile von der INSERT-Anweisung eingefügt wird, schlägt die Anweisung fehl und eine Fehlermeldung wird zurückgegeben. Dieses Verhalten kann sich in zukünftigen Firebird-Versionen ändern.

Beispiele:

```
INSERT INTO Scholars (
  firstname,
  lastname,
  address,
```

```
    phone,  
    email)  
VALUES (  
    'Henry',  
    'Higgins',  
    '27A Wimpole Street',  
    '3231212',  
    NULL)  
RETURNING lastname, fullname, id;  
  
INSERT INTO Dumbbells (firstname, lastname, iq)  
    SELECT fname, lname, iq  
FROM Friends  
    ORDER BY iq ROWS 1  
    RETURNING id, firstname, iq  
INTO :id, :fname, :iq;
```

Hinweise:

- RETURNING wird nur unterstützt für VALUES-Inserts und Singleton- SELECT-Inserts.
- In DSQL gibt eine Anweisung mit einer RETURNING-Klausel *immer* genau eine Zeile zurück. Wenn tatsächlich kein Datensatz eingefügt wurde, sind die Felder in dieser Zeile alle NULL. Dieses Verhalten kann sich in einer späteren Version von Firebird ändern. Wenn in PSQL keine Zeile eingefügt wurde, wird nichts zurückgegeben und die Zielvariablen behalten ihre vorhandenen Werte bei.

Einfügen in BLOB-Spalten

Das Einfügen in BLOB-Spalten ist nur unter folgenden Umständen möglich:

1. Die Client-Anwendung hat mit der Firebird-API spezielle Vorkehrungen für solche Einsätze getroffen. In diesem Fall ist der *modus operandi* anwendungsspezifisch und außerhalb des Geltungsbereichs dieses Handbuchs.
2. Der eingegebene Wert ist eine Textzeichenfolge von nicht mehr als 32767 Byte.

Achtung

Wenn der Wert kein String-Literal ist, achten Sie auf Verkettungen, da die Ausgabe des Ausdrucks die maximale Länge überschreiten kann.

3. Sie nutzen die Form „INSERT ... SELECT“ und eine oder mehr Spalten im Rückgabesatz sind BLOBs.

UPDATE

Verwendet für: Ändern von Zeilen in Tabellen und Sichten

Verfügbar in: DSQL, ESQL, PSQL

Syntax:

```

UPDATE target [[AS] alias]
SET col = newval [, col = newval ...]
[WHERE {search-conditions | CURRENT OF cursorname}]
[PLAN plan_items]
[ORDER BY sort_items]
[ROWS <m> [TO <n>]]
[RETURNING <returning_list> [INTO <variables>]]

<returning_list>      ::=  ret_value [, ret_value ...]
<variables>          ::=  :varname [, :varname ...]
    
```

Tabelle 6.12. Arguments for the UPDATE Statement Parameters

| Argument | Beschreibung |
|-------------------|--|
| target | Der Name der Tabelle oder Sicht, in der die Datensätze aktualisiert werden |
| alias | Alias für den Tisch oder die Ansicht |
| col | Name oder Alias einer Spalte in der Tabelle oder Sicht |
| newval | Neuer Wert für eine Spalte, die von der Anweisung in der Tabelle oder Sicht aktualisiert werden soll |
| search-conditions | Eine Suchbedingung, die den Satz der zu aktualisierenden Zeilen begrenzt |
| cursorname | Der Name des Cursors, über den die zu aktualisierende Zeile(n) positioniert wird |
| plan_items | Klauseln im Abfrageplan |
| sort_items | Spalten, die in einer ORDER BY-Klausel aufgeführt sind |
| m, n | Integer-Ausdrücke zur Begrenzung der Anzahl der zu aktualisierenden Zeilen |
| ret_value | Ein Wert, der in der RETURNING-Klausel zurückgegeben werden soll |
| varname | Name einer lokalen PSQL-Variablen |

Beschreibung: Die Anweisung UPDATE ändert Werte in einer Tabelle oder in einer oder mehreren Tabellen, die einer Sicht zugrunde liegen. Die betroffenen Spalten sind in der Klausel SET angegeben. Die betroffenen Zeilen können durch die Klauseln WHERE und ROWS eingeschränkt sein. Wenn weder WHERE noch ROWS vorhanden ist, werden alle Datensätze in der Tabelle aktualisiert.

Verwendung eines Alias

Wenn Sie einer Tabelle oder einer Ansicht einen Alias zuweisen, *muss* der Alias verwendet werden, wenn Spalten angegeben werden, und auch in Spaltenreferenzen, die in anderen Klauseln enthalten sind.

Beispiele:

Korrekte Verwendung:

```
update Fruit set soort = 'pisang' where ...
```

```
update Fruit set Fruit.soort = 'pisang' where ...
```

```
update Fruit F set soort = 'pisang' where ...
```

```
update Fruit F set F.soort = 'pisang' where ...
```

Nicht möglich:

```
update Fruit F set Fruit.soort = 'pisang' where ...
```

Die SET-Klausel

In der Klausel SET werden die Zuweisungsausdrücke, die die Spalten mit den festzulegenden Werten enthalten, durch Kommata getrennt. In einer Zuweisungsphrase befinden sich Spaltennamen auf der linken Seite und die Werte oder Ausdrücke, die die Zuweisungswerte enthalten, befinden sich auf der rechten Seite. Eine Spalte darf nur einmal in der Klausel SET enthalten sein.

Ein Spaltenname kann in Ausdrücken auf der rechten Seite verwendet werden. Der alte Wert der Spalte wird immer in diesen Werten auf der rechten Seite verwendet, auch wenn der Spalte bereits in der SET-Klausel ein neuer Wert zugewiesen wurde.

Hier ist ein Beispiel: Daten in der TSET-Tabelle:

```
A B
----
1 0
2 0
```

Die Anweisung

```
UPDATE tset SET a = 5, b = a
```

ändert die Werte zu

```
A B
----
5 1
5 2
```

Beachten Sie, dass die alten Werte (1 und 2) verwendet werden, um die Spalte b zu aktualisieren, auch nachdem der Spalte ein neuer Wert zugewiesen wurde (5).

Anmerkung

Dies war nicht immer so. Vor der Version 2.5 erhielten Spalten ihre neuen Werte sofort nach der Zuweisung. Das war jedoch kein standardmäßiges Verhalten, und wurde in Version 2.5 behoben.

Um die Kompatibilität mit Legacy-Code zu erhalten, enthält die Konfigurationsdatei `firebird.conf` den Parameter `OldSetClauseSemantics`, der auf `True` (1) gesetzt werden kann, um das alte, fehlerhafte Verhalten wiederherzustellen. Es ist eine vorübergehende Maßnahme, der Parameter wird in Zukunft entfernt.

Die WHERE-Klausel

Die WHERE-Klausel legt die Bedingungen fest, die die Datensatzgruppe für ein *searched update* begrenzen.

Wenn in PSQL ein benannter Cursor zum Aktualisieren eines Satzes verwendet wird und die Klausel WHERE CURRENT OF verwendet wird, ist die Aktion auf die Zeile beschränkt, in der sich der Cursor gerade befindet. Dies ist ein *positioniertes Update*.

Anmerkung

Die Klausel WHERE CURRENT OF ist nur in PSQL verfügbar, da es keine Anweisung zum Erstellen und Bearbeiten eines expliziten Cursors in DSQL gibt. Gesuchte Updates sind natürlich auch in PSQL verfügbar.

Beispiele:

```
UPDATE People
  SET firstname = 'Boris'
  WHERE lastname = 'Johnson';

UPDATE employee e
  SET salary = salary * 1.05
  WHERE EXISTS(
    SELECT *
      FROM employee_project ep
     WHERE e.emp_no = ep.emp_no);

UPDATE addresses
  SET city = 'Saint Petersburg', citycode = 'PET'
  WHERE city = 'Leningrad'

UPDATE employees
  SET salary = 2.5 * salary
  WHERE title = 'CEO'
```

Bei String-Literalen, mit denen der Parser Hilfe beim Interpretieren des Zeichensatzes der Daten benötigt, kann die [Einführersyntax](#) verwendet werden. Dem Zeichenfolgenliteral ist der Zeichensatzname vorangestellt, dem ein Unterstrich vorangestellt ist:

```
-- notice the '_' prefix

UPDATE People
SET name = _ISO8859_1 'Hans-Jörg Schäfer'
WHERE id = 53662
```

Das Problem mit dem „instabilen Cursor“

In Firebird muss bis zu dieser Version ein Implementierungsfehler geachtet werden, der diese Art von Aktualisierungen betrifft, wenn die WHERE-Bedingungen das IN (*select-expr*) und die *select-expr* in der Form SELECT FIRST *n* oder SELECT ... ROWS vorliegt. Zum Beispiel


```
UPDATE T
  SET ...
  WHERE ID IN (SELECT FIRST 1 ID FROM T)
```

liebevoll als die „unendliche Update-Schleife“ bekannt, wird fortlaufend Zeilen aktualisieren, immer und immer wieder, bis der Server hängen bleibt.

Quarks wie dieses können sich auf datenverändernde DML-Operationen auswirken, meistens wenn die Auswahlbedingungen eine Unterabfrage betreffen. Fälle wurden gemeldet, bei denen die Sortierreihenfolge die Erwartungen beeinträchtigt, ohne dass eine Unterabfrage erforderlich ist. Dies geschieht, weil DML-Anweisungen in den Ausführungsebenen anstelle eines stabilen „Zielsatzes“ und anschließendem Ausführen der Datenänderungen für jedes gesetzte Element implizite Cursor zum Ausführen der Operationen für die Zeile verwenden, die derzeit die Bedingungen erfüllt, ohne zu wissen, ob diese Zeile zuvor die Bedingung nicht erfüllt hat oder bereits aktualisiert wurde. Also, mit einem einfachen Beispielmuster:

```
UPDATE T SET <fields> = <values>
  WHERE <conditions>
```

the execution works as:

```
FOR SELECT <values> FROM T
  WHERE <conditions>
  INTO <tmp_vars> AS CURSOR <cursor>
DO
  UPDATE T SET <fields> = <tmp_vars>
  WHERE CURRENT OF <cursor>
```

Die Implementierung von Firebird stimmt nicht mit den SQL-Standards überein, nach denen ein stabiles Set eingerichtet werden muss, bevor Daten geändert werden. Versionen von Firebird ab V.3 werden dem Standard entsprechen.

Die ORDER BY- und ROWS-Klauseln

Die Klauseln ORDER BY und ROWS sind nur sinnvoll, wenn sie zusammen verwendet werden. Sie können jedoch auch getrennt verwendet werden.

Wenn ROWS ein Argument m hat, sind die zu aktualisierenden Zeilen auf die ersten m Zeilen beschränkt.

Wichtige Punkte:

- Wenn $m >$ der Anzahl der Zeilen, die verarbeitet werden, wird der gesamte Zeilensatz aktualisiert
- Wenn $m = 0$, wird keine Zeile aktualisiert
- Wenn $m < 0$, tritt ein Fehler auf und die Aktualisierung schlägt fehl

Wenn zwei Argumente verwendet werden, m und n , begrenzt ROWS die Zeilen, die aktualisiert werden, auf Zeilen von m bis n einschließlich. Beide Argumente sind Ganzzahlen und beginnen bei 1.

Wichtige Punkte:

- Wenn $m >$ der Anzahl der zu verarbeitenden Zeilen, werden keine Zeilen aktualisiert
- Wenn $n >$ der Anzahl der Zeilen, werden Zeilen von m bis zum Ende des Satzes werden aktualisiert

- Wenn $m < 1$ oder $n < 1$, tritt ein Fehler auf und das Update schlägt fehl
- Wenn $n = m - 1$, werden keine Zeilen aktualisiert.
- If $n < m - 1$, ein Fehler tritt auf und die Aktualisierung schlägt fehl

ROWS-Beispiel:

```
UPDATE employees
SET salary = salary + 50
ORDER BY salary ASC
ROWS 20
```

Die RETURNING-Klausel

Eine Anweisung UPDATE mit *höchstens einer Zeile* kann RETURNING enthalten, um einige Werte aus der aktualisierten Zeile zurückzugeben. RETURNING kann Daten aus jeder Zeile enthalten, nicht zwangsläufig aus der Zeile, die gerade aktualisiert wird. Es kann Literale enthalten, die nicht mit Spalten verknüpft sind, wenn dies erforderlich ist.

Wenn der RETURNING-Satz Daten aus der aktuellen Zeile enthält, melden die zurückgegebenen Werte Änderungen, die in den BEFORE UPDATE Triggern vorgenommen wurden, nicht jedoch in AFTER UPDATE-Triggern.

Die Kontextvariablen OLD.fieldname und NEW.fieldname können als Spaltennamen verwendet werden. Wenn OLD. oder NEW. nicht angegeben ist, sind die zurückgegebenen Spaltenwerte die in NEW..

In DSQL gibt eine Anweisung mit RETURNING immer eine einzelne Zeile zurück. Wenn die Anweisung keine Datensätze aktualisiert, enthalten die zurückgegebenen Werte NULL. Dieses Verhalten kann sich in zukünftigen Firebird-Versionen ändern.

Die INTO-Unterklausel

In PSQL kann die Klausel INTO verwendet werden, um die zurückgegebenen Werte an lokale Variablen zu übergeben. Es ist nicht in DSQL verfügbar. Wenn keine Datensätze aktualisiert werden, wird nichts zurückgegeben, und die in RETURNING angegebenen Variablen behalten ihre vorherigen Werte bei.

Anmerkung

Wenn ein Wert zurückgegeben und einer NEW-Kontextvariablen zugewiesen wird, ist die Verwendung eines Doppelpunktpräfixes nicht zulässig. Dies ist beispielsweise ungültig:

```
...
into :var1, :var2, :new.id
```

und dies gültig:

```
...
into :var1, :var2, new.id
```

RETURNING-Beispiel (DSQL):

```
UPDATE Scholars
SET firstname = 'Hugh', lastname = 'Pickering'
WHERE firstname = 'Henry' and lastname = 'Higgins'
RETURNING id, old.lastname, new.lastname
```

Aktualisieren von BLOB-Spalten

Das Aktualisieren einer BLOB-Spalte ersetzt immer den gesamten Inhalt. Sogar die BLOB-ID, das „Handle“ welches direkt in der Spalte gespeichert wird, ändert sich. BLOBs können aktualisiert werden wenn:

1. Die Client-Anwendung mit der Firebird-API spezielle Vorkehrungen für diese Operation getroffen hat. In diesem Fall ist der *modus operandi* anwendungsspezifisch und außerhalb des Geltungsbereichs dieses Handbuchs.
2. Der neue Wert eine Textzeichenfolge von höchstens 32767 Byte ist. Bitte beachten Sie: Wenn der Wert kein String-Literal ist, achten Sie auf Verkettungen, da diese die maximale Länge überschreiten können.
3. Die Quelle selbst eine BLOB-Spalte oder allgemeiner ein Ausdruck ist, der ein BLOB zurückgibt.
4. Sie die Anweisung INSERT CURSOR verwenden (nur ESQL).

UPDATE OR INSERT

Verwendet für: Updating an existing record in a table or, if it does not exist, inserting it

Verfügbar in: DSQL, PSQL

Syntax:

```
UPDATE OR INSERT INTO
  {target} [(column_list)]
VALUES (value_list)
[MATCHING (column_list)]
[RETURNING values [INTO variables]]

<column_list>      ::= colname [, colname ...]
<value_list>       ::= value  [, value   ...]
<ret_values>       ::= ret_value [, ret_value ...]
<variables>        ::= :varname [, :varname ...]
```

Tabelle 6.13. Arguments for the UPDATE OR INSERT Statement Parameters

| Argument | Beschreibung |
|----------|---|
| target | The name of the table or view where the record[s] is to be updated or a new record inserted |
| colname | Name of a column in the table or view |

| Argument | Beschreibung |
|-----------|---|
| value | An expression whose value is to be used for inserting or updating the table |
| ret_value | An expression returned in the RETURNING clause |
| varname | Variable name—PSQL only |

Beschreibung: UPDATE OR INSERT inserts a new record or updates one or more existing records. The action taken depends on the values provided for the columns in the MATCHING clause (or, if the latter is absent, in the primary key). If there are records found matching those values, they are updated. If not, a new record is inserted.

MATCHING PK IS NOT DISTINCT NULL

Beschränkungen

- If the table has no PK, the MATCHING clause becomes mandatory.
- In the MATCHING list as well as in the update/insert column list, each column name may occur only once.
- The „INTO <variables>“ subclause is only available in PSQL.
- When values are returned into the context variable NEW, this name must not be preceded by a colon (, :“).

The RETURNING clause

The optional RETURNING clause, if present, need not contain all the columns mentioned in the statement and may also contain other columns or expressions. The returned values reflect any changes that may have been made in BEFORE triggers, but not those in AFTER triggers. OLD.*fieldname* and NEW.*fieldname* may both be used in the list of columns to return; for field names not preceded by either of these, the new value is returned.

In DSQL, a statement with a RETURNING clause *always* returns exactly one row. If a RETURNING clause is present and more than one matching record is found, an error is raised. This behaviour may change in a later version of Firebird.

Beispiel: Modifying data in a table, using UPDATE OR INSERT in a PSQL module. The return value is passed to a local variable, whose colon prefix is not optional.

```
UPDATE OR INSERT INTO Cows (Name, Number, Location)
VALUES ('Suzy Creamcheese', 3278823, 'Green Pastures')
MATCHING (Number)
RETURNING rec_id into :id;
```

The „Unstable Cursor“ Problem

Because of the way the execution of data-changing DML is implemented in Firebird, up to and including this version, the sets targeted for updating sometimes produce unexpected results. For more information, refer to [The „Unstable Cursor“ Problem](#) in the UPDATE section.

DELETE

Verwendet für: Löschen von Zeilen aus einer Tabelle oder Ansicht

Verfügbar in: DSQL, ESQL, PSQL

Syntax:

```
DELETE
  FROM {target} [[AS] alias]
  [WHERE {search-conditions | CURRENT OF cursorname}]
  [PLAN plan_items]
  [ORDER BY sort_items]
  [ROWS <m> [TO <n>]]
  [RETURNING <returning_list> [INTO <variables>]]

<m>, <n>      ::= Any expression evaluating to an integer.
<returning_list> ::= ret_value [, ret_value ...]
<variables>  ::= :varname [, :varname ...]
```

Tabelle 6.14. Argumente der DELETE-Statement-Parameter

| Argument | Beschreibung |
|-------------------|--|
| target | Der Name der Tabelle oder Sicht, aus der die Datensätze gelöscht werden sollen |
| alias | Alias für die Zieltabelle oder -ansicht |
| search-conditions | Suchbedingung, die den Satz von Zeilen begrenzt, die zum Löschen vorgesehen sind |
| cursorname | Der Name des Cursors, in dem der aktuelle Datensatz zum Löschen positioniert ist |
| plan_items | Abfrageplanklausel |
| sort_items | ORDER BY-Klausel |
| m, n | Integer-Ausdrücke zum Begrenzen der Anzahl der gelöschten Zeilen |
| ret_value | Ein Ausdruck, der in der RETURNING-Klausel zurückgegeben werden soll |
| varname | Name einer PSQL-Variablen |

Beschreibung: DELETE entfernt Zeilen aus einer Datenbanktabelle oder aus einer oder mehreren Tabellen, die einer Sicht zugrunde liegen. WHERE und ROWS-Klauseln können die Anzahl der gelöschten Zeilen begrenzen. Wenn weder WHERE noch ROWS vorhanden ist, löscht DELETE alle Zeilen in der Beziehung.

Aliases

Wenn für die Zieltabelle oder -sicht ein Alias angegeben ist, muss dieser verwendet werden, um alle Feldnamenreferenzen in der Anweisung DELETE zu qualifizieren.

Beispiele:

Unterstützte Verwendung:

```
delete from Cities where name starting 'Alex'
```

```
delete from Cities where Cities.name starting 'Alex'
```

```
delete from Cities C where name starting 'Alex'
```

```
delete from Cities C where C.name starting 'Alex'
```

Nicht möglich:

```
delete from Cities C where Cities.name starting 'Alex'
```

WHERE

Die WHERE-Klausel legt die Bedingungen fest, die den Satz von Datensätzen für ein *searched delete* begrenzen.

Wenn in PSQL ein benannter Cursor zum Löschen eines Satzes verwendet wird und die Klausel WHERE CURRENT OF verwendet wird, ist die Aktion auf die Zeile beschränkt, in der sich der Cursor gerade befindet. Dies ist ein *positioniertes Update*.

Anmerkung

Die Klausel WHERE CURRENT OF ist nur in PSQL und ESQL verfügbar, da es keine Anweisung zum Erstellen und Bearbeiten eines expliziten Cursors in DSQL gibt. Gesuchte Löschungen sind natürlich auch in PSQL verfügbar.

Beispiele:

```
DELETE FROM People
  WHERE firstname <> 'Boris' AND lastname <> 'Johnson'
```

```
DELETE FROM employee e
  WHERE NOT EXISTS(
    SELECT *
    FROM employee_project ep
    WHERE e.emp_no = ep.emp_no);
```

```
DELETE FROM Cities
  WHERE CURRENT OF Cur_Cities;  -- ESQL and PSQL only
```

PLAN

Eine PLAN-Klausel erlaubt dem Benutzer die Operation manuell zu optimieren.

Beispiel:

```
DELETE FROM Submissions
WHERE date_entered < '1-Jan-2002'
PLAN (Submissions INDEX ix_subm_date);
```

ORDER BY und ROWS

Die Klausel ORDER BY sortiert die Menge vor dem eigentlichen Löschen. Es ist nur in Kombination mit ROWS sinnvoll, aber auch ohne gültig.

Die Klausel ROWS begrenzt die Anzahl der gelöschten Zeilen. Integer-Literale oder beliebige Integer-Ausdrücke können für die Argumente m und n verwendet werden.

Wenn ROWS ein Argument m hat, sind die zu löschenden Zeilen auf die ersten m Zeilen beschränkt.

Wichtige Punkte:

- Wenn $m >$ der Anzahl zu verarbeitender Zeilen, wird der gesamte Zeilensatz gelöscht
- Wenn $m = 0$, werden keine Zeilen gelöscht
- Wenn $m < 0$, tritt ein Fehler auf und die Löschung schlägt fehl

Werden die zwei Argumente m und n verwendet, begrenzt ROWS die Anzahl der zu löschenden Zeilen, inklusive m bis n . Beide Argumente sind Ganzzahlen und beginnen bei 1.

Wichtige Punkte:

- Wenn $m >$ der Anzahl der zu verarbeitenden Zeilen, wird keine Zeile gelöscht
- Wenn $m > 0$ und \leq der Anzahl der Zeilen im Satz und n außerhalb dieser Werte liegt, werden die Zeilen von m bis zum Ende des Satzes gelöscht
- Wenn $m < 1$ oder $n < 1$, wird ein Fehler ausgegeben und das Löschen schlägt fehl
- Wenn $n = m - 1$, werden keine Zeilen gelöscht
- Wenn $n < m - 1$, wird ein Fehler ausgegeben und das Löschen schlägt fehl

Beispiele:

Löschen des ältesten Kaufs:

```
DELETE FROM Purchases
ORDER BY date ROWS 1
```

Löschen der höchsten Kundennummer(n):

```
DELETE FROM Sales
ORDER BY custno DESC ROWS 1 to 10
```

Löschen aller Verkäufe, ORDER BY-Klausel ist sinnlos:

```
DELETE FROM Sales
ORDER BY custno DESC
```

Löschen eines Datensatzes beginnend vom Ende, z.B. von Z...:

```
DELETE FROM popgroups
ORDER BY name DESC ROWS 1
```

Löschen der fünf ältesten Gruppen:

```
DELETE FROM popgroups
ORDER BY formed ROWS 5
```

Es wird keine Sortierung (ORDER BY) angegeben, so dass 8 gefundene Datensätze, beginnend mit dem fünften, gelöscht werden:

```
DELETE FROM popgroups
ROWS 5 TO 12
```

RETURNING

Eine Anweisung DELETE, die *in höchstens einer Zeile* löscht, kann optional eine Klausel RETURNING enthalten, um Werte aus der gelöschten Zeile zurückzugeben. Die Klausel muss, falls vorhanden, nicht alle Spalten der Beziehung enthalten und kann auch andere Spalten oder Ausdrücke enthalten.

Hinweise

- In DSQL gibt eine Anweisung mit RETURNING immer einen Singleton zurück, niemals einen Satz mit mehreren Zeilen. Wenn keine Datensätze gelöscht werden, enthalten die zurückgegebenen Spalten NULL. Dieses Verhalten kann sich in zukünftigen Firebird-Versionen ändern
- Die INTO-Klausel ist nur in PSQL verfügbar
 - Wenn die Zeile nicht gelöscht wird, wird nichts zurückgegeben und die Zielvariablen behalten ihre Werte bei

Beispiele:

```
DELETE FROM Scholars
WHERE firstname = 'Henry' and lastname = 'Higgins'
RETURNING lastname, fullname, id
```

```
DELETE FROM Dumbbells
ORDER BY iq DESC
ROWS 1
RETURNING lastname, iq into :lname, :iq;
```


Das Problem mit dem „instabilen Cursor“

Aufgrund der Art und Weise, in der die Ausführung der datenverändernden DML in Firebird bis einschließlich dieser Version implementiert wird, führen die zum Löschen bestimmten Sätze manchmal zu unerwarteten Ergebnissen. Weitere Informationen finden Sie unter *Das Problem mit dem „instabilen Cursor“ Problem* im Abschnitt UPDATE.

MERGE

Verwendet für: Daten aus einer Quellenmenge in eine Zielbeziehung zusammenführen

Verfügbar in: DSQL, PSQL

Syntax:

```
MERGE INTO target [[AS] target-alias]
  USING source [[AS] source-alias]
  ON join-condition
  WHEN MATCHED THEN UPDATE SET colname = value [, colname = value ...]
  WHEN NOT MATCHED THEN INSERT [(<columns>)] VALUES (<values>)

<columns> ::= colname [, colname ...]
<values>  ::= value  [, value  ...]
```

Tabelle 6.15. Argumente der MERGE-Statement-Parameter

| Argument | Beschreibung |
|-----------------|---|
| target | Name der Zielbeziehung (Tabelle oder änderbare Sicht) |
| source | Datenquelle. Es kann eine Tabelle, eine Ansicht, eine gespeicherte Prozedur oder eine abgeleitete Tabelle sein |
| target-alias | Alias für die Zielbeziehung (Tabelle oder änderbare Sicht) |
| source-alias | Alias für die Quellbeziehung oder den Quellsatz |
| join-conditions | Die (ON) Bedingung(en) zum Abgleich der Quelldatensätze mit denen im Ziel |
| colname | Name einer Spalte in der Zielbeziehung |
| value | Der Wert, der einer Spalte in der Zieltabelle zugewiesen ist. Es ist ein Ausdruck, der ein Literalwert, eine PSQL-Variable, eine Spalte aus der Quelle oder eine kompatible Kontextvariable sein kann |

Beschreibung

Das MERGE-Statement führt Daten in einer Tabelle oder aktualisierbaren Sicht zusammen. Die Quelle kann eine Tabelle, eine Sicht oder ein allgemeines „SELECT FROM“ sein. Jeder Quelldatensatz wird verwendet, um einen oder mehrere Zieldatensätze zu aktualisieren, einen neuen Datensatz in die Zieltabelle einzufügen oder keinen.

Die durchgeführte Aktion hängt von der angegebenen Join-Bedingung und den WHEN-Klauseln ab. Die Bedingung enthält normalerweise einen Vergleich der Felder in den Quell- und Zielbeziehungen.

Hinweise

Nur eine der WHEN-Klauseln kann bereitgestellt werden. Dies wird sich in der nächsten Hauptversion von Firebird ändern, wenn Compound-Matching-Bedingungen unterstützt werden.

WHEN NOT MATCHED wird aus der Quellensicht ausgewertet, d. h. der Tabelle oder dem Satz, der in USING angegeben ist. Die ist zwangsläufig so, da INSERT ausgeführt wird, wenn der Quelldatensatz keinem Zieldatensatz entspricht. Wenn es einen Zieldatensatz gibt, der keinem Quelldatensatz entspricht, wird natürlich nichts unternommen.

Derzeit gibt die Variable ROW_COUNT den Wert 1 zurück, auch wenn mehr als ein Datensatz geändert oder eingefügt wird. Einzelheiten und Fortschritt finden Sie unter [Tracker ticket CORE-4400](#).

ACHTUNG :: Eine weitere Unregelmäßigkeit!

Wenn die Klausel WHEN MATCHED vorhanden ist und mehrere Datensätze einem einzelnen Datensatz in der Zieltabelle entsprechen, wird ein UPDATE für diesen Zieldatensatz je übereinstimmender Quelle ausgeführt, wobei jedes nachfolgende Update das vorherige überschreibt. Dieses Verhalten entspricht nicht dem SQL: 2003-Standard, der erfordert, dass diese Situation eine Ausnahme (einen Fehler) auslöst.

Beispiele:

```
MERGE INTO books b
  USING purchases p
  ON p.title = b.title and p.type = 'bk'
  WHEN MATCHED THEN
    UPDATE SET b.desc = b.desc || ';' || p.desc
  WHEN NOT MATCHED THEN
    INSERT (title, desc, bought) values (p.title, p.desc, p.bought)
```

```
MERGE INTO customers c
  USING (SELECT * from customers_delta WHERE id > 10) cd
  ON (c.id = cd.id)
  WHEN MATCHED THEN
    UPDATE SET name = cd.name
  WHEN NOT MATCHED THEN
    INSERT (id, name) values (cd.id, cd.name)
```

```
MERGE INTO numbers
  USING (
    WITH RECURSIVE r(n) AS (
      SELECT 1 FROM rdb$database
      UNION ALL
      SELECT n+1 FROM r WHERE n < 200
    )
    SELECT n FROM r
  ) t
  ON numbers.num = t.n
  WHEN NOT MATCHED THEN
    INSERT(num) VALUES(t.n);
```

Das Problem mit dem „instabilen Cursor“

Aufgrund der Art und Weise, in der die Ausführung der datenverändernden DML in Firebird bis einschließlich dieser Version implementiert wird, führen die für das Zusammenführen ausgewählten Sets manchmal zu unerwarteten Ergebnissen. Für weitere Informationen vergleichen Sie auch *Das Problem mit dem „instabilen Cursor“* im Abschnitt UPDATE.

EXECUTE PROCEDURE

Verwendet für: Ausführen einer gespeicherten Prozedur

Verfügbar in: DSQL, ESQL, PSQL

Syntax:

```
EXECUTE PROCEDURE procname
  [<inparam> [, <inparam> ...]] | [(<inparam> [, <inparam> ...])]
  [RETURNING_VALUES <outvar> [, <outvar> ...] | (<outvar> [, <outvar> ...])]

<outvar> ::= [:]varname
```

Tabelle 6.16. Argument der EXECUTE PROCEDURE-Statement-Parameter

| Argument | Beschreibung |
|----------|---|
| procname | Name der gespeicherten Prozedur |
| inparam | Ein Ausdruck, der den deklarierten Datentyp eines Eingabeparameters auswertet |
| varname | Eine PSQL-Variable zum Empfangen des Rückgabewerts |

Beschreibung: Führt eine *ausführbare gespeicherte Prozedur* aus, die eine Liste aus einem oder mehreren Eingabeparametern verwendet, wenn sie für die Prozedur definiert sind, und eine einreihige Menge von Ausgabewerten zurückgibt, wenn sie für die Prozedur definiert sind.

„Ausführbare“ gespeicherte Prozedur

Das Statement EXECUTE PROCEDURE wird am häufigsten verwendet, um die Art gespeicherte Prozedur aufzurufen, die geschrieben wird, um eine datenmodifizierende Aufgabe auf der Serverseite auszuführen—das sind die Prozeduren, die kein SUSPEND-Statement im Code enthalten. Sie können so entworfen werden, dass sie eine Ergebnismenge zurückgeben, die nur aus einer Zeile besteht, die normalerweise über eine Gruppe von RETURNING_VALUES()-Variablen an eine andere gespeicherte Prozedur übergeben wird, die sie aufruft. Clientenschnittstellen verfügen normalerweise über einen API-Wrapper, der beim Aufruf von EXECUTE PROCEDURE in DSQL die Ausgabewerte in einen einzeiligen Puffer abrufen kann.

Das Aufrufen der anderen Prozedurart—eine „auswählbare“—ist möglich mit EXECUTE PROCEDURE, aber es gibt nur die erste Zeile eines Ausgabesatzes zurück, der fast sicher als mehrzeilig ausgelegt ist. Auswählbare gespeicherte Prozeduren werden von einer Anweisung SELECT aufgerufen, die eine Ausgabe erzeugt, die sich wie eine virtuelle Tabelle verhält.

Hinweise

- In PSQL und DSQL können Eingabeparameter beliebige Ausdrücke sein, die in den erwarteten Typ aufgelöst werden.
- Obwohl nach dem Namen der gespeicherten Prozedur keine Klammern erforderlich sind, um die Eingabeparameter zu umschließen, wird ihre Verwendung aus Gründen einer guten Verwaltung empfohlen.
- Wenn Ausgabeparameter in einer Prozedur definiert wurden, kann die Klausel RETURNING_VALUES in PSQL verwendet werden, um sie über eine Liste zuvor deklarierter Variablen abzurufen, die mit den definierten Ausgabeparametern in Reihenfolge, Datentyp und Nummer übereinstimmen.
- Die Liste der RETURNING_VALUES kann optional in runde Klammern gesetzt werden und ihre Verwendung wird empfohlen.
- Wenn DSQL-Anwendungen EXECUTE PROCEDURE mithilfe der Firebird-API oder eines Wrappers aufrufen, wird ein Puffer für den Empfang der Ausgabezeile vorbereitet, und die Klausel RETURNING_VALUES wird nicht verwendet.

Beispiele:

In PSQL mit optionalen Doppelpunkten und ohne optionale Klammern:

```
EXECUTE PROCEDURE MakeFullName
  :FirstName, :MiddleName, :LastName
RETURNING_VALUES :FullName;
```

In Firebirds Befehlszeilenprogramm *isql* mit Literalparametern und optionalen Klammern:

```
EXECUTE PROCEDURE MakeFullName ('J', 'Edgar', 'Hoover');
```

Hinweis: In *isql*, wird RETURNING_VALUES nicht verwendet. Alle Ausgabewerte werden von der Anwendung erfasst und automatisch angezeigt.

Ein PSQL-Beispiel mit Ausdrucksparametern und optionalen Klammern:

```
EXECUTE PROCEDURE MakeFullName
  ('Mr./Mrs. ' || FirstName, MiddleName, upper(LastName))
RETURNING_VALUES (FullName);
```

EXECUTE BLOCK

Verwendet für: Erstellen eines „anonymen“ Blocks von PSQL-Code in DSQL zur sofortigen Ausführung

Verfügbar in: DSQL

Syntax:

```

EXECUTE BLOCK [(inparams)]
    [RETURNS (outparams)]
AS
    [declarations]
BEGIN
    [PSQL statements]
END

<i>inparams</i>          ::= <i>param_decl</i> = ? [, <i>inparams</i> ]
<i>outparams</i>       ::= <i>param_decl</i>          [, <i>outparams</i>]
<i>param_decl</i>     ::= <i>paramname</i> <i>type</i> [NOT NULL] [COLLATE <i>collation</i>]
<i>type</i>           ::= <i>datatype</i> | [TYPE OF] <i>domain</i> | TYPE OF COLUMN <i>rel.col</i>

<i>datatype</i> ::=
    {SMALLINT | INTEGER | BIGINT}
    | {FLOAT | DOUBLE PRECISION}
    | {DATE | TIME | TIMESTAMP}
    | {DECIMAL | NUMERIC} [(precision [, scale])]
    | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
    [CHARACTER SET <i>charset</i>]
    | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(size)]
    | BLOB [SUB_TYPE {<i>subtype_num</i> | <i>subtype_name</i>}]
    [SEGMENT SIZE <i>seglen</i>] [CHARACTER SET <i>charset</i>]
    | BLOB [(<i>seglen</i> [, <i>subtype_num</i>])]

<i>declarations</i> ::= <i>declare_item</i> [<i>declare_item ...</i>]
<i>declare_item</i> ::= <i>declare_var</i> | <i>declare_cursor</i>

```

Tabelle 6.17. Argument der EXECUTE BLOCK-Statement-Parameter

| Argument | Beschreibung |
|----------------|--|
| param_decl | Name und Beschreibung eines Eingabe- oder Ausgabeparameters |
| declarations | Ein Abschnitt zum Deklarieren von lokalen Variablen und benannten Cursors |
| declare_var | Lokale Variablendeklaration |
| declare_cursor | Deklaration eines benannten Cursors |
| paramname | Der Name eines Eingabe- oder Ausgabeparameters des prozeduralen Blocks mit bis zu 31 Zeichen. Der Name muss unter Ein- und Ausgabeparametern und lokalen Variablen im Block eindeutig sein |
| datatype | SQL-Datentyp |
| collation | Sortierreihenfolge |
| domain | Domain |
| rel | Name einer Tabelle oder Ansicht |
| col | Name einer Spalte in einer Tabelle oder Sicht |
| precision | Präzision. Von 1 bis 18 |
| scale | Rahmen. Von 0 bis 18. Es muss kleiner oder gleich <i>precision</i> sein |

| Argument | Beschreibung |
|--------------|--|
| size | Die maximale Größe einer Zeichenfolge in Zeichen |
| charset | Zeichensatz |
| subtype_num | BLOB-Subtypennummer |
| subtype_name | mnemotechnischer Name des BLOB-Subtyps |
| seglen | Segmentgröße, kann nicht größer als 65.535 sein |

Beschreibung: Führt einen PSQL-Block aus, als wäre es eine gespeicherte Prozedur, optional mit Eingabe- und Ausgabeparametern und Variablendeklarationen. Dies ermöglicht dem Benutzer „on-the-fly“ PSQL innerhalb eines DSQL-Kontexts auszuführen.

Beispiele:

In diesem Beispiel werden die Zahlen 0 bis 127 und die entsprechenden ASCII-Zeichen in die Tabelle ASCII_TABLE eingefügt:

```
EXECUTE BLOCK
AS
declare i INT = 0;
BEGIN
  WHILE (i < 128) DO
  BEGIN
    INSERT INTO AsciiTable VALUES (:i, ascii_char(:i));
    i = i + 1;
  END
END
```

Das nächste Beispiel berechnet das geometrische Mittel zweier Zahlen und gibt es an den Benutzer zurück:

```
EXECUTE BLOCK (x DOUBLE PRECISION = ?, y DOUBLE PRECISION = ?)
RETURNS (gmean DOUBLE PRECISION)
AS
BEGIN
  gmean = SQRT(x*y);
  SUSPEND;
END
```

Da dieser Block Eingabeparameter hat, muss er zuerst vorbereitet werden. Dann können die Parameter eingestellt und der Block ausgeführt werden. Es hängt von der Client-Software ab, wie dies durchgeführt werden muss. Auch wenn dies möglich ist—beachten Sie die folgenden Hinweise.

Unser letztes Beispiel nimmt zwei ganzzahlige Werte, `smallest` und `largest`. Für alle Zahlen im Bereich `smallest ... largest` gibt der Block die Zahl selbst, sein Quadrat, seine dritte und seine vierte Potenz aus.

```
EXECUTE BLOCK (smallest INT = ?, largest INT = ?)
RETURNS (number INT, square BIGINT, cube BIGINT, fourth BIGINT)
AS
BEGIN
  number = smallest;
  WHILE (number <= largest) DO
  BEGIN
    square = number * number;
    cube   = number * square;
    fourth = number * cube;
    SUSPEND;
    number = number + 1;
  END
END
```

Auch hier hängt es von der Client-Software ab, ob und wie Sie die Parameterwerte einstellen können.

Eingabe- und Ausgabeparameter

Das Ausführen eines Blocks ohne Eingabeparameter sollte mit jedem Firebird-Client möglich sein, der es dem Benutzer erlaubt, seine eigenen DSQL-Anweisungen einzugeben. Wenn es Eingabeparameter gibt, werden die Dinge komplizierter: Diese Parameter müssen ihre Werte erhalten, nachdem die Anweisung vorbereitet wurde, aber bevor sie ausgeführt wird. Dies erfordert spezielle Voraussetzungen, die nicht jede Client-Anwendung bietet. (Firebirds eigenes isql, zum Beispiel, nicht.)

Der Server akzeptiert nur Fragezeichen („?“) als Platzhalter für die Eingabewerte, also nicht „:a“, „:MyParam“ etc., oder Literalwerte. Client-Software unterstützt möglicherweise die Form „:xxx“ und wandelt es vor dem Senden an den Server um.

Wenn der Block Ausgabeparameter hat, *müssen* Sie SUSPEND verwenden, sonst wird nichts zurückgegeben.

Die Ausgabe erfolgt immer in Form einer Ergebnismenge, genau wie bei einer Anweisung mit SELECT. Sie können weder RETURNING_VALUES verwenden noch einen Block mit INTO in Variablen anwenden, auch wenn es nur eine Ergebniszeile gibt.

PSQL Links

Weitere Informationen zu Parameter- und Variablendeklarationen und *<PSQL-Statements>* liefert Ihnen Kapitel 7, *Prozedurale SQL-Anweisungen (PSQL)*.

Für *<Deklarationen>* im Speziellen, vgl. [DECLARE \[VARIABLE\]](#) und [DECLARE CURSOR](#) für die exakte Syntax.

Statement-Terminatoren

Einige SQL-Statement-Editoren—insbesondere das *isql*-Dienstprogramm, das mit Firebird geliefert wird, und möglicherweise einige Editoren von Drittanbietern—verwenden eine interne Konvention, die erfordert, dass alle

Anweisungen mit einem Semikolon abgeschlossen werden. Dies erzeugt einen Konflikt mit der PSQL-Syntax beim Codieren in diesen Umgebungen. Wenn Sie mit diesem Problem und seiner Lösung nicht vertraut sind, lesen Sie die Details im Kapitel PSQL im Abschnitt [Umschalten des Terminators in *isql*](#).

Prozedurale SQL-Anweisungen (PSQL)

Prozedurales SQL (PSQL) ist eine prozedurale Erweiterung von SQL. Diese Sprachuntermenge wird zum Schreiben von gespeicherten Prozeduren, Triggern und PSQL-Blöcken verwendet.

PSQL bietet alle grundlegenden Konstrukte traditioneller strukturierter Programmiersprachen und enthält auch DML-Anweisungen (SELECT, INSERT, UPDATE, DELETE usw.), in einigen Fällen mit geringfügigen Änderungen der Syntax.

Elemente der PSQL

Eine prozedurale Erweiterung kann Deklarationen von lokalen Variablen und Cursors, Zuweisungen, bedingten Anweisungen, Schleifen, Anweisungen zum Abrufen von benutzerdefinierten Ausnahmen, Fehlerbehandlung und Senden von Nachrichten (Ereignissen) an Clientanwendungen enthalten. Trigger haben Zugriff auf spezielle Kontextvariablen, zwei Arrays, die die NEW-Werte für alle Spalten während der Einfüge- und Aktualisierungsaktivität bzw. die OLD-Werte während der Aktualisierungs- und Löscharbeiten speichern.

Anweisungen, die Metadaten ändern (DDL), sind in PSQL nicht verfügbar.

DML-Anweisungen mit Parametern

Wenn DML-Anweisungen (SELECT, INSERT, UPDATE, DELETE usw.) im Rumpf des Moduls (Prozedur, Trigger oder Block) Parameter verwenden, können nur benannte Parameter verwendet werden und sie müssen „existieren“ bevor die Anweisung diese verwenden kann. Sie können verfügbar gemacht werden, indem sie entweder als Ein- oder Ausgabeparameter im Header des Moduls oder als lokale Variablen in DECLARE [VARIABLE]-Anweisungen im unteren Headerbereich deklariert werden.

Wenn eine DML-Anweisung mit Parametern im PSQL-Code enthalten ist, muss dem Parameternamen in den meisten Situationen ein Doppelpunkt („:“) vorangestellt werden. Der Doppelpunkt ist in PSQL-spezifischer Anweisungssyntax wie Zuweisungen und Bedingungen optional. Das Doppelpunktpräfix für Parameter ist nicht erforderlich, wenn gespeicherte Prozeduren von einem anderen PSQL-Modul oder in DSQL aufgerufen werden.

Transaktionen

Gespeicherte Prozeduren werden im Kontext der Transaktion ausgeführt, in der sie aufgerufen werden. Trigger werden als ein intrinsischer Teil der Operation der DML-Anweisung ausgeführt: ihre Ausführung befindet sich

also innerhalb des gleichen Transaktionskontextes wie die Anweisung selbst. Einzelne Transaktionen werden für Datenbank-Trigger gestartet.

Anweisungen, die Transaktionen starten und beenden, sind in PSQL nicht verfügbar, aber es ist möglich, eine Anweisung oder einen Anweisungsblock in einer autonomen Transaktion auszuführen.

Modulstruktur

PSQL-Codemodule bestehen aus einem Header und einem Body. Die DDL-Anweisungen zum Definieren dieser sind *komplexe Anweisungen*; das heißt, sie sind Bestandteile einer einzigen Anweisung, die Blöcke von mehreren Anweisungen umfasst. Diese Anweisungen beginnen mit einem Verb (CREATE, ALTER, DROP, RECREATE, CREATE oder ALTER) und enden mit die letzten END-Anweisung des Bodys.

Der Modul-Header

Der Header gibt den Modulnamen an und definiert alle Parameter und Variablen, die im Rumpf verwendet werden. Gespeicherte Prozeduren und PSQL-Blöcke können Ein- und Ausgangsparameter haben. Trigger haben keine Ein- oder Ausgangsparameter.

Der Header eines Triggers zeigt das Datenbankereignis (Einfügen, Aktualisieren oder Löschen oder eine Kombination) und die Betriebsphase (VOR oder NACH diesem Ereignis) an, die dazu führt, dass dieser „ausgelöst wird“.

Der Modul-Body

Der Rumpf eines PSQL-Moduls ist ein Block von Anweisungen, die wie ein Programm in einer logischen Reihenfolge ablaufen. Ein Anweisungsblock ist in einer BEGIN- und einer END-Anweisung enthalten. Der Hauptblock BEGIN ... END kann beliebig viele andere BEGIN ... END-Blöcke enthalten, sowohl eingebettete als auch sequenzielle. Alle Anweisungen außer BEGIN und END werden durch Semikolons (;) abgeschlossen. Kein anderes Zeichen ist als Terminator für PSQL-Anweisungen gültig.

Umschalten des Terminators in *isql*

Hier werden wir ein wenig abschweifen, um zu erklären, wie man das Terminatorzeichen im Dienstprogramm *isql* umschaltet, um es zu ermöglichen, PSQL-Module in dieser Umgebung zu definieren, ohne mit *isql* selbst in Konflikt zu geraten, da *isql* dasselbe Zeichen, Semikolon (;), als eigenen Anweisungsabschluss verwendet.

isql-Befehl **SET TERM**:

Verwendet für: Ändern des Terminatorzeichens, um Konflikte mit dem Terminatorzeichen in PSQL-Anweisungen zu vermeiden

Verfügbar in: nur in ISQL

Syntax:

```
SET TERM <new_terminator> <old_terminator>
```

Tabelle 7.1. SET TERM-Parameter

| Argument | Beschreibung |
|----------------|------------------|
| new_terminator | Neuer Terminator |
| old_terminator | Alter Terminator |

Wenn Sie Ihre Trigger und gespeicherten Prozeduren in *isql* schreiben, entweder in der interaktiven Schnittstelle oder in Skripten, müssen Sie eine SET TERM-Anweisung ausführen, um das normale *isql*-Anweisungsterminal vom Semikolon zu einem anderen Zeichen oder einer kurzen Zeichenfolge umzuschalten. Hierdurch werden Konflikte mit dem nicht änderbaren Semikolon-Terminator in PSQL zu vermeiden. Der Wechsel zu einem alternativen Terminator muss durchgeführt werden, bevor Sie beginnen, PSQL-Objekte zu definieren oder Ihre Skripte auszuführen.

Der alternative Terminator kann eine beliebige Zeichenkette sein, mit Ausnahme eines Leerzeichens, eines Apostrophs oder des aktuellen Terminatorzeichens. Bei jedem Buchstabenzeichen wird zwischen Groß- und Kleinschreibung unterschieden.

Beispiel: Ändern Sie das Standard-Semikolon in '^' (Caret) und verwenden Sie es, um eine Stored Procedure-Definition zu übergeben: Zeichen als alternatives Terminatorzeichen:

```
SET TERM ^;

CREATE OR ALTER PROCEDURE SHIP_ORDER (
    PO_NUM CHAR(8))
AS
BEGIN
    /* Stored procedure body */
END^

/* Other stored procedures and triggers */

SET TERM ;^

/* Other DDL statements */
```

Gespeicherte Prozeduren

Eine gespeicherte Prozedur ist ein Programm, das in den Datenbankmetadaten zur Ausführung auf dem Server gespeichert ist. Eine gespeicherte Prozedur kann durch gespeicherte Prozeduren (einschließlich sich selbst), Trigger und Clientanwendungen aufgerufen werden. Eine Prozedur, die sich selbst aufruft, heißt *rekursiv*.

Vorteile von gespeicherten Prozeduren

Gespeicherten Prozeduren besitzen die folgenden Vorteile:

1. Modularität—Anwendungen, die mit der Datenbank arbeiten, können die gleiche gespeicherte Prozedur verwenden, wodurch die Größe des Anwendungscodes reduziert wird und eine Codeduplizierung vermieden wird.
2. Vereinfachte Anwendungsunterstützung—Wenn eine gespeicherte Prozedur geändert wird, werden Änderungen sofort allen Host-Anwendungen angezeigt, ohne dass sie bei unveränderten Parametern neu kompiliert werden müssen.
3. Verbesserte Leistung—Da gespeicherte Prozeduren auf einem Server statt auf dem Client ausgeführt werden, wird der Netzwerkverkehr reduziert, wodurch die Leistung verbessert wird.

Varianten der gespeicherten Prozeduren

Firebird unterstützt zwei Arten der gespeicherten Prozeduren: *executable* (ausführbar) *selectable* (abfragbar).

Ausführbare Prozeduren

Ausführbare Prozeduren ändern normalerweise Daten in einer Datenbank. Sie können Eingabeparameter empfangen und einen einzigen Satz von Ausgabeparametern (RETURNS) zurückgeben. Sie werden mit der Anweisung EXECUTE PROCEDURE aufgerufen. Siehe auch [ein Beispiel für eine ausführbare gespeicherte Prozedur](#) am Ende des [Abschnitts CREATE PROCEDURE](#) von Kapitel 5.

Abfragbare Prozeduren

Abfragbare bzw. auswählbare gespeicherte Prozeduren rufen normalerweise Daten aus einer Datenbank ab und geben eine beliebige Anzahl von Zeilen an den Aufrufer zurück. Der Aufrufer erhält die Ausgabe Zeile für Zeile aus einem Zeilenpuffer, der von der Datenbank-Engine darauf vorbereitet wird.

Auswählbare Prozeduren können nützlich sein, um komplexe Datensätze zu erhalten, die mit regulären DSQL SELECT-Abfragen oft nicht oder nur schwer oder zu langsam abgerufen werden können. Typischerweise iteriert diese Art der Prozedur durch einen Schleifenprozess des Extrahierens von Daten, wobei sie möglicherweise transformiert wird, bevor die Ausgangsvariablen (Parameter) bei jeder Iteration der Schleife mit frischen Daten gefüllt werden. Eine SUSPEND-Anweisung am Ende der Iteration füllt den Puffer und wartet darauf, dass der Aufrufer die Zeile abfragt. Die Ausführung der nächsten Iteration der Schleife beginnt, wenn der Puffer gelöscht wurde.

Auswählbare Prozeduren können Eingabeparameter haben, und der Ausgabesatz wird durch die Klausel RETURNS im Header angegeben.

Eine wählbare gespeicherte Prozedur wird mit einer SELECT-Anweisung aufgerufen. Siehe auch [ein Beispiel für eine abfragbare gespeicherte Prozedur](#) am Ende des [Abschnitts CREATE PROCEDURE](#) von Kapitel 5.

Erstellen einer gespeicherten Prozedur

Die Syntax zum Erstellen ausführbarer gespeicherter Prozeduren und wählbarer gespeicherter Prozeduren ist exakt gleich. Der Unterschied liegt in der Logik des Programmcodes.

Syntax (partiell):

```
CREATE PROCEDURE procname
[(<inparam> [, <inparam> ...])]
[RETURNS (<outparam> [, <outparam> ...])]
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END
```

Der Header einer gespeicherten Prozedur muss den Prozedurnamen enthalten und muss unter den Namen gespeicherter Prozeduren, Tabellen und Ansichten eindeutig sein. Es kann auch einige Ein- und Ausgabeparameter definieren. Eingabeparameter werden nach dem Prozedurnamen in Klammern angegeben. Ausgabeparameter, die für auswählbare Prozeduren obligatorisch sind, sind innerhalb einer Klausel RETURNS eingeklammert.

Das letzte Element im Header (oder das erste Element im Textkörper, abhängig von Ihrer Ansicht darüber, wo die Grenze liegt) umfasst eine oder mehrere Deklarationen von lokalen Variablen und / oder benannten Cursors, die Ihre Prozedur möglicherweise erfordert.

Nach den Deklarationen folgt der Hauptblock BEGIN ... END, der den PSQL-Code der Prozedur beschreibt. Innerhalb dieses Blocks könnten PSQL- und DML-Anweisungen, Ablaufsteuerungsblöcke, Sequenzen anderer BEGIN ... END-Blöcke einschließlich eingebetteter Blöcke sein. Blöcke, einschließlich des Hauptblocks, können leer sein und die Prozedur wird trotzdem kompiliert. Es ist nicht ungewöhnlich, ein Verfahren in Stufen aus einer Gliederung zu entwickeln.

Weitere Informationen zum Erstellen gespeicherter Prozeduren: Siehe auch [CREATE PROCEDURE](#) in Kapitel 5, *Data Definition (DDL) Statements*.

Anpassen einer gespeicherten Prozedur

Eine vorhandene gespeicherte Prozedur kann geändert werden, um die Sätze von Ein- und Ausgabeparametern und alles im Prozedurhauptteil zu ändern.

Syntax (partiell):

```
ALTER PROCEDURE procname
[(<inparam> [, <inparam> ...])]
```

```
[RETURNS (<outparam> [, <outparam> ...])]  
AS  
[<declarations>]  
BEGIN  
[<PSQL_statements>]  
END
```

Weitere Informationen zum Ändern gespeicherter Prozeduren: Siehe auch [ALTER PROCEDURE](#), [CREATE OR ALTER PROCEDURE](#), [RECREATE PROCEDURE](#), in Kapitel 5, *Data Definition (DDL) Statements*.

Löschen einer gespeicherte Prozedur

Die Anweisung DROP PROCEDURE wird verwendet um gespeicherte Prozeduren zu löschen.

Syntax (vollständig):

```
DROP PROCEDURE procname;
```

Weitere Informationen zum Löschen gespeicherter Prozeduren: See [DROP PROCEDURE](#) in Kapitel 5, *Data Definition (DDL) Statements*.

Gespeicherte Funktionen (Stored Functions)

Gespeicherte PSQL-Skalarfunktionen werden in dieser Version nicht unterstützt, sie kommen jedoch in Firebird 3. In Firebird 2.5 und niedriger können Sie stattdessen eine abfragbare gespeicherte Prozedur schreiben, die ein Skalarergebnis zurückgibt, und SELECT aus Ihrer DML-Abfrage oder Unterabfrage.

Beispiel:

```
SELECT  
  PSQL_FUNC(T.col1, T.col2) AS col3,  
  col3  
FROM T
```

kann ersetzt werden durch

```
SELECT  
  (SELECT output_column FROM PSQL_PROC(T.col1)) AS col3,  
  col2  
FROM T
```

oder

```
SELECT  
  output_column AS col3,
```

```
col2,
FROM T
LEFT JOIN PSQL_PROC(T.col1)
```

PSQL-Blöcke

Ein in sich abgeschlossener, unbenannter („anonymous“) Block von PSQL-Code kann dynamisch in DSQL unter Verwendung der Syntax EXECUTE BLOCK ausgeführt werden. Der Header eines anonymen PSQL-Blocks kann optional Eingabe- und Ausgabeparameter enthalten. Der Körper kann lokale Variablen und Cursordeklarationen enthalten. Ein Block von PSQL-Anweisungen folgt.

Ein anonymes PSQL-Block wird nicht definiert und als Objekt gespeichert, im Gegensatz zu gespeicherten Prozeduren und Triggern. Er wird zur Laufzeit ausgeführt und kann nicht auf sich selbst verweisen.

Genau wie gespeicherte Prozeduren können anonyme PSQL-Blöcke verwendet werden, um Daten zu verarbeiten und Daten aus der Datenbank abzurufen.

Syntax (unvollständig):

```
EXECUTE BLOCK
[(<inparam> = ? [, <inparam> = ? ...])]
[RETURNS (<outparam> [, <outparam> ...])]
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END
```

Tabelle 7.2. PSQL Block Parameters

| Argument | Beschreibung |
|-----------------|--|
| inparam | Beschreibung der Eingabeparameter |
| outparam | Beschreibung der Ausgangsparameter |
| declarations | Ein Abschnitt zum Deklarieren lokaler Variablen und benannter Cursor |
| PSQL statements | PSQL- und DML-Anweisungen |

Weiterlesen: Siehe auch [EXECUTE BLOCK](#) für weitere Details.

Trigger

Ein Trigger ist eine andere Form von ausführbarem Code, der in den Metadaten der Datenbank zur Ausführung durch den Server gespeichert wird. Ein Trigger kann nicht direkt aufgerufen werden. Er wird automatisch aufgerufen („gefeuert“), wenn Datenänderungsereignisse mit einer bestimmten Tabelle oder Sicht (View) auftreten.

Ein Trigger gilt für genau eine Tabelle oder Sicht und nur eine *Phase* in einem Ereignis (VOR oder NACH dem Ereignis). Ein einzelner Trigger kann nur dann ausgelöst werden, wenn ein bestimmtes Datenänderungsereignis auftritt (INSERT / UPDATE / DELETE) oder wenn es auf mehr als eines dieser Ereignisse angewendet werden soll.

Ein DML-Trigger wird im Kontext der Transaktion ausgeführt, in der die datenändernde DML-Anweisung ausgeführt wird. Bei Triggern, die auf Datenbankereignisse reagieren, ist die Regel unterschiedlich: Für einige von ihnen wird eine Standardtransaktion gestartet.

Reihenfolge der Ausführung

Für jede Phase-Ereignis-Kombination kann mehr als ein Trigger definiert werden. Die Reihenfolge, in der sie ausgeführt werden (bekannt als „firing order“, kann explizit mit dem optionalen Argument POSITION in der Triggerdefinition angegeben werden.) Sie haben 32.767 Nummern zur Auswahl. Die niedrigsten Positionsnummern feuern zuerst.

Wenn eine Klausel POSITION weggelassen wird oder mehrere übereinstimmende Ereignisphasen-Trigger die gleiche Positionsnummer haben, werden die Trigger in alphabetischer Reihenfolge ausgelöst.

DML-Trigger

DML-Trigger sind solche, die ausgelöst werden, wenn eine DML-Operation den Datenstatus ändert: Zeilen in Tabellen ändern, neue Zeilen einfügen oder Zeilen löschen. Sie können sowohl für Tabellen als auch für Ansichten definiert werden.

Trigger-Optionen

Für die Ereignis-Phasen-Kombination für Tabellen und Ansichten stehen sechs Basisoptionen zur Verfügung:

| | |
|---|-----------------|
| Bevor eine neue Zeile eingefügt wird | (BEFORE INSERT) |
| Nachdem eine neue Zeile eingefügt wurde | (AFTER INSERT) |
| Bevor eine Zeile aktualisiert wird | (BEFORE UPDATE) |
| Nachdem eine Zeile aktualisiert wurde | (AFTER UPDATE) |
| Bevor eine Zeile gelöscht wird | (BEFORE DELETE) |
| Nachdem eine Zeile gelöscht wurde | (AFTER DELETE) |

Diese Basisformulare dienen zum Erstellen von Einzelphasen- / Einzelereignisauslösern. Firebird unterstützt auch Formulare zum Erstellen von Auslösern für eine Phase und mehrere Ereignisse, z. B. BEFORE INSERT ODER UPDATE ODER DELETE, oder AFTER UPDATE ODER DELETE: Die Kombinationen unterliegen Ihrer Wahl.

Anmerkung

„Multiphasen“-Trigger, wie BEFORE OR AFTER..., sind nicht möglich.

Kontextvariablen OLD und NEW

Für DML-Trigger bietet die Firebird-Engine Zugriff auf Sätze von OLD- und NEW-Kontextvariablen. Jedes ist ein Array der Werte der gesamten Zeile: eine für die Werte, wie sie vor dem Datenänderungsereignis sind (die BEFORE-Phase) und eine für die Werte, wie sie nach dem Ereignis sein werden (die AFTER-Phase). Sie werden in Anweisungen referenziert, die das Formular NEW.column_name bzw. OLD.column_name verwenden. column_name kann eine beliebige Spalte in der Definition der Tabelle sein, nicht nur die, die gerade aktualisiert wird.

Die Variablen NEW und OLD unterliegen einigen Regeln:

- In allen Triggern ist der OLD-Wert schreibgeschützt
- In BEFORE UPDATE und BEFORE INSERT Code wird der NEW-Wert gelesen / geschrieben, sofern es sich nicht um eine COMPUTED BY-Spalte handelt
- In INSERT-Triggern sind Verweise auf die OLD-Variablen ungültig und lösen eine Ausnahme aus
- In DELETE-Triggern sind Verweise auf die NEW-Variablen ungültig und lösen eine Ausnahme aus
- In allen AFTER-Triggercodes sind die NEW-Variablen schreibgeschützt

Datenbank-Trigger

Ein mit einer Datenbank oder einem Transaktionsereignis verknüpfter Trigger kann für die folgenden Ereignisse definiert werden:

| | | |
|---|---------------------------|--|
| Verbindung mit einer Datenbank herstellen | (ON CONNECT) | Bevor der Trigger ausgeführt wird, wird automatisch eine Standardtransaktion gestartet |
| Trennen von einer Datenbank | (ON DISCONNECT) | Bevor der Trigger ausgeführt wird, wird automatisch eine Standardtransaktion gestartet |
| Wenn eine Transaktion gestartet wird | (ON TRANSACTION START) | Der Trigger wird im aktuellen Transaktionskontext ausgeführt |
| Wenn eine Transaktion übergeben wird | (ON TRANSACTION COMMIT) | Der Trigger wird im aktuellen Transaktionskontext ausgeführt |
| Wenn eine Transaktion abgebrochen wird | (ON TRANSACTION ROLLBACK) | Der Trigger wird im aktuellen Transaktionskontext ausgeführt |

Trigger erstellen

Syntax:

```
CREATE TRIGGER trigname {
    <relation_trigger_legacy>
    | <relation_trigger_sql2003>
    | <database_trigger> }
```

```

AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END

<relation_trigger_legacy> ::= FOR {tablename | viewname}
[ACTIVE | INACTIVE]
{BEFORE | AFTER} <mutation_list>
[POSITION number]

<relation_trigger_sql2003> ::= [ACTIVE | INACTIVE]
{BEFORE | AFTER} <mutation_list>
[POSITION number]
ON {tablename | viewname}

<database_trigger> ::= [ACTIVE | INACTIVE]
ON db_event
[POSITION number]

<mutation_list> ::= <mutation> [OR <mutation>
    [OR <mutation>]]

<mutation> ::= { INSERT | UPDATE | DELETE }

<db_event> ::=
    CONNECT
    | DISCONNECT
    | TRANSACTION START
    | TRANSACTION COMMIT
    | TRANSACTION ROLLBACK
    
```

Der Header muss einen Namen für den Trigger enthalten, der unter den Triggernamen eindeutig ist. Er muss das Ereignis oder die Ereignisse enthalten, die den Auslöser auslösen. Für einen DML-Trigger müssen Sie außerdem die Ereignisphase und den Namen der Tabelle oder Ansicht angeben, die den Trigger „besitzen“ soll.

Der Rumpf des Triggers kann durch die Deklarationen von lokalen Variablen und Cursors, falls vorhanden, geleitet werden. Innerhalb des umschließenden Hauptblocks von BEGIN ... END befinden sich ein oder mehrere Blöcke von PSQL-Anweisungen, die leer sein können.

Weitere Informationen zum Erstellen von Triggern: Siehe "[CREATE TRIGGER](#)" in Kapitel 5, *Data Definition (DDL) Statements*.

Trigger ändern

Das Ändern der Status-, Phasen-, Tabellen- oder Ansichtereignisse, der Auslöseposition und des Codes im Rumpf eines DML-Triggers ist möglich. Sie können jedoch einen DML-Trigger nicht ändern, um ihn in einen Datenbank-Trigger zu konvertieren, und umgekehrt. Jedes nicht angegebene Element wird von ALTER TRIGGER nicht geändert. Die alternativen Anweisungen CREATE OR ALTER TRIGGER und RECREATE TRIGGER ersetzen die ursprüngliche Triggerdefinition vollständig.

Syntax:

```
ALTER TRIGGER trigname
```

```
[ACTIVE | INACTIVE]
[ {BEFORE | AFTER} <mutation_list> ]
[ POSITION number ]
[
  AS
  [<declarations>]
  BEGIN
  [<PSQL_statements>]
  END
]

<mutation_list> ::=
  <mutation> [OR <mutation> [OR <mutation>]]

<mutation> ::= { INSERT | UPDATE | DELETE }

<db_event> ::=
  CONNECT
  | DISCONNECT
  | TRANSACTION START
  | TRANSACTION COMMIT
  | TRANSACTION ROLLBACK
```

Weitere Informationen zum Ändern von Triggern: Siehe [ALTER TRIGGER](#), [CREATE OR ALTER TRIGGER](#), [RECREATE TRIGGER](#) in Kapitel 5, *Data Definition (DDL) Statements*.

Trigger löschen

Die Anweisung DROP TRIGGER dient zum Löschen von Triggern.

Syntax (vollständig):

```
DROP TRIGGER trigname;
```

Weitere Informationen zum Löschen von Triggern: Siehe [DROP TRIGGER](#) in Kapitel 5, *Data Definition (DDL) Statements*.

Schreiben des Body-Codes

In diesem Abschnitt werden die prozeduralen SQL-Sprachkonstrukte und -Anweisungen näher betrachtet, die zum Codieren des Rumpfs einer gespeicherten Prozedur, eines Triggers oder eines anonymen PSQL-Blocks verfügbar sind.

Doppelpunkt-Markierungspräfix (:)

Das Doppelpunkt-Markierungspräfix (:) wird in PSQL verwendet, um einen Verweis auf eine Variable in einer DML-Anweisung zu markieren. Der Doppelpunkt-Marker ist vor Variablennamen in anderem Code nicht erforderlich und sollte niemals auf Kontextvariablen angewendet werden.

Zuweisungs-Statements

Verwendet für: Zuweisen eines Werts zu einer Variablen

Verfügbar in: PSQL

Syntax:

```
varname = <value_expr>
```

Tabelle 7.3. Zuweisungs-Statement-Parameter

| Argument | Beschreibung |
|------------|---|
| varname | Name eines Parameters oder einer lokalen Variablen |
| value_expr | Ein Ausdruck, eine Konstante oder eine Variable, dessen Wert in den gleichen Datentyp wie <varname> |

PSQL verwendet das Äquivalenzsymbol (=) als Zuweisungsoperator. Die Zuweisungsanweisung weist der Variablen links vom Operator den rechten SQL-Ausdruckswert zu. Der Ausdruck kann ein beliebiger gültiger SQL-Ausdruck sein: Er kann Literale, interne Variablenamen, Arithmetik-, logische und Zeichenfolgenoperationen, Aufrufe von internen Funktionen oder externe Funktionen (UDFs) enthalten.

Beispiel mit Zuweisungsanweisungen:

```
CREATE PROCEDURE MYPROC (
  a INTEGER,
  b INTEGER,
  name VARCHAR (30)
)
RETURNS (
  c INTEGER,
  str VARCHAR(100))
AS
BEGIN
  -- assigning a constant
  c = 0;
  str = '';
  SUSPEND;
  -- assigning expression values
  c = a + b;
  str = name || CAST(b AS VARCHAR(10));
  SUSPEND;
  -- assigning expression value
  -- built by a query
  c = (SELECT 1 FROM rdb$database);
  -- assigning a value from a context variable
  str = CURRENT_USER;
  SUSPEND;
```

END

Siehe auch: [DECLARE VARIABLE](#)

DECLARE CURSOR

Verwendet für: Deklarieren eines benannten Cursors

Verfügbar in: PSQL

Syntax:

```
DECLARE [VARIABLE] cursorname CURSOR FOR (<select>) [FOR UPDATE]
```

Tabelle 7.4. DECLARE CURSOR-Statement-Parameter

| Argument | Beschreibung |
|------------|------------------|
| cursorname | Name des Cursors |
| select | SELECT-Statement |

Die Anweisung `DECLARE CURSOR ... FOR` bindet einen benannten Cursor an die Ergebnismenge, die in der in der Klausel `FOR` angegebenen `SELECT`-Anweisung ermittelt wurde. Im Body-Code kann der Cursor geöffnet werden, um zeilenweise durch die Ergebnismenge zu gehen und zu schließen. Während der Cursor geöffnet ist, kann der Code positionierte Aktualisierungen und Löschungen unter Verwendung der Anweisung `WHERE CURRENT OF` für `UPDATE` oder `DELETE` durchführen.

Cursor-Idiosynkrasien

- Die optionale Klausel `FOR UPDATE` kann in der `SELECT`-Anweisung enthalten sein, ihre Abwesenheit verhindert jedoch nicht die erfolgreiche Ausführung einer positionierten Aktualisierung oder Löschung.
- Es sollte darauf geachtet werden, dass die Namen von deklarierten Cursors nicht mit Namen in Konflikt geraten, die später in Anweisungen für `AS CURSOR`-Klauseln verwendet werden.
- Wenn der Cursor nur zum Durchlaufen der Ergebnismenge benötigt wird, ist es fast immer einfacher und weniger fehleranfällig, eine Anweisung `FOR SELECT` mit der Klausel `AS CURSOR` zu verwenden. Deklarierte Cursor müssen zum Abrufen von Daten explizit geöffnet und geschlossen werden. Die Kontextvariable `ROW_COUNT` muss nach jedem Abruf überprüft werden. Wenn der Wert Null ist, muss die Schleife beendet werden. Eine `FOR SELECT`-Anweisung überprüft dies automatisch.

Dennoch bieten deklarierte Cursor ein hohes Maß an Kontrolle über sequentielle Ereignisse und ermöglichen die parallele Verwaltung mehrerer Cursor.

- Das `SELECT`-Statement kann Parameter enthalten. Zum Beispiel:

```
SELECT NAME || :SFX FROM NAMES WHERE NUMBER = :NUM
```

Jeder Parameter muss zuvor als PSQL-Variable deklariert worden sein, auch wenn sie als Ein- und Ausgabeparameter entstehen. Wenn der Cursor geöffnet wird, wird dem Parameter der aktuelle Wert der Variablen zugewiesen.

Achtung!

Wenn sich der Wert einer PSQL-Variablen, die in der SELECT-Anweisung verwendet wird, während der Schleife ändert, kann der neue Wert (jedoch nicht immer) für die verbleibenden Zeilen verwendet werden. Es ist besser, solche Situationen nicht unbeabsichtigt entstehen zu lassen. Wenn Sie dieses Verhalten wirklich benötigen, sollten Sie Ihren Code sorgfältig testen, um sicherzustellen, dass Sie genau wissen, wie sich Änderungen in der Variablen auf das Ergebnis auswirken.

Beachten Sie besonders, dass das Verhalten möglicherweise vom Abfrageplan abhängt, insbesondere von den verwendeten Indizes. Es gibt derzeit keine strengen Regeln für solche Situationen, aber das könnte sich in zukünftigen Versionen von Firebird ändern.

Beispiel für benannte Cursor

1. Declaring a named cursor in the trigger.

```
CREATE OR ALTER TRIGGER TBU_STOCK
BEFORE UPDATE ON STOCK
AS
  DECLARE C_COUNTRY CURSOR FOR (
    SELECT
      COUNTRY,
      CAPITAL
    FROM COUNTRY
  );
BEGIN
  /* PSQL statements */
END
```

2. Eine Sammlung von Skripten zum Erstellen von Ansichten mit einem PSQL-Block unter Verwendung von benannten Cursors.

```
EXECUTE BLOCK
RETURNS (
  SCRIPT BLOB SUB_TYPE TEXT)
AS
DECLARE VARIABLE FIELDS VARCHAR(8191);
DECLARE VARIABLE FIELD_NAME TYPE OF RDB$FIELD_NAME;
DECLARE VARIABLE RELATION RDB$RELATION_NAME;
DECLARE VARIABLE SOURCE TYPE OF COLUMN RDB$RELATIONS.RDB$VIEW_SOURCE;
DECLARE VARIABLE CUR_R CURSOR FOR (
  SELECT
    RDB$RELATION_NAME,
    RDB$VIEW_SOURCE
  FROM
    RDB$RELATIONS
  WHERE
```

```

        RDB$VIEW_SOURCE IS NOT NULL);
-- Declaring a named cursor where
-- a local variable is used
DECLARE CUR_F          CURSOR FOR (
    SELECT
        RDB$FIELD_NAME
    FROM
        RDB$RELATION_FIELDS
    WHERE
        -- It is important that the variable must be declared earlier
        RDB$RELATION_NAME = :RELATION);
BEGIN
    OPEN CUR_R;
    WHILE (1 = 1) DO
    BEGIN
        FETCH CUR_R
        INTO :RELATION, :SOURCE;
        IF (ROW_COUNT = 0) THEN
            LEAVE;

        FIELDS = NULL;
        -- The CUR_F cursor will use the value
        -- of the RELATION variable initiated above
        OPEN CUR_F;
        WHILE (1 = 1) DO
        BEGIN
            FETCH CUR_F
            INTO :FIELD_NAME;
            IF (ROW_COUNT = 0) THEN
                LEAVE;
            IF (FIELDS IS NULL) THEN
                FIELDS = TRIM(FIELD_NAME);
            ELSE
                FIELDS = FIELDS || ', ' || TRIM(FIELD_NAME);
        END
        CLOSE CUR_F;

        SCRIPT = 'CREATE VIEW ' || RELATION;

        IF (FIELDS IS NOT NULL) THEN
            SCRIPT = SCRIPT || ' (' || FIELDS || ')';

        SCRIPT = SCRIPT || ' AS ' || ASCII_CHAR(13);
        SCRIPT = SCRIPT || SOURCE;

        SUSPEND;
    END
    CLOSE CUR_R;
END

```

Siehe auch: [OPEN](#), [FETCH](#), [CLOSE](#)

DECLARE VARIABLE

Verwendet für: Deklaration einer lokalen Variablen

Verfügbar in: PSQL

Syntax:

```

DECLARE [VARIABLE] <varname>
  {<datatype> | <domain> | TYPE OF {<domain> | COLUMN <rel.col>}}
  [NOT NULL] [CHARACTER SET <charset>] [COLLATE <collation>]
  [{DEFAULT | = } <initvalue>];

<datatype> ::=
  {SMALLINT | INTEGER | BIGINT}
  | {FLOAT | DOUBLE PRECISION}
  | {DATE | TIME | TIMESTAMP}
  | {DECIMAL | NUMERIC} [(precision [, scale])]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
  [CHARACTER SET charset]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING]
  [(size)]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
  [SEGMENT SIZE seglen] [CHARACTER SET charset]
  | BLOB [(seglen [, subtype_num])]

<initvalue> ::= <literal> | <context_var>
    
```

Tabelle 7.5. DECLARE VARIABLE-Statement-Parameter

| Argument | Beschreibung |
|--------------|---|
| varname | Name der lokalen Variable |
| datatype | Ein SQL-Datentyp |
| domain | Der Name einer bestehenden Domain in dieser Datenbank |
| rel.col | Beziehungsname (Tabelle oder Sicht) in dieser Datenbank und der Name einer Spalte in dieser Beziehung |
| precision | Präzision. Von 1 bis 18 |
| scale | Rahmen. Von 0 bis 18 muss es kleiner oder gleich der Genauigkeit sein |
| size | Die maximale Größe einer Zeichenfolge in Zeichen |
| subtype_num | BLOB-Untertyp-Nummer |
| subtype_name | Mnemonischer Name des BLOB-Untertyp |
| seglen | Segmentgröße, nicht größer als 65.535 |
| initvalue | Anfangswert für diese Variable |
| literal | Literal eines Typs, der mit dem Typ der lokalen Variablen kompatibel ist |
| context_var | Jede Kontextvariable, deren Typ mit dem Typ der lokalen Variablen kompatibel ist |
| charset | Zeichensatz |

| Argument | Beschreibung |
|-----------|--------------|
| collation | Sortierfolge |

Die Anweisung DECLARE [VARIABLE] wird zum Deklarieren einer lokalen Variable verwendet. Das Schlüsselwort VARIABLE kann weggelassen werden. Je ein DECLARE [VARIABLE]-Statement ist für jede Variable notwendig. Jede beliebige Anzahl von DECLARE [VARIABLE]-Statements kann in jeglicher Reihenfolge eingefügt werden. Der Name jeder lokalen Variable muss eindeutig innerhalb der lokalen Variablen und Ausgabeparametern in der Moduldeklaration sein.

Datentypen für Variablen

Eine lokale Variable kann von einem anderen SQL-Typ als ein Array sein.

- Ein Domainname kann als Typ angegeben werden und die Variable erbt alle ihre Attribute.
- Wenn die TYPE OF <domain>-Klausel verwendet wird, erbt die Variable nur den Datentyp der Domain und gegebenenfalls ihre Zeichensatz- und Sortierattribute. Alle Standardwerte oder Einschränkungen wie NOT NULL oder CHECK-Einschränkungen werden nicht vererbt.
- Wenn die Option "TYPE OF COLUMN" <relation.column> verwendet wird, um Daten aus einer Spalte in einer Tabelle oder Sicht zu „leihen“, wird die Variable nur den Datentyp der Spalte erben, und gegebenenfalls ihren Zeichensatz und Sortierattribute. Alle anderen Attribute werden ignoriert.

NOT NULL-Constraint: Die Variable kann bei Bedarf auf NOT NULL beschränkt werden. Wenn eine Domain als Datentyp angegeben wurde und bereits die NOT NULL-Einschränkung enthält, ist sie nicht erforderlich. Bei den anderen Formen, einschließlich der Verwendung einer Domain, die nullwertfähig ist, sollte das NOT NULL-Attribut bei Bedarf eingefügt werden.

CHARACTER SET- und COLLATE-Klauseln: Sofern nicht anders angegeben, sind der Zeichensatz und die Sortierfolge einer String-Variablen die Standardeinstellungen der Datenbank. Eine Klausel CHARACTER SET kann bei Bedarf eingefügt werden, um Zeichenkettendaten zu verarbeiten, die sich in einem anderen Zeichensatz befinden. Eine gültige Sortierreihenfolge (COLLATE-Klausel) kann ebenfalls mit oder ohne die Zeichensatzklausel eingeschlossen werden.

Initialisieren einer Variablen: Lokale Variablen sind NULL, wenn die Ausführung des Moduls beginnt. Sie können initialisiert werden, sodass ein Start- oder Standardwert verfügbar ist, wenn sie zum ersten Mal referenziert werden. Die Form DEFAULT <initvalue> kann verwendet werden, oder nur der Zuweisungsoperator, "=": = <initvalue>. Der Wert kann ein beliebiges Typ-kompatibles Literal oder eine Kontextvariable sein.

Wichtig

Stellen Sie sicher, dass Sie diese Klausel für alle Variablen verwenden, die auf NOT NULL festgelegt sind und ansonsten keinen Standardwert haben.

Beispiele für verschiedene Möglichkeiten, lokale Variablen zu deklarieren:

```
CREATE OR ALTER PROCEDURE SOME_PROC
AS
-- Declaring a variable of the INT type
DECLARE I INT;
-- Declaring a variable of the INT type that does not allow NULL
DECLARE VARIABLE J INT NOT NULL;
-- Declaring a variable of the INT type with the default value of 0
```

```

DECLARE VARIABLE K INT DEFAULT 0;
-- Declaring a variable of the INT type with the default value of 1
DECLARE VARIABLE L INT = 1;
-- Declaring a variable based on the COUNTRYNAME domain
DECLARE FARM_COUNTRY COUNTRYNAME;
-- Declaring a variable of the type equal to the COUNTRYNAME domain
DECLARE FROM_COUNTRY TYPE OF COUNTRYNAME;
-- Declaring a variable with the type of the CAPITAL column in the COUNTRY table
DECLARE CAPITAL TYPE OF COLUMN COUNTRY.CAPITAL;
BEGIN
  /* PSQL statements */
END

```

Siehe auch: [Datentypen und Unterdatentypen, Benutzerdefinierte Datentypen — Domains, CREATE DOMAIN](#)

BEGIN ... END

Verwendet für: Einen Block von Anweisungen abgrenzen

Verfügbar in: PSQL

Syntax:

```

<block> ::=
BEGIN
  <compound_statement>
  [ <compound_statement>
  ... ]
END

<compound_statement> ::= { <block> | <statement>; }

```

Das Konstrukt BEGIN ... END ist eine zweiteilige Anweisung, die einen Block von Anweisungen umhüllt, die als eine Codeeinheit ausgeführt werden. Jeder Block beginnt mit der Halb-Anweisung BEGIN und endet mit der anderen Halb-Anweisung END. Blöcke können in unbegrenzter Tiefe verschachtelt werden. Sie können leer sein, so dass sie als Stubs fungieren können, ohne dass Dummy-Anweisungen geschrieben werden müssen.

Die Anweisungen BEGIN und END haben keine Zeilenabschlußzeichen. Wenn jedoch ein PSQL-Modul im Dienstprogramm *isql* definiert oder geändert wird, muss für diese Anwendung der letzten END-Anweisung ein eigenes Terminatorzeichen folgen, das zuvor umgeschaltet wurde. SET TERM auf eine andere Zeichenfolge als ein Semikolon. Dieser Terminator ist nicht Teil der PSQL-Syntax.

Die letzte oder äußerste END-Anweisung in einem Trigger beendet den Trigger. Was die letzte Anweisung END in einer gespeicherten Prozedur macht, hängt vom Typ der Prozedur ab:

- In einer wählbaren Prozedur gibt die endgültige Anweisung END die Steuerung an den Aufrufer zurück und gibt SQLCODE 100 zurück, um anzugeben, dass keine weiteren Zeilen abgerufen werden müssen.
- In einer ausführbaren Prozedur gibt die endgültige Anweisung END die Kontrolle an den Aufrufer zurück, zusammen mit den aktuellen Werten aller definierten Ausgabeparameter.

Beispiel: Eine Beispielprozedur aus der Datenbank `employee.fdb`, die die einfache Verwendung der Blöcke BEGIN ... END zeigt:

```
SET TERM ^;
CREATE OR ALTER PROCEDURE DEPT_BUDGET (
    DNO CHAR(3))
RETURNS (
    TOT DECIMAL(12,2))
AS
    DECLARE VARIABLE SUMB DECIMAL(12,2);
    DECLARE VARIABLE RDNO CHAR(3);
    DECLARE VARIABLE CNT INTEGER;
BEGIN
    TOT = 0;

    SELECT
        BUDGET
    FROM
        DEPARTMENT
    WHERE DEPT_NO = :DNO
    INTO :TOT;

    SELECT
        COUNT(BUDGET)
    FROM
        DEPARTMENT
    WHERE HEAD_DEPT = :DNO
    INTO :CNT;

    IF (CNT = 0) THEN
        SUSPEND;

    FOR
        SELECT
            DEPT_NO
        FROM
            DEPARTMENT
        WHERE HEAD_DEPT = :DNO
        INTO :RDNO
    DO
        BEGIN
            EXECUTE PROCEDURE DEPT_BUDGET(:RDNO)
            RETURNING_VALUES :SUMB;
            TOT = TOT + SUMB;
        END

    SUSPEND;
END^
SET TERM ;^
```

Siehe auch: [EXIT](#), [LEAVE](#), [SET TERM](#)

IF ... THEN ... ELSE

Verwendet für: Bedingte Sprünge

Verfügbar in: PSQL

Syntax:

```
IF (<condition>)
  THEN <single_statement> ; | BEGIN <compound_statement> END
[ELSE <single_statement> ; | BEGIN <compound_statement> END]
```

Tabelle 7.6. IF ... THEN ... ELSE Parameters

| Argument | Beschreibung |
|--------------------|--|
| condition | Eine logische Bedingung, die TRUE, FALSE oder UNKNOWN zurückgibt |
| single_statement | Eine einzelne Anweisung wurde mit einem Semikolon abgeschlossen |
| compound_statement | Zwei oder mehr Anweisungen, die in BEGIN ... END verpackt sind |

Die bedingte Sprunganweisung IF ... THEN wird verwendet, um den Ausführungsprozess in einem PSQL-Modul zu verzweigen. Die Bedingung ist immer in Klammern eingeschlossen. Wenn es den Wert TRUE zurückgibt, verzweigt die Ausführung in die Anweisung oder den Anweisungsblock nach dem Schlüsselwort THEN. Wenn eine ELSE vorhanden ist und die Bedingung FALSE oder UNKNOWN zurückgibt, verzweigt die Ausführung in die Anweisung oder den Anweisungsblock danach.

Verzweigungen mit mehreren Unterverzweigungen

PSQL bietet keine Multi-Branch-Sprünge wie CASE oder SWITCH. Nichtsdestoweniger ist die CASE-Suchanweisung von DSQL in PSQL verfügbar und kann zumindest einige Anwendungsfälle in der Art eines Schalters erfüllen:

```
CASE <test_expr>
  WHEN <expr> THEN result
  [WHEN <expr> THEN result ...]
  [ELSE defaultresult]
END

CASE
  WHEN <bool_expr> THEN result
  [WHEN <bool_expr> THEN result ...]
  [ELSE defaultresult]
END
```

Beispiel in PSQL:

```
...
C = CASE
  WHEN A=2 THEN 1
  WHEN A=1 THEN 3
  ELSE 0
END;
...
```

Beispiel: Ein Beispiel mit der IF-Anweisung. Angenommen, die Variablen FIRST, LINE2 und LAST wurden früher deklariert.

```

...
IF (FIRST IS NOT NULL) THEN
    LINE2 = FIRST || ' ' || LAST;
ELSE
    LINE2 = LAST;
...

```

Siehe auch: [WHILE ... DO](#), [CASE](#)

WHILE ... DO

Verwendet für: Schleifenkonstrukte

Verfügbar in: PSQL

Syntax:

```

WHILE <condition> DO
<single_statement> ; | BEGIN <compound_statement> END

```

Tabelle 7.7. WHILE ... DO Parameters

| Argument | Beschreibung |
|--------------------|--|
| condition | Eine logische Bedingung, die TRUE, FALSE oder UNKNOWN zurückgibt |
| single_statement | Eine einzelne Anweisung wurde mit einem Semikolon abgeschlossen |
| compound_statement | Zwei oder mehr Anweisungen, die in BEGIN ... END verpackt sind |

Eine WHILE-Anweisung implementiert das Schleifenkonstrukt in PSQL. Die Anweisung oder der Anweisungsblock wird ausgeführt, bis die Bedingung TRUE zurückgibt. Schleifen können beliebig tief verschachtelt werden.

Beispiel: Eine Prozedur, die die Summe der Zahlen von 1 bis I berechnet, zeigt, wie das Schleifenkonstrukt verwendet wird.

```

CREATE PROCEDURE SUM_INT (I INTEGER)
RETURNS (S INTEGER)
AS
BEGIN
    s = 0;
    WHILE (i > 0) DO

```

```
BEGIN
  s = s + i;
  i = i - 1;
END
END
```

Ausführen der Prozedur in *isql*:

```
EXECUTE PROCEDURE SUM_INT(4);
```

Das Ergebnis ist:

```
S
=====
10
```

Siehe auch: [IF ... THEN ... ELSE](#), [LEAVE](#), [EXIT](#), [FOR SELECT](#), [FOR EXECUTE STATEMENT](#)

LEAVE

Verwendet für: Eine Schleife beenden

Verfügbar in: PSQL

Syntax:

```
[label:]
<loop>
BEGIN
  ...
  LEAVE [label];
  ...
END
<loop_stmt> ::=
  FOR <select_stmt> INTO <var_list> DO
  | FOR EXECUTE STATEMENT ... INTO <var_list> DO
  | WHILE (<condition>)} DO
```

Tabelle 7.8. LEAVE-Statement-Parameter

| Argument | Beschreibung |
|-------------|--|
| label | Label |
| select_stmt | SELECT-Statement |
| condition | Eine logische Bedingung, die TRUE, FALSE oder UNKNOWN zurückgibt |

Eine LEAVE-Anweisung beendet sofort die innere Schleife einer WHILE oder FOR Schleifenanweisung. Der Parameter LABEL ist optional.

LEAVE kann auch zum Beenden von äußeren Schleifen führen. Code wird weiterhin von der ersten Anweisung nach der Beendigung des äußeren Schleifenblocks ausgeführt.

Beispiele:

1. Eine Schleife verlassen, wenn bei einem Einfügen in die NUMBERS-Tabelle ein Fehler auftritt. Der Code wird weiterhin von der Zeile C = 0 ausgeführt.

```

...
WHILE ( B < 10 ) DO
BEGIN
    INSERT INTO NUMBERS(B)
VALUES (:B);
    B = B + 1;
    WHEN ANY DO
    BEGIN
        EXECUTE PROCEDURE LOG_ERROR (
            CURRENT_TIMESTAMP,
            'ERROR IN B LOOP' );
        LEAVE;
    END
END
END
C = 0;
...

```

2. Ein Beispiel für die Verwendung von Labels in der LEAVE-Anweisung. LEAVE LOOPA beendet die äußere Schleife und LEAVE LOOPB beendet die innere Schleife. Beachten Sie, dass die einfache Anweisung LEAVE ausreichen würde, um die innere Schleife zu beenden.

```

...
STMT1 = 'SELECT NAME FROM FARMS';
LOOPA:
FOR EXECUTE STATEMENT :STMT1
INTO :FARM DO
BEGIN
    STMT2 = 'SELECT NAME ' || 'FROM ANIMALS WHERE FARM = ''';
    LOOPB:
    FOR EXECUTE STATEMENT :STMT2 || :FARM || ''
    INTO :ANIMAL DO
    BEGIN
        IF (ANIMAL = 'FLUFFY') THEN
            LEAVE LOOPB;
        ELSE IF (ANIMAL = FARM) THEN
            LEAVE LOOPA;
        ELSE
            SUSPEND;
    END
END
END
...

```

Siehe auch: [EXIT](#)

EXIT

Verwendet für: Beenden der Modulausführung

Verfügbar in: PSQL

Syntax:

```
EXIT;
```

Die Anweisung EXIT bewirkt, dass die Ausführung der Prozedur oder des Triggers von jedem Punkt des Codes zur endgültigen END-Anweisung springt, wodurch das Programm beendet wird.

Beispiel: Verwenden der EXIT-Anweisung in einer auswählbaren Prozedur:

```
CREATE PROCEDURE GEN_100
RETURNS (
  I INTEGER
)
AS
BEGIN
  I = 1;
  WHILE (1=1) DO
  BEGIN
    SUSPEND;
    IF (I=100) THEN
      EXIT;
    I = I + 1;
  END
END
```

Siehe auch: [LEAVE](#), [SUSPEND](#)

SUSPEND

Verwendet für: Übergeben der Ausgabe an den Puffer und Aussetzen der Ausführung, während darauf gewartet wird, dass der Aufrufer sie abrufen

Verfügbar in: PSQL

Syntax:

```
SUSPEND;
```


Die Anweisung `SUSPEND` wird in einer auswählbaren gespeicherten Prozedur verwendet, um die Werte von Ausgabeparametern an einen Puffer zu übergeben und die Ausführung anzuhalten. Die Ausführung bleibt ausgesetzt, bis die aufrufende Anwendung den Inhalt des Puffers abrufen. Die Ausführung wird von der Anweisung direkt nach der `SUSPEND`-Anweisung fortgesetzt. In der Praxis ist dies wahrscheinlich eine neue Iteration eines Schleifenprozesses.

Wichtige Hinweise

1. Anwendungen, die Schnittstellen verwenden, die die API umschließen, führen die Abrufe von auswählbaren Prozeduren transparent aus.
2. Wenn eine `SUSPEND`-Anweisung in einer ausführbaren gespeicherten Prozedur ausgeführt wird, entspricht dies der Ausführung der Anweisung `EXIT`, was zu einer sofortigen Beendigung der Prozedur führt.
3. `SUSPEND` „unterbricht“ die Atomizität des Blocks, in dem es sich befindet. Wenn in einer auswählbaren Prozedur ein Fehler auftritt, werden Anweisungen, die nach der endgültigen `SUSPEND`-Anweisung ausgeführt werden, zurückgesetzt. Anweisungen, die vor der endgültigen `SUSPEND`-Anweisung ausgeführt wurden, werden erst zurückgesetzt, wenn die Transaktion zurückgesetzt wird.

Beispiel: Verwenden der Anweisung `SUSPEND` in einer auswählbaren Prozedur:

```
CREATE PROCEDURE GEN_100
RETURNS (
  I INTEGER
)
AS
BEGIN
  I = 1;
  WHILE (1=1) DO
  BEGIN
    SUSPEND;
    IF (I=100) THEN
      EXIT;
    I = I + 1;
  END
END
```

Siehe auch: [EXIT](#)

EXECUTE STATEMENT

Verwendet für: Ausführen dynamisch erstellter SQL-Anweisungen

Verfügbar in: PSQL

Syntax:

```
<execute_statement> ::= EXECUTE STATEMENT <argument>
[<option> ...]
[INTO <variables>]
```

```

<argument> ::= paramless_stmt
            | (paramless_stmt)
            | (<stmt_with_params>) (<param_values>)

<param_values> ::= <named_values> | <positional_values>

<named_values> ::= paramname := value_expr
                [, paramname := value_expr ...]

<positional_values> ::= value_expr [, value_expr ...]

<option> ::= WITH {AUTONOMOUS | COMMON} TRANSACTION
            | WITH CALLER PRIVILEGES
            | AS USER user
            | PASSWORD password
            | ROLE role
            | ON EXTERNAL [DATA SOURCE] <connect_string>

<connect_string> ::= [<hostspec>] {filepath | db_alias}

<hostspec> ::= <tcpip_hostspec> | <NamedPipes_hostspec>

<tcpip_hostspec> ::= hostname:

<NamePipes_hostspec> ::= \\hostname\

<variables> ::= [:]varname [, [:]varname ...]
    
```

Tabelle 7.9. EXECUTE STATEMENT-Statement-Parameter

| Argument | Beschreibung |
|-------------------|--|
| paramless_stmt | Literale Zeichenfolge oder Variable, die eine nicht parametrisierte SQL-Abfrage enthält |
| stmt_with_params | Literale Zeichenfolge oder Variable, die eine parametrisierte SQL-Abfrage enthält |
| paramname | Name des SQL-Abfrageparameters |
| value_expr | SQL-Ausdruck, der in einen Wert aufgelöst wird |
| user | Nutzername. Dies kann eine Zeichenfolge, CURRENT_USER oder eine Zeichenfolgenvariable sein |
| password | Passwort. Es kann eine Zeichenfolge oder eine Zeichenfolgevariable sein |
| role | Rolle. Dies kann eine Zeichenfolge, CURRENT_ROLE oder eine Zeichenfolgenvariable sein |
| connection_string | Verbindungszeichenfolge Es kann eine Zeichenfolge oder eine Zeichenfolgevariable sein |
| filepath | Pfad zur primären Datenbankdatei |
| db_alias | Datenbankalias |
| hostname | Computernamen oder IP-Adresse |

| Argument | Beschreibung |
|----------|--------------|
| varname | Variable |

Die Anweisung EXECUTE STATEMENT verwendet einen Zeichenfolgenparameter und führt ihn wie eine DSQL-Anweisung aus. Wenn die Anweisung Daten zurückgibt, kann sie über eine INTO -Klausel an lokale Variablen übergeben werden.

Parametrisierte Anweisungen

Sie können die Parameter — entweder benannt oder positional — in der DSQL-Anweisungsfolge verwenden. Jedem Parameter muss ein Wert zugewiesen werden.

Spezielle Regeln für parametrisierte Anweisungen

1. Benannte und positionale Parameter können nicht in einer Abfrage gemischt werden
2. Wenn die Anweisung Parameter hat, müssen sie beim Aufruf von EXECUTE STATEMENT in Klammern stehen, unabhängig davon, ob sie direkt als Strings, als Variablennamen oder als Ausdrücke verwendet werden
3. Jedem benannten Parameter muss in der Anweisungszeichenfolge ein Doppelpunkt (:) vorangestellt werden, jedoch nicht, wenn dem Parameter ein Wert zugewiesen ist
4. Positionsparameter müssen ihre Werte in derselben Reihenfolge erhalten, in der sie im Abfragetext erscheinen
5. Der Zuweisungsoperator für Parameter ist der Spezialoperator " := ", ähnlich dem Zuweisungsoperator in Pascal
6. Jeder benannte Parameter kann mehrmals in der Anweisung verwendet werden, sein Wert muss jedoch nur einmal zugewiesen werden
7. Bei Positionsparametern muss die Anzahl der zugewiesenen Werte genau der Anzahl der Parameterplatzhalter (Fragezeichen) in der Anweisung entsprechen

Beispiele: Mit benannten Parametern:

```

...
DECLARE license_num VARCHAR(15);
DECLARE connect_string VARCHAR (100);
DECLARE stmt VARCHAR (100) =
  'SELECT license
   FROM cars
   WHERE driver = :driver AND location = :loc';
BEGIN
  ...
  SELECT connstr
  FROM databases
  WHERE cust_id = :id

```

```

INTO connect_string;
...
FOR
  SELECT id
  FROM drivers
  INTO current_driver
DO
BEGIN
  FOR
    SELECT location
    FROM driver_locations
    WHERE driver_id = :current_driver
    INTO current_location
  DO
  BEGIN
    ...
    EXECUTE STATEMENT (stmt)
      (driver := current_driver,
       loc := current_location)
    ON EXTERNAL connect_string
    INTO license_num;
    ...
  
```

Derselbe Code mit Positionsparametern:

```

DECLARE license_num VARCHAR (15);
DECLARE connect_string VARCHAR (100);
DECLARE stmt VARCHAR (100) =
  'SELECT license
  FROM cars
  WHERE driver = ? AND location = ?';
BEGIN
  ...
  SELECT connstr
  FROM databases
  WHERE cust_id = :id
  into connect_string;
  ...
  FOR SELECT id
    FROM drivers
    INTO current_driver
  DO
  BEGIN
    FOR
      SELECT location
      FROM driver_locations
      WHERE driver_id = :current_driver
      INTO current_location
    DO
    BEGIN
      ...
      EXECUTE STATEMENT (stmt)
        (current_driver, current_location)
      ON EXTERNAL connect_string
      INTO license_num;
      ...
    
```

WITH {AUTONOMOUS | COMMON} TRANSACTION

Üblicherweise lief die ausgeführte SQL-Anweisung immer innerhalb der aktuellen Transaktion, und dies ist immer noch der Standardwert. WITH AUTONOMOUS TRANSACTION bewirkt, dass eine separate Transaktion mit denselben Parametern wie die aktuelle Transaktion gestartet wird. Es wird festgeschrieben, wenn die Anweisung ohne Fehler ausgeführt wird und andernfalls zurückgesetzt wird. WITH COMMON TRANSACTION verwendet, wenn möglich, die aktuelle Transaktion.

Wenn die Anweisung in einer separaten Verbindung ausgeführt werden muss, wird eine bereits gestartete Transaktion innerhalb dieser Verbindung verwendet, sofern verfügbar. Andernfalls wird eine neue Transaktion mit den gleichen Parametern wie die aktuelle Transaktion gestartet. Alle neuen Transaktionen, die unter dem „COMMON“-Regime gestartet wurden, werden mit der aktuellen Transaktion festgeschrieben oder zurückgesetzt.

WITH CALLER PRIVILEGES

Standardmäßig wird die SQL-Anweisung mit den Berechtigungen des aktuellen Benutzers ausgeführt. Die Angabe von WITH CALLER PRIVILEGES fügt dazu die Privilegien der aufrufenden Prozedur oder des Triggers hinzu, so als ob die Anweisung direkt von der Routine ausgeführt würde. WITH WITH CALLER PRIVILEGES hat keine Auswirkung, wenn die Klausel ON EXTERNAL ebenfalls vorhanden ist.

ON EXTERNAL [DATA SOURCE]

Mit ON EXTERNAL [DATA SOURCE] wird die SQL-Anweisung in einer separaten Verbindung zu derselben oder einer anderen Datenbank ausgeführt, möglicherweise sogar auf einem anderen Server. Wenn die Verbindungszeichenfolge NULL oder " (leere Zeichenfolge) ist, wird die gesamte Klausel ON EXTERNAL [DATA SOURCE] als abwesend betrachtet und die Anweisung wird für die aktuelle Datenbank ausgeführt.

Verbindungspooling

- Externe Verbindungen, die durch Anweisungen WITH COMMON TRANSACTION (der Standardwert) hergestellt werden, bleiben geöffnet, bis die aktuelle Transaktion beendet wird. Sie können durch nachfolgende Aufrufe an EXECUTE STATEMENT wiederverwendet werden, aber nur, wenn die Verbindungszeichenfolge genau gleich ist, einschließlich case
- Externe Verbindungen, die durch Anweisungen `<datenbank>WITH AUTONOMOUS TRANSACTION</datenbank>` hergestellt werden, werden geschlossen, sobald die Anweisung ausgeführt wurde
- Beachten Sie, dass Statements unter WITH AUTONOMOUS TRANSACTION-Verbindungen, die zuvor von Anweisungen unter WITH COMMON TRANSACTION geöffnet wurden, wiederverwendet werden. Wenn dies geschieht, bleibt die wiederverwendete Verbindung nach der Ausführung der Anweisung offen. (Dies geschieht, da es mindestens eine nicht-abgeschlossene Transaktion gibt!)

Transaktionspooling

- Wenn WITH COMMON TRANSACTION aktiviert ist, werden Transaktionen so oft wie möglich wiederverwendet. Sie werden zusammen mit der aktuellen Transaktion festgeschrieben oder zurückgesetzt

- Wenn WITH AUTONOMOUS TRANSACTION angegeben ist, wird immer eine neue Transaktion für die Anweisung gestartet. Diese Transaktion wird unmittelbar nach der Ausführung der Anweisung festgeschrieben oder zurückgesetzt

Ausnahmebehandlung

Ausnahmebehandlung: Wenn ON EXTERNAL verwendet wird, erfolgt die zusätzliche Verbindung immer über einen sogenannten externen Provider, auch wenn die Verbindung zur aktuellen Datenbank besteht. Eine der Folgen ist, dass Ausnahmen nicht auf die übliche Art und Weise abgefangen werden können. Jede von der Anweisung verursachte Ausnahme wird entweder in einen eds_connection- oder einen eds_statement-Fehler enden. Um sie in Ihrem PSQL-Code abzufangen, müssen Sie WHEN GDSCODE eds_connection, WHEN GDSCODE eds_statement oder WHEN ANY verwenden.

Anmerkung

Ohne ON EXTERNAL werden Ausnahmen auf die übliche Weise abgefangen, selbst wenn eine zusätzliche Verbindung zur aktuellen Datenbank hergestellt wird.

Verschiedene Hinweise

- Der für die externe Verbindung verwendete Zeichensatz ist der gleiche wie für die aktuelle Verbindung
- Zweiphasen-Commits werden nicht unterstützt

AS USER, PASSWORD und ROLE

Die optionalen Klauseln AS USER, PASSWORD und ROLE erlauben die Angabe unter welchem Benutzer und unter welcher Rolle das SQL-Statement ausgeführt wird. Die Methode der Benutzeranmeldung und die Existenz einer separaten offenen Verbindung hängt von dem Vorhandensein und den Werten der Klauseln ON EXTERNAL [DATA SOURCE], AS USER, PASSWORD und ROLE ab:

- Wenn ON EXTERNAL verwendet wird, wird immer eine neue Verbindung aufgebaut und:
 - Wenn mindestens eines von AS USER, PASSWORD und ROLE vorhanden ist, wird die native Authentifizierung mit den angegebenen Parameterwerten versucht (lokal oder remote abhängig von der Verbindungszeichenfolge). Für fehlende Parameter werden keine Standardwerte verwendet
 - Wenn alle drei nicht vorhanden sind und die Verbindungszeichenfolge keinen Hostnamen enthält, wird die neue Verbindung auf dem lokalen Host mit demselben Benutzer und derselben Rolle wie die aktuelle Verbindung hergestellt. Der Begriff "lokal" bedeutet hier "auf der gleichen Maschine wie der Server". Dies ist nicht unbedingt der Standort des Clients
 - Wenn alle drei nicht vorhanden sind und die Verbindungszeichenfolge einen Hostnamen enthält, wird eine vertrauenswürdige Authentifizierung auf dem Remote-Host versucht (aus der Perspektive des Servers wiederum "Remote"). Wenn dies erfolgreich ist, gibt das Remote-Betriebssystem den Benutzernamen an (normalerweise das Betriebssystemkonto, unter dem der Firebird-Prozess ausgeführt wird).
- Fehlt ON EXTERNAL:
 - Wenn mindestens eines von AS USER, PASSWORD und ROLE vorhanden ist, wird eine neue Verbindung zur aktuellen Datenbank mit den angegebenen Parameterwerten geöffnet. Für fehlende Parameter werden keine Standardwerte verwendet
 - Wenn alle drei nicht vorhanden sind, wird die Anweisung innerhalb der aktuellen Verbindung ausgeführt

Hinweis

Wenn ein Parameterwert NULL oder " (leere Zeichenfolge) ist, wird der gesamte Parameter als abwesend betrachtet. Darüber hinaus gilt AS USER als abwesend, wenn der Wert gleich CURRENT_USER und ROLE wenn es identisch mit CURRENT_ROLE ist.

Vorsicht mit EXECUTE STATEMENT

1. Es gibt keine Möglichkeit, die Syntax der enthaltenen Anweisung zu überprüfen
2. Es gibt keine Abhängigkeitsprüfungen, um festzustellen, ob Tabellen oder Spalten gelöscht wurden
3. Obwohl die Leistung in Schleifen in Firebird 2.5 erheblich verbessert wurde, ist die Ausführung immer noch erheblich langsamer als wenn dieselben Anweisungen direkt gestartet werden
4. Rückgabewerte werden streng auf den Datentyp überprüft, um unvorhersehbare Ausnahmen für das Typcasting zu vermeiden. Beispielsweise würde die Zeichenfolge '1234' in eine Ganzzahl, 1234, konvertiert, aber 'abc' würde einen Konvertierungsfehler ergeben

Alles in allem sollte diese Funktion sehr vorsichtig verwendet werden und Sie sollten immer die Vorbehalte berücksichtigen. Wenn Sie das gleiche Ergebnis mit PSQL und / oder DSQL erzielen können, ist dies fast immer vorzuziehen.

Siehe auch: [FOR EXECUTE STATEMENT](#)

FOR SELECT

Verwendet für: Zeilenweises Durchlaufen einer ausgewählten Ergebnismenge

Verfügbar in: PSQL

Syntax:

```
FOR <select_stmt> [AS CURSOR cursorname]
DO {<single_statement> | BEGIN <compound_statement> END}
```

Tabelle 7.10. FOR SELECT-Statement-Parameter

| Argument | Beschreibung |
|--------------------|--|
| select_stmt | SELECT-Statement |
| cursorname | Name des Cursors. Dieser muss eindeutig unter den Cursor-Namen im PSQL-Modul (gespeicherte Prozedur, Trigger oder PSQL-Block) sein |
| single_statement | Eine einzelne Anweisung, die mit einem Doppelpunkt abgeschlossen wird und die gesamte Verarbeitung für diese FOR-Schleife ausführt |
| compound_statement | Ein Anweisungsblock, der in BEGIN ... END eingeschlossen ist und der die gesamte Verarbeitung für diese FOR-Schleife ausführt |

Ein FOR SELECT-Statement

- ruft jede Zeile sequenziell aus der Ergebnismenge ab und führt die Anweisung oder den Anweisungsblock in der Zeile aus. In jeder Iteration der Schleife werden die Feldwerte der aktuellen Zeile in vordefinierte Variablen kopiert.

Mit der Klausel AS CURSOR können positionierte Löschungen und Aktualisierungen durchgeführt werden, siehe unten

- kann andere FOR SELECT-Anweisungen einbetten
- kann benannte Parameter enthalten, die zuvor in der DECLARE VARIABLE-Anweisung deklariert werden müssen, oder als Eingabe- oder Ausgabeparameter der Prozedur vorhanden sein
- erfordert eine INTO-Klausel, die sich am Ende der SELECT ... FROM ...-Spezifikation befindet. In jeder Iteration der Schleife werden die Feldwerte in der aktuellen Zeile in die Liste der Variablen kopiert, die in der Klausel INTO angegeben sind. Die Schleife wird wiederholt, bis alle Zeilen abgerufen wurden. Danach wird sie beendet
- kann mit einem LEAVE-Statement beendet werden, bevor alle Zeilen abgeholt wurden.

Der undeklarierte Cursor

Die optionale AS CURSOR-Klausel behandelt den Satz in der FOR SELECT-Struktur als nicht deklarierten benannten Cursor, der mit der WHERE CURRENT OF-Klausel bearbeitet werden kann, innerhalb der Anweisung oder des Blocks nach dem Befehl DO, um die aktuelle Zeile zu löschen oder zu aktualisieren, bevor die Ausführung zur nächsten Iteration übergeht.

Weitere Punkte, die in Bezug auf nicht deklarierte Cursor berücksichtigt werden müssen:

1. Die Anweisungen OPEN, FETCH und CLOSE können nicht auf einen Cursor angewendet werden, der durch die Klausel AS CURSOR angezeigt wird
2. Das Argument cursorname, das einer Klausel AS CURSOR zugeordnet ist, darf nicht mit Namen kollidieren, die von den Anweisungen DECLARE VARIABLE oder DECLARE CURSOR am Anfang der body-Codes, noch mit anderen Cursors, die durch eine Klausel AS CURSOR erstellt wurden
3. Die optionale Klausel FOR UPDATE in der Anweisung SELECT ist für ein positioniertes Update nicht erforderlich

Beispiele für die Verwendung von FOR SELECT:

1. Eine einfache Schleife durch Abfrageergebnisse:

```
CREATE PROCEDURE SHOWNUMS
RETURNS (
  AA INTEGER,
  BB INTEGER,
  SM INTEGER,
  DF INTEGER)
AS
BEGIN
  FOR SELECT DISTINCT A, B
    FROM NUMBERS
    ORDER BY A, B
  INTO AA, BB
```



```

DO
BEGIN
    SM = AA + BB;
    DF = AA - BB;
    SUSPEND;
END
END

```

2. Geschachtelte FOR SELECT-Schleife:

```

CREATE PROCEDURE RELFIELDS
RETURNS (
    RELATION CHAR(32),
    POS INTEGER,
    FIELD CHAR(32))
AS
BEGIN
    FOR SELECT RDB$RELATION_NAME
        FROM RDB$RELATIONS
        ORDER BY 1
        INTO :RELATION
    DO
        BEGIN
            FOR SELECT
                RDB$FIELD_POSITION + 1,
                RDB$FIELD_NAME
            FROM RDB$RELATION_FIELDS
            WHERE
                RDB$RELATION_NAME = :RELATION
            ORDER BY RDB$FIELD_POSITION
            INTO :POS, :FIELD

            DO
                BEGIN
                    IF (POS = 2) THEN
                        RELATION = ' ';

                    SUSPEND;
                END
            END
        END
    END
END

```

3. Verwenden Sie die AS CURSOR-Klausel, um einen Cursor für das positionierte Löschen eines Datensatzes zu verwenden:

```

CREATE PROCEDURE DELTOWN (
    TOWNTODELETE VARCHAR(24))
RETURNS (
    TOWN VARCHAR(24),
    POP INTEGER)
AS
BEGIN
    FOR SELECT TOWN, POP
        FROM TOWNS
        INTO :TOWN, :POP AS CURSOR TCUR

```

```
DO
BEGIN
  IF (:TOWN = :TOWNTODELETE) THEN
    -- Positional delete
    DELETE FROM TOWNS
    WHERE CURRENT OF TCUR;
  ELSE
    SUSPEND;
  END
END
```

Siehe auch: [DECLARE CURSOR](#), [LEAVE](#), [SELECT](#), [UPDATE](#), [DELETE](#)

FOR EXECUTE STATEMENT

Verwendet für: Ausführen von dynamisch erstellten SQL-Anweisungen, die einen Zeilensatz zurückgeben

Verfügbar in: PSQL

Syntax:

```
FOR <execute_statement> DO {<single_statement> | BEGIN <compound_statement> END}
```

Tabelle 7.11. FOR EXECUTE STATEMENT-Statement-Parameter

| Argument | Beschreibung |
|--------------------|--|
| execute_stmt | Ein EXECUTE STATEMENT-String |
| single_statement | Eine einzelne Anweisung, die mit einem Doppelpunkt abgeschlossen wird und die gesamte Verarbeitung für diese FOR-Schleife ausführt |
| compound_statement | Ein Anweisungsblock, der in BEGIN ... END eingeschlossen ist und der die gesamte Verarbeitung für diese FOR-Schleife ausführt |

Die Anweisung FOR EXECUTE STATEMENT wird in Analogie zu FOR SELECT verwendet, um die Ergebnismenge einer dynamisch ausgeführten Abfrage, die mehrere Zeilen zurückgibt, zu durchlaufen.

Beispiel: Ausführen einer dynamisch erstellten Abfrage SELECT, die einen Datensatz zurückgibt:

```
CREATE PROCEDURE DynamicSampleThree (
  Q_FIELD_NAME VARCHAR(100),
  Q_TABLE_NAME VARCHAR(100)
) RETURNS(
  LINE VARCHAR(32000)
)
AS
  DECLARE VARIABLE P_ONE_LINE VARCHAR(100);
BEGIN
  LINE = '';
  FOR
    EXECUTE STATEMENT
```

```
'SELECT T1.' || :Q_FIELD_NAME ||
' FROM ' || :Q_TABLE_NAME || ' T1 '
INTO :P_ONE_LINE
DO
IF (:P_ONE_LINE IS NOT NULL) THEN
LINE = :LINE || :P_ONE_LINE || ' ';
SUSPEND;
END
```

Siehe auch: [EXECUTE STATEMENT](#)

OPEN

Verwendet für: Öffnen eines deklarierten Cursors

Verfügbar in: PSQL

Syntax:

```
OPEN cursorname;
```

Tabelle 7.12. OPEN Statement Parameter

| Argument | Beschreibung |
|------------|---|
| cursorname | Name des Cursors. Ein Cursor mit diesem Namen muss zuvor mit einer DECLARE CURSOR-Anweisung deklariert werden |

Eine OPEN -Anweisung öffnet einen zuvor deklarierten Cursor, führt die für sie deklarierte SELECT -Anweisung aus und macht den ersten Datensatz zum abzurufenden Ergebnisdatensatz. OPEN kann nur auf zuvor in einer DECLARE VARIABLE-Anweisung deklarierte Cursor angewendet werden.

Anmerkung

Wenn die für den Cursor deklarierte Anweisung SELECT über Parameter verfügt, müssen sie als lokale Variablen deklariert sein oder als Ein- oder Ausgabeparameter vor dem Deklarieren des Cursors vorhanden sein. Wenn der Cursor geöffnet wird, wird dem Parameter der aktuelle Wert der Variablen zugewiesen.

Beispiele:

1. Verwenden der OPEN-Anweisung:

```
SET TERM ^;

CREATE OR ALTER PROCEDURE GET_RELATIONS_NAMES
RETURNS (
RNAME CHAR(31)
)
AS
DECLARE C CURSOR FOR (
SELECT RDB$RELATION_NAME
```

```

        FROM RDB$RELATIONS);
BEGIN
  OPEN C;
  WHILE (1 = 1) DO
  BEGIN
    FETCH C INTO :RNAME;
    IF (ROW_COUNT = 0) THEN
      LEAVE;
    SUSPEND;
  END
  CLOSE C;
END^

SET TERM ;^

```

2. Eine Sammlung von Skripten zum Erstellen von Ansichten mit einem PSQL-Block mit benannten Cursors:

```

EXECUTE BLOCK
RETURNS (
  SCRIPT BLOB SUB_TYPE TEXT)
AS
DECLARE VARIABLE FIELDS VARCHAR(8191);
DECLARE VARIABLE FIELD_NAME TYPE OF RDB$FIELD_NAME;
DECLARE VARIABLE RELATION RDB$RELATION_NAME;
DECLARE VARIABLE SOURCE TYPE OF COLUMN RDB$RELATIONS.RDB$VIEW_SOURCE;
-- named cursor
DECLARE VARIABLE CUR_R CURSOR FOR (
  SELECT
    RDB$RELATION_NAME,
    RDB$VIEW_SOURCE
  FROM
    RDB$RELATIONS
  WHERE
    RDB$VIEW_SOURCE IS NOT NULL);
-- named cursor with local variable
DECLARE CUR_F CURSOR FOR (
  SELECT
    RDB$FIELD_NAME
  FROM
    RDB$RELATION_FIELDS
  WHERE
    -- Important! The variable shall be declared earlier
    RDB$RELATION_NAME = :RELATION);
BEGIN
  OPEN CUR_R;
  WHILE (1 = 1) DO
  BEGIN
    FETCH CUR_R
    INTO :RELATION, :SOURCE;
    IF (ROW_COUNT = 0) THEN
      LEAVE;

    FIELDS = NULL;
    -- The CUR_F cursor will use
    -- variable value of RELATION initialized above
    OPEN CUR_F;

```

```

WHILE (1 = 1) DO
BEGIN
    FETCH CUR_F
    INTO :FIELD_NAME;
    IF (ROW_COUNT = 0) THEN
        LEAVE;
    IF (FIELDS IS NULL) THEN
        FIELDS = TRIM(FIELD_NAME);
    ELSE
        FIELDS = FIELDS || ', ' || TRIM(FIELD_NAME);
    END
    CLOSE CUR_F;

    SCRIPT = 'CREATE VIEW ' || RELATION;

    IF (FIELDS IS NOT NULL) THEN
        SCRIPT = SCRIPT || ' (' || FIELDS || ')';

    SCRIPT = SCRIPT || ' AS ' || ASCII_CHAR(13);
    SCRIPT = SCRIPT || SOURCE;

    SUSPEND;
END
CLOSE CUR_R;
END

```

Siehe auch: [DECLARE CURSOR](#), [FETCH](#), [CLOSE](#)

FETCH

Verwendet für: Abrufen aufeinanderfolgender Datensätze aus einem Datensatz, der mit einem Cursor abgerufen wurde

Verfügbar in: PSQL

Syntax:

```

FETCH cursorname INTO [:]varname [, [:]varname ...];

```

Tabelle 7.13. FETCH-Statement-Parameter

| Argument | Beschreibung |
|------------|--|
| cursorname | Name des Cursors Ein Cursor mit diesem Namen muss zuvor mit einer DECLARE CURSOR-Anweisung deklariert und durch eine OPEN-Anweisung geöffnet werden. |
| varname | Variablenname |

Eine FETCH-Anweisung ruft die erste und die folgenden Zeilen aus der Ergebnismenge des Cursors ab und weist PSQL-Variablen die Spaltenwerte zu. Die Anweisung FETCH kann nur mit einem Cursor verwendet werden, der mit der Anweisung DECLARE CURSOR deklariert wurde.

Die INTO-Klausel ruft Daten aus der aktuellen Zeile des Cursors ab und lädt sie in PSQL-Variablen.

Um zu überprüfen, ob alle Datensatzzeilen abgerufen wurden, gibt die Kontextvariable ROW_COUNT die Anzahl der Zeilen zurück, die von der Anweisung abgerufen wurden. Es ist positiv, bis alle Zeilen überprüft wurden. Ein ROW_COUNT von 1 gibt an, dass der nächste Abruf der letzte sein wird.

Beispiel: Verwenden der FETCH-Anweisung:

```

SET TERM ^;

CREATE OR ALTER PROCEDURE GET_RELATIONS_NAMES
RETURNS (
  RNAME CHAR(31)
)
AS
  DECLARE C CURSOR FOR (
    SELECT RDB$RELATION_NAME
    FROM RDB$RELATIONS);
BEGIN
  OPEN C;
  WHILE (1 = 1) DO
  BEGIN
    FETCH C INTO :RNAME;
    IF (ROW_COUNT = 0) THEN
      LEAVE;
    SUSPEND;
  END
  CLOSE C;
END^

SET TERM ;^

```

Siehe auch: [DECLARE CURSOR](#), [OPEN](#), [CLOSE](#)

CLOSE

Verwendet für: Einen deklarierten Cursor schließen

Verfügbar in: PSQL

Syntax:

```
CLOSE cursorname;
```

Tabelle 7.14. CLOSE-Statement-Parameter

| Argument | Beschreibung |
|------------|--|
| cursorname | Name des Cursors. Ein Cursor mit diesem Namen muss zuvor mit einer DECLARE CURSOR-Anweisung deklariert und durch eine OPEN-Anweisung geöffnet werden |

Eine Anweisung CLOSE schließt einen geöffneten Cursor. Alle Cursor, die noch geöffnet sind, werden automatisch geschlossen, nachdem der Modulcode ausgeführt wurde. Nur ein Cursor, der mit DECLARE CURSOR deklariert wurde, kann mit einer CLOSE-Anweisung geschlossen werden.

Beispiel: Verwenden der CLOSE-Anweisung:

```

SET TERM ^;

CREATE OR ALTER PROCEDURE GET_RELATIONS_NAMES
RETURNS (
  RNAME CHAR(31)
)
AS
  DECLARE C CURSOR FOR (
    SELECT RDB$RELATION_NAME
    FROM RDB$RELATIONS);
BEGIN
  OPEN C;
  WHILE (1 = 1) DO
  BEGIN
    FETCH C INTO :RNAME;
    IF (ROW_COUNT = 0) THEN
      LEAVE;
    SUSPEND;
  END
  CLOSE C;
END^

```

Siehe auch: [DECLARE CURSOR](#), [OPEN](#), [FETCH](#)

IN AUTONOMOUS TRANSACTION

Verwendet für: Eine Anweisung oder einen Block von Anweisungen in einer autonomen Transaktion ausführen

Verfügbar in: PSQL

Syntax:

```

IN AUTONOMOUS TRANSACTION DO <compound_statement>

```

Tabelle 7.15. IN AUTONOMOUS TRANSACTION Statement Parameter

| Argument | Beschreibung |
|--------------------|---|
| compound_statement | Ein Statement oder ein Block von Statements |

Eine Anweisung IN AUTONOMOUS TRANSACTION ermöglicht die Ausführung einer Anweisung oder eines Anweisungsblocks in einer autonomen Transaktion. Code, der in einer autonomen Transaktion ausgeführt wird, wird unmittelbar nach seiner erfolgreichen Ausführung unabhängig vom Status seiner übergeordneten Transaktion festgeschrieben. Dies kann erforderlich sein, wenn bestimmte Vorgänge nicht zurückgesetzt werden sollen, auch wenn in der übergeordneten Transaktion ein Fehler auftritt.

Eine autonome Transaktion hat dieselbe Isolationsstufe wie ihre übergeordnete Transaktion. Jede Ausnahme, die im Block des autonomen Transaktionscodes ausgelöst wird, führt dazu, dass die autonome Transaktion zurückgesetzt wird und alle vorgenommenen Änderungen storniert werden. Wenn der Code erfolgreich ausgeführt wird, wird die autonome Transaktion festgeschrieben.

Beispiel: Verwenden einer autonomen Transaktion in einem Trigger für das Datenbankereignis ON CONNECT, um alle Verbindungsversuche einschließlich der fehlgeschlagenen zu protokollieren:

```
CREATE TRIGGER TR_CONNECT ON CONNECT
AS
BEGIN
  -- Logging all attempts to connect to the database
  IN AUTONOMOUS TRANSACTION DO
    INSERT INTO LOG(MSG)
      VALUES ('USER ' || CURRENT_USER || ' CONNECTS.');
```

```
IF (CURRENT_USER IN (SELECT
                      USERNAME
                      FROM
                      BLOCKED_USERS)) THEN
  BEGIN
    -- Logging that the attempt to connect
    -- to the database failed and sending
    -- a message about the event
    IN AUTONOMOUS TRANSACTION DO
      BEGIN
        INSERT INTO LOG(MSG)
          VALUES ('USER ' || CURRENT_USER || ' REFUSED.');
```

```
        POST_EVENT 'CONNECTION ATTEMPT' || ' BY BLOCKED USER!';
      END
      -- now calling an exception
      EXCEPTION EX_BADUSER;
    END
  END
END
```

Siehe auch: [Transaktionskontrolle](#)

POST_EVENT

Verwendet für: Benachrichtigung von Listening-Clients über Datenbankereignisse in einem Modul

Verfügbar in: PSQL

Syntax:

```
POST_EVENT event_name;
```

Tabelle 7.16. POST_EVENT Statement Parameter

| Argument | Beschreibung |
|------------|--|
| event_name | Ereignisname (Nachricht) ist auf 127 Byte beschränkt |

Die Anweisung `POST_EVENT` benachrichtigt den Ereignismanager über das Ereignis, das es in einer Ereignistabelle speichert. Wenn die Transaktion festgeschrieben wird, benachrichtigt der Ereignismanager Anwendungen, die ihr Interesse an dem Ereignis signalisieren.

Der Ereignisname kann eine Art Code oder eine kurze Nachricht sein: Die Auswahl ist offen, da sie nur eine Zeichenfolge von bis zu 127 Bytes ist.

Der Inhalt der Zeichenfolge kann ein Zeichenfolgeliteral, eine Variable oder ein beliebiger gültiger SQL-Ausdruck sein, der in eine Zeichenfolge aufgelöst wird.

Beispiel: Benachrichtigung der zuhörenden Anwendungen über das Einfügen eines Datensatzes in die SALES-Tabelle:

```
SET TERM ^;  
CREATE TRIGGER POST_NEW_ORDER FOR SALES  
ACTIVE AFTER INSERT POSITION 0  
AS  
BEGIN  
    POST_EVENT 'new_order';  
END^  
SET TERM ;^
```

Abfangen und Behandeln von Fehlern

Firebird hat ein nützliches Lexikon von PSQL-Anweisungen und -Ressourcen, um Fehler in Modulen einzufangen und sie zu behandeln. Intern implementierte Ausnahmen existieren, um die Ausführung anzuhalten, wenn jeder Standardfehler in DDL, DSQL und der physischen Umgebung auftritt.

Systemausnahmen

Eine Ausnahme ist eine Nachricht, die generiert wird, wenn ein Fehler auftritt.

Alle Ausnahmen, die von Firebird behandelt werden, haben vordefinierte numerische Werte für Kontextvariablen (Symbole) und Textnachrichten, die ihnen zugeordnet sind. Fehlermeldungen werden standardmäßig in Englisch ausgegeben. Lokalisierte Firebird-Builds sind verfügbar, in denen Fehlermeldungen in andere Sprachen übersetzt werden.

Vollständige Auflistungen der Systemausnahmen finden Sie in *Anhang B: Fehlercodes und Meldungen*:

- `SQLSTATE` Fehlercodes und Beschreibungen
- `GDSCODE` Fehlercodes, `SQLCODEs` und Beschreibungen

Benutzerdefinierte Ausnahmen

Benutzerdefinierte Ausnahmen können in der Datenbank als permanente Objekte deklariert und im PSQL-Code aufgerufen werden, um bestimmte Fehler zu signalisieren, zum Beispiel, um bestimmte Geschäftsregeln durch-

zusetzen. Eine benutzerdefinierte Ausnahme besteht aus einem Bezeichner und einer Standardnachricht von ungefähr 1000 Byte. Weitere Informationen finden Sie unter [CREATE EXCEPTION](#).

Im PSQL-Code werden Ausnahmen mit der WHEN-Anweisung behandelt. Bei der Behandlung einer Ausnahme im Code wird entweder das Problem in situ behoben oder es wird übergangen. Bei beiden Lösungen kann die Ausführung fortgesetzt werden, ohne dass eine Ausnahmebedingungsricht an den Client zurückgegeben wird.

Eine Ausnahme führt dazu, dass die Ausführung im Block beendet wird. Anstatt die Ausführung an die END-Anweisung zu übergeben, bewegt sich die Prozedur durch Ebenen von verschachtelten Blöcken nach außen, beginnend mit dem Block, in dem die Ausnahme abgefangen wird, nach dem Code des Handlers, der diese Ausnahme „kennt“. Sie stoppt die Suche, wenn die erste WHEN-Anweisung gefunden wird, die diese Ausnahme verarbeiten kann.

EXCEPTION

Verwendet für: Eine benutzerdefinierte Ausnahme auslösen oder eine Ausnahme erneut auslösen

Verfügbar in: PSQL

Syntax:

```
EXCEPTION [exception_name [custom_message]]
```

Tabelle 7.17. EXCEPTION-Statement-Parameter

| Argument | Beschreibung |
|----------------|---|
| exception_name | Name der Ausnahme |
| custom_message | Alternativer Nachrichtentext, der an die Aufruferschnittstelle zurückgegeben wird, wenn eine Ausnahme ausgelöst wird. Die maximale Länge der Textnachricht beträgt 1.021 Byte |

Eine Anweisung EXCEPTION löst die benutzerdefinierte Ausnahme mit dem angegebenen Namen aus. Ein alternativer Nachrichtentext von bis zu 1.021 Byte kann optional den Standardnachrichtentext der Ausnahme überschreiben.

Die Ausnahmebedingung kann in der Anweisung behandelt werden, indem sie nur mit einem bestimmten WHEN ... DO-Handler belassen wird und dem Trigger oder der gespeicherten Prozedur erlaubt wird, alle Operationen zu beenden und rückgängig zu machen. Die aufrufende Anwendung erhält den alternativen Nachrichtentext, sofern einer angegeben wurde. Andernfalls empfängt es die ursprünglich für diese Ausnahme definierte Nachricht.

Innerhalb des Ausnahmebehandlungsblocks — und nur innerhalb davon — kann die abgefangene Ausnahme erneut ausgelöst werden, indem die Anweisung EXCEPTION ohne Parameter ausgeführt wird. Wenn der Befehl außerhalb des Blocks liegt, hat die erneut aufgerufene EXCEPTION-Anweisung keine Auswirkung.

Anmerkung

Benutzerdefinierte Ausnahmen werden in der Systemtabelle `RDB$EXCEPTIONS` gespeichert.

Beispiele:

1. Eine Ausnahme mit dynamisch erzeugtem Text auslösen:

```
...
EXCEPTION EX_BAD_TYPE
  'Incorrect record type with id ' || new.id;
...
```

2. Eine Ausnahme für eine Bedingung in der gespeicherten SHIP_ORDER-Prozedur auslösen:

```
CREATE OR ALTER PROCEDURE SHIP_ORDER (
  PO_NUM CHAR(8))
AS
DECLARE VARIABLE ord_stat CHAR(7);
DECLARE VARIABLE hold_stat CHAR(1);
DECLARE VARIABLE cust_no INTEGER;
DECLARE VARIABLE any_po CHAR(8);
BEGIN
  SELECT
    s.order_status,
    c.on_hold,
    c.cust_no
  FROM
    sales s, customer c
  WHERE
    po_number = :po_num AND
    s.cust_no = c.cust_no
  INTO :ord_stat,
    :hold_stat,
    :cust_no;

  IF (ord_stat = 'shipped') THEN
    EXCEPTION order_already_shipped;
  /* Other statements */
END
```

3. Eine Ausnahme bei einer Bedingung auslösen und die ursprüngliche Nachricht durch eine alternative Nachricht ersetzen:

```
CREATE OR ALTER PROCEDURE SHIP_ORDER (
  PO_NUM CHAR(8))
AS
DECLARE VARIABLE ord_stat CHAR(7);
DECLARE VARIABLE hold_stat CHAR(1);
DECLARE VARIABLE cust_no INTEGER;
DECLARE VARIABLE any_po CHAR(8);
BEGIN
```

```

SELECT
    s.order_status,
    c.on_hold,
    c.cust_no
FROM
    sales s, customer c
WHERE
    po_number = :po_num AND
    s.cust_no = c.cust_no
INTO :ord_stat,
     :hold_stat,
     :cust_no;

IF (ord_stat = 'shipped') THEN
    EXCEPTION order_already_shipped
    'Order status is "' || ord_stat || "'";
/* Other statements */
END

```

4. Einen Fehler protokollieren und erneut in den WHEN-Block werfen:

```

CREATE PROCEDURE ADD_COUNTRY (
    ACountryName COUNTRYNAME,
    ACurrency VARCHAR(10) )
AS
BEGIN
    INSERT INTO country (country,
                        currency)
VALUES (:ACountryName,
        :ACurrency);
    WHEN ANY DO
    BEGIN
        -- write an error in log
        IN AUTONOMOUS TRANSACTION DO
            INSERT INTO ERROR_LOG (PSQL_MODULE,
                                   GDS_CODE,
                                   SQL_CODE,
                                   SQL_STATE)
VALUES ('ADD_COUNTRY',
        GDSCODE,
        SQLCODE,
        SQLSTATE);
        -- Re-throw exception
    EXCEPTION;
    END
END

```

Siehe auch: [CREATE EXCEPTION, WHEN ... DO](#)

WHEN ... DO

Verwendet für: Eine Ausnahme abfangen und den Fehler behandeln

Verfügbar in: PSQL

Syntax:

```
WHEN {<error> [, <error> ...] | ANY}
DO <compound_statement>

<error> ::= {
    EXCEPTION exception_name
    | SQLCODE number
    | GDSCODE errcode
}
```

Tabelle 7.18. WHEN ... DO-Statement-Parameter

| Argument | Beschreibung |
|--------------------|---|
| exception_name | Name der Ausnahme |
| number | SQLCODE Fehler-Code |
| errcode | Symbolischer GDSCODE-Fehlernamen |
| compound_statement | Ein Statement oder ein Block von Statements |

Die Anweisung WHEN ... DO wird verwendet, um Fehler und benutzerdefinierte Ausnahmen zu behandeln. Die Anweisung erfasst alle Fehler und benutzerdefinierten Ausnahmen, die nach dem Schlüsselwort WHEN aufgeführt sind. Wenn WHEN das Schlüsselwort ANY folgt, fängt die Anweisung jeden Fehler oder jede benutzerdefinierte Ausnahme ab, auch wenn sie bereits in einer WHEN-Anweisung weiter oben im Block behandelt wurden.

Der WHEN ... DO-Block muss sich am Ende eines Anweisungsblocks befinden, vor der Anweisung END des Blocks.

Auf das Schlüsselwort DO folgt eine Anweisung oder ein Anweisungsblock innerhalb eines BEGIN ... END-Wrappers, der die Ausnahme behandelt. Die Kontextvariablen SQLCODE, GDSCODE und SQLSTATE stehen im Kontext dieser Anweisung oder dieses Blocks zur Verfügung. Die Anweisung EXCEPTION ohne Parameter kann auch in diesem Kontext verwendet werden, um den Fehler oder die Ausnahme erneut zu werfen.

Bezüglich GDSCODE

Das Argument für die Klausel WHEN GDSCODE ist der symbolische Name, der der intern definierten Ausnahme zugeordnet ist, z. B. grant_obj_notfound für den GDS-Fehler 335544551.

Nach der DO-Klausel wird eine weitere GDSCODE-Kontextvariable, die den numerischen Code enthält, für die Verwendung in der Anweisung oder dem Anweisungsblock verfügbar, die den Error-Handler codieren. Dieser numerische Code ist erforderlich, wenn Sie eine GDSCODE-Ausnahme mit einem gezielten Fehler vergleichen möchten.

Die WHEN ... DO-Anweisung oder der WHEN ... DO-Block werden niemals ausgeführt, es sei denn, eines der Ereignisse, auf die die Bedingungen abzielen, wird zur Laufzeit ausgeführt. Wenn die Anweisung ausgeführt wird, wird die Ausführung fortgesetzt, so als ob kein Fehler aufgetreten wäre: Der Fehler oder die benutzerdefinierte Ausnahme beendet weder die Operationen des Triggers noch der gespeicherten Prozedur.

Wenn jedoch die WHEN ... DO-Anweisung oder der WHEN ... DO-Block nichts zum Behandeln oder Beheben des Fehlers tut, wird die DML-Anweisung (SELECT, INSERT, UPDATE, DELETE, MERGE), die den Fehler verursacht hat zurückgerollt, und keine der Anweisungen darunter im selben Anweisungsblock wird ausgeführt.

Wichtig

1. Wenn der Fehler nicht durch eine der DML-Anweisungen (SELECT, INSERT, UPDATE, DELETE, MERGE) verursacht wird, wird der gesamte Block der Anweisungen zurückgesetzt, nicht nur der, der einen Fehler verursacht hat. Alle Operationen in der WHEN ... DO-Anweisung werden ebenfalls zurückgesetzt. Dieselbe Einschränkung gilt für die Anweisung EXECUTE PROCEDURE. Lesen Sie eine interessante Diskussion über das Phänomen im Firebird Tracker-Ticket [CORE-4483](#).
2. Bei auswählbaren gespeicherten Prozeduren bleiben Ausgabezeilen, die bereits in früheren Iterationen einer FOR SELECT ... DO ... SUSPEND-Schleife an den Client übergeben wurden erhalten, wenn beim Abrufen von Zeilen eine Ausnahme ausgelöst wird.

Anwendungsbereiche einer WHEN ... DO Anweisung

Eine Anweisung WHEN ... DO fängt Fehler und Ausnahmen im aktuellen Anweisungsblock ab. Es fängt auch ähnliche Ausnahmen in verschachtelten Blöcken ab, wenn diese Ausnahmen nicht in ihnen behandelt wurden.

Alle Änderungen, die vor der Anweisung vorgenommen wurden, die den Fehler verursacht hat, sind für eine WHEN ... DO-Anweisung sichtbar. Wenn Sie jedoch versuchen, sie in einer autonomen Transaktion zu protokollieren, sind diese Änderungen nicht verfügbar, da die Transaktion, bei der die Änderungen stattfanden, zu dem Zeitpunkt, zu dem die autonome Transaktion gestartet wird, nicht festgeschrieben ist. Das untere Beispiel 4 zeigt dieses Verhalten.

Tipp

Bei der Behandlung von Ausnahmen ist es manchmal wünschenswert, die Ausnahme zu behandeln, indem eine Protokollnachricht geschrieben wird, um den Fehler zu markieren, und die Ausführung über den fehlerhaften Datensatz hinaus fortgesetzt wird. Protokolle können in reguläre Tabellen geschrieben werden, aber es gibt ein Problem damit: Die Protokolldatensätze „verschwinden“, wenn ein nicht behandelter Fehler dazu führt, dass das Modul nicht mehr ausgeführt wird und ein Rollback erfolgt. Die Verwendung von [externen Tabellen](#) kann hier nützlich sein, da Daten, die an sie geschrieben werden, transaktionsunabhängig sind. Die verknüpfte externe Datei ist immer noch vorhanden, unabhängig davon, ob der Gesamtprozess erfolgreich ist oder nicht.

Beispiele zur Verwendung von WHEN...DO:

1. Ersetzen des Standardfehlers durch einen benutzerdefinierten Fehler:

```
CREATE EXCEPTION COUNTRY_EXIST ' ';
SET TERM ^;
CREATE PROCEDURE ADD_COUNTRY (
    ACountryName COUNTRYNAME,
    ACurrency VARCHAR(10) )
AS
BEGIN
    INSERT INTO country (country, currency)
    VALUES (:ACountryName, :ACurrency);

    WHEN SQLCODE -803 DO
        EXCEPTION COUNTRY_EXIST 'Country already exists!';
```

```
END^
SET TERM ^;
```

2. Einen Fehler protokollieren und erneut in den WHEN-Block werfen:

```
CREATE PROCEDURE ADD_COUNTRY (
    ACountryName COUNTRYNAME,
    ACurrency VARCHAR(10) )
AS
BEGIN
    INSERT INTO country (country,
                        currency)
    VALUES (:ACountryName,
            :ACurrency);
    WHEN ANY DO
    BEGIN
        -- write an error in log
        IN AUTONOMOUS TRANSACTION DO
            INSERT INTO ERROR_LOG (PSQL_MODULE,
                                   GDS_CODE,
                                   SQL_CODE,
                                   SQL_STATE)
            VALUES ('ADD_COUNTRY',
                    GDSCODE,
                    SQLCODE,
                    SQLSTATE);
        -- Re-throw exception
        EXCEPTION;
    END
END
```

3. Behandeln mehrerer Fehler in einem WHEN-Block

```
...
WHEN GDSCODE GRANT_OBJ_NOTFOUND,
      GDSCODE GRANT_FLD_NOTFOUND,
      GDSCODE GRANT_NOPRIV,
      GDSCODE GRANT_NOPRIV_ON_BASE
DO
BEGIN
    EXECUTE PROCEDURE LOG_GRANT_ERROR(GDSCODE);
    EXIT;
END
...
```

Siehe auch: [EXCEPTION](#), [CREATE EXCEPTION](#), [SQLCODE](#) und [GDSCODE](#) Fehlercodes und Meldungen (1) und [SQLSTATE](#) Fehlercodes und Meldungen

Kapitel 8

Eingebaute Funktionen und Variablen

In diesem Abschnitt wird die große Sammlung der Kontextvariablen, Skalar- und Aggregat-Funktionen beschrieben.

Kontextvariablen

CURRENT_CONNECTION

Verfügbar in: DSQL, PSQL

Beschreibung: CURRENT_CONNECTION enthält den eindeutigen Bezeichner der aktuellen Verbindung.

Typ: INTEGER

Beispiele:

```
select current_connection from rdb$database
```

```
execute procedure P_Login(current_connection)
```

Der Wert in CURRENT_CONNECTION wird in der Datenbank Headerseite gespeichert und bei einer Wiederherstellung auf 0 zurückgesetzt. Seit Version 2.1 wird dieser Wert bei jeder neuen Verbindung inkrementiert. (In vorigen Versionen, wurde der Wert nur inkrementiert, wenn der Client diesen während der Sitzung ausgelesen hat.) Daraus resultiert, dass CURRENT_CONNECTION die Anzahl der Verbindungen seit dem Anlegen —oder der aktuellsten Wiederherstellung— der Datenbank anzeigt.

CURRENT_DATE

Verfügbar in: DSQL, PSQL, ESQL

Beschreibung: CURRENT_DATE gibt das aktuelle Serverdatum zurück.

Typ: DATE

Syntax:

```
CURRENT_DATE
```


Beispiele:

```
select current_date from rdb$database
-- ergibt z.B. 2011-10-03
```

Hinweise:

- Innerhalb eines PSQL-Moduls (Prozedur, Trigger oder ausführbarer Block) bleibt der Wert von CURRENT_DATE bei jedem Lesen konstant. Wenn mehrere Module einander aufrufen oder auslösen, bleibt der Wert während der gesamten Dauer des äußersten Moduls konstant. Wenn Sie einen fortschreitenden Wert in PSQL benötigen (z. B. um Zeitintervalle zu messen), verwenden Sie **TODAY**.

CURRENT_ROLE

Verfügbar in: DSQL, PSQL

Beschreibung: CURRENT_ROLE ist eine Kontextvariable, die die Rolle des derzeit verbundenen Benutzers enthält. Ist keine Rolle aktiv, ist CURRENT_ROLE NONE.

Typ: VARCHAR(31)

Beispiel:

```
if (current_role <> 'MANAGER')
  then exception only_managers_may_delete;
else
  delete from Customers where custno = :custno;
```

CURRENT_ROLE repräsentiert immer eine gültige Rolle oder NONE. Wenn sich ein Benutzer mit einer nicht-existenten Rolle verbindet, wird die Engine die Rolle stillschweigend auf NONE setzen, ohne einen Fehler zurückzugeben.

CURRENT_TIME

Verfügbar in: DSQL, PSQL, ESQL

Beschreibung: CURRENT_TIME gibt die derzeitige Serverzeit zurück. In Versionen vor 2.0, ist der Bruchteil immer „.0000“, was eine effektive Genauigkeit von 0 Dezimalstellen ergibt. Ab Firebird 2.0 können Sie eine Genauigkeit angeben, wenn Sie diese Variable abfragen. Der Standardwert ist immer noch 0 Dezimalstellen, d.h. Sekundengenauigkeit.

Typ: TIME

Syntax:

```
CURRENT_TIME [(precision)]
precision ::= 0 | 1 | 2 | 3
```

Das optionale Argument *precision* wird in ESQL nicht unterstützt.

Tabelle 8.1. CURRENT_TIME Parameter

| Parameter | Beschreibung |
|-------------|--|
| Genauigkeit | Präzision. Der Standardwert ist 0. In ESQL nicht unterstützt.. |

Beispiele:

```
select current_time from rdb$database
-- ergibt z.B. 14:20:19.6170
```

```
select current_time(2) from rdb$database
-- ergibt z.B. 14:20:23.1200
```

Hinweise:

- Im Gegensatz zu CURRENT_TIME wurde der Standardwert von CURRENT_TIMESTAMP auf 3 Dezimalstellen geändert. Somit ist CURRENT_TIMESTAMP nicht länger die genaue Summe aus CURRENT_DATE und CURRENT_TIME, es sei denn, Sie geben die Genauigkeit selbst an.
- Innerhalb eines PSQL-Moduls (Prozedur, Trigger oder ausführbarer Block) bleibt der Wert von CURRENT_TIME bei jedem Lesen konstant. Wenn mehrere Module einander aufrufen oder auslösen, bleibt der Wert während der gesamten Dauer des äußersten Moduls konstant. Wenn Sie einen fortschreitenden Wert in PSQL benötigen (z. B. um Zeitintervalle zu messen), verwenden Sie 'NOW'.

CURRENT_TIMESTAMP

Verfügbar in: DSQL, PSQL, ESQL

Beschreibung: CURRENT_TIMESTAMP gibt das aktuelle Datum und die Uhrzeit des aktuellen Servers zurück. In Versionen vor 2.0 war der Bruchteil immer „.0000“, was eine effektive Genauigkeit von 0 Dezimalstellen ergab. Ab Firebird 2.0 können Sie eine Genauigkeit angeben, wenn Sie diese Variable abfragen. Der Standardwert ist 3 Dezimalstellen, d. H. Millisekunden.

Typ: TIMESTAMP

Syntax:

```
CURRENT_TIMESTAMP [(precision)]
precision ::= 0 | 1 | 2 | 3
```

Das optionale Argument *precision* wird in ESQL nicht unterstützt.

Tabelle 8.2. CURRENT_TIMESTAMP Parameter

| Parameter | Beschreibung |
|-------------|---|
| Genauigkeit | Präzision. Der Standardwert ist 0. In ESQL nicht unterstützt. |

Beispiele:

```
select current_timestamp from rdb$database
-- ergibt z.B. 2008-08-13 14:20:19.6170
```

```
select current_timestamp(2) from rdb$database
-- ergibt z.B. 2008-08-13 14:20:23.1200
```

Hinweise:

- Die Standardpräzision von CURRENT_TIME ist immer noch 0 Dezimalstellen, sodass in Firebird 2.0 und höher CURRENT_TIMESTAMP nicht mehr die exakte Summe von CURRENT_DATE und CURRENT_TIME ergibt, außer Sie geben explizit eine Genauigkeit an.
- Innerhalb eines PSQL-Moduls (Prozedur, Trigger oder ausführbarer Block) bleibt der Wert von CURRENT_TIMESTAMP bei jedem Lesen konstant. Wenn mehrere Module einander aufrufen oder auslösen, bleibt der Wert während der gesamten Dauer des äußersten Moduls konstant. Wenn Sie einen fortschreitenden Wert in PSQL benötigen (z. B. um Zeitintervalle zu messen), verwenden Sie **'NOW'**.

CURRENT_TRANSACTION

Verfügbar in: DSQL, PSQL

Beschreibung: CURRENT_TRANSACTION beinhaltet den eindeutigen Bezeichner der aktuellen Transaktion.

Typ: INTEGER

Beispiele:

```
select current_transaction from rdb$database
```

```
New.Txn_ID = current_transaction;
```

Der Wert von CURRENT_TRANSACTION wird in der Datenbank-Header-Seite vorgehalten und beim Wiederherstellen auf 0 zurückgesetzt. Dieser wird mit jeder neuen Transaktion inkrementiert.

CURRENT_USER

Verfügbar in: DSQL, PSQL

Beschreibung: CURRENT_USER ist eine Kontextvariable, die den Namen des aktuell verbundenen Benutzers enthält. Diese ist vollständig äquivalent zu **USER**.

Typ: VARCHAR(31)

Beispiel:

```
create trigger bi_customers for customers before insert as
begin
  New.added_by = CURRENT_USER;
  New.purchases = 0;
end
```

DELETING

Verfügbar in: PSQL

Beschreibung: Nur in Triggern verfügbar. DELETING gibt an, ob der Trigger durch eine Löschoperation (DELETE) ausgelöst wurde. Vorgesehen für den Einsatz in [Multi-Aktions-Trigger](#).

Typ: boolean

Beispiel:

```
if (deleting) then
begin
  insert into Removed_Cars (id, make, model, removed)
    values (old.id, old.make, old.model, current_timestamp);
end
```

GDSCODE

Verfügbar in: PSQL

Beschreibung: In einem „WHEN ... DO“-Fehlerbehandlungsblock, enthält die GDSCODE-Kontextvariable die numerische Repräsentation des derzeitigen Firebird-Fehlercodes. Vor Firebird 2.0, wurde GDSCODE nur innerhalb eines WHEN GDSCODE-Handlers gesetzt. Nun kann es in WHEN ANY, WHEN SQLCODE und WHEN EXCEPTION auch nicht-null sein, vorausgesetzt, die Bedingung, die den Fehler verursacht, entspricht einem Firebird Fehlercode. Außerhalb der Fehler-Handler ist GDSCODE immer 0. Außerhalb von PSQL existiert es überhaupt nicht.

Typ: INTEGER

Beispiel:

```
when gdscode grant_obj_notfound, gdscode grant_fld_notfound,
  gdscode grant_nopriv, gdscode grant_nopriv_on_base
do
begin
  execute procedure log_grant_error(gdscode);
  exit;
end
```

Hinweis

Nach WHEN GDSCODE müssen Sie symbolische Namen wie z.B. grant_obj_notfound etc. verwenden. Jedoch ist die Kontextvariable GDSCODE ein INTEGER. Wenn Sie es mit einem bestimmten Fehler vergleichen möchten, muss der numerische Wert verwendet werden, z.B. 335544551 für grant_obj_notfound.

INSERTING

Verfügbar in: PSQL

Beschreibung: Nur in Triggern verfügbar. INSERTING gibt an, ob der Trigger durch eine Einfügeoperation (INSERTING) ausgelöst wurde. Vorgesehen für den Einsatz in [Multi-Aktions-Trigger](#).

Typ: boolean

Beispiel:

```
if (inserting or updating) then
begin
  if (new.serial_num is null) then
    new.serial_num = gen_id(gen_serials, 1);
end
```

NEW

Verfügbar in: PSQL, triggers only

Beschreibung: NEW beinhaltet die neue Version einer Datenbankzeile, die gerade eingefügt oder aktualisiert wurde. Seit Firebird 2.0 steht diese nur noch im Lesemodus in AFTER-Triggern zur Verfügung.

Typ: Datenzeile

Anmerkung

In Multi-Aktions-Triggern—eingeführt in Firebird 1.5—ist NEW immer verfügbar. Wurde der Trigger jedoch durch ein DELETE ausgelöst, existiert keine neue Version des Datensatzes. In diesem Falle wird das Lesen von NEW immer NULL zurückgeben; das Schreiben hierin wird in einem Laufzeitfehler resultieren.

'NOW'

Verfügbar in: DSQL, PSQL, ESQL

Geändert in: 2.0

Beschreibung: 'NOW' ist keine Variable, sondern ein String-Literal. Es ist jedoch speziell in dem Sinne, dass, wenn Sie es mittels CAST() zu einem Datum / Uhrzeit-Typ umwandeln, Sie das aktuelle Datum und / oder Uhrzeit erhalten. Der Bruchteil der Zeit war immer „.0000“, was eine effektive Sekundengenauigkeit ergab. Seit Firebird 2.0 beträgt die Genauigkeit 3 Dezimalstellen, d.h. Millisekunden. 'NOW' unterscheidet nicht zwischen Groß- und Kleinschreibung, und die Engine ignoriert beim Casting führende oder nachgestellte Leerzeichen.

Hinweis: Bitte beachten Sie, dass diese Abkürzungsausdrücke sofort bei der Analyse ausgewertet werden und gleich bleiben, solange die Anweisung vorbereitet bleibt. Selbst wenn eine Abfrage mehrere Male ausgeführt wird, ändert sich der Wert für z.B. „timestamp 'now'“ nicht, egal wie viel Zeit vergeht. Wenn Sie den Wert für den Fortschritt benötigen (d.h. bei jedem Aufruf ausgewertet werden), verwenden Sie eine vollständige Umwandlung.

Typ: CHAR(3)

Beispiele:

```
select 'Now' from rdb$database
-- ergibt 'Now'
```

```
select cast('Now' as date) from rdb$database
-- ergibt z.B. 2008-08-13
```

```
select cast('now' as time) from rdb$database
-- ergibt z.B. 14:20:19.6170
```

```
select cast('NOW' as timestamp) from rdb$database
-- ergibt z.B. 2008-08-13 14:20:19.6170
```

Kurzschreibweisen für Casts von Datums- und Zeit-Datentypen für die letzten drei Statements:

```
select date 'Now' from rdb$database
select time 'now' from rdb$database
select timestamp 'NOW' from rdb$database
```

Hinweise:

- 'NOW' gibt immer die aktuelle Uhrzeit bzw. das aktuelle Datum zurück, auch in PSQL-Modulen, in denen `CURRENT_DATE`, `CURRENT_TIME` und `CURRENT_TIMESTAMP` während der gesamt Laufzeit der äußeren Routine den selben Wert liefern. Dies macht 'NOW' nützlich zum Messen von Zeitintervallen in Triggern, Prozeduren und ausführbaren Blöcken.
- Außer in der oben genannten Situation, ist das Lesen von `CURRENT_DATE`, `CURRENT_TIME` und `CURRENT_TIMESTAMP` dem Casting grundsätzlich vorzuziehen 'NOW'. Seien Sie sich jedoch bewusst, dass `CURRENT_TIME` standardmäßig sekundengenau ist; um millisekundengenaue Genauigkeit zu erhalten, nutzen Sie `CURRENT_TIME(3)`.

OLD

Verfügbar in: PSQL, triggers only

Beschreibung: OLD beinhaltet die existierende Version einer Datenbankzeile, gerade vor dem Aktualisieren oder Löschen. Seit Firebird 2.0 ist dies nur lesend.

Typ: Datenzeile

Anmerkung

In Multi-Aktions-Triggern —eingeführt in Firebird 1.5— ist OLD immer verfügbar. Wird der Trigger jedoch durch ein INSERT ausgelöst, existiert offensichtlich keine Vor-Version des Datensatzes. In diesem Falle, gibt OLD immer NULL zurück; schreiben wird in einem Laufzeitfehler enden.

ROW_COUNT

Verfügbar in: PSQL

Geändert in: 2.0

Beschreibung: Die Kontextvariable `ROW_COUNT` enthält die Anzahl der Zeilen, die durch das letzte DML-Statement (INSERT, UPDATE, DELETE, SELECT oder FETCH) im derzeitigen Trigger, Stored Procedure oder ausführbaren Block betroffen sind.

Typ: INTEGER

Beispiel:

```
update Figures set Number = 0 where id = :id;
if (row_count = 0) then
  insert into Figures (id, Number) values (:id, 0);
```

Verhalten von *SELECT* und *FETCH*:

- Nach einem einzelnen *SELECT*, ist *ROW_COUNT* gleich 1, sofern eine Datenzeile zurückgegeben wurde, andernfalls 0.
- In einer *FOR SELECT*-Schleife wird *ROW_COUNT* mit jeder Iteration inkrementiert (beginnend bei 0 vor dem ersten Durchlauf).
- Nach einem *FETCH* durch einen Cursor, ist *ROW_COUNT* gleich 1, falls eine Datenzeile zurückgegeben wurde und andernfalls 0. Werden weitere Datenzeilen durch den gleichen Cursor geholt, wird *ROW_COUNT* nicht über 1 hinaus inkrementiert.
- In Firebird 1.5.x ist *ROW_COUNT*0 nach jeder Art von *SELECT*-Anweisung.

Anmerkung

ROW_COUNT kann nicht verwendet werden um die Anzahl der betroffenen Zeilen eines *EXECUTE STATEMENT*- oder *EXECUTE PROCEDURE*-Befehls zu erhalten.

SQLCODE

Verfügbar in: PSQL

Veraltet ab: 2.5.1

Beschreibung: In einem „WHEN ... DO“-Fehlerbehandlungsblock enthält die *SQLCODE*-Kontextvariable den aktuellen SQL-Fehlercode. Vor Firebird 2.0 wurde *SQLCODE* nur in *WHEN SQLCODE*- und *WHEN ANY*-Handlern. Nun kann es auch in *WHEN GDSCODE* und *WHEN EXCEPTION* ungleich null sein, vorausgesetzt, die Bedingung, die den Fehler auslöst, entspricht einem SQL-Fehlercode. Außerhalb der Error-Handler ist *SQLCODE* immer 0. Außerhalb PSQL ist es überhaupt nicht vorhanden.

Typ: INTEGER

Beispiel:

```
when any
do
begin
  if (sqlcode <> 0) then
    Msg = 'An SQL error occurred!';
  else
    Msg = 'Something bad happened!';
  exception ex_custom Msg;
end
```

Wichtiger Hinweis: SQLCODE ist im Rahmen des SQL-2003-konformen *SQLSTATE*-Statuscodes veraltet. Die Unterstützung für SQLCODE und WHEN SQLCODE wird in einer späteren Firebird-Version entfernt werden.

SQLSTATE

Verfügbar in: PSQL

Aufgenommen in: 2.5.1

Beschreibung: In einem „WHEN ... DO“-Error-Handler enthält die SQLSTATE-Kontextvariable den 5 Zeichen langen, SQL-2003-konformen Statuscode, der vom Statement erzeugt wurde, das den Fehler verursacht hat. Außerhalb der Error-Handler ist SQLSTATE immer '00000'. Außerhalb PSQL ist es nicht verfügbar.

Typ: CHAR(5)

Beispiel:

```
when any
do
begin
    Msg = case sqlstate
        when '22003' then 'Numeric value out of range.'
        when '22012' then 'Division by zero.'
        when '23000' then 'Integrity constraint violation.'
        else 'Something bad happened! SQLSTATE = ' || sqlstate
    end;
    exception ex_custom Msg;
end
```

Hinweise:

- SQLSTATE soll SQLCODE ersetzen. Letzteres ist jetzt in Firebird veraltet und wird in einer zukünftigen Version verschwinden.
- Firebird unterstützt (noch) nicht die Syntax „WHEN SQLSTATE ... DO“. Sie müssen WHEN ANY verwenden und die Variable SQLSTATE im Handler testen.
- Jeder SQLSTATE-Code ist die Verkettung einer 2-Zeichen-Klasse und einer 3-Zeichen-Unterklasse. Die Klassen 00 (erfolgreicher Abschluss), 01 (Warnung) und 02 (keine Daten) repräsentieren *Abschlussbedingungen*. Jeder Statuscode außerhalb dieser Klassen ist eine *Ausnahme*. Da die Klassen 00, 01 und 02 keinen Fehler verursachen, werden sie niemals in der SQLSTATE-Variablen angezeigt.
- Für eine vollständige Auflistung der SQLSTATE-Codes, konsultieren Sie bitte den Abschnitt [SQLSTATE Fehlercodes und Meldungen](#) in *Appendix B: Fehlercodes und Meldungen*.

'TODAY'

Verfügbar in: DSQL, PSQL, ESQL

Beschreibung: 'TODAY' ist keine Variable, sondern ein String-Literal. Es ist jedoch speziell im Sinne dass Sie das aktuelle Datum erhalten, wenn Sie ein CAST() zu einem Datum / einer Zeit durchführen. 'TODAY' unterscheidet nicht zwischen Groß- und Kleinschreibung. Die Engine ignoriert führende oder nachstehende Leerzeichen beim Umwandeln.

Typ: CHAR(5)

Beispiele:

```
select 'Today' from rdb$database
-- ergibt 'Today'
```

```
select cast('Today' as date) from rdb$database
-- ergibt z.B. 2011-10-03
```

```
select cast('TODAY' as timestamp) from rdb$database
-- ergibt z.B. 2011-10-03 00:00:00.0000
```

Kurzschreibweisen für Casts von Datums- und Zeit-Datentypen für die letzten beiden Statements:

```
select date 'Today' from rdb$database
select timestamp 'TODAY' from rdb$database
```

Hinweise:

- 'TODAY' gibt immer das aktuelle Datum zurück, auch in PSQL-Modulen, in denen CURRENT_DATE, CURRENT_TIME und CURRENT_TIMESTAMP den gleichen Rückgabewert während der gesamten Dauer der äußersten Routine hat. Dies macht 'TODAY' nützlich für die Messung von Zeitintervallen in Triggern, Prozeduren und ausführbaren Blöcken (zumindest, wenn Ihre Prozeduren für mehrere Tage läuft.).
- Außer in der oben genannten Situation ist das Lesen von CURRENT_DATE im Allgemeinen besser als das Konvertieren von 'NOW'.

'TOMORROW'

Verfügbar in: DSQL, PSQL, ESQL

Beschreibung: 'TOMORROW' ist keine Variable, sondern ein String-Literal. Es ist jedoch speziell im Sinne dass Sie das Datum des nächsten Tages erhalten, wenn Sie ein CAST() zu einem Datum / einer Zeit durchführen. 'TOMORROW' unterscheidet nicht zwischen Groß- und Kleinschreibung. Die Engine ignoriert führende oder nachstehende Leerzeichen beim Umwandeln. Siehe auch 'TODAY'.

Typ: CHAR(8)

Beispiele:

```
select 'Tomorrow' from rdb$database
-- ergibt 'Tomorrow'
```

```
select cast('Tomorrow' as date) from rdb$database
-- ergibt z.B. 2011-10-04
```

```
select cast('TOMORROW' as timestamp) from rdb$database
-- ergibt z.B. 2011-10-04 00:00:00.0000
```

Kurzschreibweisen für Casts von Datums- und Zeit-Datentypen für die letzten beiden Statements:

```
select date 'Tomorrow' from rdb$database
```

```
select timestamp 'TOMORROW' from rdb$database
```

UPDATING

Verfügbar in: PSQL

Beschreibung: Nur in Triggern verfügbar. UPDATING gibt an, ob der Trigger durch eine Aktualisierungsoperation (UPDATING) ausgelöst wurde. Vorgesehen für den Einsatz in [Multi-Aktions-Trigger](#).

Typ: boolean

Beispiel:

```
if (inserting or updating) then
begin
  if (new.serial_num is null) then
    new.serial_num = gen_id(gen_serials, 1);
end
```

'YESTERDAY'

Verfügbar in: DSQL, PSQL, ESQL

Beschreibung: 'YESTERDAY' ist keine Variable, sondern ein String-Literal. Es ist jedoch speziell im Sinne dass Sie das aktuelle Datum erhalten, wenn Sie ein CAST() zu einem Datum / einer Zeit durchführen. 'YESTERDAY' unterscheidet nicht zwischen Groß- und Kleinschreibung. Die Engine ignoriert führende oder nachstehende Leerzeichen beim Umwandeln. Siehe auch ['TODAY'](#).

Typ: CHAR(9)

Beispiele:

```
select 'Yesterday' from rdb$database
-- ergibt 'Tomorrow'
```

```
select cast('Yesterday as date) from rdb$database
-- ergibt z.B. 2011-10-02
```

```
select cast('YESTERDAY' as timestamp) from rdb$database
-- ergibt z.B. 2011-10-02 00:00:00.0000
```

[Kurzschreibweisen für Casts von Datums- und Zeit-Datentypen](#) für die letzten beiden Statements:

```
select date 'Yesterday' from rdb$database
select timestamp 'YESTERDAY' from rdb$database
```

USER

Verfügbar in: DSQL, PSQL

Beschreibung: USER ist eine Kontextvariable, die den aktuellen Namen des derzeit verbundenen Benutzers vorhält. Es ist vollständig äquivalent zu `CURRENT_USER`.

Typ: VARCHAR(31)

Beispiel:

```
create trigger bi_customers for customers before insert as
begin
  New.added_by = USER;
  New.purchases = 0;
end
```

Skalarfunktionen

Upgrader: BITTE LESEN!

Eine große Anzahl von Funktionen, die in früheren Versionen von Firebird als externe Funktionen (UDFs) implementiert wurden, wurden schrittweise als interne (eingebaute) Funktionen neu implementiert. Wenn eine externe Funktion mit dem gleichen Namen wie eine integrierte Funktion in Ihrer Datenbank deklariert ist, bleibt sie dort und überschreibt alle internen Funktionen desselben Namens.

Um die internen Funktionen verfügbar sind, müssen Sie entweder ein `DROP` der UDF durchführen oder mittels `ALTER EXTERNAL FUNCTION` den Namen der UDF ändern.

Funktionen zum Arbeiten mit Kontextvariablen

RDB\$GET_CONTEXT()

Anmerkung

RDB\$GET_CONTEXT und sein Gegenpart RDB\$SET_CONTEXT sind vordefinierte UDFs. Sie werden hier als interne Funktionen geführt, da sie immer präsent sind—der Benutzer muss nichts tun, damit diese verfügbar sind.

Verfügbar in: DSQL, PSQL * Als deklariertes UDF sollte es in ESQL verfügbar sein

Beschreibung: Ruft den Wert einer Kontextvariablen aus einem der Namespaces SYSTEM, USER_SESSION und USER_TRANSACTION auf.

Syntax:

```
RDB$GET_CONTEXT ('<namespace>', '<varname>')
```

<namespace> ::= SYSTEM | USER_SESSION | USER_TRANSACTION
<varname> ::= Eine Zeichenfolge, bei der die Groß- und Kleinschreibung beachtet wird
Maximal 80 Zeichen

Tabelle 8.3. RDB\$GET_CONTEXT-Funktionsparameters

| Parameter | Beschreibung |
|-----------|--|
| namespace | Namespace |
| varname | Variablenamen. Groß- und Kleinschreibung Die maximale Länge beträgt 80 Zeichen |

Rückgabebetyp: VARCHAR(255)

Die Namespaces: Die Namespaces USER_SESSION und USER_TRANSACTION sind initial leer. Der Benutzer kann hierin Variablen erstellen und mittels RDB\$SET_CONTEXT() festlegen und diese mit RDB\$GET_CONTEXT() zurückgeben lassen. Der Zugriff auf den Namespace SYSTEM erfolgt nur lesend. Dieser enthält eine Menge vordefinierter Variablen, die in folgender Tabelle aufgelistet sind.

Tabelle 8.4. Kontextvariablen im SYSTEM-Namespace

| | |
|------------------|---|
| DB_NAME | Entweder der vollständige Pfad der Datenbank oder—falls das Verbinden mittels des Pfades deaktiviert ist—dessen Alias. |
| NETWORK_PROTOCOL | Das Verbindungsprotokoll: 'TCPv4', 'WNET', 'XNET' oder NULL. |
| CLIENT_ADDRESS | Für TCPv4 ist dies die IP-Adresse. Für XNET die lokale Prozess-ID. Für alle anderen Protokolle ist diese Variable NULL. |
| CURRENT_USER | Identisch zur globalen Variable CURRENT_USER . |
| CURRENT_ROLE | Identisch zur globalen Variable CURRENT_ROLE . |
| SESSION_ID | Identisch zur globalen Variable CURRENT_CONNECTION |
| TRANSACTION_ID | Identisch zur globalen Variable CURRENT_TRANSACTION |
| ISOLATION_LEVEL | Die Isolationsstufe der aktuellen Transaktion: 'READ COMMITTED', 'SNAPSHOT' oder 'CONSISTENCY'. |
| ENGINE_VERSION | Die (Server-) Version der Firebird-Engine. Hinzugefügt in 2.1. |

Rückgabewerte und Fehlerverhalten: Wenn die abgefragte Variable im angegebenen Namespace vorhanden ist, wird ihr Wert als eine Zeichenfolge mit max. 255 Zeichen. Wenn der Namespace nicht vorhanden ist oder wenn Sie versuchen, auf eine nicht vorhandene Variable im Namespace SYSTEM zuzugreifen, wird ein Fehler ausgegeben. Wenn Sie eine nicht vorhandene Variable in einem der anderen Namespaces abfragen, wird NULL zurückgegeben. Sowohl Namespaces als auch Variablenamen müssen als single-quoted, case-sensitive, nicht NULL-Strings angegeben werden.

Beispiele:

```
select rdb$get_context('SYSTEM', 'DB_NAME') from rdb$database
```

```
New.UserAddr = rdb$get_context('SYSTEM', 'CLIENT_ADDRESS');
```

```
insert into MyTable (TestField)
values (rdb$get_context('USER_SESSION', 'MyVar'))
```

Siehe auch: [RDB\\$SET_CONTEXT\(\)](#)

RDB\$SET_CONTEXT()

Anmerkung

RDB\$SET_CONTEXT und dessen Gegenpart RDB\$GET_CONTEXT sind vordefinierte UDFs. Sie werden hier als interne Funktionen geführt, da sie immer präsent sind—der Benutzer muss nichts tun, damit diese verfügbar sind.

Verfügbar in: DSQL, PSQL * Als deklariertes UDF sollte es in ESQL verfügbar sein

Beschreibung: Erstellt, setzt oder löscht eine Variable in einem der vom Benutzer beschreibbaren Namespaces USER_SESSION und USER_TRANSACTION.

Syntax:

```
RDB$SET_CONTEXT ('<namespace>', '<varname>', <value> | NULL)

<namespace> ::= USER_SESSION | USER_TRANSACTION
<varname>   ::= A case-sensitive string of max. 80 characters
<value>     ::= A value of any type, as long as it's castable
               to a VARCHAR(255)
```

Tabelle 8.5. RDB\$SET_CONTEXT-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| namespace | Namespace |
| varname | Variablenamen. Groß- und Kleinschreibung Die maximale Länge beträgt 80 Zeichen |
| value | Daten eines beliebigen Typs, sofern sie in VARCHAR (255) umgewandelt werden können |

Rückgabetyt: INTEGER

Die Namespaces: Die Namespaces USER_SESSION und USER_TRANSACTION sind initial leer. Der Benutzer kann hierin Variablen erstellen und mittels RDB\$SET_CONTEXT() festlegen und diese mit RDB\$GET_CONTEXT() zurückgeben lassen. Der Kontext USER_SESSION ist an die derzeitige Verbindung gebunden Variablen in USER_TRANSACTION existieren nur in der Transaktion, in der sie erstellt wurden. Wenn die Transaktion endet, werden der Kontext und alle hierin definierten Variablen zerstört.

Rückgabewerte und Fehlerverhalten: Die Funktion gibt 1 zurück, wenn die Variable bereits vor dem Aufruf vorhanden, und 0, falls dies nicht der Fall war. Um eine Variable aus einem Kontext zu entfernen, setzen Sie sie auf NULL. Wenn der angegebene Namespace nicht existiert, wird ein Fehler ausgelöst. Sowohl Namespaces als auch Variablenamen müssen als nicht-NULL-Zeichenketten ind einzelne Anführungszeichen eingegeben werden. Beachten Sie dabei Groß- und Kleinschreibung.

Beispiele:

```
select rdb$set_context('USER_SESSION', 'MyVar', 493) from rdb$database
```

```
rdb$set_context('USER_SESSION', 'RecordsFound', RecCounter);
```

```
select rdb$set_context('USER_TRANSACTION', 'Savepoints', 'Yes')
from rdb$database
```

Hinweise:

- Die maximale Anzahl der Variablen in einem einzelnen Kontext beträgt 1000.
- Alle USER_TRANSACTION-Variablen überleben das [ROLLBACK RETAIN](#) (siehe ROLLBACK-Optionen) oder [ROLLBACK TO SAVEPOINT](#) unverändert, unabhängig zu welchem Zeitpunkt der Transaktion diese gesetzt wurden.
- Aufgrund seiner UDF-ähnlichen Eigenschaft kann RDB\$SET_CONTEXT —nur in PSQL— wie eine void-Funktion aufgerufen werden, ohne das Ergebnis wie im zweiten Beispiel oben zuzuweisen. Reguläre interne Funktionen erlauben diese Art der Verwendung nicht.

Siehe auch: [RDB\\$GET_CONTEXT\(\)](#)

Mathematische Funktionen

ABS()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
ABS (number)
```

Tabelle 8.6. ABS Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------------|
| number | Ausdruck eines numerischen Typs |

Rückgabety: Numerisch

Beschreibung: Gibt den absoluten Wert des Arguments zurück.

ACOS()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
ACOS (number)
```

Tabelle 8.7. ACOS Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| number | Ausdruck eines numerischen Typs im Bereich [-1; 1] |

Rückgabotyp: DOUBLE PRECISION

Beschreibung: Gibt den Arkuskosinus des Arguments zurück.

- Das Ergebnis ist ein Winkel im Bereich [0, pi].
- Ist das Argument außerhalb der Bereichs [-1, 1], wird NaN zurückgegeben.

ASIN()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
ASIN ( number )
```

Tabelle 8.8. ASIN Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| number | Ausdruck eines numerischen Typs im Bereich [-1; 1] |

Rückgabotyp: DOUBLE PRECISION

Beschreibung: Gibt den Arkussinus des Arguments zurück.

- Das Ergebnis ist ein Winkel im Bereich [-pi/2, pi/2].
- Liegt das Argument außerhalb des Bereichs [-1, 1], wird NaN zurückgegeben.

ATAN()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
ATAN ( number )
```

Tabelle 8.9. ATAN Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------------|
| number | Ausdruck eines numerischen Typs |

Rückgabotyp: DOUBLE PRECISION

Beschreibung: Die Funktion ATAN gibt den Arcustangens des Arguments zurück. Das Ergebnis ist ein Winkel im Bereich $\langle -\pi/2, \pi/2 \rangle$.

ATAN2()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
ATAN2 (y, x)
```

Tabelle 8.10. ATAN2-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------------|
| x | Ausdruck eines numerischen Typs |
| y | Ausdruck eines numerischen Typs |

Rückgabotyp: DOUBLE PRECISION

Beschreibung: Gibt den Winkel zurück, dessen Sinus-Kosinus-Verhältnis durch die beiden Argumente gegeben ist und dessen Sinus- und Kosinus-Zeichen den Vorzeichen der Argumente entsprechen. Dies ermöglicht Ergebnisse über den gesamten Kreis einschließlich der Winkel $-\pi/2$ und $\pi/2$.

- Das Ergebnis ist ein Winkel im Bereich $[-\pi, \pi]$.
- Ist x negativ, ist das Ergebnis π , falls y gleich 0 ist, und $-\pi$ falls y gleich -0 ist.
- Wenn sowohl y als auch x 0 sind, ist das Ergebnis bedeutungslos. Beginnend mit Firebird 3 wird ein Fehler ausgelöst, wenn beide Argumente 0 sind. Bei Version 2.5.4 ist es in niedrigeren Versionen immer noch nicht behoben. Für weitere Details, besuchen Sie [Tracker-Ticket CORE-3201](#).

Hinweise:

- Eine vollständig äquivalente Beschreibung dieser Funktion ist die folgende: $\text{ATAN2}(y, x)$ ist ein Winkel zwischen der positiven X-Achse und der Linie vom Ursprung zum Punkt (x, y) . Damit wird offensichtlich, dass $\text{ATAN2}(0, 0)$ nicht definiert ist.
- Ist x größer als 0, ist $\text{ATAN2}(y, x)$ das gleiche wie $\text{ATAN}(y/x)$.
- Wenn Sinus und Kosinus des Winkels bereits bekannt sind, gibt $\text{ATAN2}(\sin, \cos)$ den Winkel zurück.

CEIL(), CEILING()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#) (Betrifft nur CEILING)

Syntax:

```
CEIL[ING] (number)
```

Tabelle 8.11. CEIL[ING]-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------------|
| number | Ausdruck eines numerischen Typs |

Rückgabety: BIGINT or DOUBLE PRECISION

Beschreibung: Gibt die kleinste ganze Zahl zurück, die größer oder gleich dem Argument ist.

Siehe auch: [FLOOR\(\)](#)

COS()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
COS (angle)
```

Tabelle 8.12. COS Funktionsparameter

| Parameter | Beschreibung |
|-----------|------------------------|
| angle | Ein Winkel in Bogenmaß |

Rückgabety: DOUBLE PRECISION

Beschreibung: Gibt den Kosinus eines Winkels zurück. Das Argument muss im Bogenmaß angegeben werden.

- Jedes nicht-NULL-Ergebnis ist—offensichtlich—im Bereich [-1, 1].

COSH()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
COSH (number)
```

Tabelle 8.13. COSH Funktionsparameter

| Parameter | Beschreibung |
|-----------|----------------------------------|
| number | Eine Zahl eines numerischen Typs |

Rückgabotyp: DOUBLE PRECISION

Beschreibung: Gibt den Hyperbelkosinus des Arguments zurück.

- Beliebiges non-NULL-Ergebnis liegt im Bereich [1, INF].

COT()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
COT (angle)
```

Tabelle 8.14. COT Funktionsparameter

| Parameter | Beschreibung |
|-----------|------------------------|
| angle | Ein Winkel in Bogenmaß |

Rückgabotyp: DOUBLE PRECISION

Beschreibung: Gibt den Kotangens eines Winkels zurück. Das Argument muss im Bogenmaß angegeben werden.

EXP()

Verfügbar in: DSQL, PSQL

Syntax:

```
EXP (number)
```

Tabelle 8.15. EXP Funktionsparameter

| Parameter | Beschreibung |
|-----------|----------------------------------|
| number | Eine Zahl eines numerischen Typs |

Rückgabotyp: DOUBLE PRECISION

Beschreibung: Gibt das natürliche Exponential zurück, e^{number}

Siehe auch: LN()

FLOOR()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
FLOOR (number)
```

Tabelle 8.16. FLOOR Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------------|
| number | Ausdruck eines numerischen Typs |

Rückgabotyp: BIGINT oder DOUBLE PRECISION

Beschreibung: Gibt die größte ganze Zahl zurück, die kleiner oder gleich dem Argument ist.

Siehe auch: CEIL()/ CEILING()

LN()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
LN (number)
```

Tabelle 8.17. LN Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------------|
| number | Ausdruck eines numerischen Typs |

Beschreibung: Gibt den natürlichen Logarithmus des Arguments zurück.

- Ein Fehler wird ausgelöst, wenn das Argument negativ oder 0 ist.

Rückgabotyp: DOUBLE PRECISION

Siehe auch: [EXP\(\)](#)

LOG()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
LOG (x, y)
```

Tabelle 8.18. LOG-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---|
| x | Base. Ein Ausdruck eines numerischen Typs |
| y | Ausdruck eines numerischen Typs |

Rückgabotyp: DOUBLE PRECISION

Beschreibung: Gibt den x-basierten Logarithmus von y zurück.

- Wenn eines der Argumente 0 oder niedriger ist, wird ein Fehler ausgelöst. (Vor 2.5 würde dies NaN, ±INF oder 0 ergeben, abhängig von den genauen Argumentwerten)
- Wenn beide Argumente 1 sind, wird NaN zurückgegeben/returned.
- Wenn $x = 1$ und $y < 1$, -INF wird zurückgegeben.
- Wenn $x = 1$ und $y > 1$, INF wird zurückgegeben.

LOG10()

Verfügbar in: DSQL, PSQL

Geändert in: 2.5

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
LOG10 (number)
```

Tabelle 8.19. LOG10 Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------------|
| number | Ausdruck eines numerischen Typs |

*Rückgabety*p: DOUBLE PRECISION

Beschreibung: Gibt den 10-basierten Logarithmus des Arguments zurück.

- Ein Fehler wird ausgelöst, wenn das Argument negativ oder 0 ist. (In Versionen vor 2.5 würden solche Werte zu NaN und INF resultieren.)

MOD()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
MOD ( a , b )
```

Tabelle 8.20. MOD-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------------|
| a | Ausdruck eines numerischen Typs |
| b | Ausdruck eines numerischen Typs |

*Rückgabety*p: INTEGER oder BIGINT

Beschreibung: Gibt den Rest einer Ganzzahldivision zurück.

- Nicht ganzzahlige Argumente werden vor der Division gerundet. Demnach ergibt „7.5 mod 2.5“ 2 (8 mod 3), nicht 0.

PI()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
PI ( )
```

*Rückgabety*p: DOUBLE PRECISION

Beschreibung: Gibt eine Annäherung an den Wert von Pi.

POWER()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
POWER (x, y)
```

Tabelle 8.21. POWER-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------------|
| x | Ausdruck eines numerischen Typs |
| y | Ausdruck eines numerischen Typs |

Rückgabetyp: DOUBLE PRECISION

Beschreibung: Gibt x hoch y zurück.

- Ist x negativ, wird ein Fehler ausgegeben.

RAND()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
RAND ( )
```

Rückgabetyp: DOUBLE PRECISION

Beschreibung: Gibt eine Zufallszahl zwischen 0 und 1 zurück.

ROUND()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
ROUND (<number> [, <scale>])
```

Tabelle 8.22. ROUND-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| number | Ausdruck eines numerischen Typs |
| scale | Eine ganze Zahl, die die Anzahl der Dezimalstellen angibt, auf die gerundet werden soll, z.B.: |

| Parameter | Beschreibung |
|-----------|---|
| | 2 zum Runden auf das nächste Vielfache von 0,01 1 zum Runden auf das nächste Vielfache von 0,1 0 zum Runden auf die nächste ganze Zahl -1 zum Runden auf das nächste Vielfache von 10 -2 zum Runden auf das nächste Vielfache von 100 |

Rückgabotyp: INTEGER, (scaled) BIGINT or DOUBLE PRECISION

Beschreibung: Rundet eine Zahl auf die nächste ganze Zahl. Wenn der Bruchteil genau 0,5 ist, ist das Runden für positive Zahlen aufwärts und für negative Zahlen abwärts. Mit dem optionalen Argument *scale* kann die Zahl anstelle von ganzen Zahlen auf Zehnerpotenzen (Zehner, Hunderter, Zehntel, Hundertstel usw.) gerundet werden.

Wichtig

- Wenn Sie das Verhalten der externen Funktion ROUND gewohnt sind, beachten Sie bitte, dass die *interne* Funktion immer die Hälfte von Null weg, d.h. abwärts für negative Zahlen, abrundet.

Beispiele: Ist das Argument *scale* vorhanden, hat das Ergebnis üblicherweise die gleiche Genauigkeit wie das erste Argument:

```
ROUND(123.654, 1) -- ergibt 123.700 (nicht 123.7)
ROUND(8341.7, -3) -- ergibt 8000.0 (nicht 8000)
ROUND(45.1212, 0) -- ergibt 45.0000 (nicht 45)
```

Andernfalls ist die Ergebnisgenauigkeit 0:

```
ROUND(45.1212) -- ergibt 45
```

SIGN()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
SIGN (number)
```

Tabelle 8.23. SIGN Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------------|
| number | Ausdruck eines numerischen Typs |

Rückgabotyp: SMALLINT

Beschreibung: Returns the sign of the argument: -1, 0 or 1.

SIN()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
SIN (angle)
```

Tabelle 8.24. SIN Funktionsparameter

| Parameter | Beschreibung |
|-----------|------------------------|
| angle | Ein Winkel im Bogenmaß |

Rückgabotyp: DOUBLE PRECISION

Beschreibung: Gibt den Sinus eines Winkels zurück. Das Argument muss im Bogenmaß angegeben werden.

- Beliebiges nicht-NULL-Ergebnis liegt—offensichtlich—im Bereich [-1, 1].

SINH()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
SINH (number)
```

Tabelle 8.25. SINH Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------------|
| number | Ausdruck eines numerischen Typs |

Rückgabotyp: DOUBLE PRECISION

Beschreibung: Gibt den Hyperbelsinus des Arguments zurück.

SQRT()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

`SQRT (number)`

Tabelle 8.26. SQRT Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------------|
| number | Ausdruck eines numerischen Typs |

Rückgabety: DOUBLE PRECISION

Beschreibung: Gibt die Quadratwurzel des Arguments zurück.

TAN()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

`TAN (angle)`

Tabelle 8.27. TAN Funktionsparameter

| Parameter | Beschreibung |
|-----------|------------------------|
| angle | Ein Winkel im Bogenmaß |

Rückgabety: DOUBLE PRECISION

Beschreibung: Gibt die Tangente eines Winkels zurück. Das Argument muss im Bogenmaß angegeben werden.

TANH()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

`TANH (number)`

Tabelle 8.28. TANH-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------------|
| number | Ausdruck eines numerischen Typs |

Rückgabotyp: DOUBLE PRECISION

Beschreibung: Gibt den Hyperbeltangens des Arguments zurück.

- Aufgrund von Rundungen liegt ein nicht-NULL-Ergebnis im Bereich [-1, 1] (mathematisch ist es <-1, 1>).

TRUNC()

Verfügbar in: DSQL, PSQL

Syntax:

```
TRUNC (<number> [, <scale>])
```

Tabelle 8.29. TRUNC-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---|
| number | Ausdruck eines numerischen Typs |
| | <p>Eine Ganzzahl, die die Anzahl der Dezimalstellen angibt, auf die die Abschneidung angewendet werden soll, z.</p> <p>2 zum Abschneiden auf das nächste Vielfache von 0,01 1 zum Abschneiden auf das nächste Vielfache von 0,1 0 zum Abschneiden auf die nächste ganze Zahl -1 zum Abschneiden auf das nächste Vielfache von 10 -2 zum Abschneiden auf das nächste Vielfache von 100</p> |

Rückgabotyp: INTEGER, (scaled) BIGINT or DOUBLE PRECISION

Beschreibung: Gibt den ganzzahligen Teil einer Zahl zurück. Mit dem optionalen Argument *scale* kann die Zahl anstelle von ganzen Zahlen auf Zehnerpotenzen (Zehner, Hunderter, Zehntel, Hundertstel usw.) abgeschnitten werden.

Hinweise:

- Wenn das Argument *scale* vorhanden ist, hat das Ergebnis normalerweise die gleiche Genauigkeit wie das erste Argument, z.B.
 - TRUNC(789.2225, 2) ergibt 789.2200 (nicht 789.22)
 - TRUNC(345.4, -2) ergibt 300.0 (nicht 300)
 - TRUNC(-163.41, 0) ergibt -163.00 (nicht -163)

Andernfalls ist die Genauigkeit 0:

- TRUNC(-163.41) ergibt -163

Wichtig

Wenn Sie das Verhalten der [externen Funktion TRUNCATE](#) untersuchen, beachten Sie bitte, dass die *interne* Funktion TRUNC immer gegen Null abschneidet, d.h. aufwärts für negative Zahlen.

Functions for Working with Strings

ASCII_CHAR()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
ASCII_CHAR (<code>)
```

Tabelle 8.30. ASCII_CHAR Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| code | Eine Ganzzahl im Bereich von 0 bis 255 |

Rückgabebetyp: [VAR]CHAR(1) CHARACTER SET NONE

Beschreibung: Gibt das ASCII-Zeichen zurück, das der im Argument übergebenen Zahl entspricht.

Wichtig

- Wenn Sie das Verhalten der UDF ASCII_CHAR gewohnt sind, die eine leere Zeichenfolge zurückgibt, wenn das Argument 0 ist, beachten Sie bitte, dass die interne Funktion hier korrekt ein Zeichen mit dem ASCII-Code 0 zurückgibt.

ASCII_VAL()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
ASCII_VAL (ch)
```

Tabelle 8.31. ASCII_VAL Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| ch | Eine Zeichenfolge des Datentyps [VAR]CHAR oder ein Text-BLOB mit der maximalen Größe von 32.767 Byte |

Rückgabotyp: SMALLINT

Beschreibung: Gibt den ASCII-Code des übergebenen Zeichens zurück.

- Wenn das Argument eine Zeichenfolge mit mehr als einem Zeichen ist, wird der ASCII-Code des ersten Zeichens zurückgegeben.
- Wenn das Argument eine leere Zeichenfolge ist, wird 0 zurückgegeben.
- Wenn das Argument NULL ist, wird NULL zurückgegeben.
- Wenn das erste Zeichen der Argument-Zeichenfolge multi-Byte ist, wird ein Fehler ausgelöst. (Ein Fehler in Firebird 2.1—2.1.3 und 2.5 führt dazu, dass ein Fehler ausgelöst wird, wenn *irgendein* Zeichen in der Zeichenfolge Multibyte ist. Dies ist in den Versionen 2.1.4 und 2.5.1 behoben.)

BIT_LENGTH()

Verfügbar in: DSQL, PSQL

Syntax:

```
BIT_LENGTH (string)
```

Tabelle 8.32. BIT_LENGTH Funktionsparameter

| Parameter | Beschreibung |
|-----------|-------------------------------------|
| string | Ein Ausdruck eines Zeichenfolgetyps |

Rückgabotyp: INTEGER

Beschreibung: Gibt die Länge in Bits der Eingabezeichenfolge an. Bei Multi-Byte-Zeichensätzen ist dies möglicherweise weniger als die Anzahl der Zeichen mal 8 mal die „formale“ Anzahl der Bytes pro Zeichen wie in RDB\$CHARACTER_SETS.

Anmerkung

Bei Argumenten vom Typ CHAR berücksichtigt diese Funktion die gesamte Länge der formalen Zeichenfolge (z. B. die deklarierte Länge eines Felds oder einer Variablen). Wenn Sie die „logische“ Bitlänge erhalten möchten, ohne die nachfolgenden Leerzeichen zu zählen, schneiden sie das Argument mittels TRIM rechtsseitig ab, bevor Sie es an BIT_LENGTH übergeben.

BLOB-Unterstützung: Seit Firebird 2.1 unterstützt diese Funktion vollständig Text BLOBs beliebiger Länge und Zeichensatz.

Beispiele:

```
select bit_length('Hello!') from rdb$database
-- ergibt 48
```

```
select bit_length(_iso8859_1 'Grüß di!') from rdb$database
-- ergibt 64: ü und ß nehmen je ein Byte in Anspruch in ISO8859_1
```

```
select bit_length
  (cast (_iso8859_1 'Grüß di!' as varchar(24) character set utf8))
from rdb$database
-- ergibt 80: ü und ß belegen je zwei Bytes in UTF8
```

```
select bit_length
  (cast (_iso8859_1 'Grüß di!' as char(24) character set utf8))
from rdb$database
-- ergibt 208: alle 24 CHAR-Positionen zählen und zwei von ihnen haben 16 Bit.
```

Siehe auch: [OCTET_LENGTH\(\)](#), [CHARACTER_LENGTH\(\)](#)

CHAR_LENGTH(), CHARACTER_LENGTH()

Verfügbar in: DSQL, PSQL

Syntax:

```
CHAR_LENGTH (str)
CHARACTER_LENGTH (string)
```

Tabelle 8.33. CHAR[ACTER]_LENGTH Funktionsparameter

| Parameter | Beschreibung |
|-----------|-------------------------------------|
| string | Ein Ausdruck eines Zeichenfolgetyps |

Rückgabotyp: INTEGER

Beschreibung: Gibt die Länge der Zeichen der Eingabezeichenfolge an.

Hinweise

- Mit Argumenten vom Typ CHAR gibt diese Funktion die formale Stringlänge (d.h. die deklarierte Länge eines Felds oder einer Variablen) zurück. Wenn Sie die „logische“ Länge erhalten möchten, ohne die nachfolgenden Leerzeichen zu zählen, [TRIM](#) das Argument, bevor es an CHAR[ACTER]_LENGTH übergeben wird.
- **BLOB-Untersützung:** Seit Firebird 2.1 unterstützt diese Funktion Text-BLOBs beliebiger Länge und beliebiger Zeichensätze.

Beispiele:

```
select char_length('Hello!') from rdb$database
-- ergibt 6
```

```
select char_length(_iso8859_1 'Grüß di!') from rdb$database
-- ergibt 8
```

```
select char_length
  (cast (_iso8859_1 'Grüß di!' as varchar(24) character set utf8))
from rdb$database
```

```
-- ergibt 8; die Tatsache, dass ü und ß jeweils zwei Bytes belegen, ist irrelevant

select char_length
  (cast (_iso8859_1 'Grüß di!' as char(24) character set utf8))
from rdb$database
-- ergibt 24: alle 24 CHAR-Positionen zählen
```

Siehe auch: [BIT_LENGTH\(\)](#), [OCTET_LENGTH\(\)](#)

HASH()

Verfügbar in: DSQL, PSQL

Syntax:

```
HASH (string)
```

Tabelle 8.34. HASH Funktionsparameter

| Parameter | Beschreibung |
|-----------|-------------------------------------|
| string | Ein Ausdruck eines Zeichenfolgetyps |

Beschreibung: Gibt einen Hash-Wert für die Eingabezeichenfolge zurück. Diese Funktion unterstützt vollständig Text-BLOBs beliebiger Länge und beliebige Zeichensätze.

Rückgabotyp: BIGINT

LEFT()

Verfügbar in: DSQL, PSQL

Syntax:

```
LEFT (string, length)
```

Tabelle 8.35. LEFT-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---|
| string | Ein Ausdruck eines Zeichenfolgetyps |
| number | Ganze Zahl. Definiert die Anzahl der zurückzugebenden Zeichen |

Rückgabotyp: VARCHAR or BLOB

Beschreibung: Gibt den äußersten linken Teil der Argument-Zeichenfolge zurück. Die Anzahl der Zeichen ist im zweiten Argument angegeben.

- Diese Funktion unterstützt vollständig TextBLOBs beliebiger Länge, einschließlich solcher mit einem Multi-Byte-Zeichensatz.

- Falls *string* ein BLOB ist, ist das Ergebnis ein BLOB. Andernfalls ist das Ergebnis ein VARCHAR(*n*), wobei *n* die Länge der Eingabezeichenfolge ist.
- Wenn das Argument *length* die Länge der Zeichenfolge überschreitet, wird die Eingabezeichenfolge unverändert zurückgegeben.
- Wenn das Argument *length* keine Ganzzahl ist, wird kaufmännisch gerundet, z.B. wird 0.5 zu 0, 1.5 wird zu 2, 2.5 wird zu 3, 3.5 wird zu 4, etc.

Siehe auch: [RIGHT\(\)](#)

LOWER()

Verfügbar in: DSQL, ESQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details unten](#)

Syntax:

```
LOWER (string)
```

Tabelle 8.36. LOWER FunktionsparameterS

| Parameter | Beschreibung |
|-----------|-------------------------------------|
| string | Ein Ausdruck eines Zeichenfolgetyps |

Rückgabotyp: (VAR)CHAR or BLOB

Beschreibung: Gibt das Kleinbuchstabenäquivalent der Eingabezeichenfolge zurück. Das genaue Ergebnis hängt vom Zeichensatz ab. Bei ASCII oder NONE zum Beispiel sind nur ASCII-Zeichen kleiner; mit OCTETS wird die gesamte Zeichenfolge unverändert zurückgegeben. Seit Firebird 2.1 unterstützt diese Funktion auch vollständig Text-BLOBs beliebiger Länge und Zeichensatzes.

Namenskonflikt

Da LOWER ein reserviertes Wort ist, hat die interne Funktion Vorrang, auch wenn die externe Funktion mit diesem Namen ebenfalls deklariert wurde. Um die (untergeordnete!) externe Funktion aufzurufen, verwenden Sie doppelte Anführungszeichen und die genaue Großschreibung, wie in "LOWER"(*str*).

Beispiel:

```
select Sheriff from Towns
where lower(Name) = 'cooper''s valley'
```

Siehe auch: [UPPER](#)

LPAD()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
LPAD(str, endlen [, padstr])
```

Tabelle 8.37. LPAD-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| str | Ein Ausdruck eines Zeichenfolgetyps |
| endlen | Länge der Ausgabezeichenfolge |
| padstr | Das Zeichen oder die Zeichenfolge, die zum Auffüllen der Quellzeichenfolge bis zur angegebenen Länge verwendet werden soll. Standard ist Leerzeichen (' ') |

Rückgabotyp: VARCHAR oder BLOB

Beschreibung: Füllt eine Zeichenfolge linksseitig mit Leerzeichen oder mit einer benutzerdefinierten Zeichenfolge, bis eine bestimmte Länge erreicht ist.

- Diese Funktion unterstützt vollständig Text-BLOBs beliebiger Länge und Zeichensätze.
- Wenn *str* ein BLOB ist, ist das Ergebnis ebenfalls ein BLOB. Andernfalls ist das Ergebnis ein VARCHAR(*endlen*).
- Wenn *padstr* angegeben wurde und gleich ' ' (Leeres Zeichen) ist, findet kein Auffüllen statt.
- Ist *endlen* kleiner als die aktuelle Länge der Zeichenkette, wird die Zeichenkette auf *endlen* Zeichen abgeschnitten, auch wenn *padstr* ein leeres Zeichen ist.

Anmerkung

In Firebird 2.1 2.1.3 waren alle Nicht-BLOB-Ergebnisse vom Typ VARCHAR(32765), was es ratsam machte, sie auf eine bescheidenere Größe zu übertragen. Dies ist nicht mehr der Fall.

Warnung

Wenn diese Funktion in einem BLOB verwendet wird, muss diese Funktion möglicherweise das gesamte Objekt in den Speicher laden. Es wird zwar versucht, den Speicherverbrauch zu begrenzen, dies kann jedoch die Leistung beeinträchtigen, wenn riesige BLOBs beteiligt sind.

Beispiele:

```
lpad ('Hello', 12)           -- ergibt '      Hello'
lpad ('Hello', 12, '-')     -- ergibt '-----Hello'
lpad ('Hello', 12, ' ')    -- ergibt 'Hello'
lpad ('Hello', 12, 'abc')   -- ergibt 'abcabcaHello'
lpad ('Hello', 12, 'abcdefghij') -- ergibt 'abcdefghHello'
lpad ('Hello', 2)          -- ergibt 'He'
lpad ('Hello', 2, '-')     -- ergibt 'He'
lpad ('Hello', 2, ' ')    -- ergibt 'He'
```

Siehe auch: [RPAD\(\)](#)

OCTET_LENGTH()

Verfügbar in: DSQL, PSQL

Syntax:

```
OCTET_LENGTH (string)
```

Tabelle 8.38. OCTET_LENGTH Funktionsparameter

| Parameter | Beschreibung |
|-----------|-------------------------------------|
| string | Ein Ausdruck eines Zeichenfolgetyps |

Rückgabotyp: INTEGER

Beschreibung: Gibt die Länge in Bytes (Oktetts) der Eingabezeichenfolge an. Bei Multi-Byte-Zeichensätzen ist dies möglicherweise weniger als die Anzahl der Zeichen mal der „formalen“ Anzahl der Bytes pro Zeichen, wie in RDB\$CHARACTER_SETS gefunden.

Anmerkung

Bei Argumenten vom Typ CHAR berücksichtigt diese Funktion die gesamte Länge der formalen Zeichenfolge (z. B. die deklarierte Länge eines Felds oder einer Variablen). Wenn Sie die „logische“ Byte-Länge erhalten möchten, ohne die nachfolgenden Leerzeichen zu zählen, [beschneiden Sie die Zeichenfolge mittels TRIM](#), bevor sie diese an OCTET_LENGTH übergeben wird.

BLOB support: Diese Funktion unterstützt vollständig Text-BLOBs beliebiger Länge und Zeichensätze.

Beispiele:

```
select octet_length('Hello!') from rdb$database
-- ergibt 6
```

```
select octet_length(_iso8859_1 'Grüß di!') from rdb$database
-- ergibt 8: ü und ß belegen ein Byte pro Zeichen in ISO8859_1
```

```
select octet_length
(cast (_iso8859_1 'Grüß di!' as varchar(24) character set utf8))
from rdb$database
-- ergibt 10: ü und ß belegen zwei Bytes je Zeichen in UTF8
```

```
select octet_length
(cast (_iso8859_1 'Grüß di!' as char(24) character set utf8))
from rdb$database
-- ergibt 26: alle 24 CHAR-Positionen zählen, und zwei Zeichen belegen 2 Bytes
```

Siehe auch: [BIT_LENGTH\(\)](#), [CHARACTER_LENGTH\(\)](#)

OVERLAY()

Verfügbar in: DSQL, PSQL

Syntax:

```
OVERLAY (string PLACING replacement FROM pos [FOR length])
```

Tabelle 8.39. OVERLAY-Funktionsparameter

| Parameter | Beschreibung |
|-------------|---|
| string | Die Zeichenfolge, in die der Austausch stattfindet |
| replacement | Ersatzzeichenfolge |
| pos | Die Position, von der aus der Austausch stattfindet (Startposition) |
| length | Die Anzahl der Zeichen, die überschrieben werden sollen |

Rückgabotyp: VARCHAR or BLOB

Beschreibung: OVERLAY() überschreibt einen Teil einer Zeichenfolge mit einer anderen Zeichenfolge. Standardmäßig ist die Anzahl der Zeichen, die aus der Host-Zeichenfolge entfernt (überschrieben) werden, gleich der Länge der Ersatzzeichenfolge. Mit dem optionalen vierten Argument kann eine andere Anzahl von Zeichen zum Entfernen angegeben werden.

- Diese Funktion unterstützt BLOBs beliebiger Länge.
- Wenn *string* oder *replacement* ein BLOB ist, ist das Ergebnis ebenfalls ein BLOB. Andernfalls ist das Ergebnis ein VARCHAR(*n*) wobei *n* die Summe der Längen von *string* und *replacement* bildet.
- Wie üblich in SQL-String-Funktionen ist *pos* 1-basierend
- Ist *pos* hinter dem Ende von *string*, wird *replacement* direkt hinter *string* platziert.
- Ist die Anzahl der Zeichen von *pos* bis zum Ende von *string* kleiner als die Länge von *replacement* (oder als Argument *length*, falls vorhanden), wird *string* an Stelle *pos* abgeschnitten und *replacement* direkt dahinter platziert.
- Der Effekt einer „FOR0“-Klausel ist, dass *replacement* einfach in *string* eingesetzt wird.
- Ist eines der Argumente NULL ist, ist das Ergebnis ebenfalls NULL.
- Wenn *pos* oder *length* keine Ganzzahl ist, wird kaufmännisch gerundet, d.h. 0.5 wird 0, 1.5 wird 2, 2.5 wird 2, 3.5 wird 4, etc.

Beispiele:

```
overlay ('Goodbye' placing 'Hello' from 2)      -- ergibt 'GHelloe'
overlay ('Goodbye' placing 'Hello' from 5)      -- ergibt 'GoodHello'
overlay ('Goodbye' placing 'Hello' from 8)      -- ergibt 'GoodbyeHello'
overlay ('Goodbye' placing 'Hello' from 20)     -- ergibt 'GoodbyeHello'

overlay ('Goodbye' placing 'Hello' from 2 for 0) -- ergibt 'GHellooodbye'
overlay ('Goodbye' placing 'Hello' from 2 for 3) -- ergibt 'GHellooby'
overlay ('Goodbye' placing 'Hello' from 2 for 6) -- ergibt 'GHello'
overlay ('Goodbye' placing 'Hello' from 2 for 9) -- ergibt 'GHello'
```

```
overlay ('Goodbye' placing '' from 4)          -- ergibt 'Goodbye'
overlay ('Goodbye' placing '' from 4 for 3)    -- ergibt 'Gooe'
overlay ('Goodbye' placing '' from 4 for 20)   -- ergibt 'Goo'

overlay ('' placing 'Hello' from 4)           -- ergibt 'Hello'
overlay ('' placing 'Hello' from 4 for 0)     -- ergibt 'Hello'
overlay ('' placing 'Hello' from 4 for 20)    -- ergibt 'Hello'
```

Warnung

Wenn in einem BLOB verwendet, muss die Funktion das gesamte Objekt in den Speicher laden. Dies kann die Leistung beeinflussen, wenn große BLOBs genutzt werden.

Siehe auch: [REPLACE\(\)](#)

POSITION()

Verfügbar in: DSQL, PSQL

Syntax:

```
POSITION (substr IN string)
| POSITION (substr, string [, startpos])
```

Tabelle 8.40. POSITION-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| substr | Der Teilstring, dessen Position gesucht werden soll |
| string | Die Zeichenfolge, die durchsucht werden soll |
| startpos | Die Position in <i>string</i> , in der die Suche beginnen soll |

Rückgabotyp: INTEGER

Beschreibung: Gibt die (1-basierte) Position des ersten Vorkommens einer Teilzeichenfolge in einer Hostzeichenfolge zurück. Mit dem optionalen dritten Argument beginnt die Suche bei einem gegebenen Offset, wobei etwaige Übereinstimmungen unberücksichtigt bleiben, die früher in der Zeichenfolge auftreten können. Wenn keine Übereinstimmung gefunden wird, ist das Ergebnis 0.

Hinweise:

- Das optionale Argument wird nur in der zweiten Syntax (Kommasyntax) unterstützt.
- Die leere Zeichenfolge wird als Teilzeichenfolge jeder Zeichenfolge betrachtet. Wenn *substr* gleich "" (leerer String) ist und ist *string* ungleich NULL, ergibt dies:
 - 1 wenn *startpos* nicht übergeben wurde;
 - *startpos* wenn *startpos* innerhalb *string* liegt;
 - 0 wenn *startpos* hinter dem Ende von *string* liegt.

Hinweis: Ein Bug in Firebird 2.1—2.1.3 und 2.5 bewirken, dass `POSITION` immer 1 zurückgibt, wenn `substr` ein Leerstring ist. Dies wurde in 2.1.4 und 2.5.1 gefixt.

- Diese Funktion unterstützt Text-BLOBSs beliebiger Größe und Zeichensatzes.

Beispiele:

```
position ('be' in 'To be or not to be')      -- ergibt 4
position ('be', 'To be or not to be')      -- ergibt 4
position ('be', 'To be or not to be', 4)   -- ergibt 4
position ('be', 'To be or not to be', 8)   -- ergibt 17
position ('be', 'To be or not to be', 18)  -- ergibt 0
position ('be' in 'Alas, poor Yorick!')    -- ergibt 0
```

Warnung

Wenn diese Funktion in einem BLOB verwendet wird, muss diese Funktion möglicherweise das gesamte Objekt in den Speicher laden. Dies kann sich auf die Leistung auswirken, wenn riesige BLOB beteiligt sind.

Siehe auch: [SUBSTRING](#)

REPLACE()

Verfügbar in: DSQL, PSQL

Syntax:

```
REPLACE (str, find, repl)
```

Tabelle 8.41. REPLACE-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---|
| str | Die Zeichenfolge, in der der Austausch stattfinden soll |
| find | Die Zeichenfolge, nach der gesucht werden soll |
| repl | Die Ersatzzeichenfolge |

Rückgabotyp: VARCHAR or BLOB

Beschreibung: Ersetzt alle Vorkommen einer Teilzeichenfolge in einer Zeichenfolge.

- Diese Funktion unterstützt Text-BLOBSs beliebiger Größe und Zeichensatzes.
- Wenn eines der Argumente ein BLOB ist, ist das Ergebnis ebenfalls ein BLOB. Andernfalls ist das Ergebnis ein `VARCHAR(n)` mit `n` berechnet aus den Längen von `str`, `find` und `repl` in der Form, dass sogar die größtmögliche Anzahl an Ersetzungen das Feld nicht überschreiten würde.
- Wenn `find` ein leerer String ist, bleibt `str` unverändert.
- Wenn `repl` ein Leerstring ist, werden alle Vorkommen von `find` in `str` entfernt.

- Ist ein Argument NULL, wird das Ergebnis immer NULL sein, auch wenn nichts ersetzt würde.

Beispiele:

```
replace ('Billy Wilder', 'il', 'oog')      -- ergibt 'Boogly Woogder'
replace ('Billy Wilder', 'il', '')        -- ergibt 'Bly Wder'
replace ('Billy Wilder', null, 'oog')     -- ergibt NULL
replace ('Billy Wilder', 'il', null)      -- ergibt NULL
replace ('Billy Wilder', 'xyz', null)     -- ergibt NULL (!)
replace ('Billy Wilder', 'xyz', 'abc')    -- ergibt 'Billy Wilder'
replace ('Billy Wilder', '', 'abc')       -- ergibt 'Billy Wilder'
```

Warnung

Wenn diese Funktion in einem BLOB verwendet wird, muss diese Funktion möglicherweise das gesamte Objekt in den Speicher laden. Dies kann sich auf die Leistung auswirken, wenn riesige BLOB beteiligt sind.

Siehe auch: [OVERLAY\(\)](#), [SUBSTRING\(\)](#), [POSITION\(\)](#), [CHAR\[ACTER\]_LENGTH\(\)](#)

REVERSE()

Verfügbar in: DSQL, PSQL

Syntax:

```
REVERSE (str)
```

Tabelle 8.42. REVERSE Funktionsparameter

| Parameter | Beschreibung |
|-----------|-------------------------------------|
| string | Ein Ausdruck eines Zeichenfolgetyps |

Rückgabety: VARCHAR

Beschreibung: Kehrt eine Zeichenfolge zurück.

Beispiele:

```
reverse ('spoonful')      -- ergibt 'lufnoops'
reverse ('Was it a cat I saw?') -- ergibt '?was I tac a ti saW'
```

Tipp

Diese Funktion ist sehr nützlich, wenn Sie Stringendungen, z.B. bei Domain-Namen oder E-Mail-Adressen:

```
create index ix_people_email on people
  computed by (reverse(email));

select * from people
  where reverse(email) starting with reverse('.br');
```

RIGHT()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
RIGHT (string, length)
```

Tabelle 8.43. RIGHT-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---|
| string | Ein Ausdruck eines Zeichenfolgetyps |
| length | Ganze Zahl. Definiert die Anzahl der zurückzugebenden Zeichen |

Rückgabotyp: VARCHAR or BLOB

Beschreibung: Gibt den rechten Teil der Argument-Zeichenfolge zurück. Die Anzahl der Zeichen ist im zweiten Argument angegeben.

- Diese Funktion unterstützt Text BLOBs beliebiger Länge, aber hat einen Fehler in den Versionen 2.1 2.1.3 und 2.5, der es scheitert mit Text BLOBs größer als 1024 Bytes die einen Multi-Byte-Zeichensatz haben. Dies wurde in den Versionen 2.1.4 und 2.5.1 behoben.
- Ist *string* ein BLOB, ist das Ergebnis ebenfalls ein BLOB. Andernfalls ist das Ergebnis ein VARCHAR(*n*), wobei *n* die Länge der Eingabezeichenfolge ist.
- Wenn das Argument *length* die Länge der Zeichenfolge überschreitet, wird die Eingabezeichenfolge unverändert zurückgegeben.
- Ist das Argument *length* keine Ganzzahl, wird kaufmännisch gerundet, d.h. 0.5 wird 0, 1.5 wird 2, 2.5 wird 2, 3.5 wird 4, etc.

Warnung

Wenn diese Funktion in einem BLOB verwendet wird, muss diese Funktion möglicherweise das gesamte Objekt in den Speicher laden. Dies kann sich auf die Leistung auswirken, wenn riesige BLOB beteiligt sind.

Siehe auch: [LEFT\(\)](#), [SUBSTRING\(\)](#)

RPAD()

Verfügbar in: DSQL, PSQL

Geändert in: 2.5 (backported to 2.1.4)

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
RPAD (str, endlen [, padstr])
```

Tabelle 8.44. RPAD-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| str | Ein Ausdruck eines Zeichenfolgetyps |
| endlen | Länge der Ausgabezeichenfolge |
| padstr | Das Zeichen oder die Zeichenfolge, die zum Auffüllen der Quellzeichenfolge bis zur angegebenen Länge verwendet werden soll. Standard ist Leerzeichen (' ') |

Rückgabety: VARCHAR or BLOB

Beschreibung: Füllt eine Zeichenfolge rechtsseitig mit Leerzeichen oder mit einer benutzerdefinierten Zeichenfolge, bis eine bestimmte Länge erreicht ist.

- Diese Funktion unterstützt vollständig Text-BLOBs beliebiger Länge und Zeichensätze.
- Wenn *str* ein BLOB ist, ist das Ergebnis ebenfalls ein BLOB. Andernfalls ist das Ergebnis ein VARCHAR(*endlen*).
- Wenn *padstr* angegeben wurde und gleich ' ' (Leeres Zeichen) ist, findet kein Auffüllen statt.
- Ist *endlen* kleiner als die aktuelle Länge der Zeichenkette, wird die Zeichenkette auf *endlen* Zeichen abgeschnitten, auch wenn *padstr* ein leeres Zeichen ist.

Anmerkung

In Firebird 2.1 2.1.3 waren alle Nicht-BLOB-Ergebnisse vom Typ VARCHAR(32765), was es ratsam machte, sie auf eine bescheidenere Größe zu übertragen. Dies ist nicht mehr der Fall.

Warnung

Wenn diese Funktion in einem BLOB verwendet wird, muss diese Funktion möglicherweise das gesamte Objekt in den Speicher laden. Es wird zwar versucht, den Speicherverbrauch zu begrenzen, dies kann jedoch die Leistung beeinträchtigen, wenn riesige BLOBs beteiligt sind.

Beispiele:

```

rpad ('Hello', 12)           -- ergibt 'Hello      '
rpad ('Hello', 12, '-')     -- ergibt 'Hello-----'
rpad ('Hello', 12, '')      -- ergibt 'Hello'
rpad ('Hello', 12, 'abc')   -- ergibt 'Helloabcabca'
rpad ('Hello', 12, 'abcdefghij') -- ergibt 'Helloabcdefghg'
rpad ('Hello', 2)          -- ergibt 'He'
rpad ('Hello', 2, '-')     -- ergibt 'He'

```

```
rpad ('Hello', 2, ' ') -- ergibt 'He'
```

Warnung

Wenn diese Funktion in einem BLOB verwendet wird, muss diese Funktion möglicherweise das gesamte Objekt in den Speicher laden. Dies kann sich auf die Leistung auswirken, wenn riesige BLOB beteiligt sind.

Siehe auch: [LPAD\(\)](#)

SUBSTRING()

Verfügbar in: DSQL, PSQL

Geändert in: 2.5.1

Syntax:

```
SUBSTRING (str FROM startpos [FOR length])
```

Tabelle 8.45. SUBSTRING-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| str | Ein Ausdruck eines Zeichenfolgetyps |
| startpos | Integer-Ausdruck, die Position, von der aus der Teilstring abgerufen werden soll |
| length | Die Anzahl der Zeichen, die nach <startpos> liegen |

Rückgabetypen: VARCHAR(*n*) oder BLOB

Beschreibung: Die Anzahl der abzurufenden Zeichen nach dem Zurückgeben der Teilzeichenfolge einer Zeichenfolge beginnend an der angegebenen Position, entweder bis zum Ende der Zeichenfolge oder mit einer bestimmten Länge.

Diese Funktion gibt den Teilstring zurück, der an der Zeichenposition *startpos* beginnt (die erste Position ist 1). Ohne das Argument FOR werden alle verbleibenden Zeichen in der Zeichenfolge zurückgegeben. Mit FOR werden *length* Zeichen oder der Rest der Zeichenfolge zurückgegeben, je nachdem, welcher Wert kürzer ist.

In Firebird 1.x müssen *startpos* und *length* Integer-Literale sein. In 2.0 und höher können sie beliebige gültige Integer-Ausdrücke sein.

Ab Firebird 2.1 unterstützt diese Funktion vollständig binäre und text BLOBs beliebiger Länge und Zeichensatz. Wenn *str* ein BLOB ist, ist das Ergebnis auch ein BLOB. Für alle anderen Argumenttypen ist das Ergebnis eine VARCHAR(*n*). Zuvor war der Ergebnistyp CHAR(*n*), wenn das Argument eine CHAR(*n*) oder ein String-Literal.

Für nicht-BLOB-Agumente entspricht die Breite des Ergebnisfelds immer der Länge von *str*, unabhängig von *startpos* und *length*. Also, `substring ('pinhead' von 4 für 2)` gibt eine VARCHAR (7) zurück, die die Zeichenkette 'he' enthält.

Ist ein Argument NULL, ist das Ergebnis NULL.

Bugs

- Wenn *str* ein BLOB ist und das *length* Argument nicht vorhanden ist, ist die Ausgabe auf 32767 Zeichen begrenzt. Problemumgehung: Geben Sie bei langen BLOB immer *char_length (str)* — oder eine ausreichend hohe Ganzzahl — als drittes Argument an, es sei denn, Sie sind sicher, dass die angeforderte Teilzeichenfolge darin passt 32767 Zeichen.

Dieser Fehler wurde in Version 2.5.1 behoben. Der Fix wurde auch nach 2.1.5 zurückportiert.

- Ein älterer Fehler in Firebird 2.0, der dazu führte, dass die Funktion „false emptystrings“ zurückgab, wenn *startpos* oder *length* NULL war, wurde behoben.

Beispiel:

```
insert into AbbrNames(AbbrName)
select substring(LongName from 1 for 3) from LongNames
```

Warnung

Wenn diese Funktion in einem BLOB verwendet wird, muss diese Funktion möglicherweise das gesamte Objekt in den Speicher laden. Dies kann sich auf die Leistung auswirken, wenn riesige BLOB beteiligt sind.

Siehe auch: [POSITION](#), [LEFT](#), [RIGHT](#), [CHAR\[ACTER\]_LENGTH](#)

TRIM()

Verfügbar in: DSQL, PSQL

Syntax:

```
TRIM ([<adjust>] str)

<adjust> ::= { [<where>] [what] } FROM

<where> ::= BOTH | LEADING | TRAILING
```

Tabelle 8.46. TRIM-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| str | Ein Ausdruck eines Zeichenfolgetyps |
| where | Die Position, von der der Teilstring entfernt werden soll—BOTH LEADING TRAILING. BOTH ist der Standard |
| what | Der Teilstring, der entfernt werden soll (mehrfach, wenn mehrere Übereinstimmungen vorkommen) vom Anfang dem Ende beider Seiten des Eingabestrings <str>. Standard ist das Leerzeichen (' ') |

Rückgabotyp: VARCHAR(*n*) or BLOB

Beschreibung: Entfernt führende und / oder nachgestellte Leerzeichen (oder optional andere Zeichenfolgen) aus der Eingabezeichenfolge. Seit Firebird 2.1 unterstützt diese Funktion vollständig Text BLOBs beliebiger Länge und Zeichensatzes.

Beispiele:

```
select trim (' Waste no space ') from rdb$database
-- ergibt 'Waste no space'
```

```
select trim (leading from ' Waste no space ') from rdb$database
-- ergibt 'Waste no space '
```

```
select trim (leading '.' from ' Waste no space ') from rdb$database
-- ergibt ' Waste no space '
```

```
select trim (trailing '!' from 'Help!!!!') from rdb$database
-- ergibt 'Help'
```

```
select trim ('la' from 'lalala I love you Ella') from rdb$database
-- ergibt ' I love you El'
```

```
select trim ('la' from 'Lalala I love you Ella') from rdb$database
-- ergibt 'Lalala I love you El'
```

Hinweise:

- Wenn *str* ein BLOB ist, ist das Ergebnis ein BLOB. Andernfalls ist das Ergebnis ein VARCHAR(*n*) mit der formalen Länge *n* des Strings *str*.
- Die zu entfernende Teilzeichenfolge darf, falls angegeben, nicht größer als 32767 Byte sein. Wenn diese Teilzeichenfolge jedoch *wiederholt* an der Kopf- oder Endstelle von *str* ist, kann die Gesamtzahl der entfernten Byte viel größer sein. (Die Einschränkung der Größe des Teilstrings wird in Firebird 3 aufgehoben.)

Warnung

Wenn diese Funktion in einem BLOB verwendet wird, muss diese Funktion möglicherweise das gesamte Objekt in den Speicher laden. Dies kann sich auf die Leistung auswirken, wenn riesige BLOB beteiligt sind.

UPPER()

Verfügbar in: DSQL, ESQL, PSQL

Syntax:

```
UPPER (str)
```

Tabelle 8.47. UPPER Funktionsparameter

| Parameter | Beschreibung |
|-----------|-------------------------------------|
| str | Ein Ausdruck eines Zeichenfolgetyps |

Rückgabotyp: (VAR)CHAR or BLOB

Beschreibung: Gibt das Großbuchstabenäquivalent der Eingabezeichenfolge zurück. Das genaue Ergebnis hängt vom Zeichensatz ab. Bei ASCII oder NONE zum Beispiel sind nur ASCII-Zeichen größer; mit OCTETS wird die gesamte Zeichenfolge unverändert zurückgegeben. Seit Firebird 2.1 unterstützt diese Funktion auch vollständig Text-BLOBs beliebiger Länge und Zeichensatzes.

Beispiele:

```
select upper(_iso8859_1 'Débâcle')
from rdb$database
-- ergibt 'DÉBÂCLE' (before Firebird 2.0: 'DÉBÂCLE')
```

```
select upper(_iso8859_1 'Débâcle' collate fr_fr)
from rdb$database
-- ergibt 'DEBACLE', following French uppercasing rules
```

Siehe auch: LOWER

Date and Time Functions

DATEADD()

Verfügbar in: DSQL, PSQL

Geändert in: 2.5

Syntax:

```
DATEADD (<args>)

<args>      ::= <amount> <unit> TO <datetime>
              | <unit>, <amount>, <datetime>

<amount>    ::= an integer expression (negative to subtract)
<unit>      ::= YEAR | MONTH | WEEK | DAY
              | HOUR | MINUTE | SECOND | MILLISECOND
<datetime>  ::= a DATE, TIME or TIMESTAMP expression
```

Tabelle 8.48. DATEADD-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| amount | Ein ganzzahliger Ausdruck vom Typ SMALLINT, INTEGER oder BIGINT. Ein negativer Wert wird subtrahiert |
| unit | Datums-/Zeit-Einheit |
| datetime | Ein Ausdruck der Typen DATE, TIME oder TIMESTAMP |

Rückgabotyp: DATE, TIME oder TIMESTAMP

Beschreibung: Fügt einem Datum / Uhrzeit-Wert die angegebene Anzahl von Jahren, Monaten, Wochen, Tagen, Stunden, Minuten, Sekunden oder Millisekunden hinzu. (Die WEEK-Einheit ist neu in 2.5.)

- Der Ergebnistyp wird durch das dritte Argument bestimmt.
- Mit TIMESTAMP und DATE-Argumenten können alle Einheiten verwendet werden. (Vor Firebird 2.5 waren Einheiten kleiner als DAY nicht erlaubt für DATE s.)
- Mit TIME-Argumenten können nur HOUR, MINUTE, SECOND und MILLISECOND genutzt werden.

Beispiele:

```
dateadd (28 day to current_date)
dateadd (-6 hour to current_time)
dateadd (month, 9, DateOfConception)
dateadd (-38 week to DateOfBirth)
dateadd (minute, 90, time 'now')
dateadd (? year to date '11-Sep-1973')
```

Siehe auch: DATEDIFF, Operationen, die Datums- und Zeitfunktionen verwenden

DATEDIFF()

Verfügbar in: DSQL, PSQL

Geändert in: 2.5

Syntax:

```
DATEDIFF (<args>)

<args>      ::= <unit> FROM <moment1> TO <moment2>
              | <unit>, <moment1>, <moment2>

<unit>      ::= YEAR | MONTH | WEEK | DAY
              | HOUR | MINUTE | SECOND | MILLISECOND
<momentN>  ::= a DATE, TIME or TIMESTAMP expression
```

Tabelle 8.49. DATEDIFF-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---|
| unit | Date/time unit |
| moment1 | Ein Ausdruck eines DATE-, TIME- oder TIMESTAMP-Typs |
| moment2 | Ein Ausdruck eines DATE-, TIME- oder TIMESTAMP-Typs |

Rückgabotyp: BIGINT

Beschreibung: Gibt die Anzahl der Jahre, Monate, Wochen, Tage, Stunden, Minuten, Sekunden oder Millisekunden zurück, die zwischen zwei Datums- / Uhrzeitwerten vergangen sind. (Die WEEK-Einheit ist neu in 2.5.)

- DATE- und TIMESTAMP-Argumente können kombiniert werden. Andere Kombinationen sind nicht erlaubt.
- Mit TIMESTAMP und DATE-Argumenten können alle Einheiten verwendet werden. (Vor Firebird 2.5 waren Einheiten kleiner als DAY nicht erlaubt für DATE s.)
- Mit TIME-Argumenten können nur HOUR, MINUTE, SECOND und MILLISECOND genutzt werden.

Berechnung:

- DATEDIFF betrachtet keine kleineren Einheiten als die, die im ersten Argument angegeben wurden. Als Ergebnis,
 - „datediff (year, date '1-Jan-2009', date '31-Dec-2009')“ ergibt 0, jedoch
 - „datediff (year, date '31-Dec-2009', date '1-Jan-2010')“ ergibt 1
- Es sieht jedoch alle *größeren* Einheiten. Somit gilt:
 - „datediff (day, date '26-Jun-1908', date '11-Sep-1973')“ ergibt 23818
- Ein negatives Ergebnis gibt an, dass *moment2* vor *moment1* liegt.

Beispiele:

```
datediff (hour from current_timestamp to timestamp '12-Jun-2059 06:00')
datediff (minute from time '0:00' to current_time)
datediff (month, current_date, date '1-1-1900')
datediff (day from current_date to cast(? as date))
```

Siehe auch: [DATEADD, Operations Using Date and Time Values](#)

EXTRACT()

Verfügbar in: DSQL, ESQL, PSQL

Syntax:

```
EXTRACT (<part> FROM <datetime>)

<part>      ::= YEAR | MONTH | WEEK
              | DAY | WEEKDAY | YEARDAY
              | HOUR | MINUTE | SECOND | MILLISECOND
<datetime> ::= ein DATE-, TIME- oder TIMESTAMP-Ausdruck
```

Tabelle 8.50. EXTRACT-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---|
| part | Datums- / Zeit-Einheit |
| datetime | Ein Ausdruck des DATE-, TIME- oder TIMESTAMP-Typs |

Rückgabetyt: SMALLINT oder NUMERIC

Beschreibung: Extrahiert und gibt ein Element aus einem Ausdruck DATE, TIME oder TIMESTAMP zurück. Diese Funktion wurde bereits in InterBase 6 hinzugefügt, aber zu diesem Zeitpunkt nicht in der *Sprachreferenz* dokumentiert.

Zurückgegebene Datentypen und Bereiche

Die zurückgegebenen Datentypen und möglichen Bereiche sind in der folgenden Tabelle aufgeführt. Wenn Sie versuchen, einen Teil zu extrahieren, der nicht im Argument Datum / Uhrzeit enthalten ist (z. B. SECOND aus einem DATE- oder YEAR aus einem TIME-Feld), tritt ein Fehler auf.

Tabelle 8.51. Typen und Bereiche der EXTRACT-Ergebnisse

| Part | Type | Bereich | Beschreibung |
|-------------|--------------|----------------|------------------------------------|
| YEAR | SMALLINT | 1—9999 | |
| MONTH | SMALLINT | 1—12 | |
| WEEK | SMALLINT | 1—53 | |
| DAY | SMALLINT | 1—31 | |
| WEEKDAY | SMALLINT | 0—6 | 0 = Sonntag |
| YEARDAY | SMALLINT | 0—365 | 0 = 1. Januar |
| HOUR | SMALLINT | 0—23 | |
| MINUTE | SMALLINT | 0—59 | |
| SECOND | NUMERIC(9,4) | 0.0000—59.9999 | enthält Millisekunde als Bruchteil |
| MILLISECOND | NUMERIC(9,1) | 0.0—999.9 | defekt in 2.1, 2.1.1 |

MILLISECOND

Beschreibung: Firebird 2.1 und höher unterstützen die Extraktion der Millisekunde aus TIME oder TIMESTAMP. Der zurückgegebene Datentyp ist NUMERIC(9,1).

Anmerkung

Wenn Sie Millisekunden aus `CURRENT_TIME` extrahieren, beachten Sie, dass diese Variable standardmäßig auf volle Sekunden auflöst, womit das Ergebnis immer 0 ist. Nutzen Sie `CURRENT_TIME(3)` oder `CURRENT_TIMESTAMP` um Millisekundengenauigkeit zu erhalten.

WEEK

Beschreibung: Firebird 2.1 und höher unterstützen die Extraktion der ISO-8601-Wochennummer aus DATUM oder TIMESTAMP. ISO-8601 Wochen beginnen an einem Montag und haben immer die vollen sieben Tage. Woche 1 ist die erste Woche, die eine Mehrheit (mindestens 4) ihrer Tage im neuen Jahr hat. Die ersten 1—3 Tage des Jahres können zur letzten Woche (52 oder 53) des vorhergehenden Jahres gehören. Ebenso kann ein Jahr, das letzte 1—3 Tage, zur Woche 1 des folgenden Jahres gehören.

Achtung

Seien Sie vorsichtig, wenn Sie die Ergebnisse WEEK und YEAR kombinieren. Zum Beispiel liegt der 30. Dezember 2008 in Woche 1 von 2009, also „extract (Woche vom Datum '30 Dec 2008 ')" gibt 1 zurück. Extrahiert jedoch YEAR gibt immer das Kalenderjahr an, welches 2008 ist. In diesem Fall stehen WEEK und YEAR im Widerspruch zueinander. Das Gleiche passiert, wenn die ersten Januartage zur letzten Woche des Vorjahres gehören.

Beachten Sie außerdem, dass WEEKDAY *nicht* ISO-8601-kompatibel ist: Es gibt 0 für Sonntag zurück, während ISO-8601 7 angibt.

Siehe auch: [Datentypen für Datum und Zeit](#)

Funktionen zur Typumwandlung

CAST()

Verfügbar in: DSQL, ESQL, PSQL

Geändert in: 2.5

Syntax:

```
CAST (expression AS <target_type>)

<target_type> ::= sql_datatype
                | [TYPE OF] domain
                | TYPE OF COLUMN relname.colname
```

Tabelle 8.52. CAST-Funktionsparameter

| Parameter | Beschreibung |
|--------------|---|
| value | SQL-Ausdruck |
| datatype | SQL-Datentyp |
| domain | |
| colname | Spaltenname eine Tabelle oder View |
| precision | Genauigkeit. Von 1 bis 18 |
| scale | Skalierung (Nachkommastellen). Von 0 bis 18—muss kleiner oder gleich der Präzision sein |
| size | Die maximale Größe einer Zeichenfolge in Zeichen |
| charset | Zeichensatz |
| subtype_num | BLOB subtype number |
| subtype_name | Mnemotechnischer Name des BLOB-Untertyps |
| seglen | Segmentgröße—diese kann nicht größer als 65.535 sein |

Rückgabebetyp: Benutzerdefiniert.

Beschreibung: CAST konvertiert einen Ausdruck in den gewünschten Datentyp oder die gewünschte Domain. Wenn die Konvertierung nicht möglich ist, wird ein Fehler ausgelöst.

Syntax der „Kurzschreibweise“

Alternative Syntax, die nur unterstützt wird, wenn ein Zeichenfolgenliteral an DATE, TIME oder TIMESTAMP übergeben wird:

```
datatype 'date/timestring'
```

Diese Syntax war bereits in InterBase verfügbar, wurde aber nie richtig dokumentiert.

Anmerkung

Die Kurzsyntax wird sofort zur Analysezeit ausgewertet, wodurch der Wert gleich bleibt, bis die Anweisung unvorbereitet ist. Für Datetime-Literale wie '12-Oktober-2012' macht dies keinen Unterschied. Für die Pseudovariablen 'NOW', 'YESTERDAY', 'TODAY' und 'TOMORROW' mag dies nicht das gewünschte Verhalten sein. Wenn Sie bei jedem Aufruf den Wert benötigen, der ausgewertet werden soll, verwenden Sie die vollständige CAST () -Syntax.

Beispiele:

Ein Voll-Syntax-Cast:

```
select cast ('12' || '-June-' || '1959' as date) from rdb$database
```

Ein String-zu-Datum-Cast in Kurzschreibweise:

```
update People set AgeCat = 'Old'
where BirthDate < date '1-Jan-1943'
```

Beachten Sie, dass Sie auch die Kurzschrift aus dem obigen Beispiel löschen können, da die Engine aus dem Kontext (Vergleich mit einem Feld DATE) versteht, wie die Zeichenfolge zu interpretieren ist:

```
update People set AgeCat = 'Old'
where BirthDate < '1-Jan-1943'
```

Aber das ist nicht immer möglich. Die folgende Darstellungsart kann nicht aufgelöst werden, da die Engine sonst eine Ganzzahl findet, die von einer Zeichenfolge abgezogen wird:

```
select date 'today' - 7 from rdb$database
```

Die folgende Tabelle zeigt die möglichen Typkonvertierungen mit CAST.

Tabelle 8.53. Mögliche Typkonvertierungen mit CAST

| Von | Zu |
|------------------|-------------------------------|
| Numerische Typen | Numerische Typen [VAR]CHAR |

| Von | Zu |
|-------------------|--|
| | BLOB |
| [VAR]CHAR BLOB | [VAR]CHAR BLOB Numerische Typen DATE TIME TIMESTAMP |
| DATE TIME | [VAR]CHAR BLOB TIMESTAMP |
| TIMESTAMP | [VAR]CHAR BLOB DATE TIME |

Beachten Sie, dass Informationen unter Umständen verloren gehen, z.B. wenn Sie eine Typkonvertierung von TIMESTAMP zu DATE durchführen. Die Tatsache, dass die Typen CAST-kompatibel sind, ist keine Garantie, dass die Umwandlung erfolgreich sein wird. „CAST(123456789 as SMALLINT)“ wird in jedem Falle in einem Fehler enden, genauso wie „CAST('Judgement Day' as DATE)“.

Umwandlung von Eingabefeldern: Seit Firebird 2.0 können Sie Anweisungsparameter in einen Datentyp umwandeln:

```
cast (? as integer)
```

Dies gibt Ihnen die Kontrolle über den Typ des von der Engine eingerichteten Eingabefeldes. Bitte beachten Sie, dass Sie bei Anweisungsparametern immer einen Vollsyntax Cast — benötigen; Kurzformen werden nicht unterstützt.

Umwandlung zu einer Domain oder dessen Typ: Firebird 2.1 und später unterstützen die Umwandlung zu einer Domain oder dessen Basistypen. Bei der Umwandlung zur Domain müssen Constraints (NOT NULL und / oder CHECK), die auf dieser basieren weiterhin die Bedingungen erfüllen, da die Umwandlung sonst fehlschlägt. CHECK ist erfüllt, wenn es zu TRUE *oder* NULL ausgewertet wird! Folgende Statements seien gegeben:

```
create domain quint as int check (value >= 5000)
select cast (2000 as quint) from rdb$database      -- (1)
select cast (8000 as quint) from rdb$database      -- (2)
select cast (null as quint) from rdb$database      -- (3)
```

Nur die erste Umwandlung (1) schlägt fehl.

Wenn der Modifikator TYPE OF verwendet wird, wird der Ausdruck in den Basistyp der Domäne umgewandelt, wobei alle Einschränkungen ignoriert werden. Mit der wie oben definierten Domäne

`<typ> quint </typ>`

sind die folgenden beiden Umwandlungen äquivalent und beide werden erfolgreich sein:

```
select cast (2000 as type of quint) from rdb$database
select cast (2000 as int) from rdb$database
```

Wenn TYPE OF mit einem (VAR)CHAR verwendet wird, werden der Zeichensatz und die Collation beibehalten:

```
create domain iso20 varchar(20) character set iso8859_1;
create domain dunl20 varchar(20) character set iso8859_1 collate du_nl;
create table zinnen (zin varchar(20));
commit;
insert into zinnen values ('Deze');
insert into zinnen values ('Die');
insert into zinnen values ('die');
insert into zinnen values ('deze');

select cast(zin as type of iso20) from zinnen order by 1;
-- ergibt Deze -> Die -> deze -> die

select cast(zin as type of dunl20) from zinnen order by 1;
-- ergibt deze -> Deze -> die -> Die
```

Warnung

Wird die Definition einer Domain geändert, werden existierende CASTs auf diese Domain oder dessen Typ ungültig. Treten diese CASTs in PSQL-Modulen auf, kann ihre Ungültigkeit entdeckt werden. Siehe auch [Das RDB\\$VALID_BLR-Feld](#) in Anhang A.

Umwandlung zu einem Spaltentyp: In Firebird 2.5 und höher ist es möglich, Ausdrücke in den Typ einer vorhandenen Tabellen- oder Ansichtsspalte umzuwandeln. Nur der Typ selbst wird verwendet; Bei String-Typen umfasst dies den Zeichensatz, nicht aber die Sortierung. Constraints und Standardwerte der Quellspalte werden nicht angewendet.

```
create table ttt (
s varchar(40) character set utf8 collate unicode_ci_ai
);
commit;

select cast ('Jag har många vänner' as type of column ttt.s) from rdb$database;
```

Warnungen

- Bei Texttypen werden Zeichensatz und Sortierung durch den Cast beibehalten—genau wie beim Casting einer Domain. Aufgrund eines Fehlers wird die Collation jedoch nicht immer berücksichtigt, wenn Vergleiche (z. B. Gleichheitsprüfungen) durchgeführt werden. In Fällen, in denen die Sortierung von Bedeutung ist, sollten Sie Ihren Code vor der Bereitstellung gründlich testen. Dieser Fehler wurde für Firebird 3 behoben.
- Wenn die Definition einer Spalte geändert wird, werden existierende CASTs zu diesem Spaltentyp ungültig. Treten diese CASTs PSQL-Modulen auf, können diese Ungültigkeiten erkannt werden. Siehe auch [Das RDB\\$VALID_BLR-Feld](#) in Anhang A.

Umwandlung von BLOB s: Erfolgreiche Umwandlungen von und zu BLOBs werden seit Firebird 2.1 unterstützt.

Funktionen für Bitweise Operationen

BIN_AND()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
BIN_AND (number, number [, number ...])
```

Tabelle 8.54. BIN_AND-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| number | Beliebiger Ganzzahlwert (Literal, smallint/integer/bigint, numeric/decimal mit Skalierung von 0) |

Rückgabotyp: SMALLINT, INTEGER or BIGINT

Anmerkung

Das SMALLINT-Ergebnis wird nur zurückgegeben, wenn alle Argumente explizit SMALLINTs oder NUMERIC(n, 0) mit $n \leq 4$; ansonsten geben kleine Ganzzahlen ein INTEGER-Ergebnis zurück.

Beschreibung: Gibt das Ergebnis der bitweisen UND-Operation für das Argument (die Argumente) zurück.

Siehe auch: [BIN_OR](#), [BIN_XOR](#)

BIN_NOT()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: NEIN

Syntax:

```
BIN_NOT (number)
```

Tabelle 8.55. BIN_NOT Funktionsparameter

| Parameter | Beschreibung |
|-----------|---|
| number | Beliebige Ganzzahl (Literal, smallint/integer/bigint, numeric/decimal mit der Skalierung von 0) |

Rückgabotyp: SMALLINT, INTEGER or BIGINT

Anmerkung

Das SMALLINT-Ergebnis wird nur zurückgegeben, wenn alle Argumente explizit SMALLINTs oder NUMERIC(n, 0) mit $n \leq 4$; ansonsten geben kleine Ganzzahlen ein INTEGER-Ergebnis zurück.

Beschreibung: Returns the result of the bitwise NOT operation on the argument, i.e., *ones complement*.

Siehe auch: [BIN_OR](#), [BIN_XOR](#) and others in this set.

BIN_OR()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>siehe Details

Syntax:

```
BIN_OR (number, number [, number ...])
```

Tabelle 8.56. BIN_OR-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---|
| number | Beliebige Ganzzahl (Literal, smallint/integer/bigint, numeric/decimal mit der Skalierung von 0) |

Rückgabebetyp: SMALLINT, INTEGER or BIGINT

Anmerkung

Das SMALLINT-Ergebnis wird nur zurückgegeben, wenn alle Argumente explizit SMALLINTs oder NUMERIC(n, 0) mit $n \leq 4$; ansonsten geben kleine Ganzzahlen ein INTEGER-Ergebnis zurück.

Beschreibung: Gibt das Ergebnis der bitweisen ODER-Operation für das Argument (die Argumente) zurück.

Siehe auch: [BIN_AND](#), [BIN_XOR](#)

BIN_SHL()

Verfügbar in: DSQL, PSQL

Syntax:

```
BIN_SHL (number, shift)
```

Tabelle 8.57. BIN_SHL-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| number | Die Zahl eines Integer-Typs |
| shift | Die Anzahl der Bits, um die der Zahlenwert verschoben wird |

Rückgabebetyp: BIGINT

Beschreibung: Returns the first argument bitwise left-shifted by the second argument, i.e. $a \ll b$ or $a \cdot 2^b$.

Siehe auch: [BIN_SHR](#)

BIN_SHR()

Verfügbar in: DSQL, PSQL

Syntax:

```
BIN_SHR (number, shift)
```

Tabelle 8.58. BIN_SHR-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| number | Die Zahl eines Integer-Typs |
| shift | Die Anzahl der Bits, um die der Zahlenwert verschoben wird |

Beschreibung: Gibt das erste Argument bitweise nach rechts verschoben durch das zweite Argument zurück, d.h. $a \gg b$ oder $a/2^b$.

- Die durchgeführte Operation ist eine arithmetische Rechtsverschiebung (SAR), was bedeutet, dass das Vorzeichen des ersten Operanden immer erhalten bleibt.

Rückgabotyp: BIGINT

Siehe auch: [BIN_SHL](#)

BIN_XOR()

Verfügbar in: DSQL, PSQL

Möglicher Namenskonflikt: JA—>[siehe Details](#)

Syntax:

```
BIN_XOR (number, number [, number ...])
```

Tabelle 8.59. BIN_XOR-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---|
| number | Beliebige Ganzzahl (Literal, smallint/integer/bigint, numeric/decimal mit der Skalierung von 0) |

Beschreibung: Gibt das Ergebnis der bitweisen XOR-Operation für das Argument (die Argumente) zurück.

Rückgabotyp: SMALLINT, INTEGER oder BIGINT

Anmerkung

Das SMALLINT-Ergebnis wird nur zurückgegeben, wenn alle Argumente explizit SMALLINTs oder NUMERIC(n, 0) mit $n \leq 4$; ansonsten geben kleine Ganzzahlen ein INTEGER-Ergebnis zurück.

Siehe auch: [BIN_AND](#), [BIN_OR](#)

Funktionen zum Arbeiten mit UUID

CHAR_TO_UUID()

Verfügbar in: DSQL, PSQL

Aufgenommen in: 2.5

Syntax:

```
CHAR_TO_UUID (ascii_uuid)
```

Tabelle 8.60. CHAR_TO_UUID Funktionsparameter

| Parameter | Beschreibung |
|------------|--|
| ascii_uuid | Eine 36-stellige Darstellung der UUID. '-' (Bindestrich) in den Positionen 9, 14, 19 und 24; gültige hexadezimale Ziffern in anderen Positionen, z. "A0bF4E45-3029-2a44-D493-4998c9b439A3" |

Rückgabetyt: CHAR(16) CHARACTER SET OCTETS

Beschreibung: Konvertiert eine für Menschen lesbare UUID-Zeichenfolge mit 36 Zeichen in die entsprechende 16-Byte-UUID.

Beispiele:

```
select char_to_uuid('A0bF4E45-3029-2a44-D493-4998c9b439A3') from rdb$database
-- ergibt A0BF4E4530292A44D4934998C9B439A3 (16-byte string)

select char_to_uuid('A0bF4E45-3029-2A44-X493-4998c9b439A3') from rdb$database
-- error: - Das für den Benutzer lesbare UUID-Argument für CHAR_TO_UUID muss eine
--          Hexadezimalstelle an Position 20 anstelle von "X (ASCII 88)
```

Siehe auch: [UUID_TO_CHAR\(\)](#), [GEN_UUID\(\)](#)

GEN_UUID()

Verfügbar in: DSQL, PSQL

Syntax:

```
GEN_UUID()
```

Rückgabetyt: CHAR(16) CHARACTER SET OCTETS

Beschreibung: Gibt eine universell eindeutige ID als 16-Byte-Zeichenfolge zurück.

Beispiel:

```
select gen_uuid() from rdb$database
-- ergibt z.B. 017347BFE212B2479C00FA4323B36320 (16-byte string)
```

Siehe auch: [UUID_TO_CHAR\(\)](#), [CHAR_TO_UUID\(\)](#)

UUID_TO_CHAR()

Verfügbar in: DSQL, PSQL

Aufgenommen in: 2.5

Syntax:

```
UUID_TO_CHAR (uuid)

uuid ::= eine Zeichenfolge bestehend aus 16 Einzelbyte-Zeichen
```

Tabelle 8.61. UUID_TO_CHAR-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--------------|
| uuid | 16-byte UUID |

Rückgabotyp: CHAR(36)

Beschreibung: Konvertiert eine 16-Byte-UUID in ihre aus 36 Zeichen bestehende, für Menschen lesbare ASCII-Darstellung.

Beispiele:

```
select uuid_to_char(x'876C45F4569B320DBC4735AC3509E5F') from rdb$database
-- ergibt '876C45F4-569B-320D-BCB4-735AC3509E5F'

select uuid_to_char(gen_uuid()) from rdb$database
-- ergibt z.B. '680D946B-45FF-DB4E-B103-BB5711529B86'

select uuid_to_char('Firebird swings!') from rdb$database
-- ergibt '46697265-6269-7264-2073-77696E677321'
```

Siehe auch: [CHAR_TO_UUID\(\)](#), [GEN_UUID\(\)](#)

Funktionen zum Arbeiten mit Generatoren (Sequenzen)

GEN_ID()

Verfügbar in: DSQL, ESQL, PSQL

Beschreibung: Inkrementiert einen Generator oder eine Sequenz und gibt den neuen Wert zurück. Ab Firebird 2.0 wird die SQL-kompatible Syntax NEXT VALUE FOR bevorzugt, außer wenn ein anderes Inkrement als 1 benötigt wird.

Rückgabebetyp: BIGINT

Syntax:

```
GEN_ID (generator-name, <step>)
```

Tabelle 8.62. GEN_ID-Funktionsparameter

| Parameter | Beschreibung |
|----------------|---|
| generator-name | Name eines Generators (Sequenz), der existiert. Wenn es in doppelten Anführungszeichen mit einer Kennung, bei der die Groß- und Kleinschreibung beachtet wird, definiert wurde, muss es in derselben Form verwendet werden, es sei denn, der Name ist ausschließlich in Großbuchstaben angegeben. |
| step | Ein Integer-Ausdruck |

Rückgabebetyp: BIGINT

Beschreibung: Inkrementiert einen Generator oder eine Sequenz und gibt den neuen Wert zurück. Wenn die Schrittweite gleich 0 ist, behält die Funktion den Wert des Generators unverändert bei und gibt den aktuellen Wert zurück.

- Ab Firebird 2.0 wird die SQL-kompatible Syntax NEXT VALUE FOR bevorzugt, außer wenn ein anderes Inkrement als 1 benötigt wird.

Beispiel:

```
new.rec_id = gen_id(gen_recnum, 1);
```

Warnung

Wenn der Wert des Schrittparameters kleiner als Null ist, wird der Wert des Generators verringert. Aber Achtung! Sie sollten mit solchen Manipulationen in der Datenbank äußerst vorsichtig sein, da sie die Datenintegrität beeinträchtigen könnten.

Siehe auch: NEXT VALUE FOR, CREATE SEQUENCE (GENERATOR)

Bedingte Funktionen

COALESCE()

Verfügbar in: DSQL, PSQL

Syntax:

```
COALESCE (<exp1>, <exp2> [, <expN> ... ])
```


Tabelle 8.63. COALESCE-Funktionsparameter

| Parameter | Beschreibung |
|---------------------|---|
| exp1, exp2 ... expN | Eine Liste von Ausdrücken aller kompatiblen Typen |

Beschreibung: Die Funktion COALESCE übernimmt zwei oder mehr Argumente und gibt den ersten Wert zurück, den nicht NULL ist. Werden alle Argumente zu NULL aufgelöst, ist das Ergebnis ebenfalls NULL.

Rückgabebetyp: Abhängig von der Eingabe.

Beispiel: In diesem Beispiel wird der Spitzname aus der Tabelle "Personen" ausgewählt. Wenn dieser NULL ist, geht es weiter zu FirstName. Wenn dieser auch NULL ist, wird „Mr./Mrs.“ verwendet. Schließlich fügt es den Familiennamen hinzu. Alles in allem versucht es, die verfügbaren Daten zu verwenden, um einen vollständigen Namen zu formulieren, der so informell wie möglich ist. Beachten Sie, dass dieses Schema nur funktioniert, wenn fehlende Spitznamen und Vornamen wirklich NULL sind: Wenn eine davon eine leere Zeichenfolge ist, gibt COALESCE diese an den Aufrufer zurück.

```
select
  coalesce (Nickname, FirstName, 'Mr./Mrs.') || ' ' || LastName
as FullName
from Persons
```

Siehe auch: [IIF](#), [NULLIF](#), [CASE](#)

DECODE()

Verfügbar in: DSQL, PSQL

Syntax:

```
DECODE(testexpr,
  expr1, result1
  expr2, result2 ...)
[, defaultresult])
```

The equivalent CASE construct:

```
CASE testexpr
  WHEN expr1 THEN result1
  [WHEN expr2 THEN result2 ...]
  [ELSE defaultresult]
END
```

Tabelle 8.64. DECODE-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| testexpr | Ein Ausdruck eines kompatiblen Typs, der mit den Ausdrücken expr1, expr2 ... exprN verglichen wird |

| Parameter | Beschreibung |
|-------------------------------|---|
| expr1, expr2, ... exprN | Ausdrücke beliebiger kompatibler Typen, auf die der <testexpr> Ausdruck wird verglichen |
| result1, result2, ... resultN | Zurückgegebene Werte eines beliebigen Typs |
| defaultresult | Der Ausdruck, der zurückgegeben werden soll, wenn keine der Bedingungen erfüllt ist |

Rückgabotyp: Verschieden

Beschreibung: DECODE ist eine Kurzform des sogenannten „**einfachen CASE**“-Konstruktes, in welchem ein Ausdruck mit einer Anzahl weiterer Ausdrücke verglichen wird, bis eine Übereinstimmung gefunden wird. Das Ergebnis wird durch den Wert bestimmt, der nach dem übereinstimmenden Ausdruck aufgeführt wird. Wenn keine Übereinstimmung gefunden wird, wird das Standardergebnis zurückgegeben, sofern vorhanden. Andernfalls wird NULL zurückgegeben.

Achtung

Die Prüfung auf Übereinstimmung wird mit dem Operator „=“ durchgeführt, was bedeutet, dass wenn <test-expr> NULL ist, keine der Ausdrücke (<expr>s) übereinstimmt, auch wenn diese NULL sind.

Beispiel:

```
select name,
       age,
       decode( upper(sex),
              'M', 'Male',
              'F', 'Female',
              'Unknown' ),
       religion
from people
```

Siehe auch: [CASE](#), [Simple CASE](#)

IIF()

Verfügbar in: DSQL, PSQL

Syntax:

```
IIF (<condition>, ResultT, ResultF)
```

Tabelle 8.65. IIF-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---------------------------|
| condition | Ein wahr- falsch-Ausdruck |

| Parameter | Beschreibung |
|-----------|--|
| resultT | Der Wert wird zurückgegeben, wenn die Bedingung wahr ist |
| resultF | Der Wert wird zurückgegeben, wenn die Bedingung falsch ist |

Rückgabotyp: Abhängig von der Eingabe.

Beschreibung: IIF benötigt drei Argumente. Wenn der erste Wert zu true lautet, wird das zweite Argument zurückgegeben. Andernfalls wird die dritte zurückgegeben.

- IIF könnte mit dem ternären Operator „?:“ in C-ähnlichen Sprachen verglichen werden.

Beispiel:

```
select iif( sex = 'M', 'Sir', 'Madam' ) from Customers
```

Anmerkung

IIF(Cond, Result1, Result2) ist eine Kurzform für „CASE WHEN Cond THEN Result1 ELSE Result2 END“.

Siehe auch: CASE, DECODE

MAXVALUE()

Verfügbar in: DSQL, PSQL

Syntax:

```
MAXVALUE (expr1 [ , ... ,exprN ])
```

Tabelle 8.66. MAXVALUE-Funktionsparameter

| Parameter | Beschreibung |
|-----------------|---------------------------------------|
| expr1 ... exprN | Liste der Ausdrücke kompatibler Typen |

Rückgabotyp: Variiert entsprechend der Eingabe—Ergebnis wird vom selben Datentyp wie der erste Ausdruck in der Liste (<Ausdruck1>) sein.

Beschreibung: Gibt den Maximalwert aus einer Liste von Zahlen-, Zeichenfolgen- oder Datums- / Uhrzeitausdrücken zurück. Diese Funktion unterstützt Text-BLOBSs beliebiger Größe und Zeichenfolge

- Löst ein Ausdruck auf NULL aus, gibt MAXVALUE ebenfalls NULL zurück. Dieses Verhalten unterscheidet sich zur Aggregatfunktion MAX.

Beispiel:

```
SELECT MAXVALUE(PRICE_1, PRICE_2) AS PRICE
FROM PRICELIST
```

Siehe auch: [MINVALUE\(\)](#)

MINVALUE()

Verfügbar in: DSQL, PSQL

Syntax:

```
MINVALUE (expr1 [, ... , exprN ])
```

Tabelle 8.67. MINVALUE-Funktionsparameter

| Parameter | Beschreibung |
|-----------------|---------------------------------------|
| expr1 ... exprN | Liste der Ausdrücke kompatibler Typen |

Rückgabetyt: Variiert entsprechend der Eingabe—Ergebnis wird vom selben Datentyp wie der erste Ausdruck in der Liste (<Ausdruck1>) sein.

Beschreibung: Gibt den Maximalwert aus einer Liste von Zahlen-, Zeichenfolgen- oder Datums- / Uhrzeitausdrücken zurück. Diese Funktion unterstützt Text-BLOBSs beliebiger Größe und Zeichenfolge

- Löst ein Ausdruck auf NULL aus, gibt MINVALUE ebenfalls NULL zurück. Dieses Verhalten unterscheidet sich zur Aggregatfunktion MIN.

Beispiel:

```
SELECT MINVALUE(PRICE_1, PRICE_2) AS PRICE
FROM PRICELIST
```

Siehe auch: [MAXVALUE\(\)](#)

NULLIF()

Verfügbar in: DSQL, PSQL

Syntax:

```
NULLIF (<exp1>, <exp2>)
```

Tabelle 8.68. NULLIF-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---|
| exp1 | Ein Ausdruck |
| exp2 | Ein anderer Ausdruck eines Datentyps, der mit <exp1> kompatibel ist |

Beschreibung: NULLIF gibt den Wert des ersten Argumentes zurück, es sei denn es ist identisch zum zweiten. In diesem Fall wird NULL zurückgegeben.

Rückgabetyt: Abhängig von der Eingabe.

Beispiel:

```
select avg( nullif(Weight, -1) ) from FatPeople
```

Dies ergibt das Durchschnittsgewicht der in FatPeople aufgeführten Personen, mit Ausnahme eines Gewichts von -1, da AVG NULL-Daten überspringt. Augescheinlich bedeutet -1 soviel wie „Gewicht unbekannt“ in dieser Tabelle. Ein normales AVG(Weight) würde die Gewichte von -1 inkludieren und das Ergebnis verfälschen.

Siehe auch: [COALESCE](#), [DECODE](#), [IIF](#), [CASE](#)

Aggregatfunktionen

Aggregatfunktionen arbeiten mit Gruppen von Datensätzen und nicht mit einzelnen Datensätzen oder Variablen. Sie werden oft in Kombination mit einer Klausel GROUP BY verwendet.

AVG()

Verfügbar in: DSQL, ESQL, PSQL

Syntax:

```
AVG ([ALL | DISTINCT] <expr>)
```

Tabelle 8.69. AVG-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| expr | Ausdruck. Es kann eine Tabellenspalte, eine Konstante, eine Variable, einen Ausdruck, eine nicht aggregierte Funktion oder eine benutzerdefinierte Funktion enthalten, die einen numerischen Datentyp zurückgibt. Aggregatfunktionen sind als Ausdrücke nicht zulässig |

Beschreibung: AVG gibt den durchschnittlichen Argumentwert in der Gruppe zurück. NULL wird ignoriert.

- Der Parameter ALL (der Standardwert) wendet die Aggregatfunktion auf alle Werte an.
- Der Parameter DISTINCT weist die AVG-Funktion an, nur eine Instanz jedes eindeutigen Werts zu berücksichtigen, unabhängig davon, wie oft dieser Wert auftritt.
- Wenn die Menge der abgerufenen Datensätze leer ist oder nur NULL enthält, ist das Ergebnis ebenfalls NULL.

Rückgabetyt: Ein numerischer Datentyp, der dem Datentyp des Arguments entspricht.

Syntax:

```
AVG (expression)
```

- Wenn die Menge der abgerufenen Datensätze leer ist oder nur NULL enthält, ist das Ergebnis ebenfalls NULL.

Beispiel:

```
SELECT
  dept_no,
  AVG(salary)
FROM employee
GROUP BY dept_no
```

Siehe auch: [SELECT](#)

COUNT()

Verfügbar in: DSQL, ESQL, PSQL

Syntax:

```
COUNT ([ALL | DISTINCT] <expr> | *)
```

Tabelle 8.70. COUNT-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| expr | Ausdruck. Es kann eine Tabellenspalte, eine Konstante, eine Variable, einen Ausdruck, eine nicht aggregierte Funktion oder eine benutzerdefinierte Funktion enthalten, die einen numerischen Datentyp zurückgibt. Aggregatfunktionen sind als Ausdrücke nicht zulässig |

Rückgabebetyp: Integer

Beschreibung: COUNT gibt die Anzahl der nicht-null-Werte einer Menge zurück.

- ALL ist der Standard: es werden alle Werte gezählt, die nicht NULL sind.
- Wenn DISTINCT angegeben wurde, werden Duplikate aus der Zählung entfernt.
- Wurde COUNT (*) anstelle eines Ausdrucks <expr> angegeben, werden alle Zeilen gezählt. COUNT (*)—
 - akzeptiert keine Parameter
 - kann nicht mit dem Schlüsselwort DISTINCT verwendet werden
 - übernimmt kein <expr>-Argument, da sein Kontext während der Definition spaltenunspezifisch ist

- zählt jede Zeile einzeln und gibt die Anzahl der Zeilen in der angegebenen Tabelle oder Gruppe zurück, ohne doppelte Zeilen zu entfernen
- zähle Zeilen, die NULL enthalten
- Wenn die Ergebnismenge leer ist oder nur NULL in der(den) angegebenen Spalte(n) enthält, ist die zurückgegebene Anzahl Null.

Beispiel:

```
SELECT
  dept_no,
  COUNT(*) AS cnt,
  COUNT(DISTINCT name) AS cnt_name
FROM employee
GROUP BY dept_no
```

Siehe auch: [SELECT](#).

LIST()

Verfügbar in: DSQL, PSQL

Geändert in: 2.5

Syntax:

```
LIST ([ALL | DISTINCT] expression [, separator])
```

Tabelle 8.71. LIST-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---|
| expr | Ausdruck. Es kann eine Tabellenspalte, eine Konstante, eine Variable, einen Ausdruck, eine Nicht-Aggregatfunktion oder eine UDF enthalten, die den String-Datentyp oder ein BLOB zurückgibt. Felder von numerischen und Datum / Uhrzeit-Typen werden in Zeichenfolgen konvertiert. Aggregatfunktionen sind als Ausdrücke nicht zulässig |
| separator | Optionales alternatives Trennzeichen, ein Zeichenfolgenausdruck. Komma ist das Standardtrennzeichen |

Rückgabotyp: BLOB

Beschreibung: LIST gibt eine Zeichenfolge zurück, die aus den Nicht-Konstantenwerten NULL und Konstanten in der Gruppe besteht, die entweder durch ein Komma oder durch ein benutzerdefiniertes Trennzeichen getrennt sind. Wenn es keine nicht-NULL-Werte gibt (dies schließt den Fall ein, dass die Gruppe leer ist), wird NULL zurückgegeben.

- ALL (der Standard) ergibt, dass alle nicht-NULL-Werte aufgelistet werden. Mit DISTINCT werden Duplikate entfernt, es sei denn *expression* ist ein BLOB.

- In Firebird 2.5 und später, kann das optionale Trennzeichen ein beliebiger String-Ausdruck sein. Damit ist es beispielsweise möglich `ascii_char(13)` als Trenner zu definieren. (Diese Verbesserung wurde auch auf 2.1.4 zurückversetzt.)
- Die Argumente `expression` und `separator` unterstützen BLOBs beliebiger Größe und beliebigen Zeichensatzes.
- Datum / Uhrzeit und numerische Argumente werden implizit vor der Verkettung in Zeichenfolgen konvertiert.
- Das Ergebnis ist ein Text-BLOB, außer `expression` ist ein BLOB eines anderen Untertyps.
- Die Reihenfolge der Listenwerte ist undefiniert. Die Reihenfolge, in der die Strings verkettet werden, wird durch die Lesereihenfolge aus der Quellenmenge bestimmt, die in Tabellen nicht allgemein definiert ist. Wenn die Reihenfolge wichtig ist, können die Quelldaten mit einer abgeleiteten Tabelle oder ähnlichem vorsortiert werden.

Beispiele:

1. Abrufen der Liste, Reihenfolge undefiniert:

```
SELECT LIST (display_name, ' ; ') FROM GR_WORK;
```

2. Abrufen der Liste in alphabetischer Reihenfolge mit einer abgeleiteten Tabelle:

```
SELECT LIST (display_name, ' ; ')
FROM (SELECT display_name
      FROM GR_WORK
      ORDER BY display_name);
```

Siehe auch: [SELECT](#)

MAX()

Verfügbar in: DSQL, ESQL, PSQL

Syntax:

```
MAX ([ALL | DISTINCT] <expr>)
```

Tabelle 8.72. MAX-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| expr | Ausdruck. Es kann eine Tabellenspalte, eine Konstante, eine Variable, einen Ausdruck, eine Nicht-Aggregatfunktion oder eine UDF enthalten. Aggregatfunktionen sind als Ausdrücke nicht zulässig. |

Rückgabotyp: Gibt ein Ergebnis desselben Datentyps als Eingabeausdruck zurück.

Beschreibung: MAX gibt das größte nicht-NULL-Element der Ergebnismenge zurück.

- Ist die Menge leer oder enthält nur NULL, ist das Ergebnis NULL.
- Wenn das Eingabeargument eine Zeichenfolge ist, gibt die Funktion den Wert zurück, der zuletzt sortiert wird, wenn COLLATE verwendet wird.
- Diese Funktion unterstützt vollständig Text BLOBs beliebiger Größe und Zeichensatz.

Anmerkung

Der Parameter DISTINCT hat keinen Sinn, wenn er mit MAX() verwendet wird und nur für die Einhaltung des Standards implementiert ist.

Beispiel:

```
SELECT
  dept_no,
  MAX(salary)
FROM employee
GROUP BY dept_no
```

Siehe auch: [MIN](#), [SELECT](#)

MIN()

Verfügbar in: DSQL, ESQL, PSQL

Syntax:

```
MIN ([ALL | DISTINCT] <expr>)
```

Tabelle 8.73. MIN-Funktionsparameter

| Parameter | Beschreibung |
|-----------|--|
| expr | Ausdruck. Es kann eine Tabellenspalte, eine Konstante, eine Variable, einen Ausdruck, eine Nicht-Aggregatfunktion oder eine UDF enthalten. Aggregatfunktionen sind als Ausdrücke nicht zulässig. |

Rückgabotyp: Gibt ein Ergebnis desselben Datentyps als Eingabeausdruck zurück.

Beschreibung: MIN gibt das größte nicht-NULL-Element der Ergebnismenge zurück.

- Ist die Menge leer oder enthält nur NULL, ist das Ergebnis NULL.
- Wenn das Eingabeargument eine Zeichenfolge ist, gibt die Funktion den Wert zurück, der zuletzt sortiert wird, wenn COLLATE verwendet wird.

- Diese Funktion unterstützt vollständig Text BLOBs beliebiger Größe und Zeichensatz.

Anmerkung

Der Parameter DISTINCT hat keinen Sinn, wenn er mit MIN() verwendet wird und nur für die Einhaltung des Standards implementiert ist.

Beispiel:

```
SELECT
  dept_no,
  MIN(salary)
FROM employee
GROUP BY dept_no
```

Siehe auch: [MAX](#), [SELECT](#)

SUM()

Verfügbar in: DSQL, ESQL, PSQL

Syntax:

```
SUM ([ALL | DISTINCT] <expr>)
```

Tabelle 8.74. SUM-Funktionsparameter

| Parameter | Beschreibung |
|-----------|---|
| expr | Numerischer Ausdruck Es kann eine Tabellenspalte, eine Konstante, eine Variable, einen Ausdruck, eine Nicht-Aggregatfunktion oder eine UDF enthalten. Aggregatfunktionen sind als Ausdrücke nicht zulässig. |

Rückgabetyt: Gibt ein Ergebnis des gleichen numerischen Datentyps wie der Eingabeausdruck zurück.

Beschreibung: SUM berechnet und gibt die Summe der nicht-Null-Werte in der Gruppe zurück.

- Wenn die Gruppe leer ist oder nur NULL enthält, ist das Ergebnis ebenfalls NULL.
- ALL ist die Standardoption — alle Werte in der Gruppe, die nicht NULL sind, werden verarbeitet. Wenn DISTINCT angegeben ist, werden Duplikate aus der Datenmenge entfernt, und die Auswertung SUM erfolgt anschließend.

Beispiel: `SELECT dept_no, SUM (salary), FROM employee GROUP BY dept_no`

Siehe auch: [SELECT](#)

Transaktionskontrolle

Alles in Firebird passiert in Transaktionen. Arbeitseinheiten sind zwischen einem Start- und einem Endpunkt isoliert. Änderungen an Daten bleiben bis zu dem Moment reversibel, zu dem die Clientanwendung den Server anweist, sie zu übertragen.

Transaktions-Statements

Firebird verfügt über ein kleines Lexikon von SQL-Anweisungen, die von Client-Anwendungen zum Starten, Verwalten, Festschreiben und Zurücksetzen (Rollback) der Transaktionen verwendet werden, die die Grenzen aller Datenbankaufgaben bilden:

SET TRANSACTION: zum Konfigurieren und Starten einer Transaktion

COMMIT: das Ende einer Arbeitseinheit signalisieren und Änderungen dauerhaft in die Datenbank schreiben

ROLLBACK: die in der Transaktion durchgeführten Änderungen rückgängig machen

SAVEPOINT: um eine Position im Protokoll der geleisteten Arbeit zu markieren, falls ein partieller Rollback benötigt wird

RELEASE SAVEPOINT: einen Sicherungspunkt löschen

SET TRANSACTION

Benutzt für: Konfiguration und Start einer Transaktion

Verfügbar in: DSQL, ESQL

Syntax:

```
SET TRANSACTION
  [NAME tr_name]
  [READ WRITE | READ ONLY]
  [[ISOLATION LEVEL] {
    SNAPSHOT [TABLE STABILITY]
    | READ COMMITTED [[NO] RECORD_VERSION] }]
  [WAIT | NO WAIT]
  [LOCK TIMEOUT seconds]
  [NO AUTO UNDO]
```

```
[IGNORE LIMBO]
[RESERVING <tables> | USING <dbhandles>]

<tables> ::= <table_spec> [, <table_spec> ...]

<table_spec> ::= tablename [, tablename ...]
  [FOR [SHARED | PROTECTED] {READ | WRITE}]

<dbhandles> ::= dbhandle [, dbhandle ...]
```

Tabelle 9.1. SET TRANSACTION-Statement-Parameter

| Parameter | Beschreibung |
|------------|--|
| tr_name | Name der Transaktion. Nur in ESQL verfügbar |
| seconds | Die Zeit in Sekunden, die die Anweisung im Falle eines Konflikts warten muss |
| tables | Die Liste der zu reservierenden Tabellen |
| dbhandles | Die Liste der Datenbanken, auf die die Datenbank zugreifen kann. Nur in ESQL verfügbar |
| table_spec | Reservierungsspezifikationen für Tabellen |
| tablename | Der Name der Tabelle, die reserviert werden soll |
| dbhandle | Der Handle der Datenbank, auf die die Datenbank zugreifen kann. Nur in ESQL verfügbar |

Die Anweisung SET TRANSACTION konfiguriert die Transaktion und startet sie. In der Regel starten nur Client-Anwendungen Transaktionen. Die Ausnahmen sind die Fälle, in denen der Server eine autonome Transaktion oder Transaktionen für bestimmte Hintergrundsystem-Threads / -Prozesse startet, z. B. das Sweeping.

Eine Clientanwendung kann beliebig viele gleichzeitig ausgeführte Transaktionen starten. Es gibt eine Beschränkung für die Gesamtzahl der ausgeführten Transaktionen in allen Clientanwendungen, die mit einer bestimmten Datenbank ab dem Zeitpunkt arbeiten, zu dem die Datenbank aus ihrer Sicherungskopie wiederhergestellt wurde oder ab dem Moment, als die Datenbank ursprünglich erstellt wurde. Die Grenze ist $2^{31}-1$ oder 2.147.483.647.

Alle Klauseln in der Anweisung SET TRANSACTION sind optional. Wenn in der Anweisung, die eine Transaktion startet, keine Klauseln angegeben sind, wird die Transaktion mit Standardwerten für den Zugriffsmodus, den Sperrauflösungsmodus und die Isolationsstufe gestartet. Dies sind:

```
SET TRANSACTION
  READ WRITE
  WAIT
  ISOLATION LEVEL SNAPSHOT;
```

Der Server weist Transaktionen sequenziell Ganzzahlen zu. Wenn ein Client eine Transaktion startet, entweder explizit definiert oder standardmäßig, sendet der Server die Transaktions-ID an den Client. Diese Nummer kann in SQL über die Kontextvariable CURRENT_TRANSACTION abgerufen werden.

Transaktionsparameter

Die wichtigsten Parameter einer Transaktion sind:

- Datenzugriffsmodus (READ WRITE, READ ONLY)
- Modus zur Sperrauflösung (WAIT, NO WAIT) mit optionaler Spezifikation des LOCK TIMEOUT
- Isolationslevel (READ COMMITTED, SNAPSHOT, TABLE STABILITY)
- ein Mechanismus zum Reservieren oder Freigeben von Tabellen (die RESERVING-Klausel)

Transaktionsname

Das optionale NAME-Attribut definiert den Namen einer Transaktion. Die Verwendung dieses Attributs ist nur in Embedded SQL verfügbar. In ESQL-Anwendungen ermöglichen benannte Transaktionen die gleichzeitige Aktivierung mehrerer Transaktionen in einer Anwendung. Wenn benannte Transaktionen verwendet werden, muss eine hostspezifische Variable mit demselben Namen für jede benannte Transaktion deklariert und initialisiert werden. Dies ist eine Einschränkung, die die dynamische Angabe von Transaktionsnamen verhindert und daher die Transaktionsbenennung in DSQL ausschließt.

Zugriffsmodus

Die zwei verfügbaren Zugriffsvarianten für Transaktionen sind READ WRITE und READ ONLY.

- Wenn der Zugriffsmodus READ WRITE lautet, können Operationen im Kontext dieser Transaktion sowohl Leseoperationen als auch Datenaktualisierungsoperationen sein. Dies ist der Standardmodus.
- Wenn der Zugriffsmodus READ ONLY ist, können im Kontext dieser Transaktion nur SELECT-Operationen ausgeführt werden. Jeder Versuch, Daten im Kontext einer solchen Transaktion zu ändern, führt zu Datenbankausnahmen. Es gilt jedoch nicht für globale temporäre Tabellen (GTT), die in READ ONLY-Transaktionen geändert werden dürfen.

Modus zur Sperrauflösung

Wenn mehrere Clientprozesse mit derselben Datenbank arbeiten, können Sperren auftreten, wenn ein Prozess nicht festgeschriebene Änderungen in einer Tabellenzeile vornimmt oder eine Zeile löscht und ein anderer Prozess versucht, dieselbe Zeile zu aktualisieren oder zu löschen. Solche Sperren heißen *Aktualisierungskonflikte* (*update conflicts*).

Sperren können in anderen Situationen auftreten, wenn mehrere Transaktionsisolationsstufen verwendet werden.

Die zwei Lock-Auflösungsmodi sind WAIT und NO WAIT.

WAIT Modus

Wenn im WAIT-Modus (Standardmodus) ein Konflikt zwischen zwei parallelen Prozessen auftritt, die gleichzeitige Datenaktualisierungen in derselben Datenbank ausführen, wartet eine WAIT-Transaktion, bis die andere Transaktion abgeschlossen ist — durch Commit (COMMIT) oder Rollback (ROLLBACK). Die Clientanwendung mit der WAIT-Transaktion wird gehalten, bis der Konflikt behoben ist.

Wenn für die WAIT-Transaktion ein LOCK TIMEOUT angegeben ist, wird das Warten nur für die in dieser Klausel angegebene Anzahl von Sekunden fortgesetzt. Wenn die Sperre am Ende des angegebenen Intervalls nicht aufgelöst wird, wird die Fehlermeldung „Lock time-out on wait transaction“ an den Client zurückgegeben.

Das Verhalten der Sperrenauflösung kann abhängig von der Transaktionsisolationsstufe etwas variieren.

NO WAIT Modus

Im NO WAIT-Modus löst eine Transaktion sofort eine Datenbankausnahme aus, wenn ein Konflikt auftritt.

Isolationslevel

Die Arbeit einer Datenbankaufgabe von anderen getrennt zu halten, ist die Frage nach der Isolation. Änderungen, die von einer Anweisung vorgenommen werden, werden für alle übrigen Anweisungen sichtbar, die innerhalb der gleichen Transaktion ausgeführt werden, unabhängig von ihrer Isolationsstufe. Änderungen, die in anderen Transaktionen ausgeführt werden, bleiben für die aktuelle Transaktion unsichtbar, solange sie nicht festgeschrieben sind. Die Isolationsstufe und manchmal auch andere Attribute bestimmen, wie Transaktionen miteinander interagieren, wenn eine andere Transaktion ihre Arbeit verrichten will.

Das ISOLATION LEVEL-Attribut definiert die Isolationsstufe für die zu startende Transaktion. Es ist der wichtigste Transaktionsparameter, um sein Verhalten gegenüber anderen gleichzeitig ausgeführten Transaktionen zu bestimmen.

Die drei in Firebird unterstützten Isolationsstufen sind:

- SNAPSHOT
- SNAPSHOT TABLE STABILITY
- READ COMMITTED mit zwei Spezifikationen (NO RECORD_VERSION und RECORD_VERSION)

SNAPSHOT-Isolationslevel

SNAPSHOT-Isolationsstufe — die Standardebene — ermöglicht es der Transaktion, nur die Änderungen zu sehen, die bereits vor dem Start festgeschrieben wurden. Alle durch gleichzeitige Transaktionen vorgenommenen festgeschriebenen Änderungen werden in einer SNAPSHOT-Transaktion nicht angezeigt, solange sie aktiv ist. Die Änderungen werden für eine neue Transaktion sichtbar, sobald die aktuelle Transaktion festgeschrieben oder vollständig zurückgesetzt wurde, jedoch nicht, wenn sie nur auf einen Sicherungspunkt zurückgesetzt wird.

Autonome Transaktionen

Änderungen, die durch autonome (autonomous) Transaktionen vorgenommen werden, werden nicht im Kontext der SNAPSHOT-Transaktion gesehen, die sie gestartet hat.

SNAPSHOT TABLE STABILITY-Isolationslevel

Die Isolationsstufe SNAPSHOT TABLE STABILITY ist am restriktivsten. Wie in SNAPSHOT sieht eine Transaktion in der SNAPSHOT TABLE STABILITY-Isolation nur die Änderungen, die vor dem Start der aktuellen Transaktion festgeschrieben wurden. Nachdem eine SNAPSHOT TABLE STABILITY gestartet wurde, können keine anderen Transaktionen Änderungen an einer Tabelle in der Datenbank vornehmen, für die Änderungen anstehen. Andere Transaktionen können andere Daten lesen, aber jeder Versuch, durch einen parallelen Prozess einzufügen, zu aktualisieren oder zu löschen, führt zu Konfliktausnahmen.

Die RESERVING-Klausel kann verwendet werden, um anderen Transaktionen zu ermöglichen, Daten in einigen Tabellen zu ändern.

Wenn eine andere Transaktion eine nicht festgeschriebene Änderung von Daten in einer Datenbanktabelle aussteht, bevor eine Transaktion mit der Isolationsstufe SNAPSHOT TABLE STABILITY gestartet wird, führt der Versuch, eine Transaktion SNAPSHOT TABLE STABILITY zu starten, zu einer Ausnahme.

READ COMMITTED Isolationslevel

Die READ COMMITTED-Isolationsstufe ermöglicht alle Datenänderungen, die andere Transaktionen festgeschrieben haben, seit sie durch die nicht festgeschriebene aktuelle Transaktion unmittelbar erkannt wurden. Nicht festgeschriebene Änderungen sind für eine READ COMMITTED-Transaktion nicht sichtbar.

Um die aktualisierte Liste von Zeilen in der Tabelle, die Sie interessieren, abzurufen, muss die SELECT-Anweisung nur — „erneut“ — angefordert werden, während sie sich noch in der nicht übergebenen READ COMMITTED-Transaktion befindet.

RECORD_VERSION

Einer von zwei modifizierenden Parametern kann für READ COMMITTED-Transaktionen spezifiziert werden, abhängig von der Art der gewünschten Konfliktlösung: RECORD_VERSION und NO RECORD_VERSION. Wie die Namen andeuten, schließen sie sich gegenseitig aus.

- NO RECORD_VERSION (der Standardwert) ist eine Art zweiphasiger Sperrmechanismus: Dadurch kann die Transaktion nicht in eine Zeile schreiben, für die ein Update aus einer anderen Transaktion ansteht.
 - Wenn NO WAIT die angegebene Sperrauflösungsstrategie ist, wird sofort ein Sperrkonfliktfehler ausgegeben.
 - Wenn WAIT angegeben ist, wartet es, bis die andere Transaktion festgeschrieben oder zurückgesetzt wird. Wenn die andere Transaktion zurückgesetzt wird oder wenn sie festgeschrieben ist und ihre Transaktions-ID älter als die ID der aktuellen Transaktion ist, ist die Änderung der aktuellen Transaktion zulässig. Ein Sperrkonfliktfehler wird zurückgegeben, wenn die andere Transaktion festgeschrieben wurde und ihre ID neuer als die der aktuellen Transaktion war.
- Wenn RECORD_VERSION angegeben ist, liest die Transaktion die letzte festgeschriebene Version der Zeile, unabhängig von anderen ausstehenden Versionen der Zeile. Die Sperrauflösungsstrategie (WAIT oder NO WAIT) hat keinen Einfluss auf das Verhalten der Transaktion beim Start.

NO AUTO UNDO

Die Option NO AUTO UNDO wirkt sich auf die Behandlung nicht verwendeter Datensatzversionen (Garbage) im Falle eines Rollbacks aus. Wenn NO AUTO UNDO markiert ist, markiert die ROLLBACK-Anweisung die Transaktion nur als Rollback, ohne die in der Transaktion erstellten unbenutzten Datensatzversionen zu löschen. Sie müssen später von der Garbage Collection aufgeräumt werden.

NO AUTO UNDO kann nützlich sein, wenn viele separate Anweisungen ausgeführt werden, die Daten in Bedingungen ändern, in denen die Transaktion üblicherweise erfolgreich abgeschlossen wird.

Die Option NO AUTO UNDO wird für Transaktionen ignoriert, bei denen keine Änderungen vorgenommen werden.

IGNORE LIMBO

Dieses Kennzeichen wird verwendet, um zu signalisieren, dass von Limbo-Transaktionen erzeugte Datensätze ignoriert werden sollen. Transaktionen werden „in der Schwebe gehalten“, wenn die zweite Phase eines zweiphasigen Commits fehlschlägt.

Historische Anmerkung

IGNORE LIMBO enthält den TPB-Parameter `isc_tpb_ignore_limbo`, der seit InterBase in der API verfügbar ist und hauptsächlich von `gfx` verwendet wird.

RESERVING

Die RESERVING-Klausel in der SET TRANSACTION-Anweisung reserviert Tabellen, die in der Tabellenliste angegeben sind. Das Reservieren einer Tabelle verhindert, dass andere Transaktionen Änderungen an ihnen vornehmen oder sogar unter Einbeziehung bestimmter Parameter Daten von ihnen lesen, während diese Transaktion ausgeführt wird.

Eine RESERVING-Klausel kann auch verwendet werden, um eine Liste von Tabellen anzugeben, die von anderen Transaktionen geändert werden können, auch wenn die Transaktion mit der Isolationsstufe SNAPSHOT TABLE STABILITY gestartet wird.

Eine RESERVING-Klausel wird verwendet, um so viele reservierte Tabellen wie erforderlich anzugeben.

Optionen für die RESERVING-Klausel

Wenn eines der Schlüsselwörter SHARED oder PROTECTED weggelassen wird, wird SHARED angenommen. Wenn die gesamte FOR-Klausel weggelassen wird, wird FOR SHARED READ angenommen. Die Namen und die Kompatibilität der vier Zugriffsoptionen zum Reservieren von Tabellen sind nicht offensichtlich.

Tabelle 9.2. Kompatibilität der Zugriffsoptionen für RESERVING

| | SHARED READ | SHARED WRITE | PROTECTED READ | PROTECTED WRITE |
|-----------------|-------------|--------------|----------------|-----------------|
| SHARED READ | Ja | Ja | Ja | Ja |
| SHARED WRITE | Ja | Ja | Nein | Nein |
| PROTECTED READ | Ja | Nein | Ja | Nein |
| PROTECTED WRITE | Ja | Nein | Nein | Nein |

Die Kombinationen dieser RESERVING-Klauselflags für den gleichzeitigen Zugriff hängen von den Isolationsstufen der gleichzeitigen Transaktionen ab:

- SNAPSHOT-Isolation

- Gleichzeitige SNAPSHOT-Transaktionen mit SHARED READ wirken sich nicht auf den Zugriff eines anderen aus
 - Eine gleichzeitige Mischung aus SNAPSHOT- und READ COMMITTED-Transaktionen mit SHARED WRITE wirkt sich nicht auf den Zugriff des anderen aus, blockiert jedoch Transaktionen mit der SNAPSHOT TABLE STABILITY-Isolation beim Lesen oder Schreiben in die angegebene Tabelle[n].
 - Gleichzeitige Transaktionen mit einer beliebigen Isolationsstufe und PROTECTED READ können nur Daten aus den reservierten Tabellen lesen. Jeder Versuch, an sie zu schreiben, führt zu einer Ausnahme
 - Mit PROTECTED WRITE können gleichzeitige Transaktionen mit der Isolation SNAPSHOT und READ COMMITTED nicht in die angegebenen Tabellen schreiben. Transaktionen mit der SNAPSHOT TABLE STABILITY-Isolation können überhaupt nicht aus den reservierten Tabellen lesen oder in diese schreiben.
- SNAPSHOT TABLE STABILITY-Isolation
 - Alle gleichzeitigen Transaktionen mit SHARED READ, unabhängig von ihrer Isolationsstufe, können aus den reservierten Tabellen lesen oder in den schreibgeschützten Tabellen schreiben (falls im READ WRITE-Modus)
 - Gleichzeitige Transaktionen mit den Isolationsstufen SNAPSHOT und READ COMMITTED und SHARED WRITE können Daten aus den Tabellen lesen und schreiben (falls im Modus READ WRITE), aber der gleichzeitige Zugriff auf diese Tabellen aus Transaktionen mit SNAPSHOT TABLE STABILITY wird vollständig blockiert, solange diese Transaktionen aktiv sind.
 - Gleichzeitige Transaktionen mit einer beliebigen Isolationsstufe und PROTECTED READ können nur aus den reservierten Tabellen lesen
 - Mit PROTECTED WRITE können gleichzeitige SNAPSHOT- und READ COMMITTED-Transaktionen die reservierten Tabellen lesen, aber nicht in diese schreiben. Der Zugriff durch Transaktionen mit der Isolationsstufe SNAPSHOT TABLE STABILITY ist vollständig blockiert.
 - READ COMMITTED-Isolation
 - Mit SHARED READ können alle gleichzeitigen Transaktionen mit einer beliebigen Isolationsstufe von den reservierten Tabellen gelesen und geschrieben werden (wenn im READ WRITE-Modus).
 - SHARED WRITE ermöglicht allen Transaktionen in der SNAPSHOT- und READ COMMITTED-Isolation das Lesen und Schreiben (falls im Modus READ WRITE) in die angegebenen Tabellen und das vollständige Sperren des Zugriffs von Transaktionen mit der Isolation SNAPSHOT TABLE STABILITY
 - Mit PROTECTED READ können gleichzeitige Transaktionen mit einer beliebigen Isolationsstufe nur aus den reservierten Tabellen gelesen werden
 - Mit PROTECTED WRITE können gleichzeitige Transaktionen in der SNAPSHOT- und READ COMMITTED-Isolation die angegebenen Tabellen lesen, aber nicht in diese schreiben. Der Zugriff von Transaktionen in der SNAPSHOT TABLE STABILITY-Isolation ist vollständig blockiert.

Tipp

In Embedded SQL kann die USING-Klausel verwendet werden, um Systemressourcen einzusparen, indem die Datenbanken beschränkt werden, auf die die Transaktion auf eine Aufzählungsliste (von Datenbanken) zugreifen kann. USING ist nicht kompatibel mit RESERVING. Eine USING-Klausel in der SET TRANSACTION-Syntax wird in DSQL nicht unterstützt.

Siehe auch: [COMMIT](#), [ROLLBACK](#)

COMMIT

Benutzt für: Transaktion festschreiben

Verfügbar in: DSQL, ESQL

Syntax:

```
COMMIT [WORK] [TRANSACTION tr_name]
[RELEASE] [RETAIN [SNAPSHOT]];
```

Tabelle 9.3. COMMIT Statement Parameter

| Parameter | Beschreibung |
|-----------|---|
| tr_name | Name der Transaktion. Nur in ESQL verfügbar |

Die COMMIT-Anweisung schreibt alle Arbeiten fest, die im Zusammenhang mit dieser Transaktion ausgeführt werden (Einfügen, Aktualisieren, Löschen, Auswählen, Ausführen von Prozeduren). Neue Datensatzversionen werden für andere Transaktionen verfügbar, und wenn die RETAIN-Klausel nicht verwendet wird, werden alle Serverressourcen freigegeben, die ihrer Arbeit zugewiesen wurden.

Wenn Konflikte oder andere Fehler in der Datenbank während des Festschreibens der Transaktion auftreten, wird die Transaktion nicht festgeschrieben, und die Gründe werden an die Benutzeranwendung zur Bearbeitung und der Möglichkeit, einen weiteren Commit zu versuchen oder die Transaktion rückgängig zu machen, zurückgegeben.

COMMIT Options

- Die optionale Klausel TRANSACTION <tr_name>, die nur in Embedded SQL verfügbar ist, gibt den Namen der Transaktion an, die festgeschrieben werden soll. Ohne die Klausel TRANSACTION wird COMMIT auf die Standardtransaktion angewendet.

Anmerkung

In ESQL-Anwendungen ermöglichen benannte Transaktionen die gleichzeitige Aktivierung mehrerer Transaktionen in einer Anwendung. Wenn benannte Transaktionen verwendet werden, muss eine hostspezifische Variable mit demselben Namen für jede benannte Transaktion deklariert und initialisiert werden. Dies ist eine Einschränkung, die die dynamische Angabe von Transaktionsnamen verhindert und daher die Transaktionsbenennung in DSQL ausschließt.

- Das optionale Schlüsselwort WORK wird nur aus Kompatibilitätsgründen mit anderen relationalen Datenbankverwaltungssystemen unterstützt, für die dies erforderlich ist.
- Das Schlüsselwort RELEASE ist nur in Embedded SQL verfügbar und ermöglicht die Trennung von allen Datenbanken, nachdem die Transaktion festgeschrieben wurde. RELEASE wird in Firebird nur zur Kompatibilität mit älteren Versionen von InterBase verwendet. Es wurde in ESQL durch die Anweisung DISCONNECT ersetzt.
- Die Klausel RETAIN [SNAPSHOT] wird für das „weiche“ Festschreiben, verschiedentlich unter den Host-Sprachen und ihren Anwendern auch als COMMIT WITH RETAIN, CommitRetaining, „warm commit“, etc. bezeichnet. Die Transaktion ist festgeschrieben, aber einige Serverressourcen bleiben erhalten, und die Trans-

aktion wird transparent mit derselben Transaktions-ID erneut gestartet. Der Status von Zeilencaches und Cursors wird beibehalten wie vor dem Soft Commit.

Für Soft-Committed-Transaktionen, deren Isolationsstufe SNAPSHOT oder SNAPSHOT TABLE STABILITY ist, wird die Ansicht des Datenbankstatus nicht aktualisiert, um Änderungen durch andere Transaktionen widerzuspiegeln, und der Benutzer der Anwendungsinstanz hat weiterhin dieselbe Ansicht wie beim ursprünglichen Start der Transaktion. Änderungen, die während der Laufzeit der zurückbehaltenen Transaktion vorgenommen wurden, sind natürlich für diese Transaktion sichtbar.

Empfehlung

Die Verwendung der Anweisung COMMIT anstelle von ROLLBACK wird zum Beenden von Transaktionen empfohlen, die nur Daten aus der Datenbank lesen, da COMMIT weniger Serverressourcen verbraucht und hilft damit bei der Optimierung der Performance nachfolgender Transaktionen.

Siehe auch: SET TRANSACTION, ROLLBACK

ROLLBACK

Benutzt für: Rollback einer Transaktion

Verfügbar in: DSQL, ESQL

Syntax:

```
ROLLBACK [WORK] [TRANSACTION tr_name]
[RETAIN [SNAPSHOT] | TO [SAVEPOINT] sp_name | RELEASE]
```

Tabelle 9.4. ROLLBACK Statement Parameters

| Parameter | Beschreibung |
|-----------|---|
| tr_name | Name der Transaktion. Nur in ESQL verfügbar |
| sp_name | Name des Sicherungspunkts. Nur in SQL verfügbar |

Die ROLLBACK-Anweisung setzt alle im Zusammenhang mit dieser Transaktion ausgeführten Arbeiten zurück (Einfügen, Aktualisieren, Löschen, Auswählen, Ausführen von Prozeduren). ROLLBACK schlägt niemals fehl und verursacht daher niemals Ausnahmen. Wenn die RETAIN-Klausel nicht verwendet wird, werden alle Serverressourcen freigegeben, die der Arbeit der Transaktion zugeordnet sind.

ROLLBACK Options

- Die optionale Klausel TRANSACTION <tr_name>, die nur in Embedded SQL verfügbar ist, gibt den Namen der Transaktion an, die festgeschrieben werden soll. Ohne die Klausel TRANSACTION wird COMMIT auf die Standardtransaktion angewendet.

Anmerkung

In ESQL-Anwendungen ermöglichen benannte Transaktionen die gleichzeitige Aktivierung mehrerer Transaktionen in einer Anwendung. Wenn benannte Transaktionen verwendet werden, muss eine hostspezifische Variable mit demselben Namen für jede benannte Transaktion deklariert und initialisiert werden. Dies ist eine Einschränkung, die die dynamische Angabe von Transaktionsnamen verhindert und daher die Transaktionsbenennung in DSQL ausschließt.

- Das optionale Schlüsselwort `WORK` wird nur aus Kompatibilitätsgründen mit anderen relationalen Datenbankverwaltungssystemen unterstützt, für die dies erforderlich ist.
- Das Schlüsselwort `RETAIN` gibt an, dass der Transaktionskontext beibehalten werden soll, obwohl die gesamte Arbeit der Transaktion rückgängig gemacht werden soll. Einige Serverressourcen bleiben erhalten und die Transaktion wird transparent mit derselben Transaktions-ID neu gestartet. Der Status von Zeilencaches und Cursors wird beibehalten, wie er vor dem „weichen“ Rollback war.

Für Transaktionen, deren Isolationsstufe `SNAPSHOT` oder `SNAPSHOT TABLE STABILITY` ist, wird die Ansicht des Datenbankstatus nicht durch das Soft-Rollback aktualisiert, um Änderungen durch andere Transaktionen widerzuspiegeln. Der Benutzer der Anwendungsinstanz hat weiterhin dieselbe Ansicht wie beim ursprünglichen Start der Transaktion. Änderungen, die während der Laufzeit der zurückbehaltenen Transaktion vorgenommen und während dieser abgeschlossen wurden, sind natürlich für diese Transaktion sichtbar.

Siehe auch: [SET TRANSACTION, COMMIT](#)

ROLLBACK TO SAVEPOINT

Die optionale Klausel `TO SAVEPOINT` in der Anweisung `ROLLBACK` gibt den Namen eines Sicherungspunkts an, auf den die Änderungen zurückgesetzt werden sollen. Der Effekt besteht darin, alle in der Transaktion vorgenommenen Änderungen rückgängig zu machen, und zwar vom erstellten Sicherungspunkt bis zu dem Zeitpunkt, an dem `ROLLBACK TO SAVEPOINT` angefordert wird.

`ROLLBACK TO SAVEPOINT` führt die folgenden Operationen aus:

- Alle Datenbankmutationen, die seit der Erstellung des Sicherungspunkts ausgeführt wurden, werden rückgängig gemacht. Benutzervariablen, die mit `RDB$SET_CONTEXT()` gesetzt wurden, bleiben unverändert.
- Alle Sicherungspunkte, die nach dem Namen erstellt wurden, werden zerstört. Savepoints, die älter als der angegebene sind, werden zusammen mit dem benannten Savepoint selbst beibehalten. Wiederholte Rollbacks zum selben Savepoint sind somit erlaubt.
- Alle impliziten und expliziten Datensatzsperrungen, die seit dem Speichern des Punkts erfasst wurden, werden freigegeben. Andere Transaktionen, die Zugriff auf nach dem Sicherungspunkt gesperrte Zeilen angefordert haben, müssen warten, bis die Transaktion festgeschrieben oder zurückgesetzt wurde. Andere Transaktionen, die die Zeilen noch nicht angefordert haben, können die entsperrten Zeilen sofort anfordern und darauf zugreifen.

Siehe auch: [SAVEPOINT](#)

SAVEPOINT

Benutzt für: Creating a savepoint

Verfügbar in: DSQL

Syntax:

```
SAVEPOINT sp_name
```

Tabelle 9.5. SAVEPOINT Statement Parameter

| Parameter | Beschreibung |
|-----------|--|
| sp_name | Name des Sicherungspunkts Nur in SQL verfügbar |

Die Anweisung SAVEPOINT erstellt einen SQL: 99-konformen Sicherungspunkt, der als Marker im „stack“ der Datenaktivitäten innerhalb einer Transaktion fungiert. Anschließend können die im „stack“ ausgeführten Tasks an diesen Savepoint rückgängig gemacht werden, wobei die frühere Arbeit und ältere Savepoints unberührt bleiben. Savepoint-Mechanismen werden manchmal als „verschachtelte Transaktionen“ bezeichnet.

Wenn bereits ein Sicherungspunkt mit demselben Namen wie der für den neuen bereitgestellte Name vorhanden ist, wird der vorhandene Sicherungspunkt gelöscht und ein neuer unter Verwendung des angegebenen Namens erstellt.

Um Änderungen auf den Sicherungspunkt zurückzuspielen, wird die Anweisung ROLLBACK TO SAVEPOINT verwendet.

Überlegungen zum Speicher

Der interne Mechanismus unterhalb der Sicherungspunkte kann große Speichermengen verbrauchen, insbesondere wenn die gleichen Zeilen mehrere Aktualisierungen in einer Transaktion erhalten. Wenn ein Sicherungspunkt nicht mehr benötigt wird, aber die Transaktion noch Arbeit verrichtet, wird die Anweisung **RELEASE SAVEPOINT** diesen löschen und somit die belegten Ressourcen freigeben.

Beispiel einer DSQL-Sitzung mit Savepoints:

```
CREATE TABLE TEST (ID INTEGER);
COMMIT;
INSERT INTO TEST VALUES (1);
COMMIT;
INSERT INTO TEST VALUES (2);
SAVEPOINT Y;
DELETE FROM TEST;
SELECT * FROM TEST; -- returns no rows
ROLLBACK TO Y;
SELECT * FROM TEST; -- returns two rows
ROLLBACK;
SELECT * FROM TEST; -- returns one row
```

Siehe auch: **ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT**

RELEASE SAVEPOINT

Benutzt für: Einen Sicherungspunkt löschen

Verfügbar in: DSQL

Syntax:

```
RELEASE SAVEPOINT sp_name [ONLY]
```

Tabelle 9.6. RELEASE SAVEPOINT-Statement-Parameter

| Parameter | Beschreibung |
|-----------|---|
| sp_name | Name des Sicherungspunkts. Nur in SQL verfügbar |

Die Anweisung `RELEASE SAVEPOINT` löscht einen benannten Savepoint und gibt damit alle Ressourcen frei. Standardmäßig werden alle Sicherungspunkte, die nach dem benannten Sicherungspunkt erstellt wurden, ebenfalls freigegeben. Der Qualifier `ONLY` weist die Engine an, nur den benannten Savepoint freizugeben.

Siehe auch: [SAVEPOINT](#)

Interne Sicherungspunkte

Standardmäßig verwendet die Engine einen automatischen Sicherungspunkt auf Transaktionsebene, um Transaktionsrollbacks durchzuführen. Wenn eine Anweisung `ROLLBACK` ausgeführt wird, werden alle in dieser Transaktion ausgeführten Änderungen über einen Sicherungspunkt auf Transaktionsebene zurückgesetzt, und die Transaktion wird dann festgeschrieben. Diese Logik reduziert die Menge an aufzuräumenden Müll (Garbage Collection), der durch zurückgerollte Transaktionen verursacht wird.

Wenn der Umfang der unter einem Sicherungspunkt auf Transaktionsebene durchgeführten Änderungen groß wird (ca. 50000 betroffene Datensätze), gibt die Engine den Sicherungspunkt auf Transaktionsebene frei und verwendet die Transaktionsinventarseite (TIP) als Mechanismus, um die Transaktion bei Bedarf zurückzusetzen.

Tipp

Wenn Sie erwarten, dass der Umfang der Änderungen in Ihrer Transaktion groß ist, können Sie die Option `NO AUTO UNDO` in Ihrer `SET TRANSACTION`-Anweisung angeben, um die Erstellung des Sicherungspunkts auf Transaktionsebene zu blockieren. Mit der API würden Sie stattdessen das TPB-Flag `isc_tpb_no_auto_undo` setzen.

Sicherungspunkte und PSQL

Transaktionssteueranweisungen sind in PSQL nicht zulässig, da dies die Atomität der Anweisung, die die Prozedur aufruft, aufheben würde. Firebird unterstützt jedoch das Auslösen und Behandeln von Ausnahmen in PSQL, sodass Aktionen, die in gespeicherten Prozeduren und Triggern ausgeführt werden, selektiv rückgängig gemacht werden können, ohne dass die gesamte Prozedur fehlschlägt.

Intern werden automatische Sicherungspunkte verwendet, um:

- alle Aktionen im Block `BEGIN ... END` rückgängig zu machen, bei dem eine Exception auftritt

- alle Aktionen rückgängig machen, die von der Prozedur oder dem Trigger ausgeführt wurden, oder für eine auswählbare Prozedur alle Aktionen, die seit dem letzten SUSPEND ausgeführt wurden, wenn die Ausführung aufgrund eines nicht erfassten Fehlers oder einer Ausnahme vorzeitig beendet wird

Jeder PSQL-Exception-Behandlungsblock ist auch durch automatische System-Savepoints begrenzt.

Anmerkung

Ein BEGIN ... END-Block erstellt keinen automatischen Sicherungspunkt. Ein Sicherungspunkt wird nur in Blöcken erstellt, die die WHEN-Anweisung zur Behandlung von Ausnahmen enthalten.

Sicherheit

Datenbanken müssen sicher sein sowie die Daten die in ihnen gespeichert werden. Firebird bietet zwei Stufen von Datensicherheitsschutz: Benutzerauthentifizierung auf Server-Ebene und SQL-Berechtigungen in Datenbanken. In diesem Kapitel erfahren Sie, wie Sie die Sicherheit auf beiden Ebenen verwalten können.

Benutzerauthentifizierung

Die Sicherheit der gesamten Datenbank hängt davon ab, einen Benutzer bei der Überprüfung seiner Autorität zu identifizieren. Diese Prozedur ist auch bekannt als *Authentifizierung*. Die Informationen über Benutzer, die für den Zugriff auf einen bestimmten Firebird-Server berechtigt sind, werden in einer speziellen Sicherheitsdatenbank gespeichert, benannt als *security2.fdb*. Jeder Datensatz in *security2.fdb* ist ein Benutzerkonto für einen Benutzer.

Ein Benutzername, bestehend aus bis zu 31 Zeichen, ist ein Groß- und Kleinschreibungsunabhängiger (case-insensitive) Bezeichner. Ein Benutzer muss ein Kennwort haben, von denen die ersten acht Zeichen signifikant sind. Während es gültig ist, ein Kennwort länger als acht Zeichen einzugeben, werden alle nachfolgenden Zeichen werden ignoriert. Bei Kennwörtern wird zwischen Groß- und Kleinschreibung unterschieden.

Wenn der Benutzer während der Verbindung der SYSDBA ist, der Datenbankeigentümer oder ein speziell privilegierter Benutzer, bekommt der Benutzer unbegrenzten Zugriff auf die Datenbank.

Besonders privilegierte Benutzer

In Firebird ist das SYSDBA-Konto ein „Superuser“, dass über jede Sicherheitbeschränkung hinausgeht. Es hat vollständigen Zugriff auf alle Objekte in allen regulären Datenbanken auf dem Server und volle Lese- / Schreibzugriffe auf die Konten in der Sicherheitsdatenbank *security2.fdb*. Kein Benutzer hat Zugriff auf die Metadaten der Sicherheitsdatenbank.

Das Standard-SYSDBA-Kennwort unter Windows und MacOS ist "masterkey" — oder "masterke", um genau zu sein, wegen der 8-stelligen Längengrenze.

Extrem wichtig!

Das Standardkennwort 'masterkey' ist über das Universum bekannt. Es sollte so schnell wie möglich geändert werden sobald die Firebird Server-Installation abgeschlossen ist.

Andere Benutzer können auf mehreren Wegen erhöhte Privilegien erwerben, von denen einige von der Betriebssystemplattform abhängig sind. Diese werden in den folgenden Abschnitten besprochen zusammengefasst in *Administratoren*.

POSIX Hosts

Bei POSIX-Systemen, einschließlich MacOSX, wird Firebird ein POSIX-Benutzerkonto so interpretieren, als wäre es ein Firebird-Benutzerkonto in seiner eigenen Sicherheitsdatenbank, vorausgesetzt der Server sieht die Client-Maschine als vertrauenswürdigen Host an und die Systembenutzerkonten auf dem Client und dem Server sind vorhanden. Um eine „vertrauenswürdige“ Beziehung mit dem Client-Host zu erstellen, müssen die entsprechenden Einträge in einer der Dateien `/etc/hosts.equiv` oder `/etc/gds_hosts.equiv` auf Firebirds Hostserver enthalten sein.

- Die Datei `hosts.equiv` enthält vertrauenswürdige Beziehungen auf Betriebssystem-Ebene, die alle Dienste umfasst (rlogin, rsh, rcp und so weiter).
- Die Datei `gds_hosts.equiv` enthält vertrauenswürdige Beziehungen nur zwischen Firebird-Hosts.

Das Format ist für beide Dateien identisch und sieht so aus:

```
hostname [username]
```

Der SYSDBA Benutzer auf POSIX

Bei POSIX-Hosts, anders als MacOSX, hat der Benutzer SYSDBA kein Standardkennwort. Wenn die vollständige Installation mit den Standard-Scripts durchgeführt wird, wird ein einmaliges Kennwort erstellt und in einer Textdatei im selben Verzeichnis, üblicherweise `/opt/firebird/`, wie `security2.fdb` gespeichert. Der Name der Kennwortdatei lautet `SYSDBA.password`.

Anmerkung

In einer Installation, die von einem verteilungsspezifischen Installer durchgeführt wird, kann der Standort der Sicherheits-Datenbank und der Kennwort-Datei von der Standard-Datei variieren.

Der root Benutzer

Der **root** Benutzer kann direkt als SYSDBA auf POSIX-Host-Systemen handeln. Firebird interpretiert **root** als ob es SYSDBA wäre und bietet Zugriff auf alle Datenbanken auf dem Server.

Windows Hosts

Auf Windows Server-fähigen Betriebssystemen können Betriebssystemkonten verwendet werden. Vertrauenswürdige Authentifizierung muss aktiviert werden, indem der Parameter *Authentication* auf *Trusted* oder *Mixed* in der Konfigurationsdatei `firebird.conf` gesetzt wird.

Auch bei vertrauenswürdiger Authentifizierung sind die Windows-Betriebssystemadministratoren nicht automatisch mit SYSDBA-Berechtigungen berechtigt, wenn sie eine Verbindung zu einer Datenbank herstellen. Um dies zu tun, muss die intern erstellte Rolle RDB\$ADMIN von SYSDBA oder dem Datenbankeigentümer geändert werden. Für Details vgl. Abschnitt *AUTO ADMIN MAPPING*.

Die eingebettete Version des Firebird-Servers unter Windows verwendet keine Server-Level-Authentifizierung. Da jedoch Objekte innerhalb einer Datenbank SQL-Berechtigungen unterliegen, ist ein gültiger Benutzername und, wenn Anwendbar, eine Rolle, in den Verbindungsparametern erforderlich.

Der Datenbankeigentümer

Der „Besitzer“ einer Datenbank ist entweder der Benutzer, der `CURRENT_USER`, welcher zum Zeitpunkt der Erstellung benutzt wurde oder der Benutzer, der als Parameter `USER` und `PASSWORD` mit der Anweisung `CREATE DATABASE` verwendet wurde.

„Besitzer“ ist kein Benutzername. Der Benutzer, der Eigentümer einer Datenbank ist, hat volle [Administratorrechte](#) in Bezug auf diese Datenbank, einschließlich des Rechtes, sie zu löschen, um es von einer Sicherung wiederherzustellen und zu aktivieren oder die Fähigkeit [AUTO ADMIN MAPPING](#) zu deaktivieren.

Anmerkung

Vor Firebird 2.1 hatte der Besitzer keine automatischen Privilegien über alle Datenbankobjekte, die von anderen Benutzern erstellt wurden.

RDB\$ADMIN-Rolle

Die intern erstellte Rolle `RDB$ADMIN` ist in jeder Datenbank vorhanden. Die Zuweisung der Rolle `RDB$ADMIN` zu einem regulären Benutzer in einer Datenbank, gewährt diesem Benutzer die Berechtigungen des `SYSDBA`, aber nur in der aktuellen Datenbank.

Die erhöhten Berechtigungen werden wirksam, wenn der Benutzer sich mit der `RDB$ADMIN`-Rolle in die Datenbank anmeldet und somit volle Kontrolle über alle Objekte in der Datenbank gibt.

Wird die `RDB$ADMIN`-Rolle in der Sicherheitsdatenbank gewährt, verleiht die Autorität die Erstellung, Bearbeitung und Löschung von Benutzerkonten.

In beiden Fällen kann der Benutzer mit den erhöhten Berechtigungen die `RDB$ADMIN`-Rolle an jeden anderen Benutzer vergeben. Mit anderen Worten wird die Angabe von `WITH ADMIN OPTION` unnötig, weil es in die Rolle eingebaut ist.

Gewähren der RDB\$ADMIN-Rolle in der Sicherheitsdatenbank

Da niemand — nicht einmal `SYSDBA` — sich mit der Sicherheitsdatenbank verbinden kann, können `GRANT`- und `REVOKE`-Anweisungen für diese Aufgabe nicht genutzt werden. Stattdessen wird die Rolle `RDB$ADMIN` gewährt und widerrufen mit den `SQL`-Anweisungen für die Benutzerverwaltung:

```
CREATE USER new_user
PASSWORD 'password'
GRANT ADMIN ROLE

ALTER USER existing_user
GRANT ADMIN ROLE

ALTER USER existing_user
REVOKE ADMIN ROLE
```

Anmerkung

GRANT ADMIN ROLE und REVOKE ADMIN ROLE sind keine Aussagen im GRANT- und REVOKE-Lexikon. Sie sind Drei-Wort-Parameter zu den Anweisungen CREATE USER und ALTER USER.

Tabelle 10.1. Parameter für die RDB\$ADMIN Rollen GRANT und REVOKE

| Parameter | Beschreibung |
|---------------|--|
| new_user | Verwenden Sie CREATE USER, Name für den neuen Benutzer. |
| existing_user | Verwenden Sie ALTER USER, Name eines vorhandenen Benutzers |
| password | Verwenden Sie CREATE USER, Kennwort für den neuen Benutzer. Das theoretische Limit sind 31 Bytes, aber nur die ersten 8 Zeichen werden berücksichtigt. |

Der Gewährende muss bereits als [Administrator](#) angemeldet sein.

Siehe auch: [CREATE USER](#) , [ALTER USER](#)

Die gleiche Aufgabe mit gsec ausführen

Eine zu verwendende Alternative ist *gsec*, mit dem Parameter `-admin` um das Attribut RDB\$ADMIN auf dem Datensatz des Benutzers zu speichern:

```
gsec -add new_user -pw password -admin yes
gsec -mo existing_user -admin yes
gsec -mo existing_user -admin no
```

Anmerkung

Abhängig vom administrativen Status des aktuellen Benutzers, können mehrere Parameter beim Aufruf von *gsec* benötigt werden, z.B. `-user` und `-pass`, oder `-trusted`.

Verwenden der RDB\$ADMIN-Rolle in der Sicherheitsdatenbank

Um Benutzerkonten über SQL zu verwalten, muss der Gewährende die RDB\$ADMIN-Rolle beim Verbinden bestimmen. Kein Benutzer kann eine Verbindung zur Sicherheitsdatenbank herstellen. Die Lösung ist also, dass sich der Benutzer mit einer Datenbank verbindet, für die dieser bereits RDB\$ADMIN-Rechte hat und übergibt die RDB\$ADMIN-Rolle mit seinen Login-Parametern. Von dort aus kann er einen beliebigen SQL-Benutzerverwaltungsbefehl ausführen.

Die SQL-Route für den Benutzer ist für jede Datenbank gesperrt, in der er nicht die RDB\$ADMIN-Rolle zugewiesen bekommen hat.

Verwenden von gsec mit RDB\$ADMIN-Rechten

Um die Benutzerverwaltung mit *gsec* durchzuführen, muss der Benutzer den zusätzlichen Schalter `-role rdb$admin` angeben.

Gewähren der RDB\$ADMIN-Rolle in einer regulären Datenbank

In einer regulären Datenbank wird die Rolle RDB\$ADMIN mit der üblichen Syntax für die Erteilung und das Widerrufen von Rollen aufgehoben und widerrufen:

```
GRANT [ROLE] RDB$ADMIN TO username  
  
REVOKE [ROLE] RDB$ADMIN FROM username
```

Um die RDB\$ADMIN-Rolle zu erteilen und zu widerrufen, muss der Gewährende als [Administrator](#) eingeloggt sein.

Siehe auch: [GRANT](#) , [REVOKE](#)

Verwenden der RDB\$ADMIN-Rolle in einer regulären Datenbank

Um seine RDB\$ADMIN-Privilegien auszuführen, nutzt der Gewährende einfach die Rolle in den Verbindungsattributen, sobald er eine Verbindung zur Datenbank herstellt.

AUTO ADMIN MAPPING

In Firebird 2.1 würden Windows-Administratoren automatisch SYSDBA Berechtigungen erhalten, wenn die vertrauenswürdige Authentifizierung für Serververbindungen konfiguriert wurde. In Firebird 2.5 geschieht dies nicht mehr automatisch. Die Einstellung der Schaltfläche AUTO ADMIN MAPPING legt nun fest, ob Administratoren automatisch SYSDBA-Rechte auf Datenbank-zu-Datenbank-Basis haben. Wenn eine Datenbank erstellt wird, ist sie standardmäßig deaktiviert.

Wenn AUTO ADMIN MAPPING in der Datenbank aktiviert ist, wird es wirksam, wenn ein Windows-Administrator

- a. eine vertrauenswürdige Authentifizierung verbindet und
- b. ohne eine Rolle spezifiziert.

Nach einer erfolgreichen „auto admin“ Verbindung wird die aktuelle Rolle auf RDB\$ADMIN gesetzt.

Auto Admin Mapping in regulären Datenbanken

So aktivieren und deaktivieren Sie die automatische Zuordnung in einer regulären Datenbank:

```
ALTER ROLE RDB$ADMIN  
SET AUTO ADMIN MAPPING -- aktivieren  
  
ALTER ROLE RDB$ADMIN  
DROP AUTO ADMIN MAPPING -- deaktivieren
```

Jede Anweisung muss von einem Benutzer mit ausreichender Berechtigung ausgestellt werden, d.h:

- der Datenbankbesitzer

- ein Administrator

In regulären Datenbanken wird der Status von AUTO ADMIN MAPPING nur zur Verbindungszeit überprüft. Wenn ein Administrator die Rolle RDB\$ADMIN während des Einloggens bereits hatte und die automatische Zuordnung aktiv ist, wird er diese Rolle für die Dauer der Sitzung behalten, auch wenn er oder jemand anderes die Zuordnung in der Zwischenzeit ausschaltet.

Ebenso wird das Umschalten auf AUTO ADMIN MAPPING nicht die aktuelle Rolle in RDB\$ADMIN für Administratoren ändern, die bereits verbunden sind.

Auto Admin Mapping in der Sicherheitsdatenbank

Es gibt keine SQL-Anweisungen, um die automatische Zuordnung in der Sicherheitsdatenbank ein- und auszuschalten. Stattdessen muss *gsec* verwendet werden:

```
gsec -mapping set
```

```
gsec -mapping drop
```

Mehr *gsec*-Schalter können erforderlich sein, je nachdem, welche Art von Log-In Sie verwendet haben, z.B. *-user* und *-pass*, oder *-trusted*.

Nur SYSDBA kann die automatische Zuordnung einstellen, wenn sie deaktiviert ist. Jeder Administrator kann es löschen (deaktivieren).

Administratoren

Allgemein kann man festhalten, dass ein Administrator ein Benutzer ist, der über ausreichende Rechte zum Lesen, Schreiben, Erstellen, Ändern oder Löschen von Objekten in einer Datenbank verfügt, auf die der Administratorstatus des Benutzers angewendet wird. Die folgende Tabelle fasst zusammen, wie „Superuser“-Privilegien in den verschiedenen Firebird Sicherheitskontexten aktiviert sind.

Tabelle 10.2. Administrator („Superuser“) Eigenschaften

| Benutzer | RDB\$ADMIN Role | Bemerkungen |
|---------------------------|---------------------------------------|--|
| SYSDBA | Auto | Besteht automatisch auf Server-Ebene. Hat volle Berechtigungen für alle Objekte in allen Datenbanken. Kann Benutzer erstellen, ändern und löschen, hat aber keinen direkten Zugriff auf die Sicherheitsdatenbank |
| <i>root</i> user on POSIX | Auto | Genau wie SYSDBA |
| Superuser on POSIX | Auto | Genau wie SYSDBA |
| Windows Administrator | Set as CURRENT_ROLE if login succeeds | Genau wie SYSDBA wenn alle folgenden Aussagen zutreffen: In der firebird.conf Datei <i>Authentifizierung = mixed / trusted</i> und Firebird neu gestartet wird, bevor fortgefahren wird. |

| Benutzer | RDB\$ADMIN Role | Bemerkungen |
|--------------------|---|---|
| | | AUTO ADMIN MAPPING Aktiviert in allen Datenbanken, in denen der Benutzer Superuser-Berechtigungen benötigt Enthält keine Rolle Login |
| Datenbankbesitzer | Auto | Wie SYSDBA, aber nur in der Datenbank, von der er der Besitzer ist |
| Regulärer Benutzer | Muss vorher gewährt werden; Muss bei der Anmeldung geliefert werden | Wie SYSDBA, aber nur in der Datenbank bzw. Datenbanken wo die Rolle gewährt wird |
| POSIX OS Benutzer | Muss vorher gewährt werden; Muss bei der Anmeldung geliefert werden | Wie SYSDBA, aber nur in der Datenbank bzw. Datenbanken wo die Rolle gewährt wird |
| Windows user | Muss vorher gewährt werden; Muss bei der Anmeldung geliefert werden | Wie SYSDBA, aber nur in der Datenbank bzw. Datenbanken wo die Rolle gewährt wird Nicht verfügbar wenn der config-Datei-Parameter <i>Authentifizierung</i> =native |

SQL-Anweisungen für die Benutzerverwaltung

In Firebird 2.5 und höher werden Benutzerkonten erstellt, geändert und gelöscht, indem eine Reihe von SQL-Anweisungen verwendet wird, die von einem Benutzer mit vollständigen Administratorrechten in der Sicherheitsdatenbank übermittelt werden können.

Anmerkung

Für einen Windows-Administrator reicht AUTO ADMIN MAPPING nur in einer regulären Datenbank aus, um die Verwaltung anderer Benutzer zu ermöglichen. Anweisungen zum Aktivieren in der Sicherheitsdatenbank finden Sie unter *Auto Admin Mapping in der Sicherheitsdatenbank*.

Nicht privilegierte Benutzer können nur die Anweisung ALTER USER verwenden und nur einige Daten in ihren eigenen Konten bearbeiten.

CREATE USER

Benutzt für: Erstellen eines Firebird-Benutzerkontos

Verfügbar in: DSQL

Syntax:

```
CREATE USER username PASSWORD 'password'
[FIRSTNAME 'firstname' ]
[MIDDLENAME 'middlename' ]
[LASTNAME 'lastname' ]
[GRANT ADMIN ROLE];
```

Tabelle 10.3. CREATE USER Statement Parameter

| Parameter | Beschreibung |
|------------|--|
| username | Benutzername. Die maximale Länge beträgt 31 Zeichen. Folgt den Regeln für reguläre Bezeichner in Firebird. Ist immer unabhängig von Groß- und Kleinschreibung (case-insensitive) |
| password | Benutzer-Kennwort. Seine theoretische Grenze sind 31 Bytes, aber nur die ersten 8 Zeichen werden berücksichtigt. Groß- / Kleinschreibung beachten |
| firstname | Optional: Vorname des Benutzers. Maximale Länge 31 Zeichen |
| middlename | Optional: Name des Benutzers. Maximale Länge 31 Zeichen |
| lastname | Optional: Nachname des Benutzers. Maximale Länge 31 Zeichen |

Verwenden Sie eine CREATE USER-Anweisung, um ein neues Firebird-Benutzerkonto zu erstellen. Der Benutzer darf nicht bereits in der Firebird-Sicherheitsdatenbank vorhanden sein, sonst wird eine primäre Schlüsselverletzungsfehlermeldung (primary key violation error message) zurückgegeben.

Das Argument `<username>` muss den Regeln für reguläre Bezeichner in Firebird entsprechen: siehe [Identifikatoren](#) im Kapitel *Struktur*. Benutzernamen sind immer unempfindlich gegen Groß- und Kleinschreibung (case-insensitive). Die Angabe eines Benutzernamens, der in doppelten Anführungszeichen eingeschlossen ist, wird keine Ausnahme (Exception) verursachen: Die Anführungszeichen werden ignoriert. Wenn ein Leerzeichen das einzige illegale Zeichen ist, wird der Benutzername auf das erste Leerzeichen zurückgeschnitten. Andere illegale Zeichen verursachen einen Ausnahme.

Die PASSWORD-Klausel gibt das Kennwort des Benutzers an. Ein Kennwort von mehr als acht Zeichen wird mit einer Warnung akzeptiert, aber überschüssige Zeichen werden ignoriert.

Mit den optionalen Klauseln FIRSTNAME, MIDDLENAME und LASTNAME können Sie weitere Benutzereigenschaften wie den Vornamen der Person, den Vornamen und den Nachnamen angeben. Sie sind einfach nur VARCHAR(31) Felder und können verwendet werden, um alles, was Sie bevorzugen zu speichern.

Wenn die GRANT ADMIN ROLE-Klausel angegeben ist, wird das neue Benutzerkonto mit den Berechtigungen der Rolle RDB\$ADMIN in der Sicherheitsdatenbank (`security2.fdb`) erstellt. Es erlaubt dem neuen Benutzer, Benutzerkonten aus jeder regulären Datenbank zu verwalten, in die er sich einloggt, aber er gewährt dem Benutzer keine besonderen Privilegien für Objekte in diesen Datenbanken.

Um ein Benutzerkonto zu erstellen, muss der aktuelle Benutzer in der Sicherheitsdatenbank [Administratorrechte](#) haben. Administratorrechte nur in regulären Datenbanken sind nicht ausreichend.

Anmerkung

CREATE / ALTER / DROP USER sind DDL-Anweisungen. Denken Sie daran, Ihre Arbeit zu Commiten. In *isql* wird der Befehl SET AUTO ON Auto-Commit auf DDL-Anweisungen aktivieren. In Drittanbieter-Tools und anderen Benutzeranwendungen muss dies nicht der Fall sein.

Beispiele:

1. Erstellen eines Benutzers mit dem Benutzernamen bigshot:

```
CREATE USER bigshot PASSWORD 'buckshot';
```

2. Erstellen Sie den Benutzer john mit zusätzlichen Eigenschaften (Vor- und Nachname):

```
CREATE USER john PASSWORD 'fYe_3Ksw'
FIRSTNAME 'John'
LASTNAME 'Doe';
```

3. Erstellen des Benutzers superuser mit Benutzerverwaltungsberechtigungen:

```
CREATE USER superuser PASSWORD 'kMn8Kjh'
GRANT ADMIN ROLE;
```

Siehe auch: [ALTER USER](#) , [DROP USER](#)

ALTER USER

Benutzt für: Ändern eines Firebird-Benutzerkontos

Verfügbar in: DSQL

Syntax:

```
ALTER USER username
{
  [SET]
  [PASSWORD 'password']
  [FIRSTNAME 'firstname']
  [MIDDLENAME 'middlename']
  [LASTNAME 'lastname']
}
[ {GRANT | REVOKE} ADMIN ROLE];
```

Tabelle 10.4. ALTER USER Statement Parameter

| Parameter | Beschreibung |
|-----------|---|
| username | Benutzername. Kann nicht geändert werden. |
| password | Benutzer-Kennwort. Seine theoretische Grenze sind 31 Bytes, aber nur die ersten 8 Zeichen werden berücksichtigt. Groß- / Kleinschreibung beachten |
| firstname | Optional: Vorname des Benutzers oder anderer optionaler Text. Max. Länge ist 31 Zeichen |

| Parameter | Beschreibung |
|------------|---|
| middlename | Optional: Zweiter Vorname des Benutzers oder anderer optionaler Text. Max. Länge ist 31 Zeichen |
| lastname | Optional: Nachname des Benutzers oder anderer optionaler Text. Max. Länge ist 31 Zeichen |

Verwenden Sie eine ALTER USER-Anweisung, um die Details im benannten Firebird-Benutzerkonto zu bearbeiten. Um das Konto eines anderen Benutzers zu ändern, muss der aktuelle Benutzer über [Administratorrechte](#) in der Sicherheitsdatenbank verfügen. Administratorrechte nur in regulären Datenbanken sind nicht ausreichend.

Jeder Benutzer kann sein eigenes Konto ändern. Nur ein Administrator kann GRANT / REVOKE ADMIN ROLE verwenden.

Alle Argumente sind optional, aber mindestens eines von ihnen muss vorhanden sein:

- Der Parameter PASSWORD dient zur Angabe eines neuen Passworts für den Benutzer
- FIRSTNAME, MIDDLENAME und LASTNAME erlauben die Aktualisierung der optionalen Benutzereigenschaften wie z.B. Vorname, Vorname und Nachname
- Hinzufügen der Klausel GRANT ADMIN ROLE gewährt dem Benutzer die Berechtigungen der Rolle RDB \$ADMIN in der Sicherheitsdatenbank(security2.fdb) und ermöglicht ihm die Konten anderer Benutzer zu verwalten. Es gewährt dem Benutzer keine besonderen Privilegien in den regulären Datenbanken.
- Hinzufügen der Klausel REVOKE ADMIN ROLE entfernt den Administrator des Benutzers in der Sicherheitsdatenbank, die nach Beendigung der Transaktion dem Benutzer die Möglichkeit gibt, jedes Benutzerkonto außer seinem eigenen zu verändern

Anmerkung

Denken Sie daran, Ihre Arbeit zu Commiten wenn Sie in einer Anwendung arbeiten, die Auto-Commit nicht beherrscht.

Beispiele:

1. Ändern des Passwortes für den Benutzer bobby und gewährt ihm Benutzerverwaltungsberechtigungen:

```
ALTER USER bobby PASSWORD '67-UiT_G8'
GRANT ADMIN ROLE;
```

2. Bearbeiten der optionalen Eigenschaften (der Vor- und Nachname) des Benutzers dan:

```
ALTER USER dan
FIRSTNAME 'No_Jack'
LASTNAME 'Kennedy';
```

3. Widerruf von Benutzerverwaltungsberechtigungen vom Benutzer dumbbell:

```
ALTER USER dumbbell
DROP ADMIN ROLE;
```

Siehe auch: [CREATE USER](#) , [DROP USER](#)

DROP USER

Benutzt für: Löschen eines Firebird-Benutzerkontos

Verfügbar in: DSQL

Syntax:

```
DROP USER username;
```

Tabelle 10.5. DROP USER Statement Parameter

| Parameter | Beschreibung |
|-----------|--------------|
| username | Benutzername |

Verwenden Sie die Anweisung DROP USER, um ein Firebird-Benutzerkonto zu löschen. Der aktuelle Benutzer benötigt [Administratorrechte](#).

Anmerkung

Denken Sie daran, Ihre Arbeit zu Commiten wenn Sie in einer Anwendung arbeiten, die Auto-Commit nicht beherrscht.

Beispiel: Löschen des Benutzers bobby:

```
DROP USER bobby;
```

Siehe auch: [CREATE USER](#) , [ALTER USER](#)

SQL-Berechtigungen

Die zweite Stufe von Firebirds Sicherheitsmodell sind SQL-Berechtigungen. Während ein erfolgreicher Login — die erste Stufe — den Zugriff eines Benutzers auf den Server und auf alle Datenbanken unter diesem Server autorisiert, bedeutet dies nicht, dass er Zugriff auf Objekte in beliebigen Datenbanken hat. Wenn ein Objekt erstellt wird, haben nur der Benutzer, der dieses erstellt hat (also der Besitzer), und Administratoren Zugriff darauf. Der Benutzer braucht *Privilegien* auf jedem Objekt, auf das er zugreifen muss. Grundsätzlich müssen die Berechtigungen explizit an einen Benutzer durch den Objektbesitzer oder einen [Administrator](#) der Datenbank *gewährt* werden.

Ein Privileg besteht aus einem DML-Zugriffstyp (SELECT, INSERT, UPDATE, DELETE, EXECUTE und REFERENZEN), den Namen eines Datenbankobjekts (Tabelle, View, Prozedur, Rolle) und den Namen des Benutzers

(Benutzer, Prozedur, Trigger, Rolle), dem es gewährt wird. Es stehen verschiedene Mittel zur Verfügung, um mehrere Arten von Zugriff auf ein Objekt für mehrere Benutzer in einer einzigen GRANT-Anweisung zu gewähren. Privilegien können von einem Benutzer mit REVOKE-Anweisungen entzogen werden.

Privilegien werden in der Datenbank gespeichert, auf die sie zutreffen und gelten nicht für andere Datenbanken.

Der Objekthinhaber

Der Benutzer, der ein Datenbankobjekt erstellt, wird zum Besitzer. Nur der Besitzer eines Objekts und Benutzer mit Administratorrechten in der Datenbank, einschließlich des Datenbankeigentümers, können das Datenbankobjekt ändern oder löschen.

Einige Eigentumsnachteile

Jeder authentifizierte Benutzer kann auf jede Datenbank zugreifen und jedes gültige Datenbankobjekt erstellen. Bis zu diesem Release wurde das Problem nicht beseitigt.

Da nicht alle Datenbankobjekte mit einem Besitzer — Domänen, externen Funktionen(UDFs), BLOB-Filtern, Generatoren(Sequenzen) und Ausnahmen — assoziiert sind, müssen autorisierte Objekte auf einem Server, der nicht ausreichend geschützt ist, als anfällig angesehen werden.

SYSDBA, der Datenbankeigentümer oder der Eigentümer des Objekts können Privilegien gewähren und von anderen Benutzern widerrufen, einschließlich Berechtigungen, um anderen Benutzern Privilegien zu gewähren. Der Prozess der Erteilung und des Widerrufs von SQL-Privilegien ist mit folgenden zwei Statements implementiert:

```
GRANT <privilege> ON <OBJECT-TYPE> <object-name>  
TO { <user-name> | ROLE <role-name> }
```

```
REVOKE <privilege> ON <OBJECT-TYPE> <object-name>  
FROM { <user-name> | ROLE <role-name> }
```

Der <OBJECT-TYPE> ist nicht für jede Art von Privileg erforderlich. Für einige Arten von Privilegien stehen zusätzliche Parameter zur Verfügung, entweder als Optionen oder als Anforderungen.

Statements zur Erteilung von Privilegien

Eine GRANT-Anweisung wird für die Erteilung von Privilegien — einschließlich Rollen — für Benutzer und andere Datenbankobjekte verwendet.

GRANT

Benutzt für: Privilegien und Rollen zuordnen

Verfügbar in: DSQL, ESQL

Syntax:

```

GRANT {
    <privileges> ON [TABLE] {tablename | viewname}
    | EXECUTE ON PROCEDURE procname
}
TO <grantee_list>
[WITH GRANT OPTION] | [{GRANTED BY | AS} [USER] grantor];

GRANT <role_granted>
TO <role_grantee_list> [WITH ADMIN OPTION]
[{{GRANTED BY | AS} [USER] grantor]

<privileges> ::= ALL [PRIVILEGES] | <privilege_list>

<privilege_list> ::= {<privilege> [, <privilege> [, ... ] ] }

<privilege> ::=
SELECT |
DELETE |
INSERT |
UPDATE [(col [, col [, ...] ] ) ] |
REFERENCES (col [, ...])

<grantee_list> ::= {<grantee> [, <grantee> [, ...] ]}

<grantee> ::=
[USER] username | [ROLE] rolename | GROUP Unix_group
| PROCEDURE procname | TRIGGER trigrname | VIEW viewname | P

<role_granted> ::= rolename [, rolename ...]

<role_grantee_list> ::= [USER] <role_grantee> [, [USER] <rol

<role_grantee> ::= {username | PUBLIC }
    
```

Tabelle 10.6. GRANT Statement Parameter

| Parameter | Beschreibung |
|------------|--|
| tablename | Der Name der Tabelle für welches das Privileg gilt |
| viewname | Der Name der View für welches das Privileg gilt |
| procname | Der Name der gespeicherten Prozedur für welches das EXECUTE-Privileg gilt oder den Namen der Prozedur für die das Privileg gewährt werden soll |
| col | Die Tabellenspalte, für die das Privileg gelten soll |
| Unix_group | Der Name einer Benutzergruppe in einem POSIX-Betriebssystem |
| username | Den Benutzernamen, dem die Berechtigungen gewährt werden oder denen die Rolle zugeordnet ist |
| rolename | Rollenamen |
| trigrname | Triggernamen |
| grantor | Der Benutzer der das Privileg gewährt. |

Eine GRANT-Anweisung gewährt ein oder mehrere Berechtigungen für Datenbankobjekte für Benutzer, Rollen, gespeicherte Prozeduren, Trigger oder Views.

Ein regulärer authentifizierter Benutzer hat keine Berechtigungen für jedes Datenbankobjekt, bis es explizit gewährt wurde. Es wird entweder an den einzelnen Benutzer oder an alle Benutzer, die als Benutzer PUBLIC gebündelt wurden, gewährt. Wenn ein Objekt erstellt wird, haben nur der Benutzer, der es erstellt hat (der Eigentümer) und **Administratoren** Berechtigungen hierauf und können anderen Benutzern, Rollen oder Objekten Privilegien gewähren.

Für verschiedene Arten von Metadatenobjekten gelten unterschiedliche Privilegien. Die verschiedenen Arten von Privilegien werden später separat beschrieben.

Die TO-Klausel

Die TO-Klausel wird für die Auflistung der Benutzer, Rollen und Datenbankobjekte (Prozeduren, Trigger und Views) verwendet, denen die in <privileges> aufgezählten Privilegien gewährt werden sollen. Die Klausel ist zwingend erforderlich.

Mit den optionalen Schlüsselwörtern USER und ROLE in der TO-Klausel können Sie genau festlegen, wer oder was das Privileg gewährt hat. Wenn ein USER- oder ROLE-Schlüsselwort nicht angegeben ist, prüft der Server auf eine Rolle mit diesem Namen und wenn es keine gibt, werden die Berechtigungen dem Benutzer ohne weitere Überprüfung gewährt.

Berechtigungen in einem ROLE-Objekt zusammenfassen

Eine Rolle ist ein „Container“-Objekt, das verwendet werden kann, um eine Sammlung von Privilegien zu verpacken. Die Verwendung der Rolle wird dann jedem Benutzer gewährt, der diese Berechtigungen erfordert. Eine Rolle kann auch einer Liste von Benutzern zugewiesen werden.

Die Rolle muss bestehen, bevor Privilegien gewährt werden können. Siehe **CREATE ROLE** im DDL-Kapitel für die Syntax und die Regeln. Die Rolle wird gepflegt durch die Erteilung von Privilegien und, wenn notwendig, deren Entzug. Wenn eine Rolle gelöscht wird (siehe **DROP ROLE**) verlieren alle Benutzer die durch die Rolle erworbenen Privilegien. Alle Privilegien, die zusätzlich zu einem betroffenen Benutzer über eine andere Grant-Anweisung gewährt wurden, bleiben erhalten.

Ein Benutzer, der eine Rolle erhält, muss diese Rolle mit seinen Anmeldeinformationen versehen, um die zugehörigen Berechtigungen auszuführen. Alle anderen Berechtigungen, die dem Benutzer eingeräumt werden, sind nicht von der Anmeldung mit einer Rolle betroffen.

Mehr als eine Rolle kann demselben Benutzer gewährt werden, aber die Anmeldung mit mehreren Rollen gleichzeitig wird nicht unterstützt.

Eine Rolle kann nur einem Benutzer zugewiesen werden.

Bitte beachten Sie:

- Wenn eine GRANT-Anweisung ausgeführt wird, wird die Sicherheitsdatenbank nicht auf die Existenz des gewährenden Benutzers geprüft. Dies ist kein Fehler: SQL-Berechtigungen betreffen die Kontrolle des Datenzugriffs für authentifizierte Benutzer, sowohl native als auch vertrauenswürdige und vertrauenswürdige Betriebssystembenutzer werden nicht in der Sicherheitsdatenbank gespeichert.
- Wenn Sie einem Datenbankobjekt, z.B. einer Prozedur, einem Trigger oder einer View, ein Privileg gewähren, müssen Sie den Objekttyp zwischen dem Schlüsselwort TO und dem Objektnamen angeben.
- Obwohl die Schlüsselwörter USER und ROLE optional sind, ist es ratsam, sie zu verwenden, um Unklarheiten zu vermeiden.

Der PUBLIC-Benutzer

Firebird hat einen vordefinierten Benutzer namens PUBLIC, der alle Benutzer repräsentiert. Berechtigungen für Operationen auf einem bestimmten Objekt, die dem Benutzer PUBLIC gewährt werden, können von jedem Benutzer ausgeübt werden, der bei der Anmeldung authentifiziert wurde.

Wichtig

Wenn dem Benutzer PUBLIC Privilegien gewährt werden, sollten sie auch vom Benutzer PUBLIC widerrufen werden.

Die WITH GRANT OPTION-Klausel

Die optionale WITH GRANT OPTION-Klausel ermöglicht es den Benutzern, die in der Benutzerliste angegeben sind, die in der Berechtigungsliste angegebenen Berechtigungen anderen Benutzern zuzuweisen.

Achtung

Es ist möglich, diese Option dem Benutzer PUBLIC zuzuordnen. Dies sollte aber niemals getan werden!

Die GRANTED BY-Klausel

Standardmäßig werden, wenn Berechtigungen in einer Datenbank gewährt werden, der aktuelle Benutzer als Gewährender aufgezeichnet. Die GRANTED BY-Klausel ermöglicht es dem aktuellen Benutzer, diese Privilegien als anderer Benutzer zu erteilen.

Wenn die REVOKE-Anweisung verwendet wird, wird es fehlschlagen, wenn der aktuelle Benutzer nicht der Benutzer ist, der in der GRANTED BY-Klausel verwendet wurde.

Alternative Syntax mit AS <username>

Die nicht standardmäßige AS-Klausel wird als Synonym für die GRANTED BY-Klausel unterstützt, um die Migration von anderen Datenbanksystemen zu vereinfachen.

Die Klauseln GRANTED BY und AS können nur vom Datenbankeigentümer und [Administratoren](#) verwendet werden. Der Objektbesitzer kann diese nicht verwenden, es sei denn, er hat auch Administratorrechte.

Privilegien auf Tabellen und Views

In der Theorie gewährt eine GRANT-Anweisung eine Berechtigung für einen Benutzer oder Objekt. In der Praxis erlaubt die Syntax, dass mehrere Berechtigungen mehreren Benutzern in einer GRANT-Anweisung erteilt werden können.

Syntax-Auszug:

```
...
<privileges> ::= ALL [PRIVILEGES] | <privilege_list>
<privilege_list> ::= {<privilege> [, <privilege> [, ... ]
```

```

<privilege> ::= {
SELECT |
DELETE |
INSERT |
UPDATE [(col [,col [, ...]])] ] ) ] |
REFERENCES (col [, col [, ...] ] )
}

```

Tabelle 10.7. Liste der Berechtigungen auf Tabellen

| Privileg | Beschreibung |
|------------|--|
| SELECT | Erlaubt den Benutzer oder das Objekt, Daten aus der Tabelle oder der View zu löschen |
| INSERT | Erlaubt den Benutzer oder das Objekt Zeilen in die Tabelle oder View hinzuzufügen(INSERT) |
| UPDATE | Erlaubt den Benutzer oder das Objekt Zeilen in der Tabelle oder View zu aktualisieren(UPDATE), die optional auf bestimmte Spalten beschränkt ist |
| <i>col</i> | (Optional) Name einer Spalte, auf die das UPDATE-Privileg des Benutzers beschränkt ist |
| DELETE | Erlaubt den Benutzer oder das Objekt, Zeilen aus der Tabelle oder der View zu löschen(DELETE) |
| REFERENCES | Erlaubt dem Benutzer oder Objekt, die angegebenen Spalten der Tabelle über einen Fremdschlüssel zu verweisen. Wenn der primäre oder eindeutige Schlüssel, auf den der Fremdschlüssel der anderen Tabelle verweist, zusammengesetzt ist, müssen alle Spalten des Schlüssels angegeben werden. |
| <i>col</i> | (Zwingend notwendig) Name einer Spalte im referenzierten Fremdschlüssel |
| ALL | Kombiniert SELECT, INSERT, UPDATE, DELETE und REFERENCES Privilegien in einem einzigen Paket |

Beispiele für GRANT <Privilegien> auf Tabellen:

1. SELECT und INSERT-Berechtigungen für den Benutzer ALEX:

```

GRANT SELECT, INSERT ON TABLE SALES
TO USER ALEX;

```

2. Das SELECT-Privileg für die MANAGER-, ENGINEER-Rollen und für den Benutzer IVAN:

```

GRANT SELECT ON TABLE CUSTOMER
TO ROLE MANAGER, ROLE ENGINEER, USER IVAN;

```

3. Alle Privilegien für die Rolle des ADMINISTRATORS, zusammen mit der Befugnis, denselben Privilegien zu gewähren:

```
GRANT ALL ON TABLE CUSTOMER
TO ROLE ADMINISTRATOR
WITH GRANT OPTION;
```

4. Die SELECT- und REFERENZ-Berechtigungen für die NAME-Spalte für alle Benutzer und Objekte:

```
GRANT SELECT, REFERENCES (NAME) ON TABLE COUNTRY
TO PUBLIC;
```

5. Das SELECT-Privileg wird dem Benutzer IVAN vom Benutzer ALEX erteilt:

```
GRANT SELECT ON TABLE EMPLOYEE
TO USER IVAN
GRANTED BY ALEX;
```

6. Gewähren der UPDATE-Berechtigung für die Spalten FIRST_NAME, LAST_NAME:

```
GRANT UPDATE (FIRST_NAME, LAST_NAME) ON TABLE EMPLOYEES
TO USER IVAN;
```

7. Gewähren der INSERT-Berechtigung für die gespeicherte Prozedur ADD_EMP_PROJ:

```
GRANT INSERT ON EMPLOYEE_PROJECT
TO PROCEDURE ADD_EMP_PROJ;
```

Das EXECUTE-Privileg

Das EXECUTE-Privileg gilt für gespeicherte Prozeduren. Es erlaubt dem Gewährenden, die gespeicherte Prozedur auszuführen und ggf. seine Ausgabe abzurufen. Im Falle von auswählbaren gespeicherten Prozeduren wirkt es etwas wie ein SELECT-Privileg, sofern dieser Stil der gespeicherten Prozedur in Reaktion auf eine SELECT-Anweisung ausgeführt wird.

Beispiel: Gewähren des EXECUTE-Privilegs auf einer gespeicherten Prozedur zu einer Rolle:

```
GRANT EXECUTE ON PROCEDURE ADD_EMP_PROJ
TO ROLE MANAGER;
```

Rollen zuordnen

Die Zuweisung einer Rolle ist vergleichbar mit der Gewährung eines Privilegs. Eine oder mehrere Rollen können einem oder mehreren Benutzern zugewiesen werden, einschließlich dem `user PUBLIC`, mit einer GRANT-Anweisung.

Die WITH ADMIN OPTION-Klausel

Mit der optionalen WITH ADMIN OPTION-Klausel können die in der Benutzerliste angegebenen Benutzer die für andere Benutzer angegebene Rolle(n) erteilen.

Achtung

Es ist möglich, diese Option dem Benutzer PUBLIC zuzuordnen. Dies sollte aber niemals getan werden!

Beispiele der Rollenzuweisung:

1. Zuweisen der DIRECTOR- und MANAGER-Rollen dem Benutzer IVAN:

```
GRANT DIRECTOR, MANAGER TO USER IVAN;
```

2. Zuweisen der ADMIN-Rolle dem Benutzer ALEX mit der Berechtigung, diese Rolle anderen Benutzern zuzuordnen:

```
GRANT MANAGER TO USER ALEX WITH ADMIN OPTION;
```

Siehe auch: [REVOKE](#)

Anweisungen zum widerrufen von Privilegien

Eine REVOKE-Anweisung wird zum Widerrufen von Berechtigungen, einschließlich Rollen, von Benutzern und anderen Datenbankobjekten verwendet.

REVOKE

Benutzt für: Widerrufen von Privilegien oder Rollenzuweisungen

Verfügbar in: DSQL, ESQL

Syntax:

```
REVOKE [GRANT OPTION FOR] {  
  <privileges> ON [TABLE] {tablename | viewname} |  
  EXECUTE ON PROCEDURE procname }  
FROM <grantee_list>  
[ {GRANTED BY | AS} [USER] grantor];  
  
REVOKE [ADMIN OPTION FOR] <role_granted>  
FROM {PUBLIC | <role_grantee_list>}  
[ {GRANTED BY | AS} [USER] grantor];  
  
REVOKE ALL ON ALL FROM <grantee_list>
```

```

<privileges> ::= ALL [PRIVILEGES] | <privilege_list>

<privilege_list> ::= {<privilege> [, <privilege> [, ... ] ] }

<privilege> ::=
SELECT |
DELETE |
INSERT |
UPDATE [(col [, col [, col [,...]] ] ) ] |
REFERENCES (col [, col [, ...] ] )

<grantee_list> ::= {<grantee> [, <grantee> [, ...] ]}

<grantee> ::=
[USER] username | [ROLE] rolename | GROUP Unix_group
| PROCEDURE procname | TRIGGER trigrname | VIEW viewname | P

<role_granted> ::= rolename [, rolename ...]

<role_grantee_list> ::= [USER] <role_grantee> [, [USER] <rol

<role_grantee> ::= {username | PUBLIC }

```

Tabelle 10.8. REVOKE Anweisung Parameter

| Parameter | Beschreibung |
|------------|---|
| tablename | Der Name der Tabelle wo das Privileg widerrufen werden soll |
| viewname | Der Name der View wo das Privileg widerrufen werden soll |
| procname | Der Name der Prozedur wo das EXECUTE-Privileg widerrufen werden soll oder der Name der Prozedur wo die Privilegien widerrufen werden sollen |
| trigrname | Triggernamen |
| col | Der Name der Spalte wo das Privileg widerrufen werden soll |
| username | Der Benutzername oder Rolle wo das Privileg widerrufen werden soll |
| rolename | Rollenname |
| Unix_group | Der Name einer Benutzergruppe in einem POSIX-Betriebssystem |
| grantor | Der Gewährende-Benutzer, in dessen Namen das Privileg abgelehnt wird |

Die REVOKE-Anweisung wird verwendet, um Privilegien von Benutzern, Rollen, gespeicherten Prozeduren, Triggern und Views zu widerrufen, die mit der GRANT-Anweisung gewährt wurden. Siehe [GRANT](#) für eine detaillierte Beschreibung der verschiedenen Arten von Privilegien.

Nur der Benutzer, der das Privileg erteilt hat, kann ihn widerrufen.

Die FROM-Klausel

Die FROM-Klausel wird verwendet, um die Liste der Benutzer, Rollen und Datenbankobjekte (Prozeduren, Trigger und Views) anzugeben, die die aufgezählten Privilegien widerrufen haben. Mit den optionalen USER und ROLE Schlüsselwörter in der FROM-Klausel können Sie genau festlegen, welcher Typ das Privileg widerrufen

soll. Wenn ein USER oder ROLE Schlüsselwort nicht angegeben ist, prüft der Server auf eine Rolle mit diesem Namen und wenn es keine gibt, werden die Berechtigungen vom Benutzer ohne weitere Überprüfung widerrufen.

Tipps

- Obwohl die Schlüsselwörter USER und ROLE optional sind, ist es ratsam, sie zu verwenden, um Unklarheiten zu vermeiden.
- Die Anweisung GRANT prüft nicht auf die Existenz des Benutzers, von dem die Berechtigungen widerrufen werden.
- Wenn Sie ein Privileg aus einem Datenbankobjekt widerrufen, müssen Sie seinen Objekttyp angeben

Widerruf von Berechtigungen des Benutzers PUBLIC

Privilegien, die dem speziellen Benutzer PUBLIC gewährt wurden, müssen vom Benutzer PUBLIC widerrufen werden. Benutzer PUBLIC bietet eine Möglichkeit, allen Benutzern gleichzeitig Privilegien zu gewähren, aber es ist nicht „eine Gruppe von Benutzern“.

Widerrufen der GRANT OPTION

Die optionale GRANT OPTION FOR-Klausel widerruft das Privileg des Benutzers, Berechtigungen zu gewähren für die Tabelle, die View, die Trigger oder die gespeicherte Prozedur für andere Benutzer oder Rollen. Es widerruft nicht das Privileg, mit dem die Grant-Option verbunden ist.

Entfernen der Berechtigung von einer oder mehreren Rollen

Eine Verwendung der Anweisung REVOKE besteht darin, Rollen zu entfernen, die einem Benutzer oder einer Gruppe von Benutzern durch eine GRANT-Anweisung zugewiesen wurden. Im Fall von mehreren Rollen und/oder mehreren Gewährenden folgt dem Verzeichnis REVOKE die Liste der Rollen, die aus der Liste der Benutzer entfernt werden, die nach der FROM-Klausel angegeben sind.

Die optionale ADMIN OPTION FOR-Klausel bietet die Möglichkeit, die „Administratorrechte“ des Gewährenden zu widerrufen, somit die Möglichkeit den anderen Benutzern dieselbe Rolle zuzuweisen, ohne das Privileg des Gewährenden der Rolle zu widerrufen.

Mehrere Rollen und Gewährende können in einer einzigen Anweisung verarbeitet werden.

Widerrufen von Privilegien von GRANTED BY

Ein Privileg, das mit der GRANTED BY-Klausel erteilt wurde, wird intern explizit dem von dieser ursprünglichen GRANT-Anweisung bezeichneten Gewährenden zugeschrieben. Um ein Privileg zu widerrufen, das durch diese Methode erhalten wurde, muss der aktuelle Benutzer entweder mit vollständigen Administratorrechten angemeldet sein oder als der Benutzer, der als <Gewährender> durch diese GRANTED BY-Klausel ist.

Anmerkung

Die gleiche Regel gilt, wenn die Syntax, die in der ursprünglichen GRANT-Anweisung verwendet wird, das synonym AS verwendet, um die Klausel anstelle des Standards GRANTED BY einzuführen.

Widerrufen von ALL ON ALL

Wenn der aktuelle Benutzer mit **Administrator**-rechte in der Datenbank angemeldet ist, kann die Anweisung

```
REVOKE ALL ON ALL FROM <grantee_list>
```

verwendet werden, um alle Privilegien (einschließlich Rollenmitgliedschaften) auf alle Objekte von einem oder mehreren Benutzern und/oder Rollen zu widerrufen. Alle Privilegien für den Benutzer werden entfernt, unabhängig davon, wer sie gewährt hat. Es ist ein schneller Weg die Privilegien zu entfernen, wenn der Zugriff auf die Datenbank für einen bestimmten Benutzer oder eine Rolle gesperrt werden muss.

Wenn der aktuelle Benutzer nicht als Administrator angemeldet ist, werden nur die Berechtigungen aufgehoben, die ursprünglich von diesem Benutzer erteilt wurden.

Die Anweisung REVOKE ALL ON ALL kann nicht verwendet werden, um Berechtigungen zu widerrufen, die für gespeicherte Prozeduren, Trigger oder Views gewährt wurden.

Anmerkung

Die GRANTED BY-Klausel wird nicht unterstützt.

Beispiele für REVOKE

1. Widerruf der Privilegien zum Lesen und Einfügen in die SALES:

```
REVOKE SELECT, INSERT ON TABLE SALES FROM USER ALEX;
```

2. Widerruf des Privilegs für das Lesen der CUSTOMER-Tabelle vom MANAGER- und ENGINEER-Rollen und vom Benutzer IVAN:

```
REVOKE SELECT ON TABLE CUSTOMER  
FROM ROLE MANAGER, ROLE ENGINEER, USER IVAN;
```

3. Widerruf von der Rolle des ADMINISTRATORS die Befugnis, anderen Benutzern oder Rollen irgendwelche Privilegien auf der CUSTOMER-Tabelle zu gewähren:

```
REVOKE GRANT OPTION FOR ALL ON TABLE CUSTOMER  
FROM ROLE ADMINISTRATOR;
```

4. Widerrufen des Privileges zum Lesen der COUNTRY-Tabelle und die Referenz auf die NAME-Spalte der COUNTRY-Tabelle von jedem Benutzer zu verweisen, über den speziellen Benutzer PUBLIC:

```
REVOKE SELECT, REFERENCES (NAME) ON TABLE COUNTRY  
FROM PUBLIC;
```

5. Das Privileg zum Lesen der EMPLOYEE-Tabelle aus dem Benutzer IVAN, das vom Benutzer ALEX erteilt wurde, widerrufen:

```
REVOKE SELECT ON TABLE EMPLOYEE  
FROM USER IVAN GRANTED BY ALEX;
```

6. Widerruf der Berechtigung zum Aktualisieren der Spalten FIRST_NAME und LAST_NAME der EMPLOYEE-Tabelle vom Benutzer IVAN:

```
REVOKE UPDATE (FIRST_NAME, LAST_NAME) ON TABLE EMPLOYEE  
FROM USER IVAN;
```

7. Zurückziehen der Berechtigung zum Einfügen von Datensätzen in die Tabelle EMPLOYEE_PROJECT aus der Prozedur ADD_EMP_PROJ:

```
REVOKE INSERT ON EMPLOYEE_PROJECT  
FROM PROCEDURE ADD_EMP_PROJ;
```

8. Das Privileg zur Ausführung des Vorgangs ADD_EMP_PROJ aus der MANAGER-Rolle widerrufen:

```
REVOKE EXECUTE ON PROCEDURE ADD_EMP_PROJ  
FROM ROLE MANAGER;
```

9. Widerruf der DIRECTOR- und MANAGER-Rollen vom Benutzer IVAN:

```
REVOKE DIRECTOR, MANAGER FROM USER IVAN;
```

10. Widerrufen Sie dem Benutzer ALEX die Berechtigung, die MANAGER-Rolle anderen Benutzern zuzuordnen:

```
REVOKE ADMIN OPTION FOR MANAGER FROM USER IVAN;
```

11. Widerruf aller Privilegien (einschließlich Rollen) auf alle Objekte vom Benutzer IVAN:

```
REVOKE ALL ON ALL FROM IVAN;
```

Nachdem diese Anweisung ausgeführt wurde, hat der Benutzer IVAN überhaupt keine Privilegien.

Siehe auch: [GRANT](#)

Anhang A: Zusatzinformationen

In diesem Anhang finden Sie Themen, auf die sich Entwickler beziehen können, um das Verständnis für Funktionen oder Änderungen zu verbessern.

Das Feld RDB\$VALID_BLR

Das Feld RDB\$VALID_BLR wurde zu den Systemtabellen RDB\$PROCEDURES und RDB\$TRIGGERS in Firebird 2.1 hinzugefügt. Sein Zweck ist, eine mögliche Ungültigkeit eines PSQL-Moduls nach einer Änderung einer Domäne oder Tabellenspalte, von der das Modul abhängt, zu signalisieren. RDB\$VALID_BLR wird auf 0 gesetzt, sobald der Code einer Prozedur oder eines Triggers ungültig durch eine solche Änderung wird.

Funktionsweise der Invalidierung

Bei Triggern und Prozeduren ergeben sich Abhängigkeiten durch die Definitionen der Tabellenspalten, auf die zugegriffen wird, und auch über alle Parameter oder Variablen, die im Modul mit der TYPE OF-Klausel definiert wurden.

Nachdem die Engine jede Domain, einschließlich der impliziten Domains, die intern hinter den Spaltendefinitionen und den Ausgabeparametern erstellt wurden, geändert hat, werden alle ihre Abhängigkeiten intern neu kompiliert.

Anmerkung

In V.2.x beinhaltet dies Prozeduren und Trigger, jedoch keine Blöcke, die in DML-Anweisungen für die Laufzeitausführung mit EXECUTE BLOCK codiert sind. Firebird 3 umfasst weitere Modultypen (gespeicherte Funktionen, Pakete).

Jedes Modul, das nicht neukompiliert werden kann, aufgrund einer Inkompatibilität durch eine Änderung einer Domain, wird als ungültig markiert („invalidated“). Dies geschieht durch Setzen von RDB\$VALID_BLR im Systemdatensatz (in RDB\$PROCEDURES oder RDB\$TRIGGERS) auf 0.

Revalidierung (setzen von RDB\$VALID_BLR auf 1) tritt auf, wenn

1. die Domain erneut geändert wird und die neue Definition kompatibel zur vorigen invalidierten Moduldefinition ist; ODER
2. das vorige invalidierte Modul angepasst wurde um zur neuen Domain-Definition zu passen

Die folgende Abfrage zeigt alle Module an, die von einer bestimmten Domain abhängig sind, und gibt deren RDB\$VALID_BLR-Status aus:

```

SELECT * FROM (
  SELECT
    'Procedure',
    rdb$procedure_name,
    rdb$valid_blr
  FROM rdb$procedures
  UNION ALL
  SELECT
    'Trigger',
    rdb$trigger_name,
    rdb$valid_blr
  FROM rdb$triggers
) (type, name, valid)
WHERE EXISTS
  (SELECT * from rdb$dependencies
   WHERE rdb$dependent_name = name
    AND rdb$depended_on_name = 'MYDOMAIN')

/* Replace MYDOMAIN with the actual domain name.
   Use all-caps if the domain was created
   case-insensitively. Otherwise, use the exact
   capitalisation. */

```

Die folgende Abfrage zeigt alle Module an, die von einer bestimmten Tabellenspalte abhängig sind, und gibt deren RDB\$VALID_BLR-Status aus:

```

SELECT * FROM (
  SELECT
    'Procedure',
    rdb$procedure_name,
    rdb$valid_blr
  FROM rdb$procedures
  UNION ALL
  SELECT
    'Trigger',
    rdb$trigger_name,
    rdb$valid_blr
  FROM rdb$triggers) (type, name, valid)
WHERE EXISTS
  (SELECT *
   FROM rdb$dependencies
   WHERE rdb$dependent_name = name
    AND rdb$depended_on_name = 'MYTABLE'
    AND rdb$field_name = 'MYCOLUMN')

```

Wichtig

Alle PSQL-Invalidierungen die durch Änderungen einer Domain oder Spalte verursacht werden, spiegeln sich im RDB\$VALID_BLR-Feld wieder. Jedoch beeinflussen andere Änderungen, wie beispielsweise die Anzahl der Ein- und Ausgabeparameter, aufgerufene Routinen usw. die Gültigkeitsfelder nicht, obwohl sie die Module potenziell invalidieren. Ein typisches Szenario ist folgendes:

1. Eine Prozedur (B) wird definiert, diese ruft eine andere Prozedur (A) auf und liest ihre Ausgabeparameter. In diesem Fall wird die Abhängigkeit in RDB\$DEPENDENCIES registriert. Danach wird Prozedur (A) angepasst. Es werden ein oder mehr Ausgabeparameter angepasst oder entfernt. Das Statement ALTER PROCEDURE A wird mit einem Fehler quittiert, sobald versucht wird ein Commit durchzuführen.
2. Eine Prozedur (B) ruft Prozedur A auf und übergibt Werte für die Eingabeparameter. Keine Abhängigkeiten werden in RDB\$DEPENDENCIES registriert. Danach werden Änderungen an den Eingabeparametern in Prozedur A durchgeführt. Fehler werden zur Laufzeit auftreten, wenn B A aufruft und es nicht-passende Eingabeparameter übergeben werden.

Weitere Hinweise Notes

- Für PSQL-Module aus früheren Firebird-Versionen (gilt für einige Systemtrigger, sogar wenn die Datenbank unter Firebird 2.1. oder höher erstellt wurde), ist RDB\$VALID_BLR NULL. Dies heißt nicht, dass ihre BLR ungültig ist.
- Die *isql*-Befehle SHOW PROCEDURES und SHOW TRIGGERS zeigen ein Sternchen in der RDB \$VALID_BLR-Spalte für alle Module, deren Wert 0 ist (ungültig). Für SHOW PROCEDURE <procname> und SHOW TRIGGER <trigname>, welche einzelne PSQL-Module sind, werden gar keien ungültigen BLR signalisiert.

Ein Hinweis zur Gleichheit

Wichtig

Diese Anmerkung über Gleichheits- und Ungleichheits-Operatoren gilt überall in der Firebird SQL-Sprache.

Der „="-Operator, welcher ausdrücklich in vielen Bedingungen verwendet wird, prüft nur Werte gegen Werte. Entsprechend dem SQL-Standard, ist NULL kein Wert und somit sind zwei NULLs weder gleich noch ungleich zueinander. Wenn Sie NULLs in Bedingungen vergleichen müssen, nutzen Sie den Operator IS NOT DISTINCT FROM. Dieser gibt wahr zurück, falls die Operanden den gleichen Wert besitzen *oder* beide NULL sind.

```
select *
  from A join B
 on A.id is not distinct from B.code
```

In Fällen in denen Sie gegen NULL innerhalb einer *ungleich*-Bedingung testen wollen, nutzen Sie IS DISTINCT FROM, nicht „<>“. Möchten Sie NULL unterschiedlich zu anderen Werten und zwei NULLs als gleich betrachten:

```
select *
  from A join B
 on A.id is distinct from B.code
```


Anhang B: Fehlercodes und Meldungen

Dieser Anhang enthält:

- **SQLSTATE Fehlercodes und Beschreibungen**
- **GDSCODE Fehlercodes, SQLCODEs und Beschreibungen**
 1. GDSCODEs 335544366 bis 335544334
 2. GDSCODEs 335544454 bis 336330760
 3. GDSCODEs 335544329 bis 335544613
 4. GDSCODEs 335544614 bis 335544689

Selbstdefinierte Exceptions

Firebird DDL unterstützt eine einfache Syntax zum Erstellen von benutzerdefinierten Exceptions für die Verwendung in PSQL-Modulen mit einem Meldungstext von bis zu 1.021 Zeichen. Für weitere Informationen, vgl. **CREATE EXCEPTION** in *DDL Statements* und, für die Verwendung, das Statement **EXCEPTION** in *PSQL Statements*.

Die Firebird SQLCODE Fehlercodes korrelieren nicht mit den Standardkonformen SQLSTATE-Codes. SQLCODE wurde viele Jahre verwendet und wird als veraltet angesehen. Der Support für SQLCODE wird vermutlich in zukünftigen Versionen entfernt.

SQLSTATE Fehlercodes und Beschreibungen

Die folgende Tabelle zeigt die Fehlercodes und Meldungen für die SQLSTATE Kontextvariablen.

Die Struktur eines SQLSTATE Fehlercodes besteht aus fünf Zeichen, die die SQL-Fehlerklasse (2 Zeichen) und die SQL-Subklasse (3 Zeichen) charakterisieren.

Tabelle B.1. SQLSTATE Fehlercodes und Meldungen

| SQLSTATE | Zugewiesene Meldung |
|-----------------------|---------------------------------------|
| SQLCLASS 00 (Success) | |
| 00000 | Success |
| SQLCLASS 01 (Warning) | |
| 01000 | General warning |
| 01001 | Cursor operation conflict |
| 01002 | Disconnect error |
| 01003 | NULL value eliminated in set function |
| 01004 | String data, right-truncated |

| SQLSTATE | Zugewiesene Meldung |
|--|---|
| 01005 | Insufficient item descriptor areas |
| 01006 | Privilege not revoked |
| 01007 | Privilege not granted |
| 01008 | Implicit zero-bit padding |
| 01100 | Statement reset to unprepared |
| 01101 | Ongoing transaction has been committed |
| 01102 | Ongoing transaction has been rolled back |
| SQLCLASS 02 (No Data) | |
| 02000 | No data found or no rows affected |
| SQLCLASS 07 (Dynamic SQL error) | |
| 07000 | Dynamic SQL error |
| 07001 | Wrong number of input parameters |
| 07002 | Wrong number of output parameters |
| 07003 | Cursor specification cannot be executed |
| 07004 | USING clause required for dynamic parameters |
| 07005 | Prepared statement not a cursor-specification |
| 07006 | Restricted data type attribute violation |
| 07007 | USING clause required for result fields |
| 07008 | Invalid descriptor count |
| 07009 | Invalid descriptor index |
| SQLCLASS 08 (Connection Exception) | |
| 08001 | Client unable to establish connection |
| 08002 | Connection name in use |
| 08003 | Connection does not exist |
| 08004 | Server rejected the connection |
| 08006 | Connection failure |
| 08007 | Transaction resolution unknown |
| SQLCLASS 0A (Feature Not Supported) | |
| 0A000 | Feature Not Supported |
| SQLCLASS 0B (Invalid Transaction Initiation) | |
| 0B000 | Invalid transaction initiation |

| SQLSTATE | Zugewiesene Meldung |
|---|--|
| SQLCLASS 0L (Invalid Grantor) | |
| 0L000 | Invalid grantor |
| SQLCLASS 0P (Invalid Role Specification) | |
| 0P000 | Invalid role specification |
| SQLCLASS 0U (Attempt to Assign to Non-Updatable Column) | |
| 0U000 | Attempt to assign to non-updatable column |
| SQLCLASS 0V (Attempt to Assign to Ordering Column) | |
| 0V000 | Attempt to assign to Ordering column |
| SQLCLASS 20 (Case Not Found For Case Statement) | |
| 20000 | Case not found for case statement |
| SQLCLASS 21 (Cardinality Violation) | |
| 21000 | Cardinality violation |
| 21S01 | Insert value list does not match column list |
| 21S02 | Degree of derived table does not match column list |
| SQLCLASS 22 (Data Exception) | |
| 22000 | Data exception |
| 22001 | String data, right truncation |
| 22002 | Null value, no indicator parameter |
| 22003 | Numeric value out of range |
| 22004 | Null value not allowed |
| 22005 | Error in assignment |
| 22006 | Null value in field reference |
| 22007 | Invalid datetime format |
| 22008 | Datetime field overflow |
| 22009 | Invalid time zone displacement value |
| 2200A | Null value in reference target |
| 2200B | Escape character conflict |
| 2200C | Invalid use of escape character |
| 2200D | Invalid escape octet |
| 2200E | Null value in array target |
| 2200F | Zero-length character string |

| SQLSTATE | Zugewiesene Meldung |
|---|---|
| 2200G | Most specific type mismatch |
| 22010 | Invalid indicator parameter value |
| 22011 | Substring error |
| 22012 | Division by zero |
| 22014 | Invalid update value |
| 22015 | Interval field overflow |
| 22018 | Invalid character value for cast |
| 22019 | Invalid escape character |
| 2201B | Invalid regular expression |
| 2201C | Null row not permitted in table |
| 22012 | Division by zero |
| 22020 | Invalid limit value |
| 22021 | Character not in repertoire |
| 22022 | Indicator overflow |
| 22023 | Invalid parameter value |
| 22024 | Character string not properly terminated |
| 22025 | Invalid escape sequence |
| 22026 | String data, length mismatch |
| 22027 | Trim error |
| 22028 | Row already exists |
| 2202D | Null instance used in mutator function |
| 2202E | Array element error |
| 2202F | Array data, right truncation |
| SQLCLASS 23 (Integrity Constraint Violation) | |
| 23000 | Integrity constraint violation |
| SQLCLASS 24 (Invalid Cursor State) | |
| 24000 | Invalid cursor state |
| 24504 | The cursor identified in the UPDATE, DELETE, SET, or GET statement is not positioned on a row |
| SQLCLASS 25 (Invalid Transaction State) | |
| 25000 | Invalid transaction state |

| SQLSTATE | Zugewiesene Meldung |
|---|---|
| 25S01 | Transaction state |
| 25S02 | Transaction is still active |
| 25S03 | Transaction is rolled back |
| SQLCLASS 26 (Invalid SQL Statement Name) | |
| 26000 | Invalid SQL statement name |
| SQLCLASS 27 (Triggered Data Change Violation) | |
| 27000 | Triggered data change violation |
| SQLCLASS 28 (Invalid Authorization Specification) | |
| 28000 | Invalid authorization specification |
| SQLCLASS 2B (Dependent Privilege Descriptors Still Exist) | |
| 2B000 | Dependent privilege descriptors still exist |
| SQLCLASS 2C (Invalid Character Set Name) | |
| 2C000 | Invalid character set name |
| SQLCLASS 2D (Invalid Transaction Termination) | |
| 2D000 | Invalid transaction termination |
| SQLCLASS 2E (Invalid Connection Name) | |
| 2E000 | Invalid connection name |
| SQLCLASS 2F (SQL Routine Exception) | |
| 2F000 | SQL routine exception |
| 2F002 | Modifying SQL-data not permitted |
| 2F003 | Prohibited SQL-statement attempted |
| 2F004 | Reading SQL-data not permitted |
| 2F005 | Function executed no return statement |
| SQLCLASS 33 (Invalid SQL Descriptor Name) | |
| 33000 | Invalid SQL descriptor name |
| SQLCLASS 34 (Invalid Cursor Name) | |
| 34000 | Invalid cursor name |
| SQLCLASS 35 (Invalid Condition Number) | |
| 35000 | Invalid condition number |
| SQLCLASS 36 (Cursor Sensitivity Exception) | |
| 36001 | Request rejected |

| SQLSTATE | Zugewiesene Meldung |
|---|--|
| 36002 | Request failed |
| SQLCLASS 37 (Invalid Identifier) | |
| 37000 | Invalid identifier |
| 37001 | Identifier too long |
| SQLCLASS 38 (External Routine Exception) | |
| 38000 | External routine exception |
| SQLCLASS 39 (External Routine Invocation Exception) | |
| 39000 | External routine invocation exception |
| SQLCLASS 3B (Invalid Save Point) | |
| 3B000 | Invalid save point |
| SQLCLASS 3C (Ambiguous Cursor Name) | |
| 3C000 | Ambiguous cursor name |
| SQLCLASS 3D (Invalid Catalog Name) | |
| 3D000 | Invalid catalog name |
| 3D001 | Catalog name not found |
| SQLCLASS 3F (Invalid Schema Name) | |
| 3F000 | Invalid schema name |
| SQLCLASS 40 (Transaction Rollback) | |
| 40000 | Ongoing transaction has been rolled back |
| 40001 | Serialization failure |
| 40002 | Transaction integrity constraint violation |
| 40003 | Statement completion unknown |
| SQLCLASS 42 (Syntax Error or Access Violation) | |
| 42000 | Syntax error or access violation |
| 42702 | Ambiguous column reference |
| 42725 | Ambiguous function reference |
| 42818 | The operands of an operator or function are not compatible |
| 42S01 | Base table or view already exists |
| 42S02 | Base table or view not found |
| 42S11 | Index already exists |
| 42S12 | Index not found |

| SQLSTATE | Zugewiesene Meldung |
|--|---|
| 42S21 | Column already exists |
| 42S22 | Column not found |
| SQLCLASS 44 (With Check Option Violation) | |
| 44000 | WITH CHECK OPTION Violation |
| SQLCLASS 45 (Unhandled User-defined Exception) | |
| 45000 | Unhandled user-defined exception |
| SQLCLASS 54 (Program Limit Exceeded) | |
| 54000 | Program limit exceeded |
| 54001 | Statement too complex |
| 54011 | Too many columns |
| 54023 | Too many arguments |
| SQLCLASS HY (CLI-specific Condition) | |
| HY000 | CLI-specific condition |
| HY001 | Memory allocation error |
| HY003 | Invalid data type in application descriptor |
| HY004 | Invalid data type |
| HY007 | Associated statement is not prepared |
| HY008 | Operation canceled |
| HY009 | Invalid use of null pointer |
| HY010 | Function sequence error |
| HY011 | Attribute cannot be set now |
| HY012 | Invalid transaction operation code |
| HY013 | Memory management error |
| HY014 | Limit on the number of handles exceeded |
| HY015 | No cursor name available |
| HY016 | Cannot modify an implementation row descriptor |
| HY017 | Invalid use of an automatically allocated descriptor handle |
| HY018 | Server declined the cancellation request |
| HY019 | Non-string data cannot be sent in pieces |
| HY020 | Attempt to concatenate a null value |
| HY021 | Inconsistent descriptor information |

| SQLSTATE | Zugewiesene Meldung |
|-------------------------------------|--|
| HY024 | Invalid attribute value |
| HY055 | Non-string data cannot be used with string routine |
| HY090 | Invalid string length or buffer length |
| HY091 | Invalid descriptor field identifier |
| HY092 | Invalid attribute identifier |
| HY095 | Invalid Function ID specified |
| HY096 | Invalid information type |
| HY097 | Column type out of range |
| HY098 | Scope out of range |
| HY099 | Nullable type out of range |
| HY100 | Uniqueness option type out of range |
| HY101 | Accuracy option type out of range |
| HY103 | Invalid retrieval code |
| HY104 | Invalid Length/Precision value |
| HY105 | Invalid parameter type |
| HY106 | Invalid fetch orientation |
| HY107 | Row value out of range |
| HY109 | Invalid cursor position |
| HY110 | Invalid driver completion |
| HY111 | Invalid bookmark value |
| HYC00 | Optional feature not implemented |
| HYT00 | Timeout expired |
| HYT01 | Connection timeout expired |
| SQLCLASS XX (Internal Error) | |
| XX000 | Internal error |
| XX001 | Data corrupted |
| XX002 | Index corrupted |

SQLCODE und GDSCODE Fehlercodes und Beschreibungen

Diese Tabelle zeigt die SQLCODE-Gruppen, die numerischen und symbolischen Werte für die GDSCODE-Fehler und deren Meldungen.

Anmerkung

SQLCODE SQLCODE wurde viele Jahre verwendet und wird als veraltet angesehen. Der Support für SQLCODE wird vermutlich in zukünftigen Versionen entfernt.

Tabelle B.2. SQLCODE und GDSCODE Fehlercodes und Meldungen (1)

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|------------------------|--|
| 101 | 335544366 | Segment | Segment buffer length shorter than expected |
| 100 | 335544338 | from_no_match | No match for first value expression |
| 100 | 335544354 | no_record | Invalid database key |
| 100 | 335544367 | segstr_eof | Attempted retrieval of more segments than exist |
| 100 | 335544374 | stream_eof | Attempt to fetch past the last record in a record stream |
| 0 | 335741039 | gfix_opt_SQL_dialect | -sql_dialect set database dialect n |
| 0 | 335544875 | bad_debug_format | Bad debug info format |
| -84 | 335544554 | nonsql_security_rel | Table/procedure has non-SQL security class defined |
| -84 | 335544555 | nonsql_security fld | Column has non-SQL security class defined |
| -84 | 335544668 | dsql_procedure_use_err | Procedure @1 does not return any values |
| -85 | 335544747 | username_too_long | The username entered is too long. Maximum length is 31 bytes |
| -85 | 335544748 | password_too_long | The password specified is too long. Maximum length is @1 bytes |
| -85 | 335544749 | username_required | A username is required for this operation |
| -85 | 335544750 | password_required | A password is required for this operation |
| -85 | 335544751 | bad_protocol | The network protocol specified is invalid |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|--------------------------------|---|
| -85 | 335544752 | dup_username_found | A duplicate user name was found in the security database |
| -85 | 335544753 | username_not_found | The user name specified was not found in the security database |
| -85 | 335544754 | error_adding_sec_record | An error occurred while attempting to add the user |
| -85 | 335544755 | error_modifying_sec_record | An error occurred while attempting to modify the user record |
| -85 | 335544756 | error_deleting_sec_record | An error occurred while attempting to delete the user record |
| -85 | 335544757 | error_updating_sec_db | An error occurred while updating the security database |
| -103 | 335544571 | dsql_constant_err | Data type for constant unknown |
| -104 | 336003075 | dsql_transitional_numeric | Precision 10 to 18 changed from DOUBLE PRECISION in SQL dialect 1 to 64-bit scaled integer in SQL dialect 3 |
| -104 | 336003077 | sql_db_dialect_dtype_unsupport | Database SQL dialect @1 does not support reference to @2 datatype |
| -104 | 336003087 | dsql_invalid_label | Label @1 @2 in the current scope |
| -104 | 336003088 | dsql_datatypes_not_comparable | Datatypes @1 are not comparable in expression @2 |
| -104 | 335544343 | invalid_blr | Invalid request BLR at offset @1 |
| -104 | 335544390 | syntaxerr | BLR syntax error: expected @1 at offset @2, encountered @3 |
| -104 | 335544425 | ctxinuse | Context already in use (BLR error) |
| -104 | 335544426 | ctxnotdef | Context not defined (BLR error) |
| -104 | 335544429 | badparnum | Bad parameter number |
| -104 | 335544440 | bad_msg_vec | - |
| -104 | 335544456 | invalid_sdl | Invalid slice description language at offset @1 |
| -104 | 335544570 | dsql_command_err | Invalid command |
| -104 | 335544579 | dsql_internal_err | Internal error |
| -104 | 335544590 | dsql_dup_option | Option specified more than once |
| -104 | 335544591 | dsql_tran_err | Unknown transaction option |

Fehlercodes und Meldungen

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|--------------------------------|--|
| -104 | 335544592 | dsql_invalid_array | Invalid array reference |
| -104 | 335544608 | command_end_err | Unexpected end of command |
| -104 | 335544612 | token_err | Token unknown |
| -104 | 335544634 | dsql_token_unk_err | Token unknown - line @1, column @2 |
| -104 | 335544709 | dsql_agg_ref_err | Invalid aggregate reference |
| -104 | 335544714 | invalid_array_id | Invalid blob id |
| -104 | 335544730 | cse_not_supported | Client/Server Express not supported in this release |
| -104 | 335544743 | token_too_long | Token size exceeds limit |
| -104 | 335544763 | invalid_string_constant | A string constant is delimited by double quotes |
| -104 | 335544764 | transitional_date | DATE must be changed to TIMESTAMP |
| -104 | 335544796 | sql_dialect_datatype_unsupport | Client SQL dialect @1 does not support reference to @2 datatype |
| -104 | 335544798 | depend_on_uncommitted_rel | You created an indirect dependency on uncommitted metadata. You must roll back the current transaction |
| -104 | 335544821 | dsql_column_pos_err | Invalid column position used in the @1 clause |
| -104 | 335544822 | dsql_agg_where_err | Cannot use an aggregate function in a WHERE clause, use HAVING instead |
| -104 | 335544823 | dsql_agg_group_err | Cannot use an aggregate function in a GROUP BY clause |
| -104 | 335544824 | dsql_agg_column_err | Invalid expression in the @1 (not contained in either an aggregate function or the GROUP BY clause) |
| -104 | 335544825 | dsql_agg_having_err | Invalid expression in the @1 (neither an aggregate function nor a part of the GROUP BY clause) |
| -104 | 335544826 | dsql_agg_nested_err | Nested aggregate functions are not allowed |
| -104 | 335544849 | malformed_string | Malformed string |
| -104 | 335544851 | command_end_err2 | Unexpected end of command- line @1, column @2 |
| -104 | 336397215 | dsql_max_sort_items | Cannot sort on more than 255 items |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|---------------------------------|---|
| -104 | 336397216 | dsql_max_group_items | Cannot group on more than 255 items |
| -104 | 336397217 | dsql_conflicting_sort_field | Cannot include the same field (@1.@2) twice in the ORDER BY clause with conflicting sorting options |
| -104 | 336397218 | dsql_derived_table_more_columns | Column list from derived table @1 has more columns than the number of items in its SELECT statement |
| -104 | 336397219 | dsql_derived_table_less_columns | Column list from derived table @1 has less columns than the number of items in its SELECT statement |
| -104 | 336397220 | dsql_derived_field_unnamed | No column name specified for column number @1 in derived table @2 |
| -104 | 336397221 | dsql_derived_field_dup_name | Column @1 was specified multiple times for derived table @2 |
| -104 | 336397222 | dsql_derived_alias_select | Internal dsql error: alias type expected by pass1_expand_select_node |
| -104 | 336397223 | dsql_derived_alias_field | Internal dsql error: alias type expected by pass1_field |
| -104 | 336397224 | dsql_auto_field_bad_pos | Internal dsql error: column position out of range in pass1_union_auto_cast |
| -104 | 336397225 | dsql_cte_wrong_reference | Recursive CTE member (@1) can refer itself only in FROM clause |
| -104 | 336397226 | dsql_cte_cycle | CTE '@1' has cyclic dependencies |
| -104 | 336397227 | dsql_cte_outer_join | Recursive member of CTE can't be member of an outer join |
| -104 | 336397228 | dsql_cte_mult_references | Recursive member of CTE can't reference itself more than once |
| -104 | 336397229 | dsql_cte_not_a_union | Recursive CTE (@1) must be an UNION |
| -104 | 336397230 | dsql_cte_nonrecurs_after_recurs | CTE '@1' defined non-recursive member after recursive |
| -104 | 336397231 | dsql_cte_wrong_clause | Recursive member of CTE '@1' has @2 clause |
| -104 | 336397232 | dsql_cte_union_all | Recursive members of CTE (@1) must be linked with another members via UNION ALL |
| -104 | 336397233 | dsql_cte_miss_nonrecursive | Non-recursive member is missing in CTE '@1' |

Fehlercodes und Meldungen

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|-------------------------------|--|
| -104 | 336397234 | dsql_cte_nested_with | WITH clause can't be nested |
| -104 | 336397235 | dsql_col_more_than_once_using | Column @1 appears more than once in USING clause |
| -104 | 336397237 | dsql_cte_not_used | CTE "@1" is not used in query |
| -105 | 335544702 | like_escape_invalid | Invalid ESCAPE sequence |
| -105 | 335544789 | extract_input_mismatch | Specified EXTRACT part does not exist in input datatype |
| -150 | 335544360 | read_only_rel | Attempted update of read-only table |
| -150 | 335544362 | read_only_view | Cannot update read-only view @1 |
| -150 | 335544446 | non_updatable | Not updatable |
| -150 | 335544546 | constant_on_view | Cannot define constraints on views |
| -151 | 335544359 | read_only_field | Attempted update of read - only column |
| -155 | 335544658 | dsql_base_table | @1 is not a valid base table of the specified view |
| -157 | 335544598 | specify_field_err | Must specify column name for view select expression |
| -158 | 335544599 | num_field_err | Number of columns does not match select list |
| -162 | 335544685 | no_dbkey | Dbkey not available for multi - table views |
| -170 | 335544512 | premisat | Input parameter mismatch for procedure @1 |
| -170 | 335544619 | extern_func_err | External functions cannot have morethan 10 parametrs |
| -170 | 335544850 | prc_out_param_mismatch | Output parameter mismatch for procedure @1 |
| -171 | 335544439 | funmismat | Function @1 could not be matched |
| -171 | 335544458 | invalid_dimension | Column not array or invalid dimensions (expected @1, encountered @2) |
| -171 | 335544618 | return_mode_err | Return mode by value not allowed for this data type |
| -171 | 335544873 | array_max_dimensions | Array data type can use up to @1 dimensions |
| -172 | 335544438 | funnotdef | Function @1 is not defined |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|---------------------------|---|
| -203 | 335544708 | dyn_fld_ambiguous | Ambiguous column reference |
| -204 | 336003085 | dsql_ambiguous_field_name | Ambiguous field name between @1 and @2 |
| -204 | 335544463 | gennotdef | Generator @1 is not defined |
| -204 | 335544502 | stream_not_defined | Reference to invalid stream number |
| -204 | 335544509 | charset_not_found | CHARACTER SET @1 is not defined |
| -204 | 335544511 | prcnnotdef | Procedure @1 is not defined |
| -204 | 335544515 | codnotdef | Status code @1 unknown |
| -204 | 335544516 | xcpnotdef | Exception @1 not defined |
| -204 | 335544532 | ref_cnstrnt_notfound | Name of Referential Constraint not defined in constraints table |
| -204 | 335544551 | grant_obj_notfound | Could not find table/procedure for GRANT |
| -204 | 335544568 | text_subtype | Implementation of text subtype @1 not located |
| -204 | 335544573 | dsql_datatype_err | Data type unknown |
| -204 | 335544580 | dsql_relation_err | Table unknown |
| -204 | 335544581 | dsql_procedure_err | Procedure unknown |
| -204 | 335544588 | collation_not_found | COLLATION @1 for CHARACTER SET @2 is not defined |
| -204 | 335544589 | collation_not_for_charset | COLLATION @1 is not valid for specified CHARACTER SET |
| -204 | 335544595 | dsql_trigger_err | Trigger unknown |
| -204 | 335544620 | alias_conflict_err | Alias @1 conflicts with an alias in the same statement |
| -204 | 335544621 | procedure_conflict_error | Alias @1 conflicts with a procedure in the same statement |
| -204 | 335544622 | relation_conflict_err | Alias @1 conflicts with a table in the same statement |
| -204 | 335544635 | dsql_no_relation_alias | There is no alias or table named @1 at this scope level |
| -204 | 335544636 | indexname | There is no index @1 for table @2 |
| -204 | 335544640 | collation_requires_text | Invalid use of CHARACTER SET or COLLATE |

Fehlercodes und Meldungen

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|--------------------------|--|
| -204 | 335544662 | dsql_blob_type_unknown | BLOB SUB_TYPE @1 is not defined |
| -204 | 335544759 | bad_default_value | Can not define a not null column with NULL as default value |
| -204 | 335544760 | invalid_clause | Invalid clause - '@1' |
| -204 | 335544800 | too_many_contexts | Too many Contexts of Relation/Procedure/Views. Maximum allowed is 255 |
| -204 | 335544817 | bad_limit_param | Invalid parameter to FIRST. Only integers >= 0 are allowed |
| -204 | 335544818 | bad_skip_param | Invalid parameter to SKIP. Only integers >= 0 are allowed |
| -204 | 335544837 | bad_substring_offset | Invalid offset parameter @1 to SUBSTRING. Only positive integers are allowed |
| -204 | 335544853 | bad_substring_length | Invalid length parameter @1 to SUBSTRING. Negative integers are not allowed |
| -204 | 335544854 | charset_not_installed | CHARACTER SET @1 is not installed |
| -204 | 335544855 | collation_not_installed | COLLATION @1 for CHARACTER SET @2 is not installed |
| -204 | 335544867 | subtype_for_internal_use | Blob sub_types bigger than 1 (text) are for internal use only |
| -205 | 335544396 | fldnotdef | Column @1 is not defined in table @2 |
| -205 | 335544552 | grant_fld_notfound | Could not find column for GRANT |
| -205 | 335544883 | fldnotdef2 | Column @1 is not defined in procedure @2 |
| -206 | 335544578 | dsql_field_err | Column unknown |
| -206 | 335544587 | dsql_blob_err | Column is not a BLOB |
| -206 | 335544596 | dsql_subselect_err | Subselect illegal in this context |
| -206 | 336397208 | dsql_line_col_error | At line @1, column @2 |
| -206 | 336397209 | dsql_unknown_pos | At unknown line and column |
| -206 | 336397210 | dsql_no_dup_name | Column @1 cannot be repeated in @2 statement |
| -208 | 335544617 | order_by_err | Invalid ORDER BY clause |
| -219 | 335544395 | relnotdef | Table @1 is not defined |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|---------------------|---|
| -219 | 335544872 | domnotdef | Domain @1 is not defined |
| -230 | 335544487 | walw_err | WAL Writer error |
| -231 | 335544488 | logh_small | Log file header of @1 too small |
| -232 | 335544489 | logh_inv_version | Invalid version of log file @1 |
| -233 | 335544490 | logh_open_flag | Log file @1 not latest in the chain but open flag still set |
| -234 | 335544491 | logh_open_flag2 | Log file @1 not closed properly; database recovery may be required |
| -235 | 335544492 | logh_diff_dbname | Database name in the log file @1 is different |
| -236 | 335544493 | logf_unexpected_eof | Unexpected end of log file @1 at offset @2 |
| -237 | 335544494 | logr_incomplete | Incomplete log record at offset @1 in log file @2 |
| -238 | 335544495 | logr_header_small2 | Log record header too small at offset @1 in log file @ |
| -239 | 335544496 | logb_small | Log block too small at offset @1 in log file @2 |
| -239 | 335544691 | cache_too_small | Insufficient memory to allocate page buffer cache |
| -239 | 335544693 | log_too_small | Log size too small |
| -239 | 335544694 | partition_too_small | Log partition size too small |
| -243 | 335544500 | no_wal | Database does not use Write-ahead Log |
| -257 | 335544566 | start_cm_for_wal | WAL defined; Cache Manager must be started first |
| -260 | 335544690 | cache_redef | Cache redefined |
| -260 | 335544692 | log_redef | Log redefined |
| -261 | 335544695 | partition_not_supp | Partitions not supported in series of log file specification |
| -261 | 335544696 | log_length_spec | Total length of a partitioned log must be specified |
| -281 | 335544637 | no_stream_plan | Table @1 is not referenced in plan |
| -282 | 335544638 | stream_twice | Table @1 is referenced more than once in plan; use aliases to distinguish |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|-------------------------------|--|
| -282 | 335544643 | dsql_self_join | The table @1 is referenced twice; use aliases to differentiate |
| -282 | 335544659 | duplicate_base_table | Table @1 is referenced twice in view; use an alias to distinguish |
| -282 | 335544660 | view_alias | View @1 has more than one base table; use aliases to distinguish |
| -282 | 335544710 | complex_view | Navigational stream @1 references a view with more than one base table |
| -283 | 335544639 | stream_not_found | Table @1 is referenced in the plan but not the from list |
| -284 | 335544642 | index_unused | Index @1 cannot be used in the specified plan |
| -291 | 335544531 | primary_key_notnull | Column used in a PRIMARY constraint must be NOT NULL |
| -292 | 335544534 | ref_cnstrnt_update | Cannot update constraints (RDB \$REF_CONSTRAINTS) |
| -293 | 335544535 | check_cnstrnt_update | Cannot update constraints (RDB \$CHECK_CONSTRAINTS) |
| -294 | 335544536 | check_cnstrnt_del | Cannot delete CHECK constraint entry (RDB\$CHECK_CONSTRAINTS) |
| -295 | 335544545 | rel_cnstrnt_update | Cannot update constraints (RDB \$RELATION_CONSTRAINTS) |
| -296 | 335544547 | invld_cnstrnt_type | Internal gds software consistency check (invalid RDB\$CONSTRAINT_TYPE) |
| -297 | 335544558 | check_constraint | Operation violates check constraint @1 on view or table @2 |
| -313 | 336003099 | upd_ins_doesnt_match_pk | UPDATE OR INSERT field list does not match primary key of table @1 |
| -313 | 336003100 | upd_ins_doesnt_match_matching | UPDATE OR INSERT field list does not match MATCHING clause |
| -313 | 335544669 | dsql_count_mismatch | Count of column list and variable list do not match |
| -314 | 335544565 | transliteration_failed | Cannot transliterate character between character sets |
| -315 | 336068815 | dyn_dtype_invalid | Cannot change datatype for column @1. Changing datatype is not supported for BLOB or ARRAY columns |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|-----------------------|--|
| -383 | 336068814 | dyn_dependency_exists | Column @1 from table @2 is referenced in @3 |
| -401 | 335544647 | invalid_operator | Invalid comparison operator for find operation |
| -402 | 335544368 | segstr_no_op | Attempted invalid operation on a BLOB |
| -402 | 335544414 | blobnotsup | BLOB and array data types are not supported for @1 operation |
| -402 | 335544427 | datnotsup | Data operation not supported |
| -406 | 335544457 | out_of_bounds | Subscript out of bounds |
| -407 | 335544435 | nullsegkey | Null segment of UNIQUE KEY |
| -413 | 335544334 | convert_error | Conversion error from string "@1" |

Tabelle B.3. SQLCODE und GDSCODE Fehlercodes und Meldungen (2)

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|---------------------------|---|
| -413 | 335544454 | nofilter | Filter not found to convert type @1 to type @2 |
| -413 | 335544860 | blob_convert_error | Unsupported conversion to target type BLOB (subtype @1) |
| -413 | 335544861 | array_convert_error | Unsupported conversion to target type ARRAY |
| -501 | 335544577 | dsql_cursor_close_err | Attempt to reclose a closed cursor |
| -502 | 336003090 | dsql_cursor_redefined | Statement already has a cursor @1 assigned |
| -502 | 336003091 | dsql_cursor_not_found | Cursor @1 is not found in the current context |
| -502 | 336003092 | dsql_cursor_exists | Cursor @1 already exists in the current context |
| -502 | 336003093 | dsql_cursor_rel_ambiguous | Relation @1 is ambiguous in cursor @2 |
| -502 | 336003094 | dsql_cursor_rel_not_found | Relation @1 is not found in cursor @2 |
| -502 | 336003095 | dsql_cursor_not_open | Cursor is not open |
| -502 | 335544574 | dsql_decl_err | Invalid cursor declaration |
| -502 | 335544576 | dsql_cursor_open_err | Attempt to reopen an open cursor |

Fehlercodes und Meldungen

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|---------------------------------|--|
| -504 | 336003089 | dsql_cursor_invalid | Empty cursor name is not allowed |
| -504 | 335544572 | dsql_cursor_err | Invalid cursor reference |
| -508 | 335544348 | no_cur_rec | No current record for fetch operation |
| -510 | 335544575 | dsql_cursor_update_err | Cursor @1 is not updatable |
| -518 | 335544582 | dsql_request_err | Request unknown |
| -519 | 335544688 | dsql_open_cursor_request | The prepare statement identifies a prepare statement with an open cursor |
| -530 | 335544466 | foreign_key | Violation of FOREIGN KEY constraint "@1" on table "@2" |
| -530 | 335544838 | foreign_key_target_doesnt_exist | Foreign key reference target does not exist |
| -530 | 335544839 | foreign_key_references_present | Foreign key references are present for the record |
| -531 | 335544597 | dsql_crdb_prepare_err | Cannot prepare a CREATE DATABASE/SCHEMA statement |
| -532 | 335544469 | trans_invalid | Transaction marked invalid by I/O error |
| -551 | 335544352 | no_priv | No permission for @1 access to @2 @3 |
| -551 | 335544790 | insufficient_svc_privileges | Service @1 requires SYSDBA permissions. Reattach to the Service Manager using the SYSDBA account |
| -552 | 335544550 | not_rel_owner | Only the owner of a table may reassign ownership |
| -552 | 335544553 | grant_nopriv | User does not have GRANT privileges for operation |
| -552 | 335544707 | grant_nopriv_on_base | User does not have GRANT privileges on base table/view for operation |
| -553 | 335544529 | existing_priv_mod | Cannot modify an existing user privilege |
| -595 | 335544645 | stream_crack | The current position is on a crack |
| -596 | 335544644 | stream_bof | Illegal operation when at beginning of stream |
| -597 | 335544632 | dsql_file_length_err | Preceding file did not specify length, so @1 must include starting page number |
| -598 | 335544633 | dsql_shadow_number_err | Shadow number must be a positive integer |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|-------------------------------|--|
| -599 | 335544607 | node_err | Gen.c: node not supported |
| -599 | 335544625 | node_name_err | A node name is not permitted in a secondary, shadow, cache or log file name |
| -600 | 335544680 | crrp_data_err | Sort error: corruption in data structure |
| -601 | 335544646 | db_or_file_exists | Database or file exists |
| -604 | 335544593 | dsql_max_arr_dim_exceeded | Array declared with too many dimensions |
| -604 | 335544594 | dsql_arr_range_error | Illegal array dimension range |
| -605 | 335544682 | dsql_field_ref | Inappropriate self-reference of column |
| -607 | 336003074 | dsql_dbkey_from_non_table | Cannot SELECT RDB\$DB_KEY from a stored procedure |
| -607 | 336003086 | dsql_udf_return_pos_err | External function should have return position between 1 and @1 |
| -607 | 336003096 | dsql_type_not_supp_ext_tab | Data type @1 is not supported for EXTERNAL TABLES. Relation '@2', field '@3' |
| -607 | 335544351 | no_meta_update | Unsuccessful metadata update |
| -607 | 335544549 | systrig_update | Cannot modify or erase a system trigger |
| -607 | 335544657 | dsql_no_blob_array | Array/BLOB/DATE data types not allowed in arithmetic |
| -607 | 335544746 | reftable_requires_pk | "REFERENCES table" without "(column)" requires PRIMARY KEY on referenced table |
| -607 | 335544815 | generator_name | GENERATOR @1 |
| -607 | 335544816 | udf_name | UDF @1 |
| -607 | 335544858 | must_have_phys_field | Can't have relation with only computed fields or constraints |
| -607 | 336397206 | dsql_table_not_found | Table @1 does not exist |
| -607 | 336397207 | dsql_view_not_found | View @1 does not exist |
| -607 | 336397212 | dsql_no_array_computed | Array and BLOB data types not allowed in computed field |
| -607 | 336397214 | dsql_only_can_subscript_array | Scalar operator used on field @1 which is not an array |
| -612 | 336068812 | dyn_domain_name_exists | Cannot rename domain @1 to @2. A domain with that name already exists |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|--------------------------|--|
| -612 | 336068813 | dyn_field_name_exists | Cannot rename column @1 to @2.A column with that name already exists in table @3 |
| -615 | 335544475 | relation_lock | Lock on table @1 conflicts with existing lock |
| -615 | 335544476 | record_lock | Requested record lock conflicts with existing lock |
| -615 | 335544507 | range_in_use | Refresh range number @1 already in use |
| -616 | 335544530 | primary_key_ref | Cannot delete PRIMARY KEY being used in FOREIGN KEY definition |
| -616 | 335544539 | integ_index_del | Cannot delete index used by an Integrity Constraint |
| -616 | 335544540 | integ_index_mod | Cannot modify index used by an Integrity Constraint |
| -616 | 335544541 | check_trig_del | Cannot delete trigger used by a CHECK Constraint |
| -616 | 335544543 | cnstrnt fld_del | Cannot delete column being used in an Integrity Constraint |
| -616 | 335544630 | dependency | There are @1 dependencies |
| -616 | 335544674 | del_last_field | Last column in a table cannot be deleted |
| -616 | 335544728 | integ_index_deactivate | Cannot deactivate index used by an integrity constraint |
| -616 | 335544729 | integ_deactivate_primary | Cannot deactivate index used by a PRIMARY/UNIQUE constraint |
| -617 | 335544542 | check_trig_update | Cannot update trigger used by a CHECK Constraint |
| -617 | 335544544 | cnstrnt fld_rename | Cannot rename column being used in an Integrity Constraint |
| -618 | 335544537 | integ_index_seg_del | Cannot delete index segment used by an Integrity Constraint |
| -618 | 335544538 | integ_index_seg_mod | Cannot update index segment used by an Integrity Constraint |
| -625 | 335544347 | not_valid | Validation error for column @1, value "@2" |
| -625 | 335544879 | not_valid_for_var | Validation error for variable @1, value "@2" |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|---------------------------|---|
| -625 | 335544880 | not_valid_for | Validation error for @1, value "@2" |
| -637 | 335544664 | dsql_duplicate_spec | Duplicate specification of @1- not supported |
| -637 | 336397213 | dsql_implicit_domain_name | Implicit domain name @1 not allowed in user created domain |
| -660 | 336003098 | primary_key_required | Primary key required on table @1 |
| -660 | 335544533 | foreign_key_notfound | Non-existent PRIMARY or UNIQUE KEY specified for FOREIGN KEY |
| -660 | 335544628 | idx_create_err | Cannot create index @1 |
| -663 | 335544624 | idx_seg_err | Segment count of 0 defined for index @1 |
| -663 | 335544631 | idx_key_err | Too many keys defined for index @1 |
| -663 | 335544672 | key_field_err | Too few key columns found for index @1 (incorrect column name?) |
| -664 | 335544434 | keytoobig | Key size exceeds implementation restriction for index "@1" |
| -677 | 335544445 | ext_err | @1 extension error |
| -685 | 335544465 | bad_segstr_type | Invalid BLOB type for operation |
| -685 | 335544670 | blob_idx_err | Attempt to index BLOB column in index @1 |
| -685 | 335544671 | array_idx_err | Attempt to index array column in index @1 |
| -689 | 335544403 | badpagtyp | Page @1 is of wrong type (expected @2, found @3) |
| -689 | 335544650 | page_type_err | Wrong page type |
| -690 | 335544679 | no_segments_err | Segments not allowed in expression index @1 |
| -691 | 335544681 | rec_size_err | New record size of @1 bytes is too big |
| -692 | 335544477 | max_idx | Maximum indexes per table (@1) exceeded |
| -693 | 335544663 | req_max_clones_exceeded | Too many concurrent executions of the same request |
| -694 | 335544684 | no_field_access | Cannot access column @1 in view @2 |
| -802 | 335544321 | arith_except | Arithmetic exception, numeric overflow, or string truncation |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|--------------------------------|--|
| -802 | 335544836 | concat_overflow | Concatenation overflow. Resulting string cannot exceed 32K in length |
| -803 | 335544349 | no_dup | Attempt to store duplicate value (visible to active transactions) in unique index "@1" |
| -803 | 335544665 | unique_key_violation | Violation of PRIMARY or UNIQUE KEY constraint "@1" on table "@2" |
| -804 | 336003097 | dsql_feature_not_supported_ods | Feature not supported on ODS version older than @1.@2 |
| -804 | 335544380 | wronumarg | Wrong number of arguments on call |
| -804 | 335544583 | dsql_sqlda_err | SQLDA missing or incorrect version, or incorrect number/type of variables |
| -804 | 335544584 | dsql_var_count_err | Count of read - write columns does not equal count of values |
| -804 | 335544586 | dsql_function_err | Function unknown |
| -804 | 335544713 | dsql_sqlda_value_err | Incorrect values within SQLDA structure |
| -804 | 336397205 | dsql_too_old_ods | ODS versions before ODS@1 are not supported |
| -806 | 335544600 | col_name_err | Only simple column names permitted for VIEW WITH CHECK OPTION |
| -807 | 335544601 | where_err | No WHERE clause for VIEW WITH CHECK OPTION |
| -808 | 335544602 | table_view_err | Only one table allowed for VIEW WITH CHECK OPTION |
| -809 | 335544603 | distinct_err | DISTINCT, GROUP or HAVING not permitted for VIEW WITH CHECK OPTION |
| -810 | 335544605 | subquery_err | No subqueries permitted for VIEW WITH CHECK OPTION |
| -811 | 335544652 | sing_select_err | Multiple rows in singleton select |
| -816 | 335544651 | ext_readonly_err | Cannot insert because the file is readonly or is on a read only medium |
| -816 | 335544715 | extfile_uns_op | Operation not supported for EXTERNAL FILE table @1 |
| -817 | 336003079 | isc_sql_dialect_conflict_num | DB dialect @1 and client dialect @2 conflict with respect to numeric precision @3 |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|----------------------------------|--|
| -817 | 336003101 | upd_ins_with_complex_view | UPDATE OR INSERT without MAT-CHING could not be used with views based on more than one table |
| -817 | 336003102 | dsql_incompatible_trigger_type | Incompatible trigger type |
| -817 | 336003103 | dsql_db_trigger_type_cant_change | Database trigger type can't be changed |
| -817 | 335544361 | read_only_trans | Attempted update during read - only transaction |
| -817 | 335544371 | segstr_no_write | Attempted write to read-only BLOB |
| -817 | 335544444 | read_only | Operation not supported |
| -817 | 335544765 | read_only_database | Attempted update on read - only database |
| -817 | 335544766 | must_be_dialect_2_and_up | SQL dialect @1 is not supported in this database |
| -817 | 335544793 | ddl_not_allowed_by_db_sql_dial | Metadata update statement is not allowed by the current database SQL dialect @1 |
| -820 | 335544356 | obsolete_metadata | Metadata is obsolete |
| -820 | 335544379 | wrong_ods | Unsupported on - disk structure for file @1; found @2.@3, support @4.@5 |
| -820 | 335544437 | wrodynver | Wrong DYN version |
| -820 | 335544467 | high_minor | Minor version too high found @1 expected @2 |
| -820 | 335544881 | need_difference | Difference file name should be set explicitly for database on raw device |
| -823 | 335544473 | invalid_bookmark | Invalid bookmark handle |
| -824 | 335544474 | bad_lock_level | Invalid lock level @1 |
| -825 | 335544519 | bad_lock_handle | Invalid lock handle |
| -826 | 335544585 | dsql_stmt_handle | Invalid statement handle |
| -827 | 335544655 | invalid_direction | Invalid direction for find operation |
| -827 | 335544718 | invalid_key | Invalid key for find operation |
| -828 | 335544678 | inval_key_posn | Invalid key position |
| -829 | 336068816 | dyn_char fld_too_small | New size specified for column @1 must be at least @2 characters |
| -829 | 336068817 | dyn_invalid_dtype_conversion | Cannot change datatype for @1. Conversion from base type @2 to @3 is not supported |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|-------------------------|--|
| -829 | 336068818 | dyn_dtype_conv_invalid | Cannot change datatype for column @1 from a character type to a non-character type |
| -829 | 336068829 | max_coll_per_charset | Maximum number of collations per character set exceeded |
| -829 | 336068830 | invalid_coll_attr | Invalid collation attributes |
| -829 | 336068852 | dyn_scale_too_big | New scale specified for column @1 must be at most @2 |
| -829 | 336068853 | dyn_precision_too_small | New precision specified for column @1 must be at least @2 |
| -829 | 335544616 | field_ref_err | Invalid column reference |
| -830 | 335544615 | field_aggregate_err | Column used with aggregate |
| -831 | 335544548 | primary_key_exists | Attempt to define a second PRIMARY KEY for the same table |
| -832 | 335544604 | key_field_count_err | FOREIGN KEY column count does not match PRIMARY KEY |
| -833 | 335544606 | expression_eval_err | Expression evaluation not supported |
| -833 | 335544810 | date_range_exceeded | Value exceeds the range for valid dates |
| -834 | 335544508 | range_not_found | Refresh range number @1 not found |
| -835 | 335544649 | bad_checksum | Bad checksum |
| -836 | 335544517 | except | Exception @1 |
| -836 | 335544848 | except2 | Exception @1 |
| -837 | 335544518 | cache_restart | Restart shared cache manager |
| -838 | 335544560 | shutwarn | Database @1 shutdown in @2 seconds |
| -841 | 335544677 | version_err | Too many versions |
| -842 | 335544697 | precision_err | Precision must be from 1 to 18 |
| -842 | 335544698 | scale_nogt | Scale must be between zero and precision |
| -842 | 335544699 | expec_short | Short integer expected |
| -842 | 335544700 | expec_long | Long integer expected |
| -842 | 335544701 | expec_ushort | Unsigned short integer expected |
| -842 | 335544712 | expec_positive | Positive value expected |
| -901 | 335740929 | gfix_db_name | Database file name (@1) already given |

Fehlercodes und Meldungen

| SQL- CO- DE | GDSCODE | Symbol | Meldung |
|----------------------------|----------------|---------------------------|---|
| -901 | 336330753 | gbak_unknown_switch | Found unknown switch |
| -901 | 336920577 | gstat_unknown_switch | Found unknown switch |
| -901 | 336986113 | fbsvcmgr_bad_am | Wrong value for access mode |
| -901 | 335740930 | gfix_invalid_sw | Invalid switch @1 |
| -901 | 335544322 | bad_dbkey | Invalid database key |
| -901 | 336986114 | fbsvcmgr_bad_wm | Wrong value for write mode |
| -901 | 336330754 | gbak_page_size_missing | Page size parameter missing |
| -901 | 336920578 | gstat_retry | Please retry, giving a database name |
| -901 | 336986115 | fbsvcmgr_bad_rs | Wrong value for reserve space |
| -901 | 336920579 | gstat_wrong_ods | Wrong ODS version, expected @1, encountered @2 |
| -901 | 336330755 | gbak_page_size_toobig | Page size specified (@1) greater than limit (16384 bytes) |
| -901 | 335740932 | gfix_incmp_sw | Incompatible switch combination |
| -901 | 336920580 | gstat_unexpected_eof | Unexpected end of database file |
| -901 | 336330756 | gbak_redir_output_missing | Redirect location for output is not speci- fied |
| -901 | 336986116 | fbsvcmgr_info_err | Unknown tag (@1) in info_svr_db_info block after isc_svc_query() |
| -901 | 335740933 | gfix_replay_req | Replay log pathname required |
| -901 | 336330757 | gbak_switches_conflict | Conflicting switches for backup/restore |
| -901 | 336986117 | fbsvcmgr_query_err | Unknown tag (@1) in isc_svc_query() results |
| -901 | 335544326 | bad_dpb_form | Unrecognized database parameter block |
| -901 | 335740934 | gfix_pgbuf_req | Number of page buffers for cache requi- red |
| -901 | 336986118 | fbsvcmgr_switch_unknown | Unknown switch "@1" |
| -901 | 336330758 | gbak_unknown_device | Device type @1 not known |
| -901 | 335544327 | bad_req_handle | Invalid request handle |
| -901 | 335740935 | gfix_val_req | Numeric value required |
| -901 | 336330759 | gbak_no_protection | Protection is not there yet |
| -901 | 335544328 | bad_segstr_handle | Invalid BLOB handle |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|----------------------------|--|
| -901 | 335740936 | gfix_pval_req | Positive numeric value required |
| -901 | 336330760 | gbak_page_size_not_allowed | Page size is allowed only on restore or create |

Tabelle B.4. SQLCODE und GDSCODE Fehlercodes und Meldungen (3)

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|------------------------|---|
| -901 | 335544329 | bad_segstr_id | Invalid BLOB ID |
| -901 | 335740937 | gfix_trn_req | Number of transactions per sweep required |
| -901 | 336330761 | gbak_multi_source_dest | Multiple sources or destinations specified |
| -901 | 335544330 | bad_tpb_content | Invalid parameter in transaction parameter block |
| -901 | 336330762 | gbak_filename_missing | Requires both input and output filenames |
| -901 | 335544331 | bad_tpb_form | Invalid format for transaction parameter block |
| -901 | 336330763 | gbak_dup_inout_names | Input and output have the same name. Disallowed |
| -901 | 335740940 | gfix_full_req | "full" or "reserve" required |
| -901 | 335544332 | bad_trans_handle | Invalid transaction handle (expecting explicit transaction start) |
| -901 | 336330764 | gbak_inv_page_size | Expected page size, encountered "@1" |
| -901 | 335740941 | gfix_username_req | User name required |
| -901 | 336330765 | gbak_db_specified | REPLACE specified, but the first file @1 is a database |
| -901 | 335740942 | gfix_pass_req | Password required |
| -901 | 336330766 | gbak_db_exists | Database @1 already exists. To replace it, use the -REP switch |
| -901 | 335740943 | gfix_subs_name | Subsystem name |
| -901 | 336723983 | gsec_cant_open_db | Unable to open database |
| -901 | 336330767 | gbak_unk_device | Device type not specified |
| -901 | 336723984 | gsec_switches_error | Error in switch specifications |
| -901 | 335740945 | gfix_sec_req | Number of seconds required |

Fehlercodes und Meldungen

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|----------------------------|---|
| -901 | 335544337 | excess_trans | Attempt to start more than @1 transactions |
| -901 | 336723985 | gsec_no_op_spec | No operation specified |
| -901 | 335740946 | gfix_nval_req | Numeric value between 0 and 32767 inclusive required |
| -901 | 336723986 | gsec_no_usr_name | No user name specified |
| -901 | 335740947 | gfix_type_shut | Must specify type of shutdown |
| -901 | 335544339 | infinap | Information type inappropriate for object specified |
| -901 | 335544340 | infona | No information of this type available for object specified |
| -901 | 336723987 | gsec_err_add | Add record error |
| -901 | 336723988 | gsec_err_modify | Modify record error |
| -901 | 336330772 | gbak_blob_info_failed | Gds_\$blob_info failed |
| -901 | 335740948 | gfix_retry | Please retry, specifying an option |
| -901 | 335544341 | infunk | Unknown information item |
| -901 | 336723989 | gsec_err_find_mod | Find / modify record error |
| -901 | 336330773 | gbak_unk_blob_item | Do not understand BLOB INFO item @1 |
| -901 | 335544342 | integ_fail | Action cancelled by trigger (@1) to preserve data integrity |
| -901 | 336330774 | gbak_get_seg_failed | Gds_\$get_segment failed |
| -901 | 336723990 | gsec_err_rec_not_found | Record not found for user: @1 |
| -901 | 336723991 | gsec_err_delete | Delete record error |
| -901 | 336330775 | gbak_close_blob_failed | Gds_\$close_blob failed |
| -901 | 335740951 | gfix_retry_db | Please retry, giving a database name |
| -901 | 336330776 | gbak_open_blob_failed | Gds_\$open_blob failed |
| -901 | 336723992 | gsec_err_find_del | Find / delete record error |
| -901 | 335544345 | lock_conflict | Lock conflict on no wait transaction |
| -901 | 336330777 | gbak_put_blr_gen_id_failed | Failed in put_blr_gen_id |
| -901 | 336330778 | gbak_unk_type | Data type @1 not understood |
| -901 | 336330779 | gbak_comp_req_failed | Gds_\$compile_request failed |

Fehlercodes und Meldungen

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|-------------------------|--|
| -901 | 336330780 | gbak_start_req_failed | Gds_\$start_request failed |
| -901 | 336723996 | gsec_err_find_disp | Find / display record error |
| -901 | 336330781 | gbak_rec_failed | gds_\$receive failed |
| -901 | 336920605 | gstat_open_err | Can't open database file @ 1 |
| -901 | 336723997 | gsec_inv_param | Invalid parameter, no switch defined |
| -901 | 335544350 | no_finish | Program attempted to exit without finishing database |
| -901 | 336920606 | gstat_read_err | Can't read a database page |
| -901 | 336330782 | gbak_rel_req_failed | Gds_\$release_request failed |
| -901 | 336723998 | gsec_op_specified | Operation already specified |
| -901 | 336920607 | gstat_sysmemex | System memory exhausted |
| -901 | 336330783 | gbak_db_info_failed | gds_\$database_info failed |
| -901 | 336723999 | gsec_pw_specified | Password already specified |
| -901 | 336724000 | gsec_uid_specified | Uid already specified |
| -901 | 336330784 | gbak_no_db_desc | Expected database description record |
| -901 | 335544353 | no_recon | Transaction is not in limbo |
| -901 | 336724001 | gsec_gid_specified | Gid already specified |
| -901 | 336330785 | gbak_db_create_failed | Failed to create database @ 1 |
| -901 | 336724002 | gsec_proj_specified | Project already specified |
| -901 | 336330786 | gbak_decomp_len_error | RESTORE: decompression length error |
| -901 | 335544355 | no_segstr_close | BLOB was not closed |
| -901 | 336330787 | gbak_tbl_missing | Cannot find table @ 1 |
| -901 | 336724003 | gsec_org_specified | Organization already specified |
| -901 | 336330788 | gbak_blob_col_missing | Cannot find column for BLOB |
| -901 | 336724004 | gsec_fname_specified | First name already specified |
| -901 | 335544357 | open_trans | Cannot disconnect database with open transactions (@ 1 active) |
| -901 | 336330789 | gbak_create_blob_failed | Gds_\$create_blob failed |
| -901 | 336724005 | gsec_mname_specified | Middle name already specified |
| -901 | 335544358 | port_len | Message length error (encountered @ 1, expected @ 2) |

Fehlercodes und Meldungen

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|-------------------------|---|
| -901 | 336330790 | gbak_put_seg_failed | Gds_\$put_segment failed |
| -901 | 336724006 | gsec_lname_specified | Last name already specified |
| -901 | 336330791 | gbak_rec_len_exp | Expected record length |
| -901 | 336724008 | gsec_inv_switch | Invalid switch specified |
| -901 | 336330792 | gbak_inv_rec_len | Wrong length record, expected @1 encountered @2 |
| -901 | 336330793 | gbak_exp_data_type | Expected data attribute |
| -901 | 336724009 | gsec_amb_switch | Ambiguous switch specified |
| -901 | 336330794 | gbak_gen_id_failed | Failed in store_blr_gen_id |
| -901 | 336724010 | gsec_no_op_specified | No operation specified for parameters |
| -901 | 335544363 | req_no_trans | No transaction for request |
| -901 | 336330795 | gbak_unk_rec_type | Do not recognize record type @1 |
| -901 | 336724011 | gsec_params_not_allowed | No parameters allowed for this operation |
| -901 | 335544364 | req_sync | Request synchronization error |
| -901 | 336724012 | gsec_incompat_switch | Incompatible switches specified |
| -901 | 336330796 | gbak_inv_bkup_ver | Expected backup version 1..8. Found @1 |
| -901 | 335544365 | req_wrong_db | Request referenced an unavailable database |
| -901 | 336330797 | gbak_missing_bkup_desc | Expected backup description record |
| -901 | 336330798 | gbak_string_trunc | String truncated |
| -901 | 336330799 | gbak_cant_rest_record | warning -- record could not be restored |
| -901 | 336330800 | gbak_send_failed | Gds_\$send failed |
| -901 | 335544369 | segstr_no_read | Attempted read of a new, open BLOB |
| -901 | 336330801 | gbak_no_tbl_name | No table name for data |
| -901 | 335544370 | segstr_no_trans | Attempted action on blob outside transaction |
| -901 | 336330802 | gbak_unexp_eof | Unexpected end of file on backup file |
| -901 | 336330803 | gbak_db_format_too_old | Database format @1 is too old to restore to |
| -901 | 335544372 | segstr_wrong_db | Attempted reference to BLOB in unavailable database |

Fehlercodes und Meldungen

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|----------------------------|--|
| -901 | 336330804 | gbak_inv_array_dim | Array dimension for column @1 is invalid |
| -901 | 336330807 | gbak_xdr_len_expected | Expected XDR record length |
| -901 | 335544376 | unres_rel | Table @1 was omitted from the transaction reserving list |
| -901 | 335544377 | uns_ext | Request includes a DSRI extension not supported in this implementation |
| -901 | 335544378 | wish_list | Feature is not supported |
| -901 | 335544382 | random | @1 |
| -901 | 335544383 | fatal_conflict | Unrecoverable conflict with limbo transaction @1 |
| -901 | 335740991 | gfix_exceed_max | Internal block exceeds maximum size |
| -901 | 335740992 | gfix_corrupt_pool | Corrupt pool |
| -901 | 335740993 | gfix_mem_exhausted | Virtual memory exhausted |
| -901 | 336330817 | gbak_open_bkup_error | Cannot open backup file @1 |
| -901 | 335740994 | gfix_bad_pool | Bad pool id. |
| -901 | 336330818 | gbak_open_error | Cannot open status and error output file @1 |
| -901 | 335740995 | gfix_trn_not_valid | Transaction state @1 not in valid range |
| -901 | 335544392 | bdbincon | Internal error |
| -901 | 336724044 | gsec_inv_username | Invalid user name (maximum 31 bytes allowed) |
| -901 | 336724045 | gsec_inv_pw_length | Warning - maximum 8 significant bytes of password used |
| -901 | 336724046 | gsec_db_specified | Database already specified |
| -901 | 336724047 | gsec_db_admin_specified | Database administrator name already specified |
| -901 | 336724048 | gsec_db_admin_pw_specified | Database administrator password already specified |
| -901 | 336724049 | gsec_sql_role_specified | SQL role name already specified |
| -901 | 335741012 | gfix_unexp_eoi | Unexpected end of input |
| -901 | 335544407 | dbbnotzer | Database handle not zero |
| -901 | 335544408 | tranotzer | Transaction handle not zero |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|--------------------------|---|
| -901 | 335741018 | gfix_recon_fail | Failed to reconnect to a transaction in database @1 |
| -901 | 335544418 | trainlim | Transaction in limbo |
| -901 | 335544419 | notinlim | Transaction not in limbo |
| -901 | 335544420 | traoutsta | Transaction outstanding |
| -901 | 335544428 | badmsgnum | Undefined message number |
| -901 | 335741036 | gfix_trn_unknown | Transaction description item unknown |
| -901 | 335741038 | gfix_mode_req | "read_only" or "read_write" required |
| -901 | 335544431 | blocking_signal | Blocking signal has been received |
| -901 | 335741042 | gfix_pzval_req | Positive or zero numeric value required |
| -901 | 335544442 | noargacc_read | Database system cannot read argument @1 |
| -901 | 335544443 | noargacc_write | Database system cannot write argument @1 |
| -901 | 335544450 | misc_interpreted | @1 |
| -901 | 335544468 | tra_state | Transaction @1 is @2 |
| -901 | 335544485 | bad_stmt_handle | Invalid statement handle |
| -901 | 336330934 | gbak_missing_block_fac | Blocking factor parameter missing |
| -901 | 336330935 | gbak_inv_block_fac | Expected blocking factor, encountered "@1" |
| -901 | 336330936 | gbak_block_fac_specified | A blocking factor may not be used in conjunction with device CT |
| -901 | 336068796 | dyn_role_does_not_exist | SQL role @1 does not exist |
| -901 | 336330940 | gbak_missing_username | User name parameter missing |
| -901 | 336330941 | gbak_missing_password | Password parameter missing |
| -901 | 336068797 | dyn_no_grant_admin_opt | User @1 has no grant admin option on SQL role @2 |
| -901 | 335544510 | lock_timeout | Lock time-out on wait transaction |
| -901 | 336068798 | dyn_user_not_role_member | User @1 is not a member of SQL role @2 |
| -901 | 336068799 | dyn_delete_role_failed | @1 is not the owner of SQL role @2 |
| -901 | 336068800 | dyn_grant_role_to_user | @1 is a SQL role and not a user |

Fehlercodes und Meldungen

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|-------------------------------|--|
| -901 | 336068801 | dyn_inv_sql_role_name | User name @1 could not be used for SQL role |
| -901 | 336068802 | dyn_dup_sql_role | SQL role @1 already exists |
| -901 | 336068803 | dyn_kywd_spec_for_role | Keyword @1 can not be used as a SQL role name |
| -901 | 336068804 | dyn_roles_not_supported | SQL roles are not supported in on older versions of the database. A backup and restore of the database is required |
| -901 | 336330952 | gbak_missing_skipped_bytes | missing parameter for the number of bytes to be skipped |
| -901 | 336330953 | gbak_inv_skipped_bytes | Expected number of bytes to be skipped, encountered "@1" |
| -901 | 336068820 | dyn_zero_len_id | Zero length identifiers are not allowed |
| -901 | 336330965 | gbak_err_restore_charset | Character set |
| -901 | 336330967 | gbak_err_restore_collation | Collation |
| -901 | 336330972 | gbak_read_error | Unexpected I/O error while reading from backup file |
| -901 | 336330973 | gbak_write_error | Unexpected I/O error while writing to backup file |
| -901 | 336068840 | dyn_wrong_gtt_scope | @1 cannot reference @2 |
| -901 | 336330985 | gbak_db_in_use | Could not drop database @1 (database might be in use) |
| -901 | 336330990 | gbak_sysmemex | System memory exhausted |
| -901 | 335544559 | bad_svc_handle | Invalid service handle |
| -901 | 335544561 | wrospbver | Wrong version of service parameter block |
| -901 | 335544562 | bad_spb_form | Unrecognized service parameter block |
| -901 | 335544563 | svcnodef | Service @1 is not defined |
| -901 | 336068856 | dyn_ods_not_supp_feature | Feature '@1' is not supported in ODS @2.@3 |
| -901 | 336331002 | gbak_restore_role_failed | SQL role |
| -901 | 336331005 | gbak_role_op_missing | SQL role parameter missing |
| -901 | 336331010 | gbak_page_buffers_missing | Page buffers parameter missing |
| -901 | 336331011 | gbak_page_buffers_wrong_param | Expected page buffers, encountered "@1" |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|---------------------------|--|
| -901 | 336331012 | gbak_page_buffers_restore | Page buffers is allowed only on restore or create |
| -901 | 336331014 | gbak_inv_size | Size specification either missing or incorrect for file @1 |
| -901 | 336331015 | gbak_file_outof_sequence | File @1 out of sequence |
| -901 | 336331016 | gbak_join_file_missing | Can't join - one of the files missing |
| -901 | 336331017 | gbak_stdin_not_supptd | standard input is not supported when using join operation |
| -901 | 336331018 | gbak_stdout_not_supptd | Standard output is not supported when using split operation |
| -901 | 336331019 | gbak_bkup_corrupt | Backup file @1 might be corrupt |
| -901 | 336331020 | gbak_unk_db_file_spec | Database file specification missing |
| -901 | 336331021 | gbak_hdr_write_failed | Can't write a header record to file @1 |
| -901 | 336331022 | gbak_disk_space_ex | Free disk space exhausted |
| -901 | 336331023 | gbak_size_lt_min | File size given (@1) is less than minimum allowed (@2) |
| -901 | 336331025 | gbak_svc_name_missing | Service name parameter missing |
| -901 | 336331026 | gbak_not_ownr | Cannot restore over current database, must be SYSDBA or owner of the existing database |
| -901 | 336331031 | gbak_mode_req | "read_only" or "read_write" required |
| -901 | 336331033 | gbak_just_data | Just data ignore all constraints etc. |
| -901 | 336331034 | gbak_data_only | Restoring data only ignoring foreign key, unique, not null & other constraints |
| -901 | 335544609 | index_name | INDEX @1 |
| -901 | 335544610 | exception_name | EXCEPTION @1 |
| -901 | 335544611 | field_name | COLUMN @1 |
| -901 | 335544613 | union_err | Union not supported |

Tabelle B.5. SQLCODE und GDSCODE Fehlercodes und Meldungen (4)

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|--------------------|----------------------------|
| -901 | 335544614 | dsql_construct_err | Unsupported DSQL construct |

Fehlercodes und Meldungen

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|---------------------------------|--|
| -901 | 335544623 | dsql_domain_err | Illegal use of keyword VALUE |
| -901 | 335544626 | table_name | TABLE @1 |
| -901 | 335544627 | proc_name | PROCEDURE @1 |
| -901 | 335544641 | dsql_domain_not_found | Specified domain or source column @1 does not exist |
| -901 | 335544656 | dsql_var_conflict | Variable @1 conflicts with parameter in same procedure |
| -901 | 335544666 | svr_version_too_old | Server version too old to support all CREATE DATABASE options |
| -901 | 335544673 | no_delete | Cannot delete |
| -901 | 335544675 | sort_err | Sort error |
| -901 | 335544703 | svcnoexe | Service @1 does not have an associated executable |
| -901 | 335544704 | net_lookup_err | Failed to locate host machine |
| -901 | 335544705 | service_unknown | Undefined service @1/@2 |
| -901 | 335544706 | host_unknown | The specified name was not found in the hosts file or Domain Name Services |
| -901 | 335544711 | unprepared_stmt | Attempt to execute an unprepared dynamic SQL statement |
| -901 | 335544716 | svc_in_use | Service is currently busy: @1 |
| -901 | 335544731 | tra_must_sweep | [no associated message] |
| -901 | 335544740 | udf_exception | A fatal exception occurred during the execution of a user defined function |
| -901 | 335544741 | lost_db_connection | Connection lost to database |
| -901 | 335544742 | no_write_user_priv | User cannot write to RDB \$USER_PRIVILEGES |
| -901 | 335544767 | blob_filter_exception | A fatal exception occurred during the execution of a blob filter |
| -901 | 335544768 | exception_access_violation | Access violation. The code attempted to access a virtual address without privilege to do so |
| -901 | 335544769 | exception_datatype_misalignment | Datatype misalignment. The attempted to read or write a value that was not stored on a memory boundary |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|----------------------------------|---|
| -901 | 335544770 | exception_array_bounds_exceeded | Array bounds exceeded. The code attempted to access an array element that is out of bounds. |
| -901 | 335544771 | exception_float_denormal_operand | Float denormal operand. One of the floating-point operands is too small to represent a standard float value. |
| -901 | 335544772 | exception_float_divide_by_zero | Floating-point divide by zero. The code attempted to divide a floating-point value by zero. |
| -901 | 335544773 | exception_float_inexact_result | Floating-point inexact result. The result of a floating-point operation cannot be represented as a decimal fraction |
| -901 | 335544774 | exception_float_invalid_operand | Floating-point invalid operand. An indeterminate error occurred during a floating-point operation |
| -901 | 335544775 | exception_float_overflow | Floating-point overflow. The exponent of a floating-point operation is greater than the magnitude allowed |
| -901 | 335544776 | exception_float_stack_check | Floating-point stack check. The stack overflowed or underflowed as the result of a floating-point operation |
| -901 | 335544777 | exception_float_underflow | Floating-point underflow. The exponent of a floating-point operation is less than the magnitude allowed |
| -901 | 335544778 | exception_integer_divide_by_zero | Integer divide by zero. The code attempted to divide an integer value by an integer divisor of zero |
| -901 | 335544779 | exception_integer_overflow | Integer overflow. The result of an integer operation caused the most significant bit of the result to carry |
| -901 | 335544780 | exception_unknown | An exception occurred that does not have a description. Exception number @1 |
| -901 | 335544781 | exception_stack_overflow | Stack overflow. The resource requirements of the runtime stack have exceeded the memory available to it |
| -901 | 335544782 | exception_sigsegv | Segmentation Fault. The code attempted to access memory without privileges |
| -901 | 335544783 | exception_sigill | Illegal Instruction. The Code attempted to perform an illegal operation |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|------------------------------|---|
| -901 | 335544784 | exception_sigbus | Bus Error. The Code caused a system bus error |
| -901 | 335544785 | exception_sigfpe | Floating Point Error. The Code caused an Arithmetic Exception or a floating point exception |
| -901 | 335544786 | ext_file_delete | Cannot delete rows from external files |
| -901 | 335544787 | ext_file_modify | Cannot update rows in external files |
| -901 | 335544788 | adm_task_denied | Unable to perform operation. You must be either SYSDBA or owner of the database |
| -901 | 335544794 | cancelled | Operation was cancelled |
| -901 | 335544797 | svcnouser | User name and password are required while attaching to the services manager |
| -901 | 335544801 | datatype_notsup | Data type not supported for arithmetic |
| -901 | 335544803 | dialect_not_changed | Database dialect not changed |
| -901 | 335544804 | database_create_failed | Unable to create database @1 |
| -901 | 335544805 | inv_dialect_specified | Database dialect @1 is not a valid dialect |
| -901 | 335544806 | valid_db_dialects | Valid database dialects are @1 |
| -901 | 335544811 | inv_client_dialect_specified | Passed client dialect @1 is not a valid dialect |
| -901 | 335544812 | valid_client_dialects | Valid client dialects are @1 |
| -901 | 335544814 | service_not_supported | Services functionality will be supported in a later version of the product |
| -901 | 335544820 | invalid_savepoint | Unable to find savepoint with name @1 in transaction context |
| -901 | 335544835 | bad_shutdown_mode | Target shutdown mode is invalid for database "@1" |
| -901 | 335544840 | no_update | Cannot update |
| -901 | 335544842 | stack_trace | @1 |
| -901 | 335544843 | ctx_var_not_found | Context variable @1 is not found in namespace @2 |
| -901 | 335544844 | ctx_namespace_invalid | Invalid namespace name @1 passed to @2 |
| -901 | 335544845 | ctx_too_big | Too many context variables |
| -901 | 335544846 | ctx_bad_argument | Invalid argument passed to @1 |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|-----------------------------|--|
| -901 | 335544847 | identifier_too_long | BLR syntax error. Identifier @1... is too long |
| -901 | 335544859 | invalid_time_precision | Time precision exceeds allowed range (0-@1) |
| -901 | 335544866 | met_wrong_gtt_scope | @1 cannot depend on @2 |
| -901 | 335544868 | illegal_prc_type | Procedure @1 is not selectable (it does not contain a SUSPEND statement) |
| -901 | 335544869 | invalid_sort_datatype | Datatype @1 is not supported for sorting operation |
| -901 | 335544870 | collation_name | COLLATION @1 |
| -901 | 335544871 | domain_name | DOMAIN @1 |
| -901 | 335544874 | max_db_per_trans_allowed | A multi database transaction cannot span more than @1 databases |
| -901 | 335544876 | bad_proc_BLR | Error while parsing procedure @1's BLR |
| -901 | 335544877 | key_too_big | Index key too big |
| -901 | 336397211 | dsql_too_many_values | Too many values (more than @1) in member list to match against |
| -901 | 336397236 | dsql_unsupp_feature_dialect | Feature is not supported in dialect @1 |
| -902 | 335544333 | bug_check | Internal gds software consistency check (@1) |
| -902 | 335544335 | db_corrupt | Database file appears corrupt (@1) |
| -902 | 335544344 | io_error | I/O error for file "@2" |
| -902 | 335544346 | metadata_corrupt | Corrupt system table |
| -902 | 335544373 | sys_request | Operating system directive @1 failed |
| -902 | 335544384 | badblk | Internal error |
| -902 | 335544385 | invpoolcl | Internal error |
| -902 | 335544387 | relbadblk | Internal error |
| -902 | 335544388 | blktoobig | Block size exceeds implementation restriction |
| -902 | 335544394 | badodsver | Incompatible version of on-disk structure |
| -902 | 335544397 | dirtypage | Internal error |
| -902 | 335544398 | waitfortra | Internal error |
| -902 | 335544399 | doubleloc | Internal error |

Fehlercodes und Meldungen

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|------------------|---|
| -902 | 335544400 | nodnotfnd | Internal error |
| -902 | 335544401 | dupnodfnd | Internal error |
| -902 | 335544402 | locnotmar | Internal error |
| -902 | 335544404 | corrupt | Database corrupted |
| -902 | 335544405 | badpage | Checksum error on database page @1 |
| -902 | 335544406 | badindex | Index is broken |
| -902 | 335544409 | trareqmis | Transaction - request mismatch (synchronization error) |
| -902 | 335544410 | badhndcnt | Bad handle count |
| -902 | 335544411 | wrotpbver | Wrong version of transaction parameter block |
| -902 | 335544412 | wroblrver | Unsupported BLR version (expected @1, encountered @2) |
| -902 | 335544413 | wrodpbver | Wrong version of database parameter block |
| -902 | 335544415 | badrelation | Database corrupted |
| -902 | 335544416 | nodetach | Internal error |
| -902 | 335544417 | notremote | Internal error |
| -902 | 335544422 | dbfile | Internal error |
| -902 | 335544423 | orphan | Internal error |
| -902 | 335544432 | lockmanerr | Lock manager error |
| -902 | 335544436 | sqlerr | SQL error code = @1 |
| -902 | 335544448 | bad_sec_info | [no associated message] |
| -902 | 335544449 | invalid_sec_info | [no associated message] |
| -902 | 335544470 | buf_invalid | Cache buffer for page @1 invalid |
| -902 | 335544471 | indexnotdefined | There is no index in table @1 with id @2 |
| -902 | 335544472 | login | Your user name and password are not defined. Ask your database administrator to set up a Firebird login |
| -902 | 335544506 | shutinprog | Database @1 shutdown in progress |
| -902 | 335544528 | shutdown | Database @1 shutdown |
| -902 | 335544557 | shutfail | Database shutdown unsuccessful |

Fehlercodes und Meldungen

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|---------------------------|--|
| -902 | 335544569 | dsql_error | Dynamic SQL Error |
| -902 | 335544653 | psw_attach | Cannot attach to password database |
| -902 | 335544654 | psw_start_trans | Cannot start transaction for password database |
| -902 | 335544717 | err_stack_limit | Stack size insufficient to execute current request |
| -902 | 335544721 | network_error | Unable to complete network request to host "@1" |
| -902 | 335544722 | net_connect_err | Failed to establish a connection |
| -902 | 335544723 | net_connect_listen_err | Error while listening for an incoming connection |
| -902 | 335544724 | net_event_connect_err | Failed to establish a secondary connection for event processing |
| -902 | 335544725 | net_event_listen_err | Error while listening for an incoming event connection request |
| -902 | 335544726 | net_read_err | Error reading data from the connection |
| -902 | 335544727 | net_write_err | Error writing data to the connection |
| -902 | 335544732 | unsupported_network_drive | Access to databases on file servers is not supported |
| -902 | 335544733 | io_create_err | Error while trying to create file |
| -902 | 335544734 | io_open_err | Error while trying to open file |
| -902 | 335544735 | io_close_err | Error while trying to close file |
| -902 | 335544736 | io_read_err | Error while trying to read from file |
| -902 | 335544737 | io_write_err | Error while trying to write to file |
| -902 | 335544738 | io_delete_err | Error while trying to delete file |
| -902 | 335544739 | io_access_err | Error while trying to access file |
| -902 | 335544745 | login_same_as_role_name | Your login @1 is same as one of the SQL role name. Ask your database administrator to set up a valid Firebird login. |
| -902 | 335544791 | file_in_use | The file @1 is currently in use by another process. Try again later |
| -902 | 335544795 | unexp_spb_form | Unexpected item in service parameter block, expected @1 |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|----------|-----------|-----------------------|--|
| -902 | 335544809 | extern_func_dir_error | Function @1 is in @2, which is not in a permitted directory for external functions |
| -902 | 335544819 | io_32bit_exceeded_err | File exceeded maximum size of 2GB. Add another database file or use a 64 bit I/O version of Firebird |
| -902 | 335544831 | conf_access_denied | Access to @1 "@2" is denied by server administrator |
| -902 | 335544834 | cursor_not_open | Cursor is not open |
| -902 | 335544841 | cursor_already_open | Cursor is already open |
| -902 | 335544856 | att_shutdown | Connection shutdown |
| -902 | 335544882 | long_login | Login name too long (@1 characters, maximum allowed @2) |
| -904 | 335544324 | bad_db_handle | Invalid database handle (no active connection) |
| -904 | 335544375 | unavailable | Unavailable database |
| -904 | 335544381 | imp_exc | Implementation limit exceeded |
| -904 | 335544386 | nopoolids | Too many requests |
| -904 | 335544389 | bufexh | Buffer exhausted |
| -904 | 335544391 | bufinuse | Buffer in use |
| -904 | 335544393 | reqinuse | Request in use |
| -904 | 335544424 | no_lock_mgr | No lock manager available |
| -904 | 335544430 | virmemexh | Unable to allocate memory from operating system |
| -904 | 335544451 | update_conflict | Update conflicts with concurrent update |
| -904 | 335544453 | obj_in_use | Object @1 is in use |
| -904 | 335544455 | shadow_accessed | Cannot attach active shadow file |
| -904 | 335544460 | shadow_missing | A file in manual shadow @1 is unavailable |
| -904 | 335544661 | index_root_page_full | Cannot add index, index root page is full |
| -904 | 335544676 | sort_mem_err | Sort error: not enough memory |
| -904 | 335544683 | req_depth_exceeded | Request depth exceeded. (Recursive definition?) |

| SQL-CODE | GDSCODE | Symbol | Meldung |
|-----------------|----------------|----------------------------|--|
| -904 | 335544758 | sort_rec_size_err | Sort record size of @1 bytes is too big ???? |
| -904 | 335544761 | too_many_handles | Too many open handles to database |
| -904 | 335544792 | service_att_err | Cannot attach to services manager |
| -904 | 335544799 | svc_name_missing | The service name was not specified |
| -904 | 335544813 | optimizer_between_err | Unsupported field type specified in BETWEEN predicate |
| -904 | 335544827 | exec_sql_invalid_arg | Invalid argument in EXECUTE STATEMENT-cannot convert to string |
| -904 | 335544828 | exec_sql_invalid_req | Wrong request type in EXECUTE STATEMENT '@1' |
| -904 | 335544829 | exec_sql_invalid_var | Variable type (position @1) in EXECUTE STATEMENT '@2' INTO does not match returned column type |
| -904 | 335544830 | exec_sql_max_call_exceeded | Too many recursion levels of EXECUTE STATEMENT |
| -904 | 335544832 | wrong_backup_state | Cannot change difference file name while database is in backup mode |
| -904 | 335544852 | partner_idx_incompat_type | Partner index segment no @1 has incompatible data type |
| -904 | 335544857 | blobtoobig | Maximum BLOB size exceeded |
| -904 | 335544862 | record_lock_not_supp | Stream does not support record locking |
| -904 | 335544863 | partner_idx_not_found | Cannot create foreign key constraint @1. Partner index does not exist or is inactive |
| -904 | 335544864 | tra_num_exc | Transactions count exceeded. Perform backup and restore to make database operable again |
| -904 | 335544865 | field_disappeared | Column has been unexpectedly deleted |
| -904 | 335544878 | concurrent_transaction | Concurrent transaction number is @1 |
| -906 | 335544744 | max_att_exceeded | Maximum user count exceeded.Contact your database administrator |
| -909 | 335544667 | drdb_completed_with_errs | Drop database completed with errors |
| -911 | 335544459 | rec_in_limbo | Record from transaction @1 is stuck in limbo |
| -913 | 335544336 | deadlock | Deadlock |

| SQL- CO- DE | GDSCODE | Symbol | Meldung |
|----------------------------|----------------|--------------------|--|
| -922 | 335544323 | bad_db_format | File @1 is not a valid database |
| -923 | 335544421 | connect_reject | Connection rejected by remote interface |
| -923 | 335544461 | cant_validate | Secondary server attachments cannot validate databases |
| -923 | 335544464 | cant_start_logging | Secondary server attachments cannot start logging |
| -924 | 335544325 | bad_dpb_content | Bad parameters on attach or create database |
| -924 | 335544441 | bad_detach | Database detach completed with errors |
| -924 | 335544648 | conn_lost | Connection lost to pipe server |
| -926 | 335544447 | no_rollback | No rollback performed |
| -999 | 335544689 | ib_error | Firebird error |

Anhang C: Reservierte Wörter und Schlüsselwörter

Reservierte Wörter sind Teil der Firebird SQL-Sprache. Sie können nicht als Bezeichner verwendet werden (z.B. als Tabellen- oder Prozedurname), es sei denn sie werden in doppelte Anführungszeichen gesetzt. Dies gilt nur für Dialekt 3. Grundsätzlich sollte dies jedoch vermieden werden.

Schlüsselwörter sind ebenfalls Teil der Sprache. Sie besitzen eine spezielle Bedeutung, sofern sie ordnungsgemäß eingesetzt werden. Sie sind jedoch nicht für die eigene und exklusive Verwendung unter Firebird reserviert. Sie können diese als Bezeichner ohne doppelte Anführungszeichen nutzen.

Reservierte Wörter

Vollständige Liste reservierter Wörter in Firebird 2.5:

| | | |
|--------------------|-------------------|---------------------|
| ADD | ADMIN | ALL |
| ALTER | AND | ANY |
| AS | AT | AVG |
| BEGIN | BETWEEN | BIGINT |
| BIT_LENGTH | BLOB | BOTH |
| BY | CASE | CAST |
| CHAR | CHAR_LENGTH | CHARACTER |
| CHARACTER_LENGTH | CHECK | CLOSE |
| COLLATE | COLUMN | COMMIT |
| CONNECT | CONSTRAINT | COUNT |
| CREATE | CROSS | CURRENT |
| CURRENT_CONNECTION | CURRENT_DATE | CURRENT_ROLE |
| CURRENT_TIME | CURRENT_TIMESTAMP | CURRENT_TRANSACTION |
| CURRENT_USER | CURSOR | DATE |
| DAY | DEC | DECIMAL |
| DECLARE | DEFAULT | DELETE |
| DELETING | DISCONNECT | DISTINCT |
| DOUBLE | DROP | ELSE |
| END | ESCAPE | EXECUTE |
| EXISTS | EXTERNAL | EXTRACT |
| FETCH | FILTER | FLOAT |
| FOR | FOREIGN | FROM |
| FULL | FUNCTION | GDSCODE |
| GLOBAL | GRANT | GROUP |
| HAVING | hour | IN |
| INDEX | INNER | INSENSITIVE |
| INSERT | INSERTING | INT |

| | | |
|------------------|-----------------|------------------|
| INTEGER | INTO | IS |
| JOIN | LEADING | LEFT |
| LIKE | LONG | LOWER |
| MAX | MAXIMUM_SEGMENT | MERGE |
| MIN | MINUTE | MONTH |
| NATIONAL | NATURAL | NCHAR |
| NO | NOT | NULL |
| NUMERIC | OCTET_LENGTH | OF |
| ON | ONLY | OPEN |
| OR | ORDER | OUTER |
| PARAMETER | PLAN | POSITION |
| POST_EVENT | PRECISION | PRIMARY |
| PROCEDURE | RDB\$DB_KEY | REAL |
| RECORD_VERSION | RECREATE | RECURSIVE |
| REFERENCES | RELEASE | RETURNING_VALUES |
| RETURNS | REVOKE | RIGHT |
| ROLLBACK | ROW_COUNT | ROWS |
| SAVEPOINT | SECOND | SELECT |
| SENSITIVE | SET | SIMILAR |
| SMALLINT | SOME | SQLCODE |
| SQLSTATE (2.5.1) | START | SUM |
| TABLE | THEN | TIME |
| TIMESTAMP | TO | TRAILING |
| TRIGGER | TRIM | UNION |
| UNIQUE | UPDATE | UPDATING |
| UPPER | USER | USING |
| VALUE | VALUES | VARCHAR |
| VARIABLE | VARYING | VIEW |
| WHEN | WHERE | WHILE |
| WITH | YEAR | |

Schlüsselwörter

Die folgenden Terme (Zeichen, Zeichenkombinationen, Wörter) haben in der DSQL von Firebird 2.5 eine spezielle Bedeutung. Einige sind außerdem reservierte Wörter, andere nicht.

| | | |
|------------|-----------|-----------|
| !< | ^< | ^= |
| ^> | , | := |
| != | !> | (|
|) | < | <= |
| <> | = | > |
| >= | | ~< |
| ~= | ~> | ABS |
| ACCENT | ACOS | ACTION |
| ACTIVE | ADD | ADMIN |
| AFTER | ALL | ALTER |
| ALWAYS | AND | ANY |
| AS | ASC | ASCENDING |
| ASCII_CHAR | ASCII_VAL | ASIN |
| AT | ATAN | ATAN2 |

| | | |
|--------------------|-------------------|---------------------|
| AUTO | AUTONOMOUS | AVG |
| BACKUP | BEFORE | BEGIN |
| BETWEEN | BIGINT | BIN_AND |
| BIN_NOT | BIN_OR | BIN_SHL |
| BIN_SHR | BIN_XOR | BIT_LENGTH |
| BLOB | BLOCK | BOTH |
| BREAK | BY | CALLER |
| CASCADE | CASE | CAST |
| CEIL | CEILING | CHAR |
| CHAR_LENGTH | CHAR_TO_UUID | CHARACTER |
| CHARACTER_LENGTH | CHECK | CLOSE |
| COALESCE | COLLATE | COLLATION |
| COLUMN | COMMENT | COMMIT |
| COMMITTED | COMMON | COMPUTED |
| CONDITIONAL | CONNECT | CONSTRAINT |
| CONTAINING | COS | COSH |
| COT | COUNT | CREATE |
| CROSS | CSTRING | CURRENT |
| CURRENT_CONNECTION | CURRENT_DATE | CURRENT_ROLE |
| CURRENT_TIME | CURRENT_TIMESTAMP | CURRENT_TRANSACTION |
| CURRENT_USER | CURSOR | DATA |
| DATABASE | DATE | DATEADD |
| DATEDIFF | DAY | DEC |
| DECIMAL | DECLARE | DECODE |
| DEFAULT | DELETE | DELETING |
| DESC | DESCENDING | DESCRIPTOR |
| DIFFERENCE | DISCONNECT | DISTINCT |
| DO | DOMAIN | DOUBLE |
| DROP | ELSE | END |
| ENTRY_POINT | ESCAPE | EXCEPTION |
| EXECUTE | EXISTS | EXIT |
| EXP | EXTERNAL | EXTRACT |
| FETCH | FILE | FILTER |
| FIRST | FIRSTNAME | FLOAT |
| FLOOR | FOR | FOREIGN |
| FREE_IT | FROM | FULL |
| FUNCTION | GDSCODE | GEN_ID |
| GEN_UUID | GENERATED | GENERATOR |
| GLOBAL | GRANT | GRANTED |
| GROUP | HASH | HAVING |
| HOUR | IF | IGNORE |
| IIF | IN | INACTIVE |
| INDEX | INNER | INPUT_TYPE |
| INSENSITIVE | INSERT | INSERTING |
| INT | INTEGER | INTO |
| IS | ISOLATION | JOIN |
| KEY | LAST | LASTNAME |
| LEADING | LEAVE | LEFT |
| LENGTH | LEVEL | LIKE |
| LIMBO | LIST | LN |
| LOCK | LOG | LOG10 |
| LONG | LOWER | LPAD |

| | | |
|----------------|------------------|------------------|
| MANUAL | MAPPING | MATCHED |
| MATCHING | MAX | MAXIMUM_SEGMENT |
| MAXVALUE | MERGE | MIDDLENAME |
| MILLISECOND | MIN | MINUTE |
| MINVALUE | MOD | MODULE_NAME |
| MONTH | NAMES | NATIONAL |
| NATURAL | NCHAR | NEXT |
| NO | NOT | NULL |
| NULLIF | NULLS | NUMERIC |
| OCTET_LENGTH | OF | ON |
| ONLY | OPEN | OPTION |
| OR | ORDER | OS_NAME |
| OUTER | OUTPUT_TYPE | OVER |
| OVERFLOW | OVERLAY | PAD |
| PAGE | PAGE_SIZE | PAGES |
| PARAMETER | PASSWORD | PI |
| PLACING | PLAN | POSITION |
| POST_EVENT | POWER | PRECISION |
| PRESERVE | PRIMARY | PRIVILEGES |
| PROCEDURE | PROTECTED | RAND |
| RDB\$DB_KEY | READ | REAL |
| RECORD_VERSION | RECREATE | RECURSIVE |
| REFERENCES | RELEASE | REPLACE |
| REQUESTS | RESERV | RESERVING |
| RESTART | RESTRICT | RETAIN |
| RETURNING | RETURNING_VALUES | RETURNS |
| REVERSE | REVOKE | RIGHT |
| ROLE | ROLLBACK | ROUND |
| ROW_COUNT | ROW_NUMBER | ROWS |
| RPAD | SAVEPOINT | SCALAR_ARRAY |
| SCHEMA | SECOND | SEGMENT |
| SELECT | SENSITIVE | SEQUENCE |
| SET | SHADOW | SHARED |
| SIGN | SIMILAR | SIN |
| SINGULAR | SINH | SIZE |
| SKIP | SMALLINT | SNAPSHOT |
| SOME | SORT | SOURCE |
| SPACE | SQLCODE | SQLSTATE (2.5.1) |
| SQRT | STABILITY | START |
| STARTING | STARTS | STATEMENT |
| STATISTICS | SUB_TYPE | SUBSTRING |
| SUM | SUSPEND | TABLE |
| TAN | TANH | TEMPORARY |
| THEN | TIME | TIMEOUT |
| TIMESTAMP | TO | TRAILING |
| TRANSACTION | TRIGGER | TRIM |
| TRUNC | TWO_PHASE | TYPE |
| UNCOMMITTED | UNDO | UNION |
| UNIQUE | UPDATE | UPDATING |
| UPPER | USER | USING |
| UUID_TO_CHAR | VALUE | VALUES |
| VARCHAR | VARIABLE | VARYING |

VIEW
WEEKDAY
WHILE
WRITE

WAIT
WHEN
WITH
YEAR

WEEK
WHERE
WORK
YEARDAY

Anhang D: Systemtabellen

Wenn Sie eine Datenbank erstellen, generiert die Firebird Engine einige Systemtabellen. Metadaten—die Beschreibungen und Eigenschaften aller Datenbankobjekte—werden in den Systemtabellen gespeichert.

Systemtabellen werden durch den Präfix RDB\$ gekennzeichnet.

Tabelle D.1. Liste der Systemtabellen

| Systemtabelle | Inhalte |
|-------------------------|---|
| RDB\$BACKUP_HISTORY | Historie der Backups, die mittels <i>nBackup</i> durchgeführt wurden. |
| RDB\$CHARACTER_SETS | Namen und Beschreibungen der verfügbaren Zeichensätze innerhalb der Datenbank |
| RDB\$CHECK_CONSTRAINTS | Querverweise zwischen den Constraint-Namen (NOT NULL-Constraints, CHECK-Constraints sowie ON UPDATE- und ON DELETE-Klauseln eines Fremdschlüssel-Constraints) und ihren systemgenerierten Triggern. |
| RDB\$COLLATIONS | Collation-Sequenz für alle Zeichensätze |
| RDB\$DATABASE | Basisinformationen über die Datenbank |
| RDB\$DEPENDENCIES | Informationen zu den Abhängigkeiten zwischen Datenbankobjekten |
| RDB\$EXCEPTIONS | Benutzerdefinierte Datenbank-Exceptions |
| RDB\$FIELDS | Spalten- und Domain-Definitionen, sowohl system- als auch benutzerdefiniert |
| RDB\$FIELD_DIMENSIONS | Dimensionen der Arrayspalten |
| RDB\$FILES | Informationen über sekundäre und Shadow-Dateien |
| RDB\$FILTERS | Informationen über BLOB-Filter |
| RDB\$FORMATS | Informationen über Änderungen der Tabellenformate |
| RDB\$FUNCTIONS | Informationen über externe Funktionen |
| RDB\$FUNCTION_ARGUMENTS | Parametereigenschaften externer Funktionen |
| RDB\$GENERATORS | Informationen über Generatoren (Sequenzen) |
| RDB\$INDEX_SEGMENTS | Segmente und Index-Positionen |
| RDB\$INDICES | Definitionen aller Datenbankindizes (System- und Benutzerdefiniert) |

| Systemtabelle | Inhalte |
|---------------------------|--|
| RDB\$LOG_FILES | In derzeitigen Versionen nicht verwendet |
| RDB\$PAGES | Informationen über Datenbankseiten |
| RDB\$PROCEDURE_PARAMETERS | Parameter der Stored Procedures |
| RDB\$PROCEDURES | Definitionen der Stored Procedures |
| RDB\$REF_CONSTRAINTS | Definitionen der referentiellen Constraints (Fremdschlüssel) |
| RDB\$RELATION_CONSTRAINTS | Definitionen aller Constraints auf Tabellenebene |
| RDB\$RELATION_FIELDS | Top-Level-Definitionen der Tabellenspalten |
| RDB\$RELATIONS | Header für Tabellen und Views |
| RDB\$ROLES | Rollendefinitionen |
| RDB\$SECURITY_CLASSES | Zugriffslisten (ACL) |
| RDB\$TRANSACTIONS | Status für Multi-Datenbank-Transaktionen |
| RDB\$TRIGGER_MESSAGES | Triggermeldungen |
| RDB\$TRIGGERS | Trigger-Definitionen |
| RDB\$TYPES | Definitionen für enumerierte Datentypen |
| RDB\$USER_PRIVILEGES | An Systembenutzer zugewiesene SQL-Privilegien |
| RDB\$VIEW_RELATIONS | Tabellen denen View-Definitionen zugewiesen wurden: eine Zeile für jede Tabelle innerhalb einer View |

RDB\$BACKUP_HISTORY

RDB\$BACKUP_HISTORY speichert die Historie der Backups, die mittels *nBackup* durchgeführt wurden.

| Spaltenname | Datentyp | Beschreibung |
|-------------------|--------------|--|
| RDB\$BACKUP_ID | INTEGER | Durch die Engine vergebene Kennung |
| RDB\$TIMESTAMP | TIMESTAMP | Zeitstempel des Backup |
| RDB\$BACKUP_LEVEL | INTEGER | Backup-Level |
| RDB\$GUID | CHAR(38) | Eindeutige Kennung |
| RDB\$SCN | INTEGER | Systemnummer (Scan) |
| RDB\$FILE_NAME | VARCHAR(255) | Vollständiger Pfad und Dateiname der Backupdatei |

RDB\$CHARACTER_SETS

RDB\$CHARACTER_SETS benennt und beschreibt die in der Datenbank verfügbaren Zeichensätze.

| Spaltenname | Datentyp | Beschreibung |
|---------------------------|-----------|---|
| RDB\$CHARACTER_SET_NAME | CHAR(31) | Name des Zeichensatzes |
| RDB\$FORM_OF_USE | CHAR(31) | Nicht verwendet |
| RDB\$NUMBER_OF_CHARACTERS | INTEGER | Die Anzahl der Zeichen im Zeichensatz. Wird nicht für existente Zeichensätze verwendet. |
| RDB\$DEFAULT_COLLATE_NAME | CHAR(31) | Der Name der Standard-Collation-Sequenz für den Zeichensatz |
| RDB\$CHARACTER_SET_ID | SMALLINT | Eindeutige Kennung des Zeichensatzes |
| RDB\$SYSTEM_FLAG | SMALLINT | Systemkennzeichen: Wert ist 1 wenn der Zeichensatz bei Erstellung der Datenbank festgelegt wurde; Wert ist 0 für einen benutzerdefinierten Zeichensatz. |
| RDB\$DESCRIPTION | BLOB TEXT | Kann die Textbeschreibung des Zeichensatzes speichern |
| RDB\$FUNCTION_NAME | CHAR(31) | Für benutzerdefinierte Zeichensätze, auf die über externe Funktionen zugegriffen wird, ist dies der Name der externen Funktion. |
| RDB\$BYTES_PER_CHARACTER | SMALLINT | Die maximale Anzahl von Bytes, die ein Zeichen repräsentieren. |

RDB\$CHECK_CONSTRAINTS

RDB\$CHECK_CONSTRAINTS enthält die Querverweise zwischen den systemgenerierten Triggern für Constraints sowie die Namen der zugewiesenen Constraints (NOT NULL-Constraints, CHECK-Constraints sowie die ON UPDATE- und ON DELETE-Klauseln in Fremdschlüssel-Constraints).

| Spaltenname | Datentyp | Beschreibung |
|----------------------|----------|--|
| RDB\$CONSTRAINT_NAME | CHAR(31) | Constraint-Name, der durch den Benutzer oder automatisch durch das System vergeben wurde. |
| RDB\$TRIGGER_NAME | CHAR(31) | Für CHECK-Constraints ist dies der Name des Triggers, der diesen Constraint erzwingt. Für NOT NULL-Constraints ist |

| Spaltenname | Datentyp | Beschreibung |
|-------------|----------|---|
| | | dies der Name der Tabelle, die diesen Constraint enthält. Für Fremdschlüssel-Constraints ist dies der Name des Trigger, der die ON UPDATE- und ON DELETE-Klauseln erzwingt. |

RDB\$COLLATIONS

RDB\$COLLATIONS speichert die Collation-Sequenzen für alle Zeichensätze.

| Spaltenname | Datentyp | Beschreibung |
|---------------------------|-----------|---|
| RDB\$COLLATION_NAME | CHAR(31) | Name der Collation-Sequenz |
| RDB\$COLLATION_ID | SMALLINT | Kennung der Collation-Sequenz. Bildet zusammen mit der Kennung des Zeichensatzes eine eindeutige Kennung. |
| RDB\$CHARACTER_SET_ID | SMALLINT | Kennung des Zeichensatzes. Bildet zusammen mit der Kennung der Collation-Sequenz eine eindeutige Kennung. |
| RDB\$COLLATION_ATTRIBUTES | SMALLINT | Collation-Eigenschaften. Dies ist eine Bitmaske, wobei das erste Bit angibt, ob nachstehende Leerzeichen in Collations berücksichtigt werden sollen (0 - NO PAD; 1 - PAD SPACE); das zweite Bit gibt an, ob die Collation sensitiv für Groß- und Kleinschreibung ist (0 - CASE SENSITIVE, 1 - CASE INSENSITIVE); das dritte Bit gibt an, ob die Collation Akzent-sensitiv ist (0 - ACCENT SENSITIVE, 1 - ACCENT SENSITIVE). Hieraus ergibt sich, dass die Collation bei einem Wert von 5 nachstehende Leerzeichen nicht berücksichtigt und Akzent-sensitiv ist. |
| RDB\$SYSTEM_FLAG | SMALLINT | Kennzeichen: der Wert 0 bedeutet benutzerdefiniert; der Wert 1 bedeutet systemdefiniert. |
| RDB\$DESCRIPTION | BLOB TEXT | Kann Textbeschreibung der Collation speichern |
| RDB\$FUNCTION_NAME | CHAR(31) | Derzeit nicht verwendet |
| RDB\$BASE_COLLATION_NAME | CHAR(31) | Der Name der Basis-Collation für diese Collation-Sequenz. |
| RDB\$SPECIFIC_ATTRIBUTES | BLOB TEXT | Beschreibt spezifische Eigenschaften. |

RDB\$DATABASE

RDB\$DATABASE speichert Basisinformationen über die Datenbank. Beinhaltet nur einen Datensatz.

| Spaltenname | Datentyp | Beschreibung |
|-------------------------|-----------|---|
| RDB\$DESCRIPTION | BLOB TEXT | Datenbankkommentar. |
| RDB\$RELATION_ID | SMALLINT | Zähler der durch jede neu erstellte Tabelle oder View um eins erhöht wird. |
| RDB\$SECURITY_CLASS | CHAR(31) | Die Sicherheitsklasse, die in Tabelle RDB\$SECURITY_CLASSES definiert wurde, um Zugriffe für die gesamte Datenbank zu begrenzen. |
| RDB\$CHARACTER_SET_NAME | CHAR(31) | Der Name des Standardzeichensatzes, der mittels der DEFAULT CHARACTER SET-Klausel während der Datenbankerstellung gesetzt wurde. NULL für den Zeichensatz NONE. |

RDB\$DEPENDENCIES

RDB\$DEPENDENCIES speichert die Abhängigkeiten zwischen Datenbankobjekten.

| Spaltenname | Datentyp | Beschreibung |
|-----------------------|----------|--|
| RDB\$DEPENDENT_NAME | CHAR(31) | Der Name der View, Prozedur, Trigger, CHECK-Constraint oder Computed Column, für die die Abhängigkeit definiert ist, z.B. das <i>abhängige</i> Objekt. |
| RDB\$DEPENDED_ON_NAME | CHAR(31) | Der Name des Objekts, von dem das definierte Objekt—Tabelle, View, Prozedur, Trigger, CHECK-Constraint oder Computed Column—abhängig ist. |
| RDB\$FIELD_NAME | CHAR(31) | Der Spaltenname im abhängigen Objekt, das auf eine View, Prozedur, Trigger, CHECK-Constraint oder Computed Column verweist. |
| RDB\$DEPENDENT_TYPE | SMALLINT | Kennzeichnet den Typ des abhängigen Objekts: 0 - Tabelle |

| Spaltenname | Datentyp | Beschreibung |
|-----------------------|----------|--|
| | | 1 - View 2 - Trigger 3 - Computed Column 4 - CHECK-Constraint 5 - Prozedur 6 - Index-Ausdruck 7 - Exception 8 - User 9 - Spalte 10 - Index |
| RDB\$DEPENDED_ON_TYPE | SMALLINT | Kennzeichnet den Typ des Objekts, auf das verwiesen wird: 0 - Tabelle (oder darin enthaltene Spalte) 1 - View 2 - Trigger 3 - Computed-Column 4 - CHECK-Constraint 5 - Prozedur (oder deren Parameter)) 6 - Index-Anweisung 7 - Exception 8 - User 9 - Spalte 10 - Index 14 - Generator (Sequence) 15 - UDF 17 - Collation |

RDB\$EXCEPTIONS

RDB\$EXCEPTIONS speichert benutzerdefinierte Datenbank-Exceptions.

| Spaltenname | Datentyp | Beschreibung |
|-----------------------|---------------|--|
| RDB\$EXCEPTION_NAME | CHAR(31) | Benutzerdefinierter Exception-Name |
| RDB\$EXCEPTION_NUMBER | INTEGER | Die eindeutige Nummer der Exception, die durch das System zugewiesen wurde |
| RDB\$MESSAGE | VARCHAR(1021) | Exception-Meldungstext |
| RDB\$DESCRIPTION | BLOB TEXT | Kann die Beschreibung der Exception speichern |
| RDB\$SYSTEM_FLAG | SMALLINT | Kennzeichen: Benutzerdefiniert = 0 Systemdefiniert = 1 oder höher |

RDB\$FIELDS

RDB\$FIELDS speichert Definitionen für Spalten und Domains, sowohl system- als auch benutzerdefiniert. Hier werden die detaillierten Dateneigenschaften für alle Spalten gespeichert.

Anmerkung

Die Spalte RDB\$FIELDS.RDB\$FIELD_NAME zeigt auf RDB\$RELATION_FIELDS.RDB\$FIELD_SOURCE, nicht auf RDB\$RELATION_FIELDS.RDB\$FIELD_NAME.

| Spaltenname | Datentyp | Beschreibung |
|------------------------|-----------|---|
| RDB\$FIELD_NAME | CHAR(31) | Der eindeutige Name der Domain. Wird durch den Benutzer festgelegt oder automatisch durch das System. Domains die durch das System erstellt wurden, beginnen mit dem Präfix 'RDB\$'. |
| RDB\$QUERY_NAME | CHAR(31) | Nicht in Verwendung |
| RDB\$VALIDATION_BLR | BLOB BLR | Die Binärsprachenrepräsentation (BLR) des SQL-Ausdrucks, der die Prüfung der CHECK-Werte in der Domain angibt |
| RDB\$VALIDATION_SOURCE | BLOB TEXT | Der originale Quelltext in SQL, der die Prüfung des CHECK-Wertes angibt |
| RDB\$COMPUTED_BLR | BLOB BLR | Die Binärsprachenrepräsentation (BLR) des SQL-Ausdrucks, welchen der Datenbankserver verwendet, wenn auf COMPUTED BY-Spalten zugegriffen wird. |
| RDB\$COMPUTED_SOURCE | BLOB TEXT | Der originale Quelltext der Anweisung, der die COMPUTED BY-Spalte definiert. |
| RDB\$DEFAULT_VALUE | BLOB BLR | Der Standardwert, sofern vorhanden, für das Feld oder die Domain, in Binärsprachenrepräsentation (BLR). |
| RDB\$DEFAULT_SOURCE | BLOB TEXT | Der Vorgabewert als Quelltext, als SQL-Konstante oder -Ausdruck. |
| RDB\$FIELD_LENGTH | SMALLINT | Spaltengröße in Bytes. FLOAT, DATE, TIME, INTEGER beanspruchen 4 Bytes. DOUBLE PRECISION, BIGINT, TIME-STAMP und BLOB beanspruchen 8 Bytes. Für CHAR- und VARCHAR-Datentypen wird die größtmögliche Anzahl Bytes beansprucht, wenn eine String-Domain (Spalte) definiert wurde. |
| RDB\$FIELD_SCALE | SMALLINT | |

| Spaltenname | Datentyp | Beschreibung |
|---------------------|----------|---|
| | | Die negative Nummer, die die Präzision für DECIMAL- und NUMERIC-Spalten festlegt—die Anzahl der Stellen nach dem Dezimalkomma |
| RDB\$FIELD_TYPE | SMALLINT | Code des Datentyps für die Spalte: 7 = SMALLINT 8 = INTEGER 10 = FLOAT 12 = DATE 13 = TIME 14 = CHAR 16 = BIGINT 27 = DOUBLE PRECISION 35 = TIMESTAMP 37 = VARCHAR 261 = BLOB Codes für DECIMAL und NUMERIC sind die gleichen wie für Integer-Typen, da diese als solche gespeichert werden. |
| RDB\$FIELD_SUB_TYPE | SMALLINT | Geben den Untertyp für BLOB-Datentypen an: 0 - undefiniert 1 - Text 2 - BLR 3 - ACL 4 - für zukünftigen Gebrauch reserviert 5 - Enkodierte Tabellenmetadatenbeschreibung 6 - Speicherung der Details für übergreifende Datenbanktransaktionen, die abnormal beendet wurden. Spezifikationen für die CHAR-Datentypen: 0 - untypisierte Daten 1 - feste Binärdaten Spezifiziert einen bestimmten Datentyp für die Integer-Datentypen (SMALLINT, INTEGER, BIGINT) und für Festkomma-Zahlen (NUMERIC, DECIMAL): 0 oder NULL - der Datentyp passt zum Wert im Feld RDB\$FIELD_TYPE 1 - NUMERIC 2 - DECIMAL |
| RDB\$MISSING_VALUE | BLOB BLR | Nicht verwendet |

| Spaltenname | Datentyp | Beschreibung |
|-----------------------|--------------|---|
| RDB\$MISSING_SOURCE | BLOB TEXT | Nicht verwendet |
| RDB\$DESCRIPTION | BLOB TEXT | Beliebiger Kommentar für Domains (Tabellenspalten) |
| RDB\$SYSTEM_FLAG | SMALLINT | Kennzeichen: der Wert 1 bedeutet, dass die Domain automatisch durch das System erstellt wurde, der Wert 0 bedeutet, die Domain wurde durch den Benutzer definiert. |
| RDB\$QUERY_HEADER | BLOB TEXT | Nicht verwendet |
| RDB\$SEGMENT_LENGTH | SMALLINT | Gibt die Länge der BLOB-Buffer in Bytes für BLOB-Spalten an. Verwendet NULL für alle anderen Datentypen. |
| RDB\$EDIT_STRING | VARCHAR(127) | Nicht verwendet |
| RDB\$EXTERNAL_LENGTH | SMALLINT | Die Länge der Spalte in Bytes, sofern diese zu einer externen Tabelle gehört. Für reguläre Tabellen immer NULL. |
| RDB\$EXTERNAL_SCALE | SMALLINT | Der Skalierungsfaktor für Integer-Felder in einer externen Tabelle; repräsentiert die Potenz von 10, die mit dem Integer multipliziert wird |
| RDB\$EXTERNAL_TYPE | SMALLINT | Der Datentyp des Feldes, wie er in der externen Tabelle vorkommt: 7 = SMALLINT 8 = INTEGER 10 = FLOAT 12 = DATE 13 = TIME 14 = CHAR 16 = BIGINT 27 = DOUBLE PRECISION 35 = TIMESTAMP 37 = VARCHAR 261 = BLOB |
| RDB\$DIMENSIONS | SMALLINT | Gibt die Anzahl der Dimensionen in einem Array an, sofern die Spalte als Array definiert wurde, sonst immer NULL. |
| RDB\$NULL_FLAG | SMALLINT | Gibt an, ob die Spalte einen leeren Wert annehmen darf (das Feld enthält dann NULL) oder nicht (das Feld enthält dann den Wert 1) |
| RDB\$CHARACTER_LENGTH | SMALLINT | Die Länge für CHAR- oder VARCHAR-Spalten in Zeichen (nicht in Bytes) |

| Spaltenname | Datentyp | Beschreibung |
|-----------------------|----------|--|
| RDB\$COLLATION_ID | SMALLINT | Die Kennung der Collation-Sequenz für eine Zeichenspalte oder -Domain. Wurde dies nicht definiert ist der Feldwert 0 |
| RDB\$CHARACTER_SET_ID | SMALLINT | Die Kennung des Zeichensatzes für eine Zeichenspalte, eine BLOB TEXT-Spalte oder -Domain |
| RDB\$FIELD_PRECISION | SMALLINT | Gibt die Gesamtzahl der Stellen für Festkomma-Datentypen (DECIMAL und NUMERIC) an. Der Wert ist 0 für Integer-Datentypen, NULL für alle anderen. |

RDB\$FIELD_DIMENSIONS

RDB\$FIELD_DIMENSIONS speichert die Dimensionen für Array-Spalten.

| Spaltenname | Datentyp | Beschreibung |
|------------------|----------|---|
| RDB\$FIELD_NAME | CHAR(31) | Der Name der Array-Spalte. Dieser muss im Feld RDB\$FIELD_NAME innerhalb der Tabelle RDB\$FIELDS. |
| RDB\$DIMENSION | SMALLINT | Kennzeichnet eine Dimension in der Array-Spalte. Die Nummerierung der Dimensionen startet bei 0. |
| RDB\$LOWER_BOUND | INTEGER | Die untere Grenze dieser Dimension. |
| RDB\$UPPER_BOUND | INTEGER | Die obere Grenze dieser Dimension. |

RDB\$FILES

RDB\$FILES speichert Informationen über sekundäre Dateien und Shadow-Dateien.

| Spaltenname | Datentyp | Beschreibung |
|----------------|--------------|--------------|
| RDB\$FILE_NAME | VARCHAR(255) | |

| Spaltenname | Datentyp | Beschreibung |
|--------------------|----------|--|
| | | Der vollständige Pfad zur Datei und der Name von entweder <ul style="list-style-type: none"> • der sekundären Datenbankdatei in Multi-datei-Datenbanken, oder • der Shadow-Datei |
| RDB\$FILE_SEQUENCE | SMALLINT | Die fortlaufende Nummer der sekundären Datei in einer Sequenz oder der Shadow-Datei innerhalb einer Shadow-Dateien-Sammlung. |
| RDB\$FILE_START | INTEGER | Die initiale Seitenzahl in der sekundären Datei oder der Shadow-Datei. |
| RDB\$FILE_LENGTH | INTEGER | Dateilänge in Datenbankseiten. |
| RDB\$FILE_FLAGS | SMALLINT | Für den internen Gebrauch |
| RDB\$SHADOW_NUMBER | SMALLINT | Nummer der Shadow-Sammlung. Wenn die Zeile eine sekundäre Datenbankdatei beschreibt, ist der Feldwert NULL, andernfalls 0. |

RDB\$FILTERS

RDB\$FILTERS speichert Informationen über BLOB-Filter.

| Spaltenname | Datentyp | Beschreibung |
|--------------------|--------------|---|
| RDB\$FUNCTION_NAME | CHAR(31) | Das eindeutige Kennzeichen für BLOB-Filter |
| RDB\$DESCRIPTION | BLOB TEXT | Dokumentation über die BLOB-Filter und die zwei Untertypen, die dieser nutzt. Geschrieben durch den Benutzer. |
| RDB\$MODULE_NAME | VARCHAR(255) | Der Name der dynamischen Bibliothek oder des Shared Object, in der der Code des BLOB-Filters steht. |
| RDB\$ENTRYPOINT | CHAR(31) | Der exportierte Name des BLOB-Filters in der Filterbibliothek. Beachten Sie, dass dies üblicherweise nicht das Gleiche wie RDB\$FUNCTION_NAME ist. Das ist die Kennung, womit der BLOB-Filter in der Datenbank deklariert wird. |

| Spaltenname | Datentyp | Beschreibung |
|----------------------|----------|--|
| RDB\$INPUT_SUB_TYPE | SMALLINT | Der BLOB-Untertyp der Daten, die durch die Funktion konvertiert werden |
| RDB\$OUTPUT_SUB_TYPE | SMALLINT | Der BLOB-Untertyp der konvertierten Daten. |
| RDB\$SYSTEM_FLAG | SMALLINT | Dieses Kennzeichen gibt an, ob der Filter ist benutzerdefiniert oder intern definiert: <ul style="list-style-type: none"> • 0 = benutzerdefiniert • 1 oder größer = intern definiert |

RDB\$FORMATS

RDB\$FORMATS speichert Informationen über Änderungen in Tabellen. Jedes Mal wenn Änderungen in den Metadaten einer Tabelle durchgeführt werden, bekommt diese eine neue Formatnummer. Wenn die Formatnummer irgendeiner Tabelle die 255 erreicht, wird die gesamte Datenbank inoperabel. Um in den normalen Betrieb zu wechseln, müssen Sie zunächst ein Backup der Datenbank mit dem Werkzeug *gbak* und anschließend eine Wiederherstellung durchführen.

| Spaltenname | Datentyp | Beschreibung |
|------------------|-------------|--|
| RDB\$RELATION_ID | SMALLINT | Kennung der Tabelle oder View |
| RDB\$FORMAT | SMALLINT | Kennung des Tabellenformats—maximal 255. Der kritische Punkt ist erreicht, wenn die Nummer 255 für eine <i>beliebige</i> Tabelle oder View erreicht. |
| RDB\$DESCRIPTOR | BLOB FORMAT | Speichert Spaltennamen und Dateneigenschaften als BLOB, so wie sie zum Zeitpunkt der Erstellung des Format-Datensatzes war. |

RDB\$FUNCTIONS

RDB\$FUNCTIONS speichert Informationen, die von der Engine für externe Funktionen (benutzerdefinierte Funktionen, UDFs) verwendet werden.

Anmerkung

In späteren Hauptversionen (Firebird 3.0 +) wird RDB\$FUNCTIONS außerdem Informationen zu Stored Functions speichern: benutzerdefinierte Funktionen, geschrieben in SQL.

| Spaltenname | Datentyp | Beschreibung |
|----------------------|--------------|--|
| RDB\$FUNCTION_NAME | CHAR(31) | Der eindeutige (deklarierte) Name der externen Funktion. |
| RDB\$FUNCTION_TYPE | SMALLINT | Derzeit nicht verwendet |
| RDB\$QUERY_NAME | CHAR(31) | Derzeit nicht verwendet |
| RDB\$DESCRIPTION | BLOB TEXT | Beliebiger Textkommentar zur externen Funktion |
| RDB\$MODULE_NAME | VARCHAR(255) | Der Name der dynamischen Bibliothek oder des Shared Object, die bzw. das den Code der externen Funktion vorhält. |
| RDB\$ENTRYPOINT | CHAR(31) | Der exportierte Name der externen Funktion in der Funktionsbibliothek. Beachten Sie, dass dies üblicherweise nicht der gleiche Name wie in RDB\$FUNCTION_NAME ist, welches wiederum die Kennung hält, mit der die externe Funktion in der Datenbank registriert ist. |
| RDB\$RETURN_ARGUMENT | SMALLINT | Die Positionsnummer des zurückgegebenen Argumentes innerhalb der Parameterliste, die sich auf die Eingabeargumente bezieht. |
| RDB\$SYSTEM_FLAG | SMALLINT | Kennzeichen zeigt an, ob der Filter benutzer- oder intern definiert wurde: <ul style="list-style-type: none"> • 0 = benutzerdefiniert • 1 = intern definiert |

RDB\$FUNCTION_ARGUMENTS

RDB\$FUNCTION_ARGUMENTS speichert die Parameter externer Funktionen und ihrer Attribute.

| Spaltenname | Datentyp | Beschreibung |
|------------------------|----------|--|
| RDB\$FUNCTION_NAME | CHAR(31) | Der eindeutige Name (deklariertes Kennzeichen) der externen Funktion |
| RDB\$ARGUMENT_POSITION | SMALLINT | Die Position des Arguments innerhalb der Argumentliste. |

| Spaltenname | Datentyp | Beschreibung |
|-----------------------|----------|--|
| RDB\$MECHANISM | SMALLINT | <p>Kennzeichen: wie wird das Argument übergeben</p> <ul style="list-style-type: none"> • 0 = by value • 1 = by reference • 2 = by descriptor • 3 = by BLOB descriptor |
| RDB\$FIELD_TYPE | SMALLINT | <p>Datentyp-Code, der für die Spalte definiert ist:</p> <p>7 = SMALLINT 8 = INTEGER 12 = DATE 13 = TIME 14 = CHAR 16 = BIGINT 27 = DOUBLE PRECISION 35 = TIMESTAMP 37 = VARCHAR 261 = BLOB 40 = CSTRING (null-terminierter Text) 45 = BLOB_ID 261 = BLOB</p> |
| RDB\$FIELD_SCALE | SMALLINT | Die Skalierung eines Integer- oder Festkomma-Arguments. Dies ist der Exponent von 10. |
| RDB\$FIELD_LENGTH | SMALLINT | <p>Argumentlänge in Bytes:</p> <p>SMALLINT = 2 INTEGER = 4 DATE = 4 TIME = 4 BIGINT = 8 DOUBLE PRECISION = 8 TIMESTAMP = 8 BLOB_ID = 8</p> |
| RDB\$FIELD_SUB_TYPE | SMALLINT | Speichert den BLOB-Untertypen für ein Argument des BLOB-Datentyps. |
| RDB\$CHARACTER_SET_ID | SMALLINT | Die Kennung des Zeichensatzes für Zeichenargumente. |
| RDB\$FIELD_PRECISION | SMALLINT | Die Anzahl der Stelle für die Präzision, die für den Datentyp des Arguments verfügbar ist. |
| RDB\$CHARACTER_LENGTH | SMALLINT | Die Länge eines CHAR- oder VARCHAR-Arguments in Zeichen (nicht in Bytes). |

RDB\$GENERATORS

RDB\$GENERATORS speichert Generatoren (Sequenzen) und hält diese aktuell.

| Spaltenname | Datentyp | Beschreibung |
|---------------------|-----------|--|
| RDB\$GENERATOR_NAME | CHAR(31) | Der eindeutige Generatorname. |
| RDB\$GENERATOR_ID | SMALLINT | Die eindeutige Kennung, die für den Generator durch das System vergeben wurde. |
| RDB\$SYSTEM_FLAG | SMALLINT | Kennzeichen: 0 = benutzerdefiniert 1 oder größer = systemdefiniert |
| RDB\$DESCRIPTION | BLOB TEXT | Kann Kommentartexte zum Generator speichern. |

RDB\$INDICES

RDB\$INDICES speichert die Definitionen benutzerdefinierter und systemdefinierter Indizes. Die Eigenschaften jeder Spalte, die zu einem Index gehören, werden in je einer Spalte innerhalb der Tabelle RDB\$INDEX_SEGMENTS vorgehalten.

| Spaltenname | Datentyp | Beschreibung |
|--------------------|----------|--|
| RDB\$INDEX_NAME | CHAR(31) | Der eindeutige Indexname, der durch den Benutzer oder automatisch durch das System vergeben wurde. |
| RDB\$RELATION_NAME | CHAR(31) | Der Name der Tabelle zu der der Index gehört. Dieser korrespondiert mit der Kennung in RDB\$RELATION_NAME.RDB\$RELATIONS |
| RDB\$INDEX_ID | SMALLINT | Die interne (System-)Kennung des Index. |
| RDB\$UNIQUE_FLAG | SMALLINT | Gibt an, ob der Index eindeutig ist: 1 - eindeutig (unique) 0 - nicht eindeutig (not unique) |

| Spaltenname | Datentyp | Beschreibung |
|------------------------|------------------|--|
| RDB\$DESCRIPTION | BLOB TEXT | Kann Kommentare zum Index speichern. |
| RDB\$SEGMENT_COUNT | SMALLINT | Die Anzahl der Segment (Spalten) des Index. |
| RDB\$INDEX_INACTIVE | SMALLINT | Gibt an, ob der Index derzeit aktiv ist: 1 - inaktiv 0 - aktiv |
| RDB\$INDEX_TYPE | SMALLINT | Unterscheidet zwischen einem Anweisungsindex (1) und einem regulären Index(0 oder null). Wird nicht in Datenbanken vor Firebird 2.0 verwendet; reguläre Indizes in aktualisierten (upgraded) Datenbanken werden üblicherweise NULL in dieser Spalte speichern. |
| RDB\$FOREIGN_KEY | CHAR(31) | Der Name des zugewiesenen Fremdschlüssel-Constraints, falls vorhanden. |
| RDB\$SYSTEM_FLAG | SMALLINT | Gibt an, ob der Index system- oder benutzerdefiniert ist: 1 oder größer - systemdefiniert 0 - benutzerdefiniert |
| RDB\$EXPRESSION_BLR | BLOB BLR | Ausdruck für einen Anweisungsindex, geschrieben in Binärsprachenrepräsentation (BLR). Wird für die Berechnung der Indexwerte zur Laufzeit verwendet. |
| RDB\$EXPRESSION_SOURCE | BLOB TEXT | Der Quellcode des Ausdrucks für einen Anweisungsindex. |
| RDB\$STATISTICS | DOUBLE PRECISION | Speichert die letzte bekannte Selektivität des gesamten Index, die durch die Ausführung eines SET STATISTICS-Statements berechnet wird. Diese wird außerdem beim ersten Öffnen der Datenbank durch den Server Neuberechnet. Die Selektivität jedes einzelnen Index-Segments wird in der Tabelle RDB\$INDEX_SEGMENTS gespeichert. |

RDB\$INDEX_SEGMENTS

RDB\$INDEX_SEGMENTS speichert die Segmente (Tabellenspalten) eines Index und ihre Position innerhalb des Schlüssels. Pro Spalte innerhalb des Index wird eine einzelne Zeile vorgehalten.

| Spaltenname | Datentyp | Beschreibung |
|---------------------|------------------|--|
| RDB\$INDEX_NAME | CHAR(31) | Der Name des Index, dem dieses Segment zugewiesen ist. Der Hauptdatensatz befindet sich in RDB\$INDICES.RDB\$INDEX_NAME. |
| RDB\$FIELD_NAME | CHAR(31) | Der Name der Spalte, die zum Index gehört, korrespondierend zur Kennung für die Tabelle und dessen Spalte in RDB\$RELATION_FIELDS.RDB\$FIELD_NAME. |
| RDB\$FIELD_POSITION | SMALLINT | Die Spaltenposition im Index. Die Positionen werden von links nach rechts festgelegt und starten bei 0. |
| RDB\$STATISTICS | DOUBLE PRECISION | Die letzte bekannte (berechnete) Selektivität dieses Spaltenindex. Je größer die Zahl ist, desto kleiner die Selektivität. |

RDB\$LOG_FILES

RDB\$LOG_FILES wird derzeit nicht verwendet.

RDB\$PAGES

RDB\$PAGES speichert Informationen über die Datenbankseiten und deren Nutzung.

| Spaltenname | Datentyp | Beschreibung |
|--------------------|----------|---|
| RDB\$PAGE_NUMBER | INTEGER | Die eindeutige Nummer der physikalisch erstellten Datenbankseiten. |
| RDB\$RELATION_ID | SMALLINT | Die Kennung der Tabelle, zu der die Seite gehört. |
| RDB\$PAGE_SEQUENCE | INTEGER | Die Nummer der Seite innerhalb der Sequenz aller Seiten in der zugehörigen Tabelle. |
| RDB\$PAGE_TYPE | SMALLINT | Gibt den Seitentyp an (Daten, Index, BLOB, etc.). Informationen für das System. |

RDB\$PROCEDURES

RDB\$PROCEDURES speichert die Definitionen für Stored Procedures, inklusive ihres PSQL-Quelltextes und ihrer Binärsprachenrepräsentation (BLR). Die nächste Tabelle RDB\$PROCEDURE_PARAMETERS speichert die Definitionen der Eingabe- und Ausgabeparameter.

| Spaltenname | Datentyp | Beschreibung |
|------------------------|-----------|---|
| RDB\$PROCEDURE_NAME | CHAR(31) | Name (Kennung) der Stored Procedure. |
| RDB\$PROCEDURE_ID | SMALLINT | Die eindeutige system-generierte Kennung. |
| RDB\$PROCEDURE_INPUTS | SMALLINT | Gibt die Anzahl der Eingabeparameter an. NULL wenn es keine gibt. |
| RDB\$PROCEDURE_OUTPUTS | SMALLINT | Gibt die Anzahl der Ausgabeparameter an. NULL wenn es keine gibt. |
| RDB\$DESCRIPTION | BLOB TEXT | Beliebiger Kommentartext, der die Prozedur beschreibt. |
| RDB\$PROCEDURE_SOURCE | BLOB TEXT | Der PSQL-Quelltext der Prozedur. |
| RDB\$PROCEDURE_BLR | BLOB BLR | Die Binärsprachenrepräsentation (BLR) des Prozedurcodes. |
| RDB\$SECURITY_CLASS | CHAR(31) | Kann die definierte Sicherheitsklasse aus der Systemtabelle RDB\$SECURITY_CLASSES aufnehmen, um Zugriffsbeschränkungen zu verwenden. |
| RDB\$OWNER_NAME | CHAR(31) | Der Benutzername des Prozedurbesitzers—der Benutzer, der CURRENT_USER war, als die Prozedur erstellt wurde. Dies kann, muss aber nicht, der Benutzername des Autors sein. |
| RDB\$RUNTIME | BLOB | Eine Metadatenbeschreibung der Prozedur, die intern für die Optimierung verwendet wird. |
| RDB\$SYSTEM_FLAG | SMALLINT | Gibt an, ob die Prozedur durch einen Benutzer (Wert 0) oder durch das System (Wert 1 oder größer) erstellt wurde. |
| RDB\$PROCEDURE_TYPE | SMALLINT | Prozedurtyp: 1 - selektierbare Stored Procedure (beinhaltet ein SUSPEND-Statement) 2 - ausführbare Stored Procedure NULL - unbekannt * |

| Spaltenname | Datentyp | Beschreibung |
|-----------------|----------|---|
| | | * gilt für Prozeduren, die vor Firebird 1.5 erstellt wurden. |
| RDB\$VALID_BLR | SMALLINT | Gibt an, ob der PSQL-Quelltext der Stored Procedure nach der letzten Anpassung mittels ALTER PROCEDURE gültig bleibt. |
| RDB\$DEBUG_INFO | BLOB | Beinhaltet Debugging-Informationen über Variablen, die in der Stored Procedure Verwendung finden. |

RDB\$PROCEDURE_PARAMETERS

RDB\$PROCEDURE_PARAMETERS speichert die Parameter einer Stored Procedure und ihrer Eigenschaften. Je Parameter wird eine eigene Zeile vorgehalten.

| Spaltenname | Datentyp | Beschreibung |
|-----------------------|-----------|---|
| RDB\$PARAMETER_NAME | CHAR(31) | Parametername |
| RDB\$PROCEDURE_NAME | CHAR(31) | Der Name der Prozedur, für die der Parameter definiert wurde. |
| RDB\$PARAMETER_NUMBER | SMALLINT | Die Folgenummer des Parameters. |
| RDB\$PARAMETER_TYPE | SMALLINT | Gibt an, ob dies ein Eingabe- (Wert 0) oder Ausgabeparameter (Wert 1) ist. |
| RDB\$FIELD_SOURCE | CHAR(31) | Der Name der benutzerdefinierten Domain, wenn eine Domain anstelle eines Datentyps referenziert wurde. Beginnt der Name mit dem Präfix 'RDB\$', wurde die Domain automatisch durch das System erstellt. |
| RDB\$DESCRIPTION | BLOB TEXT | Kann Kommentartexte zum Parameter speichern. |
| RDB\$SYSTEM_FLAG | SMALLINT | Gibt an, ob der Parameter durch das System (Wert 1 oder größer) oder durch den Benutzer definiert wurde (Wert 0) |
| RDB\$DEFAULT_VALUE | BLOB BLR | Der Vorgabewert des Parameters in Binärsprachenrepräsentation (BLR). |
| RDB\$DEFAULT_SOURCE | BLOB TEXT | Der Vorgabewert des Parameters als PSQL-Code. |
| RDB\$COLLATION_ID | SMALLINT | Die Kennung der Collation-Sequenz, die für Zeichenparameter verwendet wird. |

| Spaltenname | Datentyp | Beschreibung |
|--------------------------|----------|--|
| RDB\$NULL_FLAG | SMALLINT | Gibt an, ob NULL erlaubt ist. |
| RDB\$PARAMETER_MECHANISM | SMALLINT | Kennzeichen: gibt an wie der Parameter übergeben wird: <ul style="list-style-type: none"> • 0 = by value • 1 = by reference • 2 = by descriptor • 3 = by BLOB descriptor |
| RDB\$FIELD_NAME | CHAR(31) | Der Name der Spalte, auf die der Parameter verweist, wenn er mit TYPE OF COLUMN anstelle eines regulären Datentyps deklariert wurde. Wird in Verbindung mit RDB\$RELATION_NAME verwendet (siehe unten). |
| RDB\$RELATION_NAME | CHAR(31) | Der Name der Tabelle, auf die der Parameter verweist, wenn er mit TYPE OF COLUMN anstelle eines regulären Datentyps deklariert wurde. |

RDB\$REF_CONSTRAINTS

RDB\$REF_CONSTRAINTS speichert die Eigenschaften für referentielle Constraints—Fremdschlüsselbeziehungen und referentielle Aktionen.

| Spaltenname | Datentyp | Beschreibung |
|----------------------|----------|---|
| RDB\$CONSTRAINT_NAME | CHAR(31) | Name des Fremdschlüssels, definiert durch den Benutzer oder automatisch durch das System. |
| RDB\$CONST_NAME_UQ | CHAR(31) | Der Name der primären oder eindeutigen Schlüsselbedingung, die durch die REFERENCES-Klausel in der Constraint-Definition verknüpft ist. |
| RDB\$MATCH_OPTION | CHAR(7) | Wird nicht verwendet. Der Wert ist in allen Fällen FULL. |
| RDB\$UPDATE_RULE | CHAR(11) | Aktionen für die referentielle Integrität, die auf Fremdschlüsseldatensätze angewendet wird, sobald der Primärschlüssel der Eltern-tabelle aktualisiert wird: RESTRICT, NO ACTION, CASCADE, SET NULL, SET DEFAULT |

| Spaltenname | Datentyp | Beschreibung |
|------------------|----------|--|
| RDB\$DELETE_RULE | CHAR(11) | Aktionen für die referentielle Integrität, die auf Fremdschlüsseldatensätze angewendet wird, sobald der Primärschlüssel der Elterntabelle gelöscht wird: RESTRICT, NO ACTION, CASCADE, SET NULL, SET DEFAULT |

RDB\$RELATIONS

RDB\$RELATIONS speichert die Top-Level-Definitionen und -Eigenschaften aller Tabellen und Views im System.

| Spaltenname | Datentyp | Beschreibung |
|--------------------|-----------|--|
| RDB\$VIEW_BLR | BLOB BLR | Speichert die Abfragespezifikation einer View in Binärsprachenrepräsentation (BLR). Das Feld speichert NULL für Tabellen. |
| RDB\$VIEW_SOURCE | BLOB TEXT | Beinhaltet den Original Quelltext der Abfrage für eine View, in SQL-Sprache. Benutzerkommentare sind inkludiert. Das Feld speichert NULL für Tabellen. |
| RDB\$DESCRIPTION | BLOB TEXT | Speichert Kommentare für die Tabelle oder View. |
| RDB\$RELATION_ID | SMALLINT | Interne Kennung der Tabelle oder View. |
| RDB\$SYSTEM_FLAG | SMALLINT | Gibt an ob die Tabelle oder View benutzer- (Wert 0) oder systemdefiniert (Wert 1 oder größer) ist. |
| RDB\$DBKEY_LENGTH | SMALLINT | Die Gesamtlänge des Datenbankschlüssels. Für eine Tabelle: 8 Bytes. Für eine View: die Anzahl aller beinhalteten Tabellen mit 8 multipliziert. |
| RDB\$FORMAT | SMALLINT | Interne Verwendung, zeigt auf den verknüpften Datensatz in RDB\$FORMATS— nicht anpassen. |
| RDB\$FIELD_ID | SMALLINT | Die Feld-ID für die nächste anzufügende Spalte. Die Zahl wird nicht dekrementiert, wenn eine Spalte gelöscht wird. |
| RDB\$RELATION_NAME | CHAR(31) | Name der Tabelle oder View. |

| Spaltenname | Datentyp | Beschreibung |
|---------------------------|--------------|--|
| RDB\$SECURITY_CLASS | CHAR(31) | Kann eine Referenz zur Sicherheitsklasse aufnehmen, die in der Tabelle RDB\$SECURITY_CLASSES definiert wurde. Damit lassen sich Zugriffsbeschränkungen für alle Benutzer dieser Tabelle oder View umsetzen. |
| RDB\$EXTERNAL_FILE | VARCHAR(255) | Der vollständige Pfad der externen Daten-datei, sofern die Tabelle mit der EXTERNAL FILE-Klausel definiert wurde. |
| RDB\$RUNTIME | BLOB | Beschreibung der Tabellenmetadaten, intern für Optimierungen verwendet. |
| RDB\$EXTERNAL_DESCRIPTION | BLOB | Kann Kommentare für die externe Datei einer externen Tabelle speichern. |
| RDB\$OWNER_NAME | CHAR(31) | Der Benutzername des Benutzers, der die Tabelle oder View erstellt hat. |
| RDB\$DEFAULT_CLASS | CHAR(31) | Standard-Sicherheitsklasse. Wird verwendet, wenn eine neue Spalte zur Tabelle hinzugefügt wurde. |
| RDB\$FLAGS | SMALLINT | Interne Kennzeichen. |
| RDB\$RELATION_TYPE | SMALLINT | Der Typ des Relationsobjekts: 0 - system- oder benutzerdefinierte Tabelle 1 - View 2 - Externe Tabelle 3 - Monitoring-Tabelle 4 - Connection-Level GTT (PRESERVE ROWS) 5 - Transaction-Level GTT (DELETE ROWS) |

RDB\$RELATION_CONSTRAINTS

RDB\$RELATION_CONSTRAINTS speichert die Definitionen aller Tabellen-Level Constraints: Primärschlüssel, UNIQUE, Fremdschlüssel, CHECK, NOT NULL.

| Spaltenname | Datentyp | Beschreibung |
|----------------------|----------|--|
| RDB\$CONSTRAINT_NAME | CHAR(31) | Der Name des Tabellen-Level Constraints. Definiert durch den Benutzer, oder automatisch durch das System erstellt. |

| Spaltenname | Datentyp | Beschreibung |
|-------------------------|----------|---|
| RDB\$CONSTRAINT_TYPE | CHAR(11) | Der Name des Constraint-Typs: PRIMARY KEY, UNIQUE, FOREIGN KEY, CHECK oder NOT NULL |
| RDB\$RELATION_NAME | CHAR(31) | Der Name der Tabelle zu der der Constraint gehört. |
| RDB\$DEFERRABLE | CHAR(3) | Derzeit in allen Fällen NO: Firebird unterstützt derzeit keine verögerten (deferrable) Constraints. |
| RDB\$INITIALLY_DEFERRED | CHAR(3) | Derzeit in allen Fällen NO. |
| RDB\$INDEX_NAME | CHAR(31) | Der Name des Index, der diesen Constraint unterstützt. Für einen CHECK- oder NOT NULL-Constraint ist der Wert NULL. |

RDB\$RELATION_FIELDS

RDB\$RELATION_FIELDS speichert die Definitionen der Tabellen- und View-Spalten.

| Spaltenname | Datentyp | Beschreibung |
|---------------------|--------------|---|
| RDB\$FIELD_NAME | CHAR(31) | Spaltenname |
| RDB\$RELATION_NAME | CHAR(31) | Der Name der Tabelle oder View zu der die Spalte gehört. |
| RDB\$FIELD_SOURCE | CHAR(31) | Name der Domain auf der die Spalte basiert, entweder benutzerdefiniert über die Tabellendefinition oder automatisch über das System erstellt, anhand der definierten Eigenschaften. Die Eigenschaften stehen in der Tabelle RDB\$FIELDS: diese Spalte verweist auf RDB\$FIELDS.RDB\$FIELD_NAME. |
| RDB\$QUERY_NAME | CHAR(31) | Derzeit nicht verwendet |
| RDB\$BASE_FIELD | CHAR(31) | Nur bei Views gefüllt. Beinhaltet den Namen der Spalte aus der Basistabelle. |
| RDB\$EDIT_STRING | VARCHAR(127) | Nicht verwendet. |
| RDB\$FIELD_POSITION | SMALLINT | Die null-basierte Position der Spalten in der Tabelle oder View, Aufzählung von links nach rechts. |
| RDB\$QUERY_HEADER | BLOB TEXT | Nicht verwendet. |

| Spaltenname | Datentyp | Beschreibung |
|---------------------|-----------|---|
| RDB\$UPDATE_FLAG | SMALLINT | Gibt an ob dies eine reguläre (Wert 1) oder berechnete (Wert 0) Spalte ist. |
| RDB\$FIELD_ID | SMALLINT | Eine ID zugewiesen durch RDB \$RELATIONS.RDB\$FIELD_ID zum Zeitpunkt als die Spalte zur View oder Tabelle hinzugefügt wurde. Sollte immer als verfügbar angesehen werden. |
| RDB\$VIEW_CONTEXT | SMALLINT | Für eine View-Spalte ist dies die interne Kennung der Basistabelle aus der das Feld stammt. |
| RDB\$DESCRIPTION | BLOB TEXT | Kommentare zur Tabellen- oder View-Spalte. |
| RDB\$DEFAULT_VALUE | BLOB BLR | Der Wert, der für die DEFAULT-Klausel der Spalte verwendet wurde, sofern einer vorhanden ist, gespeichert als Binärsprachenrepräsentation (BLR). |
| RDB\$SYSTEM_FLAG | SMALLINT | Gibt an, ob die Spalte benutzer- (Wert 0) oder systemdefiniert (Wert 1 oder größer) ist. |
| RDB\$SECURITY_CLASS | CHAR(31) | Kann auf eine in RDB \$SECURITY_CLASSES definierte Sicherheitsklasse verweisen, um Zugriffsbeschränkungen für alle Benutzer dieser Spalte anzuwenden. |
| RDB\$COMPLEX_NAME | CHAR(31) | Nicht verwendet. |
| RDB\$NULL_FLAG | SMALLINT | Gibt an ob die Spalte null zulässt (NULL) oder nicht (Wert 1) |
| RDB\$DEFAULT_SOURCE | BLOB TEXT | Der Quelltext einer DEFAULT-Klausel, wenn vorhanden. |
| RDB\$COLLATION_ID | SMALLINT | Die Kennung der Collation-Sequenz des Zeichensatzes für die Spalte, sofern dies nicht die Vorgabe-Collation ist. |

RDB\$ROLES

RDB\$ROLES speichert die Rollen, die in der Datenbank definiert wurden.

| Spaltenname | Datentyp | Beschreibung |
|----------------|----------|--------------|
| RDB\$ROLE_NAME | CHAR(31) | Rollenname |

| Spaltenname | Datentyp | Beschreibung |
|------------------|-----------|---|
| RDB\$OWNER_NAME | CHAR(31) | Der Benutzername des Rolleneigentümers. |
| RDB\$DESCRIPTION | BLOB TEXT | Speichert Kommentare zur Rolle. |
| RDB\$SYSTEM_FLAG | SMALLINT | Systemkennzeichen. |

RDB\$SECURITY_CLASSES

RDB\$SECURITY_CLASSES speichert die Zugriffslisten.

| Spaltenname | Datentyp | Beschreibung |
|---------------------|-----------|---|
| RDB\$SECURITY_CLASS | CHAR(31) | Name der Sicherheitsklasse. |
| RDB\$ACL | BLOB ACL | Die Zugriffsliste, die sich auf die Sicherheitsklasse bezieht. Listet Benutzer und ihre Berechtigungen auf. |
| RDB\$DESCRIPTION | BLOB TEXT | Speichert Kommentare zur Sicherheitsklasse. |

RDB\$TRANSACTIONS

RDB\$TRANSACTIONS speichert die Zustände verteilter Transaktionen und anderer Transaktionen, die für ein zweiphasiges Commit mit einer expliziten Vorbereitungsnachricht vorgesehen wurden.

| Spaltenname | Datentyp | Beschreibung |
|------------------------------|-----------|--|
| RDB\$TRANSACTION_ID | INTEGER | Die eindeutige Kennung der verfolgten Transaktion. |
| RDB\$TRANSACTION_STATE | SMALLINT | Transaktionsstatus: 0 - in limbo 1 - committed 2 - rolled back |
| RDB\$TIMESTAMP | TIMESTAMP | Nicht verwendet. |
| RDB\$TRANSACTION_DESCRIPTION | BLOB | Beschreibt die vorbereitete Transaktion und kann eine benutzerdefinierte Meldung sein, |

| Spaltenname | Datentyp | Beschreibung |
|-------------|----------|--|
| | | die an <code>isc_prepare_transaction2</code> übergeben wurde, auch wenn diese keine verteilte Transaktion ist. Diese kann Verwendung finden, wenn eine verlorene Verbindung nicht wiederhergestellt werden kann. |

RDB\$TRIGGERS

RDB\$TRIGGERS speichert Triggerdefinitionen für alle Tabellen und View.

| Spaltenname | Datentyp | Beschreibung |
|-----------------------|----------|---|
| RDB\$TRIGGER_NAME | CHAR(31) | Triggername |
| RDB\$RELATION_NAME | CHAR(31) | Der Name der Tabelle oder View zu der der Trigger gehört. NULL wenn der Trigger auf ein Datenbankereignis angewandt wird („database trigger“) |
| RDB\$TRIGGER_SEQUENCE | SMALLINT | Position dieses Triggers in der Sequenz. Null bedeutet normalerweise, dass keine Sequenzposition angegeben wurde. |
| RDB\$TRIGGER_TYPE | SMALLINT | Der Ereignistyp, bei dem der Trigger ausgelöst wird: 1 - before insert 2 - after insert 3 - before update 4 - after update 5 - before delete 6 - after delete 17 - before insert or update 18 - after insert or update 25 - before insert or delete 26 - after insert or delete 27 - before update or delete 28 - after update or delete 113 - before insert or update or delete 114 - after insert or update or delete 8192 - on connect 8193 - on disconnect 8194 - on transaction start 8195 - on transaction commit 8196 - on transaction rollback |

| Spaltenname | Datentyp | Beschreibung |
|---|-----------|--|
| Das Erkennen des genauen RDB \$TRIGGER_TYPE-Codes ist etwas komplizierter, da dies eine Bit-map nutzt, die für die Berechnung die Phase und das Ereignis sowie die Reihenfolge der Definitionen berücksichtigt. Für die Neugierigen wird die Kalkulation in Mark Rot-teveels Blog unter erklärt . | | |
| RDB\$TRIGGER_SOURCE | BLOB TEXT | Speichert den Quellcode des Triggers in PSQL. |
| RDB\$TRIGGER_BLR | BLOB BLR | Speichert den Quellcode des Triggers in Binärsprachenrepräsentation (BLR). |
| RDB\$DESCRIPTION | BLOB TEXT | Kommentartext zum Trigger. |
| RDB\$TRIGGER_INACTIVE | SMALLINT | Gibt an, ob der Trigger derzeit inaktiv (1) oder aktiv (0) ist. |
| RDB\$SYSTEM_FLAG | SMALLINT | Kennzeichen: Gibt an, ob der Trigger benutzer- (Wert 0) oder systemdefiniert (Wert 1 oder größer) ist. |
| RDB\$FLAGS | SMALLINT | Interne Verwendung |
| RDB\$VALID_BLR | SMALLINT | Gibt an, ob der Text des Triggers nach der letzten Änderung mittels ALTER TRIGGER gültig bleibt. |
| RDB\$DEBUG_INFO | BLOB | Beinhaltet Debugging-Informationen über die im Trigger genutzten Variablen. |

RDB\$TRIGGER_MESSAGES

RDB\$TRIGGER_MESSAGES speichert die Triggermeldungen.

| Spaltenname | Datentyp | Beschreibung |
|---------------------|---------------|--|
| RDB\$TRIGGER_NAME | CHAR(31) | Der Name des Triggers, zu dem die Meldung gehört |
| RDB\$MESSAGE_NUMBER | SMALLINT | Die Nummer der Meldung innerhalb des Triggers (von 1 bis 32.767) |
| RDB\$MESSAGE | VARCHAR(1023) | Text der Triggermeldung |

RDB\$TYPES

RDB\$TYPES speichert die definierten Listen enumerierter Typen, die im gesamten System verwendet werden.

| Spaltenname | Datentyp | Beschreibung |
|------------------|-----------|--|
| RDB\$FIELD_NAME | CHAR(31) | Enumerierter Typname. Jeder Typname beinhaltet seinen eigenen Typensatz, z.B. Objekttypen, Datentypen, Zeichensätze, Triggertypen, BLOB-Untertypen, etc. |
| RDB\$TYPE | SMALLINT | Die Kennung des Objekttyps. In jedem Aufzählungstyp wird eine eindeutige Zahlenreihe verwendet. Beispielsweise werden in der Auswahl aus dem unter RDB \$OBJECT_TYPE in RDB\$FIELD_NAME geführten Satz einige Objekttypen aufgelistet: 0 - TABLE 1 - VIEW 2 - TRIGGER 3 - COMPUTED_FIELD 4 - VALIDATION 5 - PROCEDURE ... |
| RDB\$TYPE_NAME | CHAR(31) | Der Name eines Elements eines Aufzählungstyps, z. B. TABELLE, ANSICHT, TRIGGER usw. im obigen Beispiel. Im Aufzählungstyp RDB \$CHARACTER_SET, speichert RDB \$TYPE_NAME die Namen der Zeichensätze. |
| RDB\$DESCRIPTION | BLOB TEXT | Beliebige Kommentartexte zu den Aufzählungstypen. |
| RDB\$SYSTEM_FLAG | SMALLINT | Kennzeichen: gibt an, ob das Typ-Element benutzer- (Wert 0) oder systemdefiniert (Wert 1 oder größer) ist. |

RDB\$USER_PRIVILEGES

RDB\$USER_PRIVILEGES speichert die SQL-Zugriffsprivilegien der Firebird-Benutzer und Privilegobjekte.

| Spaltenname | Datentyp | Beschreibung |
|--------------------|----------|--|
| RDB\$USER | CHAR(31) | Der Benutzer oder das Objekt, dem bzw. der diese Berechtigung erteilt wird. |
| RDB\$GRANTOR | CHAR(31) | Der Benutzer, der die Berechtigung erteilt. |
| RDB\$PRIVILEGE | CHAR(6) | Das hier gewährte Privileg: A - lle (alle Privilegien) S - select (Abfrage von Daten) I - insert (Datensätze einfügen) D - delete (Datensätze löschen) R - references (Fremdschlüssel) U - update (Datensätze aktualisieren) E - executing (Prozeduren) |
| RDB\$GRANT_OPTION | SMALLINT | Gibt an, ob die Berechtigung WITH GRANT OPTION im Privileg enthalten ist: 1 - enthalten 0 - nicht enthalten |
| RDB\$RELATION_NAME | CHAR(31) | Der Objektname (Tabelle, View, Prozedur oder Rolle) dem das Privileg zugewiesen wurde (ON). |
| RDB\$FIELD_NAME | CHAR(31) | Der Name der Spalte zu dem das Privileg gehört, für Spaltenbasierte Berechtigungen (ein UPDATE- oder REFERENCES-Privileg). |
| RDB\$USER_TYPE | SMALLINT | Gibt den Typ des Benutzers (ein Benutzer, eine Prozedur, eine View, etc.) an, dem das Privileg zugewiesen wurde (TO). |
| RDB\$OBJECT_TYPE | SMALLINT | Gibt den Typ des Objekts an, dem das Privileg zugewiesen wurde (ON). |

RDB\$VIEW_RELATIONS

RDB\$VIEW_RELATIONS speichert die Tabellen, die in der View-Definition referenziert werden. Pro Tabelle wird ein Datensatz verwendet.

| Spaltenname | Datentyp | Beschreibung |
|--------------------|----------|--|
| RDB\$VIEW_NAME | CHAR(31) | Viewname |
| RDB\$RELATION_NAME | CHAR(31) | Der Name der Tabelle, die in der View referenziert wird. |

| Spaltenname | Datentyp | Beschreibung |
|--------------------|-----------------|--|
| RDB\$VIEW_CONTEXT | SMALLINT | Der Alias, der für die View-Spalte im Code der Abfragedefinition in Binärsprachenrepräsentation (BLR) verwendet wird |
| RDB\$CONTEXT_NAME | CHAR(255) | Der Text, der mit dem in der Spalte RDB\$VIEW_CONTEXT gemeldeten Alias verknüpft ist. |

Anhang E: Monitoring Tables

Firebird überwacht Aktivitäten innerhalb der Datenbank und macht diese für Benutzerabfragen über die Monitoringtabellen verfügbar. Die Definitionen dieser Tabellen sind immer in der Datenbank verfügbar, alle mit dem Präfix MON\$. Die Tabellen sind virtuell: die Daten werden erst bei der Abfrage durch den Benutzer bereitgestellt. Dies ist auch der Grund, weshalb es keinen Sinn macht, Trigger hierfür zu erstellen!

Der Schlüssel zum Verstehen des Überwachungsmechanismus ist ein *activity snapshot*. Dieser Snapshot gibt den derzeitigen Status der Datenbank zu Beginn der Transaktion wieder, in welcher die Abfrage der Monitoringtabellen ausgeführt wird. Es werden einige Informationen über die Datenbank selbst, aktive Verbindungen, Benutzer, vorbereitete Transaktionen, laufende Abfragen und mehr zurückgegeben.

Der Snapshot wird erstellt, sobald die Monitoringtabelle das erste Mal abgefragt wird. Er wird bis zum Ende der aktuellen Transaktion vorgehalten. Damit wird eine stabile, konsistente Ansicht zum Abfragen über mehrere Tabellen gewährleistet. Das heißt, Monitoringtabellen verhalten sich immer als seien sie in SNAPSHOT TABLE STABILITY-Isolation („Konsistenz“), auch wenn die derzeitige Transaktion in einem niedrigeren Isolationslevel gestartet wird.

Um den Snapshot zu aktualisieren, muss die aktuelle Transaktion abgeschlossen werden und die Monitoringtabellen müssen in einem neuen Transaktionskontext erneut ausgeführt werden.

Zugriffssicherheit

- SYSDBA und der Datenbankbesitzer habe Vollzugriff auf alle Informationen, die über die Monitoringtabellen verfügbar sind
- Reguläre Benutzer können nur Informationen zu ihren eigenen Verbindungen einsehen; andere Verbindungen sind nicht sichtbar

Warnung

In stark belasteten Umgebungen kann das Sammeln von Informationen über die Monitoringtabellen einen negativen Einfluss auf die Systemleistung haben.

Tabelle E.1. Liste der Monitoringtabellen

| Systemtabelle | Inhalte |
|------------------------|---|
| MON\$ATTACHMENTS | Informationen über aktive Datenbankattachments |
| MON\$CALL_STACK | Stackaufrufe von aktiven Abfragen durch Stored Procedures und Trigger |
| MON\$CONTEXT_VARIABLES | Informationen zu benutzerdefinierten Kontextvariablen |
| MON\$DATABASE | Informationen über die Datenbank, welche durch die CURRENT_CONNECTION verbunden ist |
| MON\$IO_STATS | Input/Output-Statistiken |

| Systemtabelle | Inhalte |
|-------------------|--|
| MON\$MEMORY_USAGE | Statistiken über den Speicherverbrauch |
| MON\$RECORD_STATS | Record-Level-Statistiken |
| MON\$STATEMENTS | Zur Ausführung vorbereitete Statements |
| MON\$TRANSACTIONS | Gestartete Transaktionen |

MON\$ATTACHMENTS

MON\$ATTACHMENTS zeigt Informationen über aktive Attachments der Datenbank an.

| Spaltenname | Datentyp | Beschreibung |
|-------------------------|--------------|--|
| MON\$ATTACHMENT_ID | INTEGER | Verbindungs-Kennung |
| MON\$SERVER_PID | INTEGER | Serverprozess-Kennung |
| MON\$STATE | SMALLINT | Verbindungsstatus: 0 - Leerlauf (idle) 1 - Aktiv (active) |
| MON\$ATTACHMENT_NAME | VARCHAR(255) | Connection String—der Dateiname und volle Pfad zur primären Datenbankdatei |
| MON\$USER | CHAR(31) | Der Name des Benutzers, der mit diese Verbindung nutzt |
| MON\$ROLE | CHAR(31) | Der angegebene Rollenname zum Zeitpunkt des Verbindungsaufbaus. Wurde beim Aufbau der Verbindung keine Rolle angegeben, enthält das Feld den Text NONE |
| MON\$REMOTE_PROTOCOL | VARCHAR(10) | Name des Remote-Protokolls |
| MON\$REMOTE_ADDRESS | VARCHAR(255) | Remote-Adresse (Adresse und Servername) |
| MON\$REMOTE_PID | INTEGER | Kennung des Client-Prozesses |
| MON\$CHARACTER_SET_ID | SMALLINT | Kennung des Zeichensatzes (vgl. RDB\$CHARACTER_SET in der Systemtabelle RDB\$TYPES) |
| MON\$TIMESTAMP | TIMESTAMP | Datum und Zeit zum Zeitpunkt des Verbindungsaufbaus. |
| MON\$GARBAGE_COLLECTION | SMALLINT | Kennzeichen für Garbage Collection (wie in der Attachment DPB definiert): 1=erlaubt (allowed), 0=nicht erlaubt (not allowed) |

| Spaltenname | Datentyp | Beschreibung |
|---------------------|--------------|--|
| MON\$REMOTE_PROCESS | VARCHAR(255) | Der volle Dateiname und Pfad zu der ausführbaren Datei, die diese Verbindung aufgebaut hat |
| MON\$STAT_ID | INTEGER | Statistik-Kennung |

Verwendung von MON\$ATTACHMENTS um eine Verbindung zu beenden

Monitoringtabellen sind nur-lesend. Jedoch hat der Server einen eingebauten Mechanismus, um Datensätze zu löschen (und nur zum Löschen) in der Tabelle MON\$ATTACHMENTS, wodurch es möglich wird, Datenbankverbindungen zu beenden.

Hinweis

- Sämtliche Aktivitäten der beendeten Verbindung werden augenblicklich gestoppt und alle aktiven Transaktionen werden zurückgerollt
- Die beendete Verbindung gibt einen Fehler mit dem Code `isc_att_shutdown` zurück
- Versuche diese Verbindung weiterzuverwenden, wird ebenfalls Fehler zurückgeben.

Beispiel: Alle Verbindungen außer der eigenen (current) beenden:

```
DELETE FROM MON$ATTACHMENTS
WHERE MON$ATTACHMENT_ID <> CURRENT_CONNECTION
```

MON\$CALL_STACK

MON\$CALL_STACK zeigt Aufrufe des Stacks durch Abfragen von Stored Procedures und Trigger an.

| Spaltenname | Datentyp | Beschreibung |
|-------------------|----------|---|
| MON\$CALL_ID | INTEGER | Aufruf-Kennzeichen |
| MON\$STATEMENT_ID | INTEGER | Das Kennzeichen des Top-Level-SQL-Statements. Dies ist das Statement, das die Kette der Aufrufe initialisiert hat. Nutzen Sie diese Kennung um die aktiven Statements in der MON\$STATEMENTS-Tabelle zu finden. |
| MON\$CALLER_ID | INTEGER | Die Kennung der aufrufenden Stored Procedure oder des aufrufenden Triggers |

| Spaltenname | Datentyp | Beschreibung |
|--------------------|-----------|---|
| MON\$OBJECT_NAME | CHAR(31) | PSQL-Objekt-Name (Module) |
| MON\$OBJECT_TYPE | SMALLINT | PSQL-Objekt-Typ (Trigger oder Stored Procedure): 2 - Trigger 5 - Stored Procedure |
| MON\$TIMESTAMP | TIMESTAMP | Datum und Zeitpunkt des Aufrufs |
| MON\$SOURCE_LINE | INTEGER | Die Zeilennummer im SQL-Statement, welches zum Zeitpunkt des Snapshots gestartet wurde |
| MON\$SOURCE_COLUMN | INTEGER | Die Spaltennummer im SQL-Statement, welches zum Zeitpunkt des Snapshots gestartet wurde |
| MON\$STAT_ID | INTEGER | Statistik-Kennung |

EXECUTE STATEMENT-Aufrufe: Informationen über Aufrufe, die mittels EXECUTE STATEMENT ausgeführt wurden, erscheinen nicht im Aufruf-Stack.

Beispiel zur Verwendung von MON\$CALL_STACK: Ermitteln des Aufruf-Stack für alle Verbindungen außer der eigenen:

```
WITH RECURSIVE
  HEAD AS (
    SELECT
      CALL.MON$STATEMENT_ID, CALL.MON$CALL_ID,
      CALL.MON$OBJECT_NAME, CALL.MON$OBJECT_TYPE
    FROM MON$CALL_STACK CALL
    WHERE CALL.MON$CALLER_ID IS NULL
    UNION ALL
    SELECT
      CALL.MON$STATEMENT_ID, CALL.MON$CALL_ID,
      CALL.MON$OBJECT_NAME, CALL.MON$OBJECT_TYPE
    FROM MON$CALL_STACK CALL
    JOIN HEAD ON CALL.MON$CALLER_ID = HEAD.MON$CALL_ID
  )
SELECT MON$ATTACHMENT_ID, MON$OBJECT_NAME, MON$OBJECT_TYPE
FROM HEAD
JOIN MON$STATEMENTS STMT ON STMT.MON$STATEMENT_ID = HEAD.MON$STATEMENT_ID
WHERE STMT.MON$ATTACHMENT_ID <> CURRENT_CONNECTION
```

MON\$CONTEXT_VARIABLES

MON\$CONTEXT_VARIABLES zeigt Infos über benutzerdefinierte Kontextvariablen an.

| Spaltenname | Datentyp | Beschreibung |
|---------------------|--------------|--|
| MON\$ATTACHMENT_ID | INTEGER | Verbindungskennung. Gültiger Wert nur für Variablen auf Verbindungsebene. Für Transaktionsebenen ist der Variablenwert NULL. |
| MON\$TRANSACTION_ID | INTEGER | Transaktionskennung. Gültiger Wert nur auf Transaktionsebene. Für Verbindungsebenen ist der Variablenwert NULL. |
| MON\$VARIABLE_NAME | VARCHAR(80) | Name der Kontextvariable |
| MON\$VARIABLE_VALUE | VARCHAR(255) | Wert der Kontextvariable |

MON\$DATABASE

MON\$DATABASE zeigt Header-Daten der Datenbank an, mit der der aktuelle Benutzer verbunden ist.

| Spaltenname | Datentyp | Beschreibung |
|-------------------------|--------------|--|
| MON\$DATABASE_NAME | VARCHAR(255) | Name und voller Pfad der primären Datenbankdatei oder der Datenbank-Alias. |
| MON\$PAGE_SIZE | SMALLINT | Datenbank Seitengröße in Bytes. |
| MON\$ODS_MAJOR | SMALLINT | Haupt-ODS-Version, z.B. 11 |
| MON\$ODS_MINOR | SMALLINT | Unter-ODS-Version, z.B. 11 |
| MON\$OLDEST_TRANSACTION | INTEGER | Nummer der ältesten (relevanten) Transaktion (oldest [interesting] transaction (OIT)) |
| MON\$OLDEST_ACTIVE | INTEGER | Nummer der ältesten aktiven Transaktion (oldest active transaction (OAT)) |
| MON\$OLDEST_SNAPSHOT | INTEGER | Nummer der Transaktion, die zum Zeitpunkt der OAT aktiv war - älteste Snapshot Transaktion (oldest snapshot transaction (OST)) |
| MON\$NEXT_TRANSACTION | INTEGER | Nummer der nächsten Transaktion zum Zeitpunkt als der Monitoring-Snapshot erstellt wurde |
| MON\$PAGE_BUFFERS | INTEGER | Die Anzahl der Seiten, die im Speicher für den Datenbank Seiten-Cache (page cache) zugewiesen wurden |
| MON\$SQL_DIALECT | SMALLINT | SQL-Dialekt der Datenbank: 1 oder 3 |

| Spaltenname | Datentyp | Beschreibung |
|---------------------|-----------|--|
| MON\$SHUTDOWN_MODE | SMALLINT | Der derzeitige Shutdown-Status der Datenbank: 0 - Die Datenbank ist online 1 - Multi-User Shutdown 2 - Single-User Shutdown 3 - Kompletter Shutdown |
| MON\$SWEEP_INTERVAL | INTEGER | Sweep-Intervall |
| MON\$READ_ONLY | SMALLINT | Dieses Kennzeichen gibt an, ob die Datenbank im Modus read-only (Wert 1) oder read-write (Wert 0) arbeitet. |
| MON\$FORCED_WRITES | SMALLINT | Gibt an, ob der Schreibmodus der Datenbank auf synchrones Schreiben (forced writes ON, Wert ist 1) oder asynchrones Schreiben (forced writes OFF, Wert ist 0) gestellt ist |
| MON\$RESERVE_SPACE | SMALLINT | Gibt an, ob reserve_space (Wert 1) oder use_all_space (Wert 0) zum Füllen der Datenbankseiten verwendet wird. |
| MON\$CREATION_DATE | TIMESTAMP | Datum und Zeit zu der die Datenbank erstellt oder wiederhergestellt wurde. |
| MON\$PAGES | BIGINT | Anzahl der zugewiesenen Seiten der Datenbank auf einem externen Gerät |
| MON\$STAT_ID | INTEGER | Statistik-Kennung |
| MON\$BACKUP_STATE | SMALLINT | Derzeitiger physikalischer Backup-Status (nBackup): 0 - normal 1 - stalled 2 - merge |

MON\$IO_STATS

MON\$IO_STATS zeigt Input/Output-Statistiken an. Die Zähler arbeiten kumulativ, gruppiert für jede Statistikgruppe.

| Spaltenname | Datentyp | Beschreibung |
|--------------|----------|-------------------|
| MON\$STAT_ID | INTEGER | Statistik-Kennung |

| Spaltenname | Datentyp | Beschreibung |
|-------------------|----------|---|
| MON\$STAT_GROUP | SMALLINT | Statistik-Gruppe: 0 - Datenbank 1 - Verbindung 2 - Transaktion 3 - Statement 4 - Aufruf (Call) |
| MON\$PAGE_READS | BIGINT | Anzahl der gelesenen Datenbankseiten |
| MON\$PAGE_WRITES | BIGINT | Anzahl der geschriebenen Datenbankseiten |
| MON\$PAGE_FETCHES | BIGINT | Anzahl der geholten (fetched) Datenbankseiten |
| MON\$PAGE_MARKS | BIGINT | Anzahl der markierten Datenbankseiten |

MON\$MEMORY_USAGE

MON\$MEMORY_USAGE zeigt Statistiken zu Speichernutzung an.

| Spaltenname | Datentyp | Beschreibung |
|-----------------------|----------|--|
| MON\$STAT_ID | INTEGER | Statistik-Kennung |
| MON\$STAT_GROUP | SMALLINT | Statistik-Gruppen: 0 - Datenbank 1 - Verbindung 2 - Transaktion 3 - Statement 4 - Aufruf (Call) |
| MON\$MEMORY_USED | BIGINT | Die Größe des genutzten Speichers in Bytes. Diese Daten beziehen sich auf die höchste Speicherzuteilung, die vom Server abgerufen wird. Dies ist nützlich, um Speicherlecks und exzessiven Speicherverbrauch in Verbindungen, Prozeduren, etc. zu ermitteln. |
| MON\$MEMORY_ALLOCATED | BIGINT | Die Größe des Speichers, der durch das Betriebssystem zugeteilt wurde. Angabe in Bytes. Diese Daten beziehen sich auf die Low-Level-Zuweisung von Speicher, die durch den Firebird Speicher-Manager abgerufen wird—die Größe des Speichers zuge- |

| Spaltenname | Datentyp | Beschreibung |
|---------------------------|----------|--|
| | | wiesen durch das Betriebssystem— womit Sie die physikalischen Speicherbedarf steuern können. |
| MON\$MAX_MEMORY_USED | BIGINT | Der größte Speicherverbrauch für dieses Objekt in Bytes. |
| MON\$MAX_MEMORY_ALLOCATED | BIGINT | Die größte Speicherreservierung für dieses Objekt durch das Betriebssystem in Bytes. |

Anmerkung

Nicht alle Datensätze dieser Tabelle haben nicht-null-Werte. MON\$DATABASE und Objekte in Beziehung auf Speicherzuweisungen haben nicht-null-Werte. Kleinere Speicherzuordnungen werden hier nicht angeführt, sondern dem Datenbankspeicherpool zugewiesen.

MON\$RECORD_STATS

MON\$RECORD_STATS zeigt Datensatz-Level-Statistiken an. Die Zähler arbeiten kumulativ, gruppiert für jede Statistikgruppe.

| Spaltenname | Datentyp | Beschreibung |
|-----------------------|----------|--|
| MON\$STAT_ID | INTEGER | Statistik-Kennung |
| MON\$STAT_GROUP | SMALLINT | Statistik-Gruppen: 0 - Datenbank 1 - Verbindung 2 - Transaktion 3 - Statement 4 - Aufruf (Call) |
| MON\$RECORD_SEQ_READS | BIGINT | Anzahl der sequenziell gelesenen Datensätze |
| MON\$RECORD_IDX_READS | BIGINT | Anzahl der mittels Index gelesenen Datensätze |
| MON\$RECORD_INSERTS | BIGINT | Anzahl der eingefügten Datensätze |
| MON\$RECORD_UPDATES | BIGINT | Anzahl der aktualisierten Datensätze |
| MON\$RECORD_DELETES | BIGINT | Anzahl der gelöschten Datensätze |
| MON\$RECORD_BACKOUTS | BIGINT | Anzahl der Datensätze für die eine neue primäre Datensatzversion während eines Rollbacks oder Savepoint-Undo erstellt wurde. |

| Spaltenname | Datentyp | Beschreibung |
|----------------------|----------|---|
| MON\$RECORD_PURGES | BIGINT | Anzahl der Datensätze für die die Versionskette nicht länger von der OAT (oldest active transaction) oder jüngeren Transaktionen benötigt wird. |
| MON\$RECORD_EXPUNGES | BIGINT | Anzahl der Datensätze, in denen die Versionskette aufgrund von Löschungen innerhalb von Transaktionen gelöscht wird, die älter als die OAT (oldest active transaction) sind |

MON\$STATEMENTS

MON\$STATEMENTS zeigt Statments an, die für die Ausführung vorbereitet wurden.

| Spaltenname | Datentyp | Beschreibung |
|---------------------|-----------|--|
| MON\$STATEMENT_ID | INTEGER | Statement-Kennung |
| MON\$ATTACHMENT_ID | INTEGER | Verbindungs-Kennung |
| MON\$TRANSACTION_ID | INTEGER | Transaktions-Kennung |
| MON\$STATE | SMALLINT | Statement-Status: 0 - Leerlauf (idle) 1 - Aktiv 2 - verzögert (stalled) |
| MON\$TIMESTAMP | TIMESTAMP | Der Zeitpunkt an dem das Statement vorbereitet wurde. |
| MON\$SQL_TEXT | BLOB TEXT | Statement-Text in SQL |
| MON\$STAT_ID | INTEGER | Statistik-Kennung |

Der Status STALLED gibt an, dass das Statement zum Zeitpunkt des Snapshots einen offenen Cursor besaß und auf den Client wartet, der weitere Datensätze abrufen.

MON\$STATEMENTS zum Stoppen einer Abfrage nutzen

Monitoringtabellen können nur gelesen werden. Jedoch hat der Server einen eingebauten Mechanismus um Datensätze in der Tabelle MON\$STATEMENTS zu löschen (und nur zum Löschen), womit es möglich ist, laufende Abfragen zu stoppen.

Hinweis

- Werden derzeit keine Abfragen in der Verbindung ausgeführt, so wird jeder Versuch eine Abfrage zu stoppen, nicht funktionieren.
- Nachdem eine Abfrage gestoppt wurde, werden Execute- und Fetch-Aufrufe der API den Fehlercode `isc_cancelled` zurückgeben.
- Nachfolgende Abfragen laufen normal weiter.

Beispiel: Stoppen aller aktiven Abfragen der angegebenen Verbindung

```
DELETE FROM MON$STATEMENTS
WHERE MON$ATTACHMENT_ID = 32
```

MON\$TRANSACTIONS

MON\$TRANSACTIONS gibt Auskunft über gestartete Transaktionen.

| Spaltenname | Datentyp | Beschreibung |
|-------------------------|-----------|--|
| MON\$TRANSACTION_ID | INTEGER | Transaktions-Kennung |
| MON\$ATTACHMENT_ID | INTEGER | Verbindungs-Kennung |
| MON\$STATE | SMALLINT | Transaktionsstatus: 0 - Leerlauf (idle) 1 - Aktiv |
| MON\$TIMESTAMP | TIMESTAMP | Zeitpunkt an dem die Transaktion gestartet wurde |
| MON\$TOP_TRANSACTION | INTEGER | Top-Level-Transaktionsnummer (Kennung) |
| MON\$OLDEST_TRANSACTION | INTEGER | Kennung der ältesten relevanten Transaktion (oldest [interesting] transaction (OIT)) |
| MON\$OLDEST_ACTIVE | INTEGER | Kennung der ältesten aktiven Transaktion (oldest active transaction (OAT)) |
| MON\$ISOLATION_MODE | SMALLINT | Isolationsmoduls (Level): 0 - Konsistenz (Snapshot für Tabellenstabilität) 1 - Konkurrierend (Snapshot) 2 - Read Committed mit Datensatzversion 3 - Read Committed ohne Datensatzversion |
| MON\$LOCK_TIMEOUT | SMALLINT | Lock-Timeout: |

Monitoring Tables

| Spaltenname | Datentyp | Beschreibung |
|------------------|----------|---|
| | | -1 - warten (ewig) 0 - nicht warten 1 oder größer - Lock-Timeout in Sekunden |
| MON\$READ_ONLY | SMALLINT | Gibt an, ob die Transaktion nur-lesend (Wert 1) oder lesend-schreibend (Wert 0) läuft |
| MON\$AUTO_COMMIT | SMALLINT | Gibt an, ob automatisches Commit für die Transaktion verwendet wird (Wert 1) oder nicht (Wert 0) |
| MON\$AUTO_UNDO | SMALLINT | Gibt an, ob der Logging-Mechanismus <i>automatisches Undo</i> für die Transaktion verwendet wird (Wert 1) oder nicht (Wert 0) |
| MON\$STAT_ID | INTEGER | Statistik-Kennung |

Anhang F: Zeichensätze und Collations

Tabelle F.1. Zeichensätze und Collations

| Zeichensatz | ID | Bytes pro Zeichen | Sortierung | Sprache |
|-------------|----|-------------------|----------------|---------------------------------------|
| ASCII | 2 | 1 | ASCII | Englisch |
| BIG_5 | 56 | 2 | BIG_5 | Chinesisch, Vietnamesisch, Koreanisch |
| CP943C | 68 | 2 | CP943C | Japanisch |
| " | " | " | CP943C_UNICODE | Japanisch |
| CYRL | 50 | 1 | CYRL | Russisch |
| " | " | " | DB_RUS | Russisch dBase |
| " | " | " | PDOX_CYRL | Russisch Paradox |
| DOS437 | 10 | 1 | DOS437 | U.S. Englisch |
| " | " | " | DB_DEU437 | Deutsch dBase |
| " | " | " | DB_ESP437 | Spanisch dBase |
| " | " | " | DB_FIN437 | Finnisch dBase |
| " | " | " | DB_FRA437 | Französisch dBase |
| " | " | " | DB_ITA437 | Italienisch dBase |
| " | " | " | DB_NLD437 | Niederländisch dBase |
| " | " | " | DB_SVE437 | Schwedisch dBase |
| " | " | " | DB_UK437 | Englisch (Groß Britanien) dBase |
| " | " | " | DB_US437 | U.S. Englisch dBase |
| " | " | " | PDOX_ASCII | Code page Paradox—ASCII |
| " | " | " | PDOX_SWEDFIN | Schwedisch / Finnisch Paradox |
| " | " | " | PDOX_INTL | International Englisch Paradox |
| DOS737 | 9 | 1 | DOS737 | Griechisch |
| DOS775 | 15 | 1 | DOS775 | Baltic |
| DOS850 | 11 | 1 | DOS850 | Latin I (ohne Euro-Zeichen) |

| Zeichensatz | ID | Bytes pro Zeichen | Sortierung | Sprache |
|--------------------|-----------|--------------------------|-------------------|----------------------------|
| " | " | " | DB_DEU850 | Deutsch |
| " | " | " | DB_ESP850 | Spanisch |
| " | " | " | DB_FRA850 | Französisch |
| " | " | " | DB_FRC850 | Französisch—Kanada |
| " | " | " | DB_ITA850 | Italienisch |
| " | " | " | DB_NLD850 | Niederländisch |
| " | " | " | DB_PTB850 | Portugisisch—Brasilien |
| " | " | " | DB_SVE850 | Schwedisch |
| " | " | " | DB_UK850 | Englisch—Groß Britanien |
| " | " | " | DB_US850 | U.S. Englisch |
| DOS852 | 45 | 1 | DOS852 | Latin II |
| " | " | " | DB_CSJ | Tschechisch dBase |
| " | " | " | DB_PLK | Polish dBase |
| " | " | " | DB_SLO | Slovak dBase |
| " | " | " | PDOX_CSJ | Tschechisch Paradox |
| " | " | " | PDOX_HUN | Hungarian Paradox |
| " | " | " | PDOX_PLK | Polish Paradox |
| " | " | " | PDOX_SLO | Slovak Paradox |
| DOS857 | 46 | 1 | DOS857 | Türkisch |
| " | " | " | DB_TRK | Türkisch dBase |
| DOS858 | 16 | 1 | DOS858 | Latin I (mit Euro-Zeichen) |
| DOS860 | 13 | 1 | DOS860 | Portugisisch |
| " | " | " | DB_PTG860 | Portugisisch dBase |
| DOS861 | 47 | 1 | DOS861 | Isländisch |
| " | " | " | PDOX_ISL | Isländisch Paradox |
| DOS862 | 17 | 1 | DOS862 | Hebräisch |
| DOS863 | 14 | 1 | DOS863 | Französisch—Kanada |
| " | " | " | DB_FRC863 | Französisch dBase—Kanada |
| DOS864 | 18 | 1 | DOS864 | Arabisch |

| Zeichensatz | ID | Bytes pro Zeichen | Sortierung | Sprache |
|--------------------|-----------|--------------------------|-------------------|---|
| DOS865 | 12 | 1 | DOS865 | Skandinavisch |
| " | " | " | DB_DAN865 | Dänisch dBase |
| " | " | " | DB_NOR865 | Norwegisch dBase |
| " | " | " | PDOX_NORDAN4 | Paradox Norwegen und Dänemark |
| DOS866 | 48 | 1 | DOS866 | Russisch |
| DOS869 | 49 | 1 | DOS869 | Modern Griechisch |
| EUCJ_0208 | 6 | 2 | EUCJ_0208 | Japanisch EUC |
| GB_2312 | 57 | 2 | GB_2312 | Vereinfachtes chinesisches (Hong Kong, Korea) |
| GB18030 | 69 | 4 | GB18030 | Chinesisch |
| " | " | " | GB18030_UNICODE | Chinesisch |
| GBK | 67 | 2 | GBK | Chinesisch |
| " | " | " | GBK_UNICODE | Chinesisch |
| ISO8859_1 | 21 | 1 | ISO8859_1 | Latin I |
| " | " | " | DA_DA | Dänisch |
| " | " | " | DE_DE | Deutsch |
| " | " | " | DU_NL | Niederländisch |
| " | " | " | EN_UK | Englisch—Groß Britanien |
| " | " | " | EN_US | U.S. Englisch |
| " | " | " | ES_ES | Spanisch |
| " | " | " | ES_ES_CI_AI | Spanisch—Groß- und Kleinschreibungsunabhängig sowie Akzentunabhängig |
| " | " | " | FI_FI | Finnisch |
| " | " | " | FR_CA | Französisch—Kanada |
| " | " | " | FR_FR | Französisch |
| " | " | " | FR_FR_CI_AI | Französisch—Groß- und Kleinschreibungsunabhängig sowie Akzentunabhängig |
| " | " | " | IS_IS | Isländisch |
| " | " | " | IT_IT | Italienisch |

| Zeichensatz | ID | Bytes pro Zeichen | Sortierung | Sprache |
|--------------------|-----------|--------------------------|-------------------|--|
| " | " | " | NO_NO | Norwegisch |
| ISO8859_1 | " | " | PT_PT | Portugiesisch |
| " | " | " | PT_BR | Portugiesisch—Brasilien |
| " | " | " | SV_SV | Schwedisch |
| ISO8859_2 | 22 | 1 | ISO8859_2 | Latin 2—Zentraleuropa (Kroatisch, tschechisch, ungarisch, polnisch, romanisch, serbisch, slowakisch, slowenisch) |
| " | " | " | CS_CZ | Tschechisch |
| " | " | " | ISO_HUN | Hungarian |
| " | " | " | ISO_PLK | Polish |
| ISO8859_3 | 23 | 1 | ISO8859_3 | Latin 3—Südeuropa (Malta, Esperanto) |
| ISO8859_4 | 34 | 1 | ISO8859_4 | Latin 4—Nordeuropa (Estnisch, lettisch, litauisch, grönländisch, lappisch) |
| ISO8859_5 | 35 | 1 | ISO8859_5 | Kyrillisch (Russisch) |
| ISO8859_6 | 36 | 1 | ISO8859_6 | Arabisch |
| ISO8859_7 | 37 | 1 | ISO8859_7 | Griechisch |
| ISO8859_8 | 38 | 1 | ISO8859_8 | Hebräisch |
| ISO8859_9 | 39 | 1 | ISO8859_9 | Latin 5 |
| ISO8859_13 | 40 | 1 | ISO8859_13 | Latin 7—Baltikum |
| " | " | " | LT_LT | Litauisch |
| KOI8R | 63 | 1 | KOI8R | Russisch—Wörterbuchsartierung |
| " | " | " | KOI8R_RU | Russisch |
| KOI8U | 64 | 1 | KOI8U | Ukrainisch—Sortierung nach Wörterbuch |
| " | " | " | KOI8U_UA | Ukrainisch |
| KSC_5601 | 44 | 2 | KSC_5601 | Koreanisch |
| " | " | " | KSC_DICTIONARY | Koreanisch—Sortierung nach Wörterbuch |
| NEXT | 19 | 1 | NEXT | Coding NeXTSTEP |
| " | " | " | NXT_DEU | Deutsch |
| " | " | " | NXT_ESP | Spanisch |
| " | " | " | NXT_FRA | Französisch |

| Zeichensatz | ID | Bytes pro Zeichen | Sortierung | Sprache |
|--------------------|-----------|--------------------------|-------------------|---|
| " | " | " | NXT_ITA | Italienisch |
| NEXT | 19 | 1 | NXT_US | U.S. Englisch |
| NONE | 0 | 1 | NONE | Neutrale code page. Umwandlung in Großschreibung wird nur für ASCII-Codes 97-122 durchgeführt. Empfehlung: Zeichensatz vermeiden. |
| OCTETS | 1 | 1 | OCTETS | Binäre Zeichenkodierung |
| SJIS_0208 | 5 | 2 | SJIS_0208 | Japanisch |
| TIS620 | 66 | 1 | TIS620 | Thailändisch |
| " | " | " | TIS620_UNICODE | Thailändisch |
| UNICODE_FSS | 3 | 3 | UNICODE_FSS | Alle englischen |
| UTF8 | 4 | 4 | UTF8 | Alle durch Unicode 4.0 unterstützte Sprachen |
| " | " | " | USC_BASIC | Alle durch Unicode 4.0 unterstützte Sprachen |
| " | " | " | UNICODE | Alle durch Unicode 4.0 unterstützte Sprachen |
| " | " | " | UNICODE_CI | Alle durch Unicode 4.0 unterstützte Sprachen—Groß- und Kleinschreibunabhängig |
| " | " | " | UNICODE_CI_AI | Alle durch Unicode 4.0 unterstützte Sprachen—Groß- und Kleinschreibunabhängig |
| WIN1250 | 51 | 1 | WIN1250 | ANSI—Zentraleuropa |
| " | " | " | BS_BA | Bosnisch |
| " | " | " | PXW_CSY | Ungarisch |
| " | " | " | PXW_HUN | Ungarisch—Sortierung nach Wörterbuch |
| " | " | " | PXW_HUNDC | Polnisch |
| " | " | " | PXW_PLK | Slovakisch |
| " | " | " | PXW_SLOV | Slowenisch |
| " | " | " | WIN_CZ | Tschechisch |
| " | " | " | WIN_CZ_CI | Tschechisch—Groß- und Kleinschreibungsunabhängig |
| " | " | " | WIN_CZ_CI_AI | Tschechisch—Groß- und Kleinschreibungsunabhängig und Akzent-unabhängig |

| Zeichensatz | ID | Bytes pro Zeichen | Sortierung | Sprache |
|--------------------|-----------|----------------------------------|-------------------|--------------------------------------|
| WIN1251 | 52 | 1 | WIN1251 | ANSI Kyrillisch |
| " | " | " | WIN1251_UA | Ukrainisch |
| " | " | " | PXW_CYRL | Paradox Kyrillisch (Russisch) |
| WIN1252 | 53 | 1 | WIN1252 | ANSI—Latin I |
| " | " | " | PXW_INTL | Englisch international |
| " | " | " | PXW_INTL850 | Paradox mehrsprachig Latin I |
| " | " | " | PXW_NORDAN4 | Norwegisch und Dänisch |
| " | " | " | PXW_SPAN | Paradox Spanisch |
| " | " | " | PXW_SWEDFIN | Schwedisch und Finnisch |
| " | " | " | WIN_PTBR | Portugiesisch—Brasilianisch |
| WIN1253 | 54 | 1 | WIN1253 | ANSI Griechisch |
| " | " | " | PXW_GREEK | Paradox Griechisch |
| WIN1254 | 55 | 1 | WIN1254 | ANSI Türkisch |
| " | " | " | PXW_TURK | Paradox Türkisch |
| WIN1255 | 58 | 1 | WIN1255 | ANSI Hebräisch |
| WIN1256 | 59 | 1 | WIN1256 | ANSI Arabisch |
| WIN1257 | 60 | 1 | WIN1257 | ANSI Baltisch |
| " | " | " | WIN1257_EE | Estnisch—Sortierung nach Wörterbuch |
| " | " | " | WIN1257_LT | Litauisch—Sortierung nach Wörterbuch |
| " | " | " | WIN1257_LV | Lettisch—Sortierung nach Wörterbuch |
| WIN1258 | 65 | 1 | WIN1258 | Vietnamesisch |

Anhang G: Lizenzhinweise

Die Inhalte dieser Dokumentation sind Gegenstand der Public Documentation License Version 1.0 (die „Lizenz“); Sie dürfen diese Dokumentation nur verwenden, sofern Sie die Bedingungen dieser Lizenz akzeptieren. Kopien der Lizenz sind verfügbar unter <http://www.firebirdsql.org/pdfmanual/pdl.pdf> (PDF) und <http://www.firebirdsql.org/manual/pdl.html> (HTML).

Die originale Dokumentation wurde unter dem Titel *Firebird 2.5 Language Reference* erfasst.

Die ursprünglichen Schreiber der Originaldokumentation sind: Paul Vinkenoog, Dmitry Yemanov und Thomas Woinke. Verfasser der ursprünglichen Texte in russisch sind Denis Simonov, Dmitry Filippov, Alexander Karpeykin, Alexey Kovyazin und Dmitry Kuzmenko.

Copyright (C) 2008-2015. Alle Rechte vorbehalten. Kontakt mit den Verfassern: paul at vinkenoog dot nl.

Verfasser und Bearbeiter des inkludierten PDL-lizenzierten Materials sind: J. Beesley, Helen Borrie, Arno Brinkman, Frank Ingermann, Vlad Khorsun, Alex Peshkov, Nikolay Samofatov, Adriano dos Santos Fernandes, Dmitry Yemanov.

Enthaltene Teile unterliegen dem Copyright (C) 2001-2015 ihrer jeweiligen Verfasser. Alle Rechte vorbehalten.

Anhang H: Dokumenthistorie

Die genaue Dateihistorie wird im `firebird-documentation`-Repository innerhalb unseres Git-Baumes geführt; zu finden unter <https://github.com/FirebirdSQL/firebird-documentation/commits/master>

Versionsgeschichte

- | | | | |
|----------|-------------------|----------|--|
| 1.000-de | 11. März 2018 | MK | Deutsche Übersetzung basierend auf der englischen Dokumentenversion 1.000. |
| 1.000 | 11. August 2017 | H.E.M.B. | Während des letzten Überprüfungszeitraums wurden keine weiteren Änderungen am Inhalt vorgenommen. Die Kapitel DML, PSQL, Funktionen und Variablen, Transaktionen und Sicherheit wurden in dieser Phase nicht überprüft. |
| 0.906 | 11. August 2016 | H.E.M.B. | Mehrere Revisionen wurden im Laufe des Jahres als Beta 1 veröffentlicht, mit Überprüfungen verschiedener Abschnitte von Paul Vinkenoog, Aage Johansen und Mark Rotteveel. Diese Überarbeitung (0.906) wartet auf die endgültige Überarbeitung einiger späterer Abschnitte, die in rot kursiv als „Editor's Note“ markiert sind. |
| 1.1 | 1. September 2015 | H.E.M.B. | Das Original stammt aus dem Russischen, übersetzt durch Dmitry Borodin (MegaTranslations). Rohübersetzungen bearbeitet und in Doc-Book konvertiert, in dieser Revision (1.1) durch Helen Borrie. Diese Revision wurde nur als PDF-Datei verteilt, für die Überprüfung von Dmitry Yemanov, u.a. Reviewer, bitte beachten Sie Kommentare wie diese: <i>Editor's note :: The sky is falling, take cover!</i> |