



Руководство по языку SQL СУБД Firebird

(Охватывает Firebird до версии 2.5.3 включительно)

v.0311201403

Спонсоры документации:



Platinum Sponsor

**МОСКОВСКАЯ
БИРЖА**

Gold Sponsor

IBSurgeon

Оглавление

Введение	10
Что содержит данный документ	10
Авторство	11
Спонсоры	11
Благодарности	11
Лицензионные замечания	12
Обновления	12
Подмножества SQL	13
Действия при ошибках	13
Диалекты SQL	15
Типы и подтипы данных	16
Целочисленные данные	18
SMALLINT	18
INTEGER	18
BIGINT	18
Типы данных с плавающей точкой	19
FLOAT	19
DOUBLE PRECISION	20
Типы данных с фиксированной точкой	20
NUMERIC	20
DECIMAL	21
Типы данных для работы с датой и временем	21
DATE	22
TIME	22
TIMESTAMP	22
Символьные типы данных	22
CHAR	26
VARCHAR	26
NCHAR	26
Бинарные типы данных	27
Массивы	28
Специальные типы данных	30
Тип данных SQL_NULL	30
Преобразование типов данных	31
Явное преобразование типов данных	31
Неявное преобразование типов данных	33
Сокращённое приведение типов даты и времени (datetime)	34
Работа с доменами	34
Создание доменов	36
Изменение доменов	36
Удаление доменов	37
Операторы DDL	38
DATABASE	38

Руководство по языку SQL СУБД Firebird

CREATE DATABASE.....	38
CONNECT	42
ALTER DATABASE	45
DROP DATABASE.....	48
SHADOW.....	49
CREATE SHADOW.....	49
DROP SHADOW.....	51
DOMAIN	52
CREATE DOMAIN	53
ALTER DOMAIN	57
DROP DOMAIN	61
TABLE	62
CREATE TABLE	62
ALTER TABLE	74
DROP TABLE	80
RECREATE TABLE	81
INDEX.....	82
CREATE INDEX.....	82
ALTER INDEX.....	85
DROP INDEX.....	87
SET STATISTICS	88
VIEW	89
CREATE VIEW	89
ALTER VIEW	92
CREATE OR ALTER VIEW.....	94
DROP VIEW.....	95
RECREATE VIEW	96
TRIGGER	97
CREATE TRIGGER	97
ALTER TRIGGER	102
CREATE OR ALTER TRIGGER	105
DROP TRIGGER	106
RECREATE TRIGGER	107
PROCEDURE.....	108
CREATE PROCEDURE.....	108
ALTER PROCEDURE.....	112
CREATE OR ALTER PROCEDURE	114
DROP PROCEDURE	115
RECREATE PROCEDURE	116
EXTERNAL FUNCTION	117
DECLARE EXTERNAL FUNCTION.....	117
ALTER EXTERNAL FUNCTION	120
DROP EXTERNAL FUNCTION	121
FILTER	122
DECLARE FILTER	122
DROP FILTER	124
SEQUENCE (GENERATOR).....	124
CREATE SEQUENCE (GENERATOR).....	125

Руководство по языку SQL СУБД Firebird

ALTER SEQUENCE.....	126
SET GENERATOR	127
DROP SEQUENCE (GENERATOR).....	128
EXCEPTION.....	129
CREATE EXCEPTION.....	129
ALTER EXCEPTION.....	130
CREATE OR ALTER EXCEPTION.....	130
DROP EXCEPTION.....	131
RECREATE EXCEPTION.....	132
COLLATION.....	132
CREATE COLLATION.....	133
DROP COLLATION.....	136
CHARACTER SET.....	137
ALTER CHARACTER SET.....	137
Операторы DML.....	139
SELECT.....	139
FIRST, SKIP и ROWS.....	140
Особенности использования.....	141
Примеры FIRST и SKIP.....	142
ROWS.....	143
Примеры с ROWS.....	144
Список полей SELECT.....	145
Примеры операторов SELECT с различными типами полей.....	147
Выборка в переменные с помощью INTO.....	149
Выражение FROM.....	150
Выборка из селективной хранимой процедуры.....	152
Выборка из производной таблицы (derived table).....	153
Выборка из общих табличных выражений (CTE).....	156
Соединения (JOINS).....	157
Внутренние (INNER) и внешние (OUTER) соединения.....	158
Обычные соединения.....	161
Соединения с явными условиями.....	161
Соединения именованными столбцами.....	162
Естественные соединения (Natural Joins).....	164
Замечание о равенстве.....	165
Перекрестное соединение (CROSS JOIN).....	166
Неоднозначные имена полей в соединениях.....	167
Соединения с хранимыми процедурами.....	167
Предложение WHERE.....	168
Предложение GROUP BY.....	172
HAVING.....	176
Предложение PLAN.....	178
Простые планы.....	179
Составные планы.....	181
UNION.....	183
Предложение ORDER BY.....	185
WITH LOCK.....	189

Руководство по языку SQL СУБД Firebird

Как сервер работает с WITH LOCK	191
Оptionальное предложение "OF <column-names>"	191
Предостережения при использовании WITH LOCK	192
Общие табличные выражения ("WITH ... AS ... SELECT").....	193
Рекурсивные CTE	195
CASE.....	198
Простой CASE	198
Поисковый CASE	199
UPDATE	201
SET	202
WHERE	203
ORDER BY и ROWS	203
RETURNING.....	204
Обновление столбцов BLOB	205
INSERT.....	205
UPDATE OR INSERT.....	206
RETURNING	207
MERGE.....	207
DELETE.....	208
Псевдонимы.....	209
WHERE	210
PLAN.....	210
ORDER BY	210
RETURNING	211
ROWS	211
EXECUTE BLOCK.....	212
Входные и выходные параметры	215
Терминатор оператора	215
EXECUTE PROCEDURE	216
Операторы PSQL.....	218
Хранимые процедуры	219
Создание хранимой процедуры	219
Изменение хранимой процедуры	220
Удаление хранимой процедуры	220
Хранимые функции	221
PSQL блоки	221
Триггеры	222
Создание триггера.....	223
Изменение триггера	224
Удаление триггера.....	225
SET TERM.....	226
Операторы языка PSQL.....	227
Оператор присваивания	227
DECLARE VARIABLE.....	228
BEGIN ... END.....	233
IF ... THEN ... ELSE	235
WHILE ... DO.....	237

Руководство по языку SQL СУБД Firebird

LEAVE.....	238
EXIT	240
SUSPEND.....	241
EXECUTE STATEMENT.....	242
FOR SELECT.....	247
FOR EXECUTE STATEMENT	250
OPEN.....	251
FETCH	253
CLOSE	255
IN AUTONOMOUS TRANSACTION	256
EXCEPTION	257
WHEN ... DO	260
POST_EVENT	263
Встроенные функции и переменные.....	264
Контекстные переменные.....	264
CURRENT_CONNECTION.....	264
CURRENT_DATE	264
CURRENT_ROLE	264
CURRENT_TIME	265
CURRENT_TIMESTAMP.....	266
CURRENT_TRANSACTION	266
CURRENT_USER.....	267
DELETING	267
GDSCODE	268
INSERTING.....	268
NEW.....	269
'NOW'	269
OLD.....	270
ROW_COUNT.....	271
SQLCODE.....	271
SQLSTATE	272
'TODAY'	273
'TOMORROW'	274
UPDATING.....	274
'YESTERDAY'	274
USER	275
Функции для работы с контекстными переменными	275
RDB\$GET_CONTEXT ().....	275
RDB\$SET_CONTEXT()	277
Скалярные функции.....	279
ABS ().....	279
ACOS ().....	280
ASCII_CHAR ()	280
ASCII_VAL ().....	280
ASIN ()	281
ATAN ()	281
ATAN2 ().....	281

Руководство по языку SQL СУБД Firebird

BIN_AND ()	282
BIN_OR ()	282
BIN_SHL ()	283
BIN_SHR ()	283
BIN_XOR ()	283
BIT_LENGTH ()	283
CAST ()	284
CEIL (), CEILING ()	285
CHAR_LENGTH (), CHARACTER_LENGTH ()	285
CHAR_TO_UUID ()	286
COALESCE ()	286
COS ()	286
COSH ()	287
COT ()	287
DATEADD ()	287
DATEDIFF ()	288
DECODE ()	290
EXP ()	290
EXTRACT ()	291
FLOOR ()	292
GEN_ID ()	292
GEN_UUID ()	293
HASH ()	294
IIF ()	294
LEFT ()	295
LN ()	295
LOG ()	296
LOG10 ()	296
LOWER ()	296
LPAD ()	297
MAXVALUE ()	298
MINVALUE ()	298
MOD ()	298
NULLIF ()	299
OCTET_LENGTH ()	299
OVERLAY ()	300
PI ()	302
POSITION ()	302
POWER ()	303
RAND ()	303
REPLACE ()	303
REVERSE ()	303
RIGHT ()	304
ROUND ()	305
RPAD ()	306
SIGN ()	306
SIN ()	307
SINH ()	307

Руководство по языку SQL СУБД Firebird

SQRT ()	307
SUBSTRING ()	308
TAN ()	308
TANH ()	309
TRIM ()	309
TRUNC ()	310
UPPER ()	311
UUID_TO_CHAR ()	312
Агрегатные функции	312
AVG ()	312
COUNT()	313
LIST ()	313
MAX ()	314
MIN ()	314
SUM ()	314
Управление транзакциями	316
RELEASE SAVEPOINT	316
ROLLBACK	316
ROLLBACK RETAIN	316
ROLLBACK TO SAVEPOINT	317
SAVEPOINT	317
Внутренние точки сохранения	319
Точки сохранения и PSQL	319
SET TRANSACTION	320
IGNORE LIMBO	320
LOCK TIMEOUT	321
NO AUTO UNDO	321
Безопасность	322
Операторы управления пользователями	324
CREATE USER	324
ALTER USER	325
DROP USER	327
Операторы управления ролями	329
CREATE ROLE	329
ALTER ROLE	330
DROP ROLE	331
Операторы назначения привилегий	332
GRANT	332
REVOKE	336
Роль RDB\$ADMIN	341
Предоставление роли RDB\$ADMIN в обычной базе данных	341
Использование роли RDB\$ADMIN в обычной базе данных	342
Предоставление роли RDB\$ADMIN в базе данных пользователей	342
Использование роли RDB\$ADMIN в базе данных пользователей	343
AUTO ADMIN MAPPING	343
Включение и выключение флага в обычной базе данных	344

Руководство по языку SQL СУБД Firebird

Включение и выключение флага в базе данных пользователей.....	344
Приложения	346
Приложение 1. Поле RDB\$VALID_BLR	346
Приложение 2. Обработка ошибок, коды и сообщения	348
Типы исключений	348
Системные исключения	349
Пользовательские исключения. Создание	349
Пользовательские исключения. Изменение и удаление	350
Коды ошибок SQLSTATE и их описание	350
Коды ошибок GDSCODE их описание, и SQLCODE	363

Глава 1

Введение

Это руководство описывает язык SQL, поддерживаемый СУБД Firebird 2.5. Известно, что СУБД Firebird начала создаваться на основе открытого кода СУБД InterBase 6.0. В период с 2000 по 2014 год было выпущено 5 основных релизов Firebird: 1.0.x.x; 1.5.x; 2.0.x; 2.1.x; 2.5.x., релиз 3.0 запланирован на 2015 год.

Однако, за всё время развития проекта Firebird, а это более 10 лет, на русском языке до сих пор не было создано единой документации. Несколько переведённых материалов по отдельным вопросам можно было найти на официальном сайте (<http://www.firebirdsql.org/en/reference-manuals/>). Но их, во-первых, очень мало, а во-вторых зачастую они уже неактуальны. Наибольшее количество русскоязычной информации о Firebird находится на сайте <http://ibase.ru/develop.htm>, за что огромное спасибо Дмитрию Кузьменко (IBSurgeon/ibase.ru).

Но единой русскоязычной документации о Firebird до сих пор не существовало. Данное «Руководство по языку SQL СУБД Firebird» — это первый русскоязычный документ, полностью освещающий все аспекты и особенности работы с языком SQL Firebird для текущей (актуальной) на сегодняшний день версии СУБД Firebird 2.5. В руководстве также приводятся практические примеры использования SQL, многие из которых взяты из реальной практики.

Что содержит данный документ

Данный документ содержит описание языка SQL Firebird. Он охватывает следующие основные области:

- Основные положения;
- Зарезервированные и ключевые слова;
- Типы и подтипы данных;
- Операторы DDL (Data Definition Language - язык создания данных);
- Операторы DML (Data Manipulation Language - язык обращения с данными);
- Операторы управления транзакциями;
- Обработка исключений;
- Операторы PSQL (Procedural SQL — процедурный SQL, используется в хранимых процедурах, триггерах и выполнимых блоках);
- Безопасность и операторы управления доступом;
- Контекстные переменные;

Руководство по языку SQL СУБД Firebird

- Операторы и предикаты (утверждения);
- Агрегатные функции;
- Встроенные функции;
- Коды ошибок и обработка исключительных ситуаций;
- UDF (User Defined Functions — функции, определённые пользователем. Также известные как внешние функции).

Вопросы, не связанные с SQL в данном документе не рассматриваются.

Авторство

В работе над руководством принимали участие:

- Симонов Денис;
- Пол Винкеуг;
- Филиппов Дмитрий;
- Еманов Дмитрий;
- Томас Воинк;
- Карпейкин Александр;
- Ковязин Алексей;
- Кузьменко Дмитрий;

Редакторы – Александр Карпейкин, Дмитрий Кузьменко, Алексей Ковязин, Денис Симонов

Спонсоры

Платиновым спонсором создания «Руководства по языку СУБД Firebird» является Московская Биржа (www.moex.com).

Московская Биржа – крупнейший в России и Восточной Европе биржевой холдинг, образованный 19 декабря 2011 года в результате слияния биржевых групп ММВБ (основана в 1992) и РТС (основана в 1995). Московская Биржа входит в двадцатку ведущих мировых площадок по объёму торгов ценными бумагами, суммарной капитализации торгуемых акций и в десятку крупнейших бирж производных финансовых инструментов.

Золотым спонсором «Руководства по языку СУБД Firebird» является IBSurgeon (iBase.ru) (www.ib-aid.com, www.ibase.ru): техническая поддержка и инструменты разработчика и администратора для СУБД Firebird.

Благодарности

Благодарим Влада Хорсуна, Александра Пешкова, Павла Зотова за помощь в

Руководство по языку SQL СУБД Firebird

создании этого документа.

Лицензионные замечания

Содержание данного Документа распространяется на условиях лицензии «Public Documentation License Version 1.0» (далее «Лицензия»); Вы можете использовать этот Документ, только если согласны с условиями Лицензии. Копии текста Лицензии доступны по адресам <http://www.firebirdsql.org/pdfmanual/pdl.pdf> (PDF) и <http://www.firebirdsql.org/manual/pdl.html> (HTML).

Оригинальное название документа «Руководство по языку SQL Firebird».

Copyright (C) 2014. Все права защищены. Адрес электронной почты для контакта: case@firebirdsql.org

Далее представлен оригинальный текст раздела, так как его перевод не имеет равноценной юридической силы.

The contents of this Documentation are subject to the Public Documentation License Version 1.0 (the “License”); you may only use this Documentation if you comply with the terms of this License. Copies of the License are available at <http://www.firebirdsql.org/pdfmanual/pdl.pdf> (PDF) and <http://www.firebirdsql.org/manual/pdl.html> (HTML).

Обновления

Так как СУБД Firebird постоянно развивается, то изменяется и улучшается его документация. Вы можете получить обновленные версии этого документа и других частей документации по Firebird на русском языке на сайте проекта документации:

<https://www.assembla.com/spaces/firebird-russian-documentation/wiki>

Глава 2

Подмножества SQL

SQL имеет четыре подмножества SQL, используемых в различных областях применения:

- Динамический SQL (DSQL, Dynamic SQL)
- Процедурный SQL (PSQL, Procedural SQL)
- Встроенный SQL (ESQL, Embedded SQL)
- Интерактивный SQL (ISQL, Interactive SQL)

Динамический SQL является основной частью языка, которая соответствует **Части 2** (SQL/Foundation – SQL/Основы) спецификации SQL. DSQL представляет собой конструкции, которые передаются клиентскими приложениями с помощью Firebird API и обрабатываются сервером базы данных.

Процедурный SQL является расширением Динамического SQL, в котором дополнительно присутствуют составные операторы, содержащие локальные переменные, присваивание, циклы и другие процедурные конструкции. PSQL относится к **Части 4** (SQL/PSM) спецификации SQL. Изначально расширение PSQL было доступно только лишь в постоянно хранимых в базе модулях (процедурах и триггерах), но сравнительно недавно они стали также доступны в Динамическом SQL (смотри EXECUTE BLOCK).

Встроенный SQL определяет DSQL подмножество, поддерживаемое средством Firebird GPRE – приложение, которое позволяет вам внедрять SQL конструкции в ваш непосредственный язык программирования (C, C++, Pascal, Cobol и так далее) и производить обработку этих внедренных конструкций в правильный вызов Firebird API. Обратите внимание, что ESQL поддерживает только часть конструкций и выражений DSQL.

Интерактивный SQL подразумевает собой язык, который может быть использован для работы с приложением командной строки Firebird ISQL для интерактивного доступа к базам данных. ISQL является обычным клиентским приложением. Для него обычный язык – это язык DSQL. Однако приложение поддерживает несколько дополнительных команд.

Оба языковых подмножества, как DSQL, так и PSQL полностью представлены в данном руководстве. Из набора инструментария ни ESQL, ни ISQL не описаны здесь отдельно, за исключением тех мест, где это не указано явно.

Действия при ошибках

Обработка любого оператора либо успешно завершается, либо прерывается из-за вызванной определёнными условиями ошибки. Обработку

Руководство по языку SQL СУБД Firebird

ошибок можно проводить как на клиентском приложении, так и на стороне сервера средствами SQL.

Подробнее смотрите в [Приложение 2. Обработка ошибок, коды и сообщения](#).

Глава 3

Диалекты SQL

SQL диалект – это термин, определяющий специфические особенности языка SQL, которые доступны во время доступа с его помощью к базе данных. SQL диалект может быть определен как на уровне базы данных, так и на уровне соединения с базой данных. В настоящее время доступны три диалекта:

- В 1-м диалекте дата и время хранятся в поле с типом данных DATE и имеется тип данных TIMESTAMP, который идентичен DATE. Двойные кавычки используются для разграничения строковых данных. Точность типов данных NUMERIC и DECIMAL меньше, чем в 3-м диалекте и в случае, если значение точности более 9, Firebird хранит такие значения как длинные значения с плавающей точкой. BIGINT не является доступным типом данных. Идентификаторы являются регистром не зависимыми. Значение генераторов хранится как 64 битное целое, а при выдаче значения урезается до 32 битного целого.
- Диалект 2 доступен только в клиентском соединении к Firebird и не может быть применен к базе данных. Он предназначен для того, чтобы помочь в отладке в случае возможных проблем с целостностью данных при проведении миграции с диалекта 1 на 3.
- Диалект 3 базы данных позволяет хранить числа (типы данных DECIMAL и NUMERIC) в базе данных как длинные значения с фиксированной точкой (масштабируемые целые числа) в случае если точность числа меньше чем 9. Тип данных TIME доступен и используется для хранения значения только **времени**. Тип данных DATE хранит информацию о дате. Тип данных BIGINT доступен в качестве целого 64-х битного типа данных. Двойные кавычки могут использоваться, но только для идентификаторов, которые являются зависимыми от регистра, а не для строковых данных, для которых используют одинарные кавычки. Значения генераторов хранятся как 64-ти битные целые значения.

Целью 1-го диалекта является обеспечение поддержки для унаследованных (пре-версия IB6) Interbase приложений для работы с Firebird. Диалект 2 используется как промежуточный и предназначен для разрешения проблем при миграции с 1-го в 3-й диалект. Для вновь разрабатываемых баз данных и приложений настоятельно рекомендуется использовать 3-й диалект. Диалект при соединении с базой данных должен быть таким же, как и базы данных. Исключением является случай миграции с 1-го в 3-й диалект, когда в строке соединения с базой данных используется 2-й диалект.

По умолчанию это руководство описывает семантику SQL третьего диалекта, если только в тексте явно не указывается диалект.

Глава 4

Типы и подтипы данных

Типы данных используются в случае:

- определения столбца в таблице базы данных в операторе CREATE TABLE или для его изменения с использованием ALTER TABLE;
- при объявлении и редактировании домена оператором CREATE DOMAIN/ALTER DOMAIN;
- при объявлении локальных переменных в хранимых процедурах, PSQL-блоках и триггерах, при указании аргументов хранимых процедур;
- при описании внешних функций (UDF – функций, определенных пользователем) для указания аргументов и возвращаемых значений.
- при явном преобразовании типов данных в качестве аргумента для функции CAST.

Данные о типах данных приведены в таблице 4.1.

Таблица 4.1. Типы данных Firebird

Название	Размер	Точность и диапазон	Описание
BIGINT	64 бита	От -2^{63} .. $2^{63} - 1$	Тип данных доступен только в 3 диалекте.
BLOB	Переменный	Нет. Размер сегмента BLOB ограничивается 64К. Максимальный размер поля BLOB 4 Гб.	Тип данных с динамически изменяемым размером для хранения больших данных, таких как графика, тексты, оцифрованные звуки. Базовая структурная единица — сегмент. Подтип Blob описывает содержимое.
CHAR(n) CHARACTER(n)	n символов (размер в байтах зависит от кодировки, кол-ва байт на символ)	от 1 до 32 767 байтов (объявленный размер может быть до 32765 включительно)	Символьный тип данных фиксированной длины. При отображении данных, строка дополняется пробелами справа до указанной длины. Конечные пробелы не хранятся в базе данных, а восстанавливаются при отображении такого столбца. Восстановление пробельных символов до максимальной длины происходит на клиенте, а не на сервере, при передаче данных по локальной сети пробелы не передаются, что позволяет уменьшить сетевой трафик. Если количество символов n не указано, то по умолчанию принимается 1.
DATE	32 бита	От 01.01.100 н.э. до	ISC_DATE

Руководство по языку SQL СУБД Firebird

Название	Размер	Точность и диапазон	Описание
		31.12.9999 н.э.	
DECIMAL (precision, scale)	Переменный (16, 32 или 64 бита)	precision = от 1 до 18, указывает, по меньшей мере, количество цифр для хранения; scale = от 0 до 18. Задаёт количество знаков после разделителя	Scale должно быть меньше или равно precision. Число с десятичной точкой, имеющей после точки scale разрядов. Пример: DECIMAL(10,3) содержит число точно в следующем формате: rrrrrrrr.sss.
DOUBLE PRECISION	64 бита	$2,225 \times 10^{-308}$ до $1,797 \times 10^{308}$	IEEE двойной точности, 15 цифр, размер зависит от платформы
FLOAT	32 бита	$1,175 \times 10^{-38}$ до $3,402 \times 10^{38}$	IEEE одинарной точности, 7 цифр
INTEGER INT	32 бита	-2 147 483 648 до 2 147 483 647	signed long
NUMERIC(precision, scale)	Переменный (16, 32 или 64 бита)	precision = от 1 до 18; <u>точное</u> количество цифр для хранения. scale = от 0 до 18; задаёт количество знаков после точки. Должно быть меньше или равно precision.	Scale должно быть меньше или равно precision. Число с десятичной точкой, имеющей после точки scale разрядов. Пример: NUMERIC(10,3) содержит число точно в следующем формате: rrrrrrrr.sss.
SMALLINT	16 бит	-32768 до 32767	signed short (word)
TIME	32 бита	0:00 до 23:59:59.9999	ISC_TIME
TIMESTAMP	64 бита	От 01.01.100 н.э. до 31.12.9999 н.э.	Включает информацию и о времени
VARCHAR(n) CHAR VARYING CHARACTER VARYING	n символов (размер в байтах зависит от кодировки, кол-ва байт на символ)	от 1 до 32 767 байтов	Размер символов в байтах с учетом их кодировки не может быть больше 32К. Начальные и конечные пробелы хранятся и не обрезаются, за исключением тех хвостовых пробелов, которые не укладываются в заявленную длину. Для этого типа данных, в отличие от CHAR (где по умолчанию предполагается количество символов 1), количество символов n обязательно должно быть указано.

Замечание о датах:

Следует иметь в виду, что временной ряд из дат прошлых веков рассматривается без учета реальных исторических фактов и так, как будто бы во всём этом диапазоне ВСЕГДА действовал только Григорианский

календарь.

Целочисленные данные

Для целых чисел используют целочисленные типы данных SMALLINT, INTEGER и BIGINT (в 3 диалекте). Firebird не поддерживает беззнаковый целочисленный тип данных.

SMALLINT

Тип данных SMALLINT представляет собой целочисленное компактное хранилище данных и применяется в случае, когда не требуется широкий диапазон возможных значений для хранения данных.

Примеры:

```
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

Тип данных INTEGER представляет собой 4-байтовое целое. Сокращенный вариант записи типа данных INT.

BIGINT

BIGINT это SQL-99-совместимый 64 битный целочисленный тип данных. Он доступен только в 3-м диалекте. При использовании *клиентом* диалекта 1, передаваемое сервером значение генератора усекается до 32-х битного целого (INTEGER). При подключении в 3-м диалекте значение генератора имеет тип BIGINT.

Числа типа BIGINT находятся в диапазоне $-2^{63} .. 2^{63} - 1$, или -9 223 372 036 854 775 808 .. 9 223 372 036 854 775 807.

Начиная с Firebird 2.5 числа типа BIGINT могут быть заданы в шестнадцатеричном виде с 9 — 16 шестнадцатеричными цифрами. Более короткие шестнадцатеричные числа интерпретируются как тип данных INTEGER.

Пример:

```
CREATE TABLE WHOLELOTTARECORDS (
  ID BIGINT NOT NULL PRIMARY KEY,
  DESCRIPTION VARCHAR(32)
);
```

Руководство по языку SQL СУБД Firebird

```
INSERT INTO MYBIGINTS VALUES (  
    -236453287458723,  
    328832607832,  
    22,  
    -56786237632476,  
    0X6F55A09D42, -- 478177959234  
    0X7FFFFFFFFFFFFFFFFF, -- 9223372036854775807  
    0XFFFFFFFFFFFFFFFF, -- -1  
    0X80000000, -- -2147483648, т.е. INTEGER  
    0X080000000, -- 2147483648, т.е. BIGINT  
    0XFFFFFFFF, -- -1, т.е. INTEGER  
    0X0FFFFFFFFF -- 4294967295, т.е. BIGINT  
);
```

Шестнадцатеричный INTEGER автоматически приводится к типу BIGINT перед вставкой в таблицу. Однако это происходит *после* установки численного значения, так 0x80000000 (8 цифр) и 0x080000000 (9 цифр) будут сохранены в разных форматах. Значение 0x80000000 (8 цифр) будет сохранено в формате INTEGER, а 0x080000000 (9 цифр) как BIGINT.

Типы данных с плавающей точкой

Типы данных с плавающей точкой являются примерами данных, которые хранятся в СУБД с точностью, *подходящей масштабу* числа. Примерами таких данных являются FLOAT и DOUBLE PRECISION (DOUBLE).

Учитывая особенности хранения чисел с плавающей точкой в базе данных, данный тип данных не рекомендуется использовать для хранения денежных данных. По тем же причинам не рекомендуется использовать столбцы с данными такого типа в качестве ключей и применять к ним ограничения уникальности.

При проверке данных столбцов с типами данных с плавающей точкой рекомендуется вместо точного равенства использовать выражения проверки вхождения в диапазон, например BETWEEN.

При использовании таких типов данных в выражениях рекомендуется крайне внимательно и серьезно подойти к вопросу округления результатов расчетов.

FLOAT

Данный тип данных обладает приблизительной точностью 7 цифр после запятой. Для обеспечения надежности хранения полагайтесь на 6 цифр.

DOUBLE PRECISION

При хранении данных, предполагается приблизительная точность 15 цифр.

Типы данных с фиксированной точкой

Данные типы данных позволяют применять их для хранения денежных значений и обеспечивают предсказуемость операций умножения и деления.

Firebird предлагает два типа данных с фиксированной точкой: NUMERIC и DECIMAL. В соответствии со стандартом оба типа ограничивают хранимое число объявленным масштабом (количеством чисел после запятой). При этом подход к тому, как ограничивается точность для типов разный: для столбцов NUMERIC точность является такой, «как объявлено», в то время как DECIMAL столбцы могут получать числа, чья точность, *по меньшей мере, равна* тому, что было объявлено.

Например, NUMERIC(4, 2) описывает число, состоящее в общей сложности из четырех цифр, включая 2 цифры после запятой; итого 2 цифры до запятой, 2 после. При записи в столбец с этим типом данных значений 3,1415 в столбце NUMERIC(4, 2) будет сохранено значение 3,14.

Для данных с фиксированной точкой общим является форма декларации, например NUMERIC(p, s). Здесь важно понять, что в этой записи *s* - это **масштаб**, а не интуитивно предсказываемое «*количество знаков после запятой*». Для «визуализации» механизма хранения данных запомните для себя процедуру:

- При сохранении в базу данных число умножается на 10 в степени *s*, превращаясь в целое;
- При чтении данных происходит обратное преобразование числа.

Способ физического хранения данных в СУБД зависит от нескольких факторов: декларируемой точности, диалекта базы данных, типа объявления (Таблица 4.2).

Таблица 4.2. Способ физического хранения вещественных чисел

Точность	Тип данных	Диалект 1	Диалект 3
1 – 4	NUMERIC	SMALLINT	SMALLINT
	DECIMAL	INTEGER	INTEGER
5 – 9	NUMERIC и DECIMAL	INTEGER	INTEGER
10 – 18	NUMERIC и DECIMAL	DOUBLE PRECISION	BIGINT

NUMERIC

Формат объявления данных:

Руководство по языку SQL СУБД Firebird

NUMERIC(p, s)

В зависимости от точности (p) и масштаба (s) СУБД хранит данные по разному (см. таблицу 4.2).

Приведем примеры того, как СУБД хранит данные в зависимости от формы их объявления:

NUMERIC (4)	SMALLINT
NUMERIC (4, 2)	SMALLINT
NUMERIC (10, 4)	DOUBLE PRECISION (для 1-го диалекта или BIGINT (для 3-го диалекта))

Примечание

Всегда надо помнить, что формат хранения данных зависит от точности. Например, вы задали тип столбца NUMERIC(2,2), предполагая, что диапазон значений в нём будет -0.99...0.99. Однако в действительности диапазон значений в столбце будет -327.68..327.68, что объясняется хранением типа данных NUMERIC(2,2) в формате SMALLINT. Фактически типы данных NUMERIC(4,2), NUMERIC(3,2) и NUMERIC(2,2) являются одинаковыми. Т.е. Для реального хранения данных в столбце с типом данных NUMERIC(2,2) в диапазоне -0.99...0.99 для него надо создавать ограничение.

DECIMAL

Формат объявления данных:

DECIMAL(p, s)

Формат хранения данных в базе во многом аналогичен NUMERIC, хотя существуют некоторые особенности.

Приведем примеры того, как СУБД хранит данные в зависимости от формы их объявления:

DECIMAL (4)	INTEGER
DECIMAL (4, 2)	INTEGER
DECIMAL (10, 4)	DOUBLE PRECISION (для 1-го диалекта) или BIGINT (для 3-го диалекта)

Типы данных для работы с датой и временем

В СУБД Firebird для работы с данными, содержащими дату и время, используются типы данных DATE, TIME, TIMESTAMP. В 3-м диалекте присутствуют все три вышеназванных типа данных, а в 1-м для операций с датой и временем доступен только тип данных DATE, который не тождественен типу данных DATE 3-го диалекта, а напоминает тип данных TIMESTAMP из 3-го

Руководство по языку SQL СУБД Firebird

диалекта.

Доли секунды, если хранятся в типах данных даты и времени, являются десятичными долями секунды.

DATE

В 3-м диалекте тип данных DATE, как это и следует предположить из названия, хранит только одну дату без времени. В 1-м диалекте нет типа данных «только дата». В случае необходимости сохранять, например, только значения даты, без времени, при записи в таблицу передавайте время к значению даты в виде литерала "00:00:00.0000".

Допустимый диапазон хранения от 01 января 100 г. н.э. до 31 декабря 9999 года.

TIME

Этот тип данных доступен только в 3-м диалекте. Позволяет хранить время дня в диапазоне от 00:00:00.0000 до 23:59:59.9999 .

При необходимости получения времени из DATE. в 1-м диалекте можно использовать функцию EXTRACT.

Примеры:

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

TIMESTAMP

Этот тип данных доступен только в 3-м диалекте, состоит из двух 32-битных слов и хранит дату со временем. Такое хранение эквивалентно типу DATE 1-го диалекта.

Символьные типы данных

В СУБД Firebird для работы с символьными данными есть тип данных фиксированной длины CHAR и строковый тип данных VARCHAR переменной длины. Максимальный размер текстовых данных, хранящийся в этих типах данных, составляет 32767 байт. Максимальное количество *символов*, которое поместится в этот объем, зависит от используемого набора символов CHARACTER SET и/или заданного порядка сортировки, который для символьных данных задается предложением COLLATE.

В случае отсутствия явного указания набора символов при описании

Руководство по языку SQL СУБД Firebird

текстового объекта базы данных будет использоваться набор символов по умолчанию, заданный при создании базы данных. При отсутствии явного указания набора символов, а также отсутствия набора символов по умолчанию в базе данных, поле получает набор символов CHARACTER SET NONE.

Если база данных будет содержать строки *только с русским алфавитом*, то для нее рекомендуется к использованию кодировка WIN1251. При ее использовании на один символ расходуется 1 байт, соответственно максимальный размер текстовых полей для данной кодировки будет 32767 символов. Для стандартных операций сортировки при работе с WIN1251 не требуется задавать порядок сортировки (COLLATE).

В настоящее время все современные средства разработки поддерживают Unicode. При возникновении необходимости использования западноевропейских текстов в строковых полях базы данных или для более экзотических алфавитов, рекомендуется работать с набором символов UTF8. При этом следует иметь в виду, что на один символ в данном наборе приходится до 4 байт. Следовательно, максимальный размер символов в символьных полях составит $32767/4$ (8192) байта на символ. При этом следует обратить внимание, что фактически значение параметра «байт на символ» зависит от диапазона, к которому принадлежит символ: английские буквы занимают 1 байт, русские буквы кодировки WIN1251 – 2 байта, остальные символы – могут занимать до 4-х байт.

Набор символов UTF8 поддерживает последнюю версию стандарта Unicode, до 4 байт на символ, поэтому для интернациональных баз рекомендуется использовать именно эту реализацию поддержки Unicode в Firebird.

При работе со строками необходимо помнить и о параметре соединения клиентской программы к базе данных. В нем также задается набор символов. В случае различия набора символов, при выдаче результата для строковых столбцов происходит автоматическая перекодировка как при передаче данных с клиента на сервер, так и в обратном направлении с сервера на клиент. То есть, совершенно нормальной является ситуация, когда база создана в кодировке WIN1251, а в настройках клиента в параметрах соединения стоит KOI8R или UTF8.

Упомянутый выше набор символов NONE относится к специальным наборам символов. Его можно охарактеризовать тем, что каждый байт является частью *строки*, но в системе хранится без указаний, к какому *фактическому* набору символов они относятся. Разбираться с такими данными должно клиентское приложение, на него возлагается ответственность в правильной трактовке символов из таких полей. Также к специальным наборам символов относится OCTETS. В этом случае данные рассматриваются как байты, которые могут в принципе не интерпретироваться как символы. OCTETS позволяет хранить бинарные данные и/или результаты работы некоторых функций Firebird. Правильное отображение данных пользователю, хранящихся в полях с CHARACTER SET OCTETS, также становится заботой клиентской стороны. При

Руководство по языку SQL СУБД Firebird

работе с подобными данными следует также помнить, что СУБД не контролирует их содержимое и возможно возникновение исключения при работе кода, когда идет попытка отображения бинарных данных в желаемой кодировке.

Каждый текстовый набор символов (CHARACTER SET) имеет последовательность сортировки (COLLATE) по умолчанию, задающий порядок сортировки и упорядочивания. Если необходимо нестандартное поведение строк при указанных выше действиях, то в описании строкового столбца может быть указан параметр COLLATE, который его опишет. Помимо описания объявления столбца, выражение COLLATE может быть добавлено в предложениях SELECT в секции WHERE, когда происходят операции сравнения больше – меньше, в секции ORDER BY при сортировке по символьному полю, а также при операциях группировки для указания специальной последовательности сортировки при выводе в предложении GROUP BY.

Для регистронезависимого поиска можно воспользоваться функцией UPPER:

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

Для строк с набором символов WIN1251 можно для этих же целей воспользоваться предложением COLLATE PXW_CYRL.

Пример:

```
WHERE FIRST_NAME COLLATE PXW_CYRL >= :FLT_NAME
```

Пример сортировки независимой от регистра символов:

```
ORDER BY NAME COLLATE PXW_CYRL
```

См. также **CONTAINING**

Ниже приведена таблица возможных последовательностей сортировки для набора символов UTF8 (таблица 4.3).

Таблица 4.3. Последовательности сортировки для UTF8

COLLATION	Комментарии
UCS_BASIC	Сортировка работает в соответствии с положением символа в таблице (бинарная): Пример: A, B, a, b, á... Добавлена в Firebird 2.0
UNICODE	Сортировка работает в соответствии с алгоритмомUCA (Unicode Collation Algorithm)(алфавитная). Пример: a, A, á, b, B... Добавлена в Firebird 2.0

Руководство по языку SQL СУБД Firebird

UTF-8	Сортировка происходит без учета регистра символа. Добавлена в Firebird 2.1
UNICODE_CI_AI	Сортировка происходит без учета регистра символа, в алфавитном порядке. Добавлена в Firebird 2.5

Пример сортировки строк для набора символов UTF8 без учета регистра символов (эквивалент COLLATE PXW_CYRL)

```
ORDER BY NAME COLLATE UNICODE_CI_AI
```

Для символьных полей, использующих нестандартный порядок сортировки, может возникнуть проблема при построении индекса по данному полю: имеется ограничение на длину поля при построении индекса – 84 байта при указании COLLATE, 252 байта – без COLLATE. Это актуально для версий Firebird меньше 2.0.

Теперь ситуация изменилась. Максимальная используемая длина ключа индекса, ранее установленная в 252 байта, теперь равна 1/4 размера страницы, т.е. от 256 до 4096 байтов. Максимальная длина индексируемой строки на 9 байтов меньше, чем максимальная длина ключа. В таблице 4.4 приведены данные для максимальной длины индексируемой строки (в символах) в зависимости от размера страницы и набора символов.

Таблица 4.4. Максимальная длина индексируемого поля (VAR)CHAR

Размер страницы	Максимальная длина индексируемой строки для набора символов			
	1 байт/символ	2 байта/символ	3 байта/символ	4 байта/символ
1024	247	123	82	61
2048	503	251	167	125
4096	1015	507	338	253
8192	2039	1019	679	509
16384	4087	2043	1362	1021

Замечание:

В кодировках, нечувствительных к регистру ("_CI"), один символ в *индексе* будет занимать не 4, а 6 (шесть) байт, поэтому максимальная длина ключа для страницы, скажем, 4096, составит 169 символов.

См. также: [CREATE DATABASE](#), [CONNECT](#), [COLLATION](#), [CREATE INDEX](#), [SELECT](#), Предложение WHERE, Предложение ORDER BY, Предложение GROUP BY.

Руководство по языку SQL СУБД Firebird

Полный список доступных наборов символов и нестандартных порядков сортировки доступен в [приложении 4](#).

CHAR

CHAR является типом данных фиксированной длины. Если введенное количество символов меньше объявленной длины, то поле дополнится концевыми пробелами. В общем случае символ заполнитель может и не являться пробелом, он зависит от набора символов, так например для набора символов OCTETS - это ноль.

Полное название типа данных CHARACTER, но при работе нет требования применять полные наименования; инструменты по работе с базой прекрасно понимают и короткие имена символьных типов данных.

Синтаксис:

```
CHAR [(длина)] [CHARACTER SET <набор>] [COLLATE <имя>]
```

В случае если не указана длина, то считается, что она равна единице.

Данный тип символьных данных можно использовать для хранения в справочниках кодов, длина которых стандартна и определенной «ширины». Примером такого может служить почтовый индекс в России – 6 символов.

VARCHAR

Является базовым строковым типом для хранения текстов переменной длины, поэтому реальный размер хранимой структуры равен фактическому размеру данных плюс 2 байта, в которых задана длина поля. Все символы, которые передаются с клиентского приложения в базу данных, считаются как значимые, включая начальные и конечные пробельные символы.

Полное название CHARACTER VARYING. Имеется и сокращенный вариант записи CHAR VARYING.

Синтаксис:

```
VARCHAR (длина) [CHARACTER SET <набор>] [COLLATE <имя>]
```

NCHAR

Представляет собой символьный тип данных фиксированной длины с предопределенным набором символов ISO8859_1.

Руководство по языку SQL СУБД Firebird

Синтаксис:

NCHAR (длина)

Синонимом является написание NATIONAL CHAR.

Аналогично доступно для строковых типов переменной длины: NATIONAL CHARACTER VARYING (NVARCHAR).

Бинарные типы данных

BLOB (Binary Large Objects, большие двоичные объекты) представляют собой сложные структуры, предназначенные для хранения текстовых и двоичных данных неопределенной длины, зачастую очень большого объема.

Синтаксис:

```
BLOB [SUB_TYPE <подтип>]
      [SEGMENT SIZE <размер сегмента>]
      [CHARACTER SET <набор символов>]
```

Сокращенный синтаксис:

```
BLOB (<размер сегмента>)
      BLOB (<размер сегмента>, <подтип>)
      BLOB (, <подтип>)
```

В случае если подтип не указан, он считается равным 0, читается как BLOB SUBTYPE 0. Общее название этого подтипа «нетипизированный двоичный BLOB».

Подтип 1 имеет алиас TEXT. Например, BLOB SUBTYPE TEXT. Это специализированный подтип, который используется для хранения текстовых данных большого объема. Для текстового подтипа BLOB может быть указан набор символов, аналогично символьному полю. Начиная с версии Firebird 2.0 для текстовых BLOB полей можно указывать порядок сортировки COLLATE.

Указание дискового сегмента BLOB является некоторым атавизмом, оно идет с тех времен, когда приложения для работы с данными BLOB писались на C (Embedded SQL) при помощи GPRE. В настоящий момент размер сегмента при работе с данными BLOB определяется клиентской частью, причем размер сегмента может превышать размер страницы данных.

Подтип BLOB отражает природу данных, записанную в столбце. Могут быть также добавлены пользовательские подтипы данных, для них зарезервирован интервал от -1 до -32768.

Максимальный размер поля BLOB ограничен 4Гб и не зависит от варианта

Руководство по языку SQL СУБД Firebird

сервера, 32 битный или 64 битный (во внутренних структурах, связанных с BLOB присутствуют 4-х байтные счетчики). Для размера страницы 4096 максимальный размер BLOB поля несколько ниже 2 Гб.

Текстовые BLOB любой длины и с любым набором символов (включая multi-byte) теперь могут быть использованы практически с любыми встроенными функциями и операторами (начиная с Firebird 2.1):

- Полная поддержка для операторов:
 - = (присвоение);
 - =, <>, <, <=, >, >= (сравнение);
 - || (конкатенация);
 - BETWEEN, IS [NOT] DISTINCT FROM, IN, ANY|SOME и ALL.
- Частичная поддержка для STARTING [WITH], LIKE и CONTAINING. (возникает ошибка, в случае если второй аргумент больше или равен 32 Кб).
- SELECT DISTINCT, ORDER BY и GROUP BY в своей работе используют BLOB ID, а не содержимое самого поля. Это одновременно и хорошо и плохо, кроме того, SELECT DISTINCT ошибочно выдает несколько значений NULL, если они присутствуют. GROUP BY ведет себя странно в том, что он объединяет одинаковые строки, если они находятся рядом, но не делает этого, если они располагаются вдали друг от друга.

По умолчанию, для каждого BLOB создается обычная запись, хранящаяся на какой-то выделенной для этого странице данных (data page). Если весь BLOB на эту страницу поместится, его называют *BLOB уровня 0*. Номер этой специальной записи хранится в записи таблицы и занимает 8 байт.

Если BLOB не помещается на одну страницу данных (data page), то его содержимое размещается на отдельных страницах, целиком выделенных для него (blob page), а в записи о BLOB помещают номера этих страниц. Это *BLOB уровня 1*.

Если массив номеров страниц с данными BLOB не помещается на страницу данных (data page), то его (массив) размещают на отдельных страницах (blob page), а в записи о BLOB помещают уже номера этих страниц. Это *BLOB уровня 2*. Уровни выше 2 не поддерживаются.

См. также: [FILTER](#), [DECLARE FILTER](#).

Массивы

Поддержка массивов в СУБД Firebird является расширением традиционной реляционной модели. Поддержка в СУБД такого инструмента позволяет проще решать некоторые задачи по обработке однотипных данных. Массивы в Firebird реализованы на базе полей типа BLOB. Массивы могут быть одномерными и многомерными.

Руководство по языку SQL СУБД Firebird

Пример:

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

Так будет создана таблица с полем типа массива из четырех целых. Индексы данного массива от 1 до 4. Для определения верхней и нижней границы значений индекса следует воспользоваться следующим *синтаксисом*:

```
[<нижняя>:<верхняя>]
```

Добавление новой размерности в синтаксисе идет через запятую. Пример создания таблицы с массивом размерности два, в котором нижняя граница значений начинается с нуля:

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

СУБД не предоставляет большого набора инструментов для работы с содержимым массивов. База данных employee.fdb, которая находится в дистрибутиве Firebird, содержит пример хранимой процедуры, показывающей возможности работы с массивами. Ниже приведен её текст:

```
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  
                /* PRINTS 'NULL' INSTEAD OF BLANKS */
```

Руководство по языку SQL СУБД Firebird

```
    LANGUAGES = 'NULL';  
    I = I +1;  
    SUSPEND;  
END  
END
```

Если приведенных выше возможностей достаточно для ваших задач, то вы вполне можете применять массивы для своих проектов. В настоящее время совершенствования механизмов обработки массивов средствами СУБД не производится.

Специальные типы данных

Тип данных SQL_NULL

Данный тип данных содержит не данные, а только состояние: NULL или NOT NULL. Также, этот тип данных не может быть использован при объявлении полей таблицы, переменных PSQL, использован в описании параметров. Этот тип данных добавлен для улучшения поддержки нетипизированных параметров в предикате IS NULL. Такая проблема возникает при использовании «отключаемых фильтров» при написании запросов следующего типа:

```
WHERE col1 = :param1 OR :param1 IS NULL
```

после обработки, на уровне API запрос будет выглядеть как

```
WHERE col1 = ? OR ? IS NULL
```

В данном случае получается ситуация, когда разработчик при написании SQL запрос рассматривает *:param1* как одну переменную, которую использует два раза, а на уровне API запрос содержит два отдельных и независимых параметра. Вдобавок к этому, сервер не может определить *тип* второго параметра, поскольку он идет в паре с *IS NULL*.

Именно для решения проблемы «? IS NULL» и был добавлен этот специальный тип данных SQL_NULL.

После введения данного специального типа данных при передаче запроса и его параметров на сервер будет работать такая схема: приложение передает параметризованные запросы на сервер в виде «?». Это делает невозможным слияние пары «одинаковых» параметров в один. Так, например, для двух фильтров (двух именованных параметров) необходимо передать четыре позиционных параметра (*далее предполагается, что читатель имеет некоторое знакомство с Firebird API*):

```
SELECT  
    SH.SIZE, SH.COLOUR, SH.PRICE
```

Руководство по языку SQL СУБД Firebird

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

После выполнения `isc_dsql_describe_bind()` `sqltype` 2-го и 4-го параметров устанавливается в `SQL_NULL`. Как уже говорилось выше, сервер Firebird не имеет никакой информации об их связи с 1-м и 3-м параметрами - это полностью прерогатива программиста. Как только значения для 1-го и 3-го параметров были установлены (или заданы как `NULL`) и запрос подготовлен, каждая пара `XSQLVARs` должна быть заполнена следующим образом:

Пользователь задал параметры

- Первый параметр (сравнение значений): `set *sqldata` в переданное значение и `*sqlind` в `0` (для `NOT NULL`);
- Второй параметр (проверка на `NULL`): `set *sqldata` в `NULL` (не `SQL_NULL`) и `*sqlind` в `0` (для `NOT NULL`).

Пользователь не задал параметры (NULL)

- Оба параметра (проверка на `NULL`): `set *sqldata` в `NULL` (не `SQL_NULL`) и `*sqlind` в `-1` (**индикация** `NULL`).

Другими словами: значение параметра сравнения всегда устанавливается как обычно. `SQL_NULL` параметр устанавливается также, за исключением случая, когда `sqldata` передаётся как `NULL`.

Преобразование типов данных

При написании выражения или при задании, например, условий сравнения, нужно стараться использовать совместимые типы данных. В случае необходимости использования смешанных данных различных типов, желательно первоначально выполнить преобразования типов, а уже потом выполнять операции.

При рассмотрении вопроса преобразования типов в Firebird большое внимание стоит уделить тому, в каком **диалекте** база данных.

Явное преобразование типов данных

В тех случаях, когда требуется выполнить явное преобразование одного типа в другой, используют функцию `CAST`.

Синтаксис:

```
CAST (<значение> | NULL AS <тип данных>),
      где <тип данных> ::= sql_datatype
```

Руководство по языку SQL СУБД Firebird

```
| [TYPE OF] <домен>  
| TYPE OF COLUMN relname.colname
```

При преобразовании к домену учитываются объявленные для него ограничения, например, NOT NULL или описанные в CHECK и если <значение> не пройдет проверку, то преобразование не удастся. В случае если дополнительно указывается TYPE OF (преобразование к базовому типу), при преобразовании игнорируются любые ограничения домена. При использовании TYPE OF с типом (VAR)CHAR набор символов и сортировка сохраняются.

При преобразовании к типу столбца допускается использовать указание столбца таблицы или представления. Используется только сам тип столбца; в случае строковых типов это включает набор символов и сортировку. Ограничения и значения по умолчанию исходного столбца не применяются.

Пример:

```
CREATE TABLE TTT (  
    S VARCHAR (40)  
    CHARACTER SET UTF8 COLLATE UNICODE_CI_AI);  
COMMIT;  
  
/* У меня много друзей (шведский) */  
SELECT  
    CAST ('Jag har många vänner' AS TYPE OF COLUMN TTT.S)  
FROM RDB$DATABASE;
```

В таблице 4.5 представлены допустимые преобразования для функции CAST.

Таблица 4.5. Допустимые преобразования для функции CAST

Из типа	В тип
Числовые типы	Числовые типы [VAR]CHAR BLOB
[VAR]CHAR BLOB	[VAR]CHAR BLOB Числовые типы DATE TIME TIMESTAMP
DATE TIME	[VAR]CHAR BLOB TIMESTAMP
TIMESTAMP	[VAR]CHAR

Руководство по языку SQL СУБД Firebird

	BLOB DATE TIME
--	----------------------

При проведении следует помнить о возможности частичной потери данных, например, при преобразовании типа данных `TIMESTAMP` в `DATE`.

Неявное преобразование типов данных

В 3-м диалекте невозможно неявное преобразование данных, здесь требуется указывать функцию `CAST` для явной трансляции одного типа в другой. Однако это не относится к операции конкатенации, при которой все другие типы данных будут неявно преобразованы к символьному.

При использовании 1-го диалекта во многих выражениях выполняется неявная трансляция одних типов в другой без применения функции `CAST`. Например, в выражении отбора в диалекте 1 можно записать:

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

и преобразование строки в дату пройдет неявно.

В 1-м диалекте можно смешивать целые данные и числовые строки, строки неявно преобразуются в целое, если это будет возможно, например:

```
2 + '1'
```

корректно выполнится. В 3-м диалекте подобное выражение вызовет ошибку, в нем потребуется запись следующего вида:

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

Таблица 4.6. Литералы с предопределенными значениями даты и времени

Литерал	Значение	Тип данных для диалекта 1	Тип данных для диалекта 3
'NOW'	Текущая дата и время	DATE	TIMESTAMP
'TODAY'	Текущая дата	DATE (с нулевым временем)	DATE (только дата)
'TOMORROW'	Текущая дата + 1	DATE (с нулевым временем)	DATE
'YESTERDAY'	Текущая дата - 1	DATE (с нулевым временем)	DATE

Руководство по языку SQL СУБД Firebird

Сокращённое приведение типов даты и времени (datetime)

При преобразовании строки в тип DATE, TIME или TIMESTAMP, Firebird позволяет использовать сокращённое “C-style” приведение типов.

Синтаксис:

```
datatype 'date/timestring'
```

Пример:

```
UPDATE PEOPLE
SET AGE_CAT = 'Old'
WHERE BIRTHDATE < DATE '1-Jan-1943';

INSERT INTO APPOINTMENTS
  (EMPLOYEE_ID, CLIENT_ID, APP_DATE, APP_TIME)
VALUES (973, 8804, DATE'today' + 2, TIME '16:00');

NEW.LASTMOD = TIMESTAMP'now';
```

Обратите внимание, что эти сокращённые выражения вычисляются сразу же во время синтаксического анализа, то есть как будто оператор уже подготовлен к выполнению. Таким образом, даже если запрос выполняется несколько раз, значение, например, для “timestamp 'now'” не изменится, независимо от того, сколько времени проходит. Если вам нужно получать нарастающее значение времени (т.е. оно должно быть оценено при каждом вызове), используйте полный синтаксис CAST. Пример использования в триггере такого выражения:

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

См. также: [Явное преобразование типов данных](#).

Работа с доменами

Домены в СУБД Firebird реализуют широко известный по многим языкам программирования инструмент «типы данных, определенные пользователем». Когда несколько таблиц в базе данных содержат поля с характеристиками одинаковыми или практически одинаковыми, то есть целесообразность сделать домен, в котором описать набор свойств поля и использовать такой набор свойств, описанный один раз, в нескольких объектах базы данных. Домены могут использоваться помимо описания полей таблиц и представлений (VIEW) и при объявлении входных и выходных параметров, а также при объявлении переменных в коде PSQL.

Определение домена содержит обязательные и необязательные атрибуты. К обязательному атрибуту относится *тип данных*. К необязательным относятся:

Руководство по языку SQL СУБД Firebird

- значение по умолчанию;
- возможности использования NULL для домена;
- ограничения CHECK для данных домена;
- набор символов (для символьных типов данных и BLOB полей);
- порядок сортировки (для символьных типов данных).

Пример описания домена:

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

См. также: [Явное преобразование типов данных](#), где описаны отличия работы механизма преобразования данных при указании доменов для опций TYPE OF и TYPE OF COLUMN.

При описании таблиц базы данных некоторые свойства столбцов, базирующихся на доменах, могут быть переопределены. Возможности переопределения атрибутов столбцов на базе доменов приведены в таблице 4.7

Таблица 4.7. Возможности переопределения атрибутов столбцов на базе доменов

Атрибут	Может переопределяться	Примечания
тип данных	нет	
значение по умолчанию	да	
текстовый набор символов	да	также может использоваться, чтобы восстановить для столбца значения по умолчанию для базы данных
текстовый порядок сортировки	да	
условия проверки CHECK	нет	для добавления в проверку новых условий, можно использовать в операторах CREATE и ALTER на уровне таблицы соответствующие предложения CHECK.
NOT NULL	нет	во многих случаях лучше оставить при описании домена возможность значения NULL, а контроль его допустимости осуществлять в описании полей на уровне таблицы.

Создание доменов

Создание домена производится командой CREATE.

Краткий синтаксис:

```
CREATE DOMAIN <имя> [AS] <тип_данных>
  [DEFAULT {<константа>|<литерал>|NULL
           |<контекстная_переменная>}]
  [NOT NULL] [CHECK (<условие_проверки>)]
  [COLLATE collation];
```

См. также: [CREATE DOMAIN](#)

Изменение доменов

Для редактирования свойств домена используют оператор ALTER DOMAIN языка определения данных (DDL).

При редактировании домена можно:

- переименовать домен;
- изменить тип данных;
- удалить текущее значение по умолчанию;
- установить новое значение по умолчанию;
- удалить текущее ограничение CHECK;
- добавить новое ограничение CHECK;

Краткий синтаксис:

```
ALTER DOMAIN <имя> [{TO <новое_имя>}]
  [{SET DEFAULT {<константа>|<литерал>|NULL
                |<контекстная_переменная>} | DROP DEFAULT}]
  [{ADD [CONSTRAINT] CHECK (<условие_проверки>)| DROP CONSTRAINT}]
  [{TYPE datatype}];
```

Пример:

```
ALTER DOMAIN STORE_GRP SET DEFAULT -1;
```

При изменении доменов следует учитывать и его зависимости: имеются ли столбцы таблиц; находятся ли в коде PSQL объявления переменных, входных и/или выходных параметров с типом этого домена. При поспешном редактировании без внимательной проверки можно сделать данный код неработоспособным!

Руководство по языку SQL СУБД Firebird

При смене в домене типа данных не допустимы преобразования, которые могут привести к потере данных. Также, например, при преобразовании varchar в integer проверьте, все ли данные, что используют данных домен, смогут пройти преобразование.

См. также: [ALTER DOMAIN](#)

Удаление доменов

Оператор DROP DOMAIN удаляет из базы данных домена при условии, что он не используется в каком либо из объектов базы данных.

Синтаксис:

```
DROP DOMAIN <имя>;
```

Важно:

Удаление домена доступно любому пользователю, подключенному к базе данных.

Пример:

```
DROP DOMAIN Test_Domain;
```

См. также: [DROP DOMAIN](#)

Глава 5

Операторы DDL

Data Definition Language (DDL) – язык описания данных. С помощью этого подмножества языка создаются, модифицируются и удаляются объекты базы данных (т.е. Метаданные).

DATABASE

В данном разделе описываются вопросы создания базы данных, подключения к существующей базе данных, изменения структуры файлов, перевод базы данных в состояние, необходимое для безопасного резервного копирования, и обратно и удаления базы данных.

CREATE DATABASE

Создание новой базы данных.

Доступно: DSQL, ESQL.

Синтаксис:

```
CREATE {DATABASE | SCHEMA} '<filespec>'
    [USER 'username' [PASSWORD 'password']]
    [PAGE_SIZE [=] size]
    [LENGTH [=] num [PAGE[S]]]
    [DEFAULT CHARACTER SET charset [COLLATION collation]]
    [<sec_file> [<sec_file> ...]]
    [DIFFERENCE FILE 'diff_file'];
```

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

<server_spec> ::= servername: | \\servername

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

Таблица 5.1. Параметры оператора CREATE DATABASE

Аргумент	Описание
filespec	Спецификация первичного файла базы данных.
server_spec	Спецификация удалённого сервера. Включает в себя имя сервера и протокол. Необходима, если база данных создаётся на удалённом сервере.
filepath	Полный путь и имя файла, включая расширение. Имя файла должно быть задано в соответствии со спецификой используемой платформы.
db_alias	Псевдоним (alias) базы данных, присутствующий

Руководство по языку SQL СУБД Firebird

Аргумент	Описание
	в файле aliases.conf
servername	Имя сервера или IP адрес, на котором создаётся база данных.
username	Имя пользователя-владельца базы данных. Может включать в себя до 31 символа. Не чувствительно к регистру.
password	Пароль пользователя-владельца базы данных. Может включать в себя до 32 символов, однако только первые 8 имеют значение. Чувствительно к регистру.
size	Размер страницы для базы данных. Допустимые значения 4096 (по умолчанию), 8192, 16384.
num	Максимальный размер первичного или вторичного файла в страницах.
charset	Задаёт набор символов по умолчанию для строковых типов данных.
collation	Сортировка для набора символов по умолчанию.
sec_file	Спецификация вторичного файла
pagenum	Номер страницы, с которой начинается вторичный файл базы данных.
diff_file	Путь и имя дельта файла.

Описание:

Оператор CREATE DATABASE создаёт новую базу данных. Вы можете использовать CREATE DATABASE или CREATE SCHEMA. Это синонимы.

База данных может состоять из одного или нескольких файлов. Первый, основной, файл называется первичным, остальные файлы – вторичными.

Примечание:

В настоящее время многофайловые базы данных являются атавизмом. Многофайловые базы данных имеет смысл использовать на старых файловых системах, в которых существует ограничение на размер любого файла. Например, в FAT32 нельзя создать файл больше 4х гигабайт.

Спецификация первичного файла - имя файла базы данных и его расширение с указанием к нему полного пути в соответствии с правилами используемой операционной системы. При создании базы данных файл базы данных должен отсутствовать. В противном случае будет выдано сообщение об ошибке и база данных не будет создана. Если полный путь к базе данных не указан, то база данных будет создана в одном из системных каталогов. В каком именно зависит от операционной системы.

Вместо полного пути к первичному файлу базы можно использовать

Руководство по языку SQL СУБД Firebird

псевдонимы (aliases). Псевдонимы описываются в файле `aliases.conf` в формате:

```
alias = filepath
```

При создании базы данных на удалённом сервере необходимо указать спецификацию удалённого сервера. Спецификация удалённого сервера зависит от используемого протокола. Если вы при создании базы данных используете протокол TCP/IP, то спецификация первичного файла должна выглядеть следующим образом.

```
servername:{filepath | db_alias}
```

Если вы при создании базы данных используете протокол под названием именованные каналы (Name Pipes), то спецификация первичного файла должна выглядеть следующим образом.

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

Необязательные предложения `USER` и `PASSWORD` задают, соответственно, имя и пароль пользователя присутствующего в базе данных безопасности `security2.fdb`. Пользователя и пароль можно не указывать, если установлены переменные окружения `ISC_USER` и `ISC_PASSWORD`. Пользователь, указанный при создании базы данных, будет её владельцем.

Необязательное предложение `PAGE_SIZE` задаёт размер страницы базы данных. Этот размер будет установлен для первичного файла и всех вторичных файлов базы данных. При вводе размера страницы БД меньшего, чем 4096, он будет автоматически изменён на 4096. Другие числа (не равные 4096, 8192 или 16384) будут изменены на ближайшее меньшее из поддерживаемых значений. Если размер страницы базы данных не указан, то по умолчанию принимается значение 4096.

Необязательное предложение `LENGTH` задаёт максимальный размер первичного или вторичного файла базы данных в страницах. При создании базы данных её первичный или вторичный файл будут занимать минимально необходимое количество страниц для хранения системных данных, не зависимо от величины, установленной в предложении `LENGTH`. Для единственного или последнего (в многофайловой базе данных) файла значение `LENGTH` никак не влияет на его размер. Файл будет автоматически увеличивать свой размер по мере необходимости.

Предложение `DEFAULT CHARACTER SET` задаёт набор символов по умолчанию для строковых типов данных. Наборы символов применяются для типов `CHAR`, `VARCHAR` и `BLOB`. По умолчанию используется набор символов `NONE`. Для набора символов по умолчанию можно также указать сортировку по умолчанию (`COLLATION`). В этом случае сортировка станет умалчиваемой для набора символов по умолчанию (т.е. для всей БД за исключением случаев

Руководство по языку SQL СУБД Firebird

использования других наборов символов).

Предложение `STARTING AT` задает номер страницы базы данных, с которой должен начинаться следующий файл базы данных. Когда предыдущий файл будет полностью заполнен данными в соответствии с заданным номером страницы, система начнет помещать вновь добавляемые данные в следующий файл базы данных.

Необязательное предложение `DIFFERENCE FILE` задаёт путь и имя дельта файла, в который будут записываться изменения, внесённые в БД после перевода её в режим «безопасного копирования» (“copy-safe”) путём выполнения команды `ALTER DATABASE BEGIN BACKUP`. Полное описание данного параметра см. в [ALTER DATABASE](#).

Для того чтобы база данных была создана в нужном вам диалекте SQL, следует перед выполнением оператора создания базы данных задать нужный диалект, выполнив оператор `SET SQL DIALECT`. По умолчанию база данных создаётся в 3 диалекте.

Примеры:

1. Создание базы данных в операционной системе Windows расположенной на диске D с размером страницы 8192. Владелец базы данных будет пользователь wizard. База данных будет в 1 диалекте, и использовать набор символов по умолчанию WIN1251.

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

2. Создание базы данных в операционной системе Linux расположенной на диске D с размером страницы 4096. Владелец базы данных будет пользователь wizard. База данных будет в 3 диалекте, и использовать набор символов по умолчанию UTF8 с умалчиваемой сортировкой UNICODE_CI_AI.

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

3. Создание базы данных на удалённом сервере baseserver расположенном по пути на который ссылается псевдоним test, описанный в файле aliases.conf. Используется протокол TCP. Владелец базы данных будет пользователь wizard. База данных будет в 3 диалекте, и использовать набор символов по умолчанию UTF8.

```
CREATE DATABASE 'baseserver:test'
```

Руководство по языку SQL СУБД Firebird

```
USER 'wizard' PASSWORD 'player'  
DEFAULT CHARACTER SET UTF8;
```

4. Создание базы данных в 3 диалекте с набором символов по умолчанию UTF8. Первичный файл будет содержать 10000 страниц с размером страницы 8192. Как только в процессе работы с базой данных первичный файл будет заполнен, СУБД будет помещать новые данные во вторичный файл test.fdb2. Аналогичные действия будут происходить и со вторым вторичным файлом. Размер последнего файла будет увеличиваться до тех пор, пока это позволяет используемая операционная система или пока не будет исчерпана память на внешнем носителе.

```
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;
```

5. Создание базы данных в 3 диалекте с набором символов по умолчанию UTF8. Первичный файл будет содержать 10000 страниц с размером страницы 8192. Как только в процессе работы с базой данных первичный файл будет заполнен, СУБД будет помещать новые данные во вторичный файл test.fdb2. Аналогичные действия будут происходить и со вторым вторичным файлом.

```
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;
```

См. также: [ALTER DATABASE](#), [DROP DATABASE](#), [CONNECT](#)

CONNECT

Подключение к существующей базе данных.

Доступно: ISQL

Синтаксис:

Руководство по языку SQL СУБД Firebird

```
CONNECT 'filespec'  
  [USER 'username'] [PASSWORD 'password']  
  [CACHE cache_size [BUFFERS]]  
  [ROLE 'rolename'];  
  
<filespec> ::= [<server_spec>]{filepath | db_alias}  
  
<server_spec> ::= servername: | \\servername\
```

Таблица 5.2. Параметры оператора CONNECT

Аргумент	Описание
filespec	Спецификация первичного файла базы данных.
server_spec	Спецификация удалённого сервера. Включает в себя имя сервера и протокол. Необходимо, если осуществляется подключение к удалённой базе данных.
filepath	Полный путь и имя файла, включая расширение. Имя файла должно быть задано в соответствии со спецификой используемой платформы.
db_alias	Псевдоним (alias) базы данных, присутствующий в файле aliases.conf
servername	Имя сервера или IP адрес, на котором находится удалённая база данных.
username	Имя пользователя-владельца базы данных. Может включать в себя до 31 символа. Не чувствительно к регистру.
password	Пароль пользователя-владельца базы данных. Может включать в себя до 32 символов, однако только первые 8 имеют значение. Чувствителен к регистру.
cache_size	Количество буферов кэш-памяти для соединения. По умолчанию 256. Максимальное значение зависит от используемой ОС.
rolename	Имя роли, с которой пользователь соединяется с базой данных. Может включать в себя до 31 символа.

Описание:

Оператор CONNECT осуществляет подключение к существующей базе данных.

В операторе соединения указывается спецификация только первичного файла базы данных, независимо от того существуют ли вторичные. Вместо полного пути к первичному файлу базы можно использовать псевдонимы

Руководство по языку SQL СУБД Firebird

(aliases). Псевдонимы описываются в файле `aliases.conf` в формате:

```
alias = filepath
```

Если в операторе подключения указан только полный путь к первичному файлу базы данных или её псевдоним, то используется локальный протокол (XNET).

```
{filepath | db_alias}
```

Если вам необходимо подключиться к удалённой базе данных, то необходимо указать спецификацию удалённого сервера. Спецификация удалённого сервера зависит от используемого протокола. Если вы хотите подключиться к базе данных, используя протокол TCP/IP, то спецификация первичного файла должна выглядеть следующим образом.

```
servername:{filepath | db_alias}
```

Если вы хотите подключиться к базе данных, используя протокол под названием именованные каналы (Name Pipes), то спецификация первичного файла должна выглядеть следующим образом.

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

Если на компьютере не заданы переменные окружения `ISC_USER` и `ISC_PASSWORD`, то обязательно нужно указывать пользователя (предложение `USER`) и пароль (предложение `PASSWORD`). Если используется не Embedded версия сервера, то пользователь должен существовать в базе данных безопасности `secutity2.fdb`. Пользователя и пароль так же можно не указывать, если используется аутентификация средствами ОС (Trusted Authentication).

В операционных системах семейства UNIX при использовании локального протокола имя пользователя и пароль можно не указывать, если вы работаете под учётной записью пользователя `root`. При подключении вы будете иметь права пользователя `SYSDBA`.

При использовании Embedded версии сервера вы можете не указывать имя пользователя или указать любого без указания пароля, поскольку в этом случае проверка существования пользователя не происходит. При подключении вы будете иметь права указанного пользователя.

Необязательное предложение `ROLE` задаёт имя роли, с которой пользователь соединяется с базой данных. Пользователю, указанному в предложении `USER` должна быть предоставлена указанная роль. При подключении пользователь будет иметь права указанного пользователя и права указанной роли.

Перед соединением с базой данных необходимо установить диалект

Руководство по языку SQL СУБД Firebird

клиента при помощи оператора SET SQL DIALECT и набор символов клиента при помощи оператора SET NAMES (обычно совпадает с набором символов, который был указан при создании базы данных). Если при подключении набор символов не был указан, то по умолчанию используется набор символов NONE.

Предупреждение:

Во избежание ошибок желательно в параметрах соединения клиента указывать диалект, который был указан при создании базы данных.

Примеры:

1. Подключение к базе данных test.fdb используя 1 диалект с набором символов WIN1251 под пользователем wizard.

```
SET SQL DIALECT 1;  
SET NAMES WIN1251;  
CONNECT 'D:\test.fdb'  
USER 'wizard' PASSWORD 'player';
```

2. Подключение к базе данных с псевдонимом test по протоколу TCP/IP под пользователем wizard и ролью MANAGER. Используется 3 диалект с набором символов UTF8.

```
SET SQL DIALECT 3;  
SET NAMES UTF8;  
CONNECT 'baseserver:test'  
USER 'wizard' PASSWORD 'player'  
ROLE 'MANAGER';
```

3. Подключение к базе данных с псевдонимом test с указанием IP адреса сервера и альтернативного порта под пользователем wizard и ролью MANAGER. Используется 3 диалект с набором символов UTF8.

```
SET SQL DIALECT 3;  
SET NAMES UTF8;  
CONNECT '192.168.0.100/3052:test'  
USER 'wizard' PASSWORD 'player'  
ROLE 'MANAGER';
```

См. Также: [CREATE DATABASE](#), [ALTER DATABASE](#), [DROP DATABASE](#)

ALTER DATABASE

Оператор ALTER DATABASE позволяет изменять структуру файлов базы данных или переключать её в "безопасное для копирования" состояние.

Руководство по языку SQL СУБД Firebird

Доступно: DSQL, ESQL.

Синтаксис:

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

<add_sec_clause> ::= ADD FILE <sec_file>

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

Таблица 5.2. Параметры оператора ALTER DATABASE

Аргумент	Описание
add_sec_clause	Инструкция для добавления вторичного файла базы данных.
sec_file	Спецификация вторичного файла.
filepath	Полный путь и имя дельта файла или вторичного файла базы данных.
pagenum	Номер страницы, с которой начинается вторичный файл базы данных.
num	Максимальный размер вторичного файла в страницах.
diff_file	Путь и имя дельта файла.

Описание:

Оператор ALTER DATABASE изменяет структуру файлов базы данных или переключает её в состояние "безопасное для копирования".

Предложение ADD FILE добавляет к базе данных вторичный файл. Для вторичного файла необходимо указывать полный путь к файлу и имя вторичного файла. Описание вторичного файла аналогично тому, что описано в операторе CREATE DATABASE.

Предложение ADD DIFFERENCE FILE задаёт путь и имя дельта файла, в который будут записываться изменения, внесённые в базу данных после перевода её в режим «безопасного копирования» («copy-safe»). Этот оператор в действительности не добавляет файла. Он просто переопределяет умалчиваемые имя и путь файла дельты. Для изменения существующих установок необходимо сначала удалить ранее указанное описание файла дельты с помощью оператора DROP DIFFERENCE FILE, а затем задать новое описание файла дельты. Если не переопределять путь и имя файла дельты, то

Руководство по языку SQL СУБД Firebird

он будет иметь тот же путь и имя, что и БД, но с расширением `.delta`.

Примечание:

При задании относительного пути или только имени файла дельты он будет создаваться в текущем каталоге сервера. Для операционных систем Windows это системный каталог.

Предложение `DROP DIFFERENCE FILE` удаляет описание (путь и имя) файла дельты, заданное ранее командой `ADD DIFFERENCE FILE`. На самом деле при выполнении этого оператора файл не удаляется. Он удаляет путь и имя файла дельты и при последующем переводе БД в режим «безопасного копирования» будут использованы значения по умолчанию (т.е. тот же путь и имя что и у файла БД, но с расширением `.delta`).

Предложение `BEGIN BACKUP` предназначено для перевода базы данных в режим «безопасного копирования» («copy-safe»). Этот оператор «замораживает» основной файл базы данных, что позволяет безопасно делать резервную копию средствами файловой системы, даже если пользователи подключены и выполняют операции с данными. При этом все изменения, вносимые пользователями в базу данных, будут записаны в отдельный файл, так называемый *дельта файл* (*delta file*).

Примечание:

Оператор `BEGIN BACKUP`, не смотря на синтаксис, не начинает резервное копирование базы данных, а лишь создаёт условия для его осуществления.

Предложение `END BACKUP` предназначено для перевода базы данных из режима «безопасного копирования» «copy-safe» в режим нормального функционирования. Этот оператор объединяет файл дельты с основным файлом базы данных и восстанавливает нормальное состояние работы, таким образом, закрывая возможность создания безопасных резервных копий средствами файловой системы. (При этом безопасное резервное копирование с помощью утилиты GBAK остаётся доступным).

Изменить структуру файлов базы данных могут:

- владелец базы данных;
- пользователь `SYSDBA`;
- любой пользователь, подключенный с ролью `RDB$ADMIN` (роль должна быть назначена пользователю);
- пользователь операционной системы `root` (Linux);
- администраторы Windows, если используется доверительная авторизация (`trusted authentication`) и включено автоматическое предоставление роли `RDB$ADMIN` администраторам Windows.

Руководство по языку SQL СУБД Firebird

Примеры:

1. Добавление вторичного файла в базу данных. Как только в предыдущем первичном или вторичных файлах будет заполнено 30000 страниц, СУБД будет помещать данные во вторичный файл test4.fdb.

```
ALTER DATABASE
ADD FILE 'D:\test.fdb4'
STARTING PAGE 30001;
```

2. Установка пути и имени файла дельты

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

3. Удаление описание файла дельты

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

4. Перевод базу данных в режим «безопасного копирования»

```
ALTER DATABASE
BEGIN BACKUP;
```

5. Возвращение базы данных в режим нормального функционирования из режима «безопасного копирования»

```
ALTER DATABASE
END BACKUP;
```

См. Также: [CREATE DATABASE](#), [DROP DATABASE](#), [CONNECT](#)

DROP DATABASE

Оператор DROP DATABASE удаляет базы данных.

Доступно: DSQL, ESQL.

Синтаксис:

```
DROP DATABASE;
```

Описание:

Оператор DROP DATABASE удаляет текущую базу данных. Перед

Руководство по языку SQL СУБД Firebird

удалением базы данных, к ней необходимо присоединиться. Оператор удаляет первичный, все вторичные файлы и все файлы теневых копий (см.).

Базу данных могут удалить:

- владелец базы данных;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

Удаление базы данных, к которой подключен клиент.

DROP DATABASE;

См. также: [CREATE DATABASE](#), [ALTER DATABASE](#), [CONNECT](#)

SHADOW

Теневая копия (shadow — дословно тень) является точной страничной копией базы данных. После создания теневой копии все изменения, сделанные в базе данных, сразу же отражаются и в теневой копии. Если по каким либо причинам первичный файл базы данных станет недоступным, то СУБД переключится на теневую копию.

В данном разделе рассматриваются вопросы создания и удаления теневых копий.

Замечание:

Это относится только к текущим операциям с базой данных, но не к новым подключениям. В случае поломки исходной базы данных администратор БД должен восстановить изначальные файлы базы данных, в том числе и с помощью файлов теневых копий. Только после этого будет возможно подключение новых клиентов.

CREATE SHADOW

Создание теневой копии.

Доступно: DSQL, ESQL

Синтаксис:

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

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

Таблица 5.3. Параметры оператора CREATE SHADOW

Аргумент	Описание
sh_num	Номер теневой копии – положительное число, идентифицирующее набор файлов теневой копии
filepath	Имя файла и путь к нему в соответствии с требованиями ОС
num	Максимальный размер теневой копии в страницах
secondary_file	Спецификация вторичного файла
page_num	Номер страницы, с которой должен начинаться вторичный файл копии

Описание:

Оператор CREATE SHADOW создаёт новую теневую копию. Теневая копия начинает дублировать базу данных сразу в момент создания этой копии.

Если выбран режим AUTO (значение по умолчанию), то в случае, когда теневая копия становится недоступной, автоматически прекращается использование этой копии и из базы данных удаляются все ссылки на неё. Работа с базой данных продолжается обычным образом без осуществления копирования в данную теневую копию.

Если выбран режим MANUAL, то в случае, когда теневая копия становится недоступной, все попытки соединения с базой данных и обращения к ней будут вызывать сообщение об ошибке до тех пор, пока теневая копия не станет доступной или пока не будет удалена администратором БД с помощью оператора DROP SHADOW.

В случае, когда теневая копия заменяет базу данных, для этой копии можно создать новую теневую копию, которая начнёт выполнять функцию резервного копирования. При создании такой копии необходимо указывать ключевое слово CONDITIONAL. Эта условная теневая копия будет заменять бывшую активную перед этим теневую копию, которая стала выполнять функцию базы данных.

Теневые копии, как и база данных, могут состоять из нескольких файлов. Количество и размер файлов теневых копий не связано с количеством и

Руководство по языку SQL СУБД Firebird

размером файлов базы данных.

Необязательное предложение `LENGTH` задаёт максимальный размер первичного или вторичного файла теневой копии в страницах. Для единственного или последнего файла теневой копии значение `LENGTH` никак не влияет на его размер. Файл будет автоматически увеличивать свой размер по мере необходимости.

Предложение `STARTING AT` задает номер страницы теневой копии, с которой должен начинаться следующий файл теневой копии. Когда предыдущий файл будет полностью заполнен данными в соответствии с заданным номером страницы, система начнет помещать вновь добавляемые данные в следующий файл теневой копии.

Для файлов теневой копии размер страницы устанавливается равным размеру страницы базы данных и не может быть изменён.

Создать теневую копию могут:

- владелец базы данных;
- пользователь `SYSDBA`;
- любой пользователь, подключенный с ролью `RDB$ADMIN` (роль должна быть назначена пользователю);
- пользователь операционной системы `root` (Linux);
- администраторы Windows, если используется доверительная авторизация (`trusted authentication`) и включено автоматическое предоставление роли `RDB$ADMIN` администраторам Windows.

Примеры:

1. Создание теневую копию базы данных с номером 1

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

2. Создание многофайловой теневой копии

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

См. Также: [CREATE DATABASE](#), [CONNECT](#), [DROP SHADOW](#)

DROP SHADOW

Удаляет теневую копию.

Руководство по языку SQL СУБД Firebird

Доступно: SQL, ESQL

Синтаксис:

```
DROP SHADOW sh_num;
```

Таблица 5.4 Параметры оператора DROP SHADOW

Аргумент	Описание
sh_num	Номер теневой копии – положительное число, идентифицирующее набор файлов теневой копии

Описание:

Оператор DROP SHADOW удаляет указанную теневую копию из базы данных, с которой установлено текущее соединение. При удалении теневой копии удаляются все связанные с ней файлы, и прекращается процесс дублирования данных в эту теневую копию.

Удалить теневую копию могут:

- владелец базы данных;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

Удаление теневой копии с номером 1.

```
DROP SHADOW 1;
```

См. также: [CREATE SHADOW](#)

DOMAIN

Домен (Domain) – один из объектов реляционной базы данных, при создании которого можно задать некоторые характеристики, а затем использовать ссылку на домен при определении столбцов таблиц, объявлении локальных переменных, входных и выходных аргументов в модулях PSQL.

В данном разделе рассматриваются синтаксис операторов создания, модификации и удаления доменов. Подробное описание доменов и их использования можно прочесть в главе [Работа с доменами](#).

CREATE DOMAIN

Создание нового домена.

Доступно: DSQL, ESQL

Синтаксис:

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

```
<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]
| {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING]
    [(size)] [<array_dim>]
| BLOB [SUB_TYPE {subtype_num | subtype_name}]
    [SEGMENT SIZE seglen] [CHARACTER SET charset]
| BLOB [(seglen [, subtype_num])]
```

```
<array_dim> ::= [x:y [,x:y ...]]
```

```
<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 <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>
}
```

Руководство по языку SQL СУБД Firebird

`<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> ...])
}
    
```

Таблица 5.4. Параметры оператора CREATE DOMAIN

Аргумент	Описание
name	Имя домена. Может содержать до 31 символа.
datatype	Тип данных SQL.
literal	Литерал.
context_var	Любая контекстная переменная, тип которой совместим с типом данных домена.
dom_condition	Условие домена.
collation	Порядок сортировки.
array_dim	Размерность массива.
x	Начальный номер элемента в массиве, положительное целое число.
y	Последний номер элемента в массиве, положительное целое число.
precision	Точность. От 1 до 18.
scale	Масштаб. От 0 до 18, должно быть меньше или равно precision.
size	Максимальный размер строки в символах.
charset	Набор символов.
subtype_num	Номер подтипа BLOB.
subtype_name	Мнемоника подтипа BLOB.
seglen	Размер сегмента, не может превышать 65535.
val	Значение.
select_one	Оператор SELECT выбирающий один столбец и возвращающий только одну строку.
select_list	Оператор SELECT выбирающий один столбец и возвращающий ноль и более строк.
select_expr	Оператор SELECT выбирающий несколько столбцов и возвращающий ноль и более строк.
expression	Выражение.
genname	Имя последовательности (генератора).
func	Встроенная или UDF функция.

Руководство по языку SQL СУБД Firebird

Описание:

Оператор CREATE DOMAIN создаёт новый домен.

В качестве типа домена можно использовать любой тип данных SQL. Для любого типа данных кроме BLOB можно указать размерность массива, если домен должен быть массивом. Размерность массива указывается в квадратных скобках. Чтобы не перепутать их с символами, обозначающими необязательные элементы, они выделены жирным шрифтом. При указании размерности массива указываются два числа через двоеточие. Первое число означает начальный номер элемента массива, второе – конечный. Если указано только одно число, то оно означает последний номер в элементе массива, а первым номером считается 1. Если у массива два и более измерения, то они перечисляются через запятую.

Для типов CHAR, VARCHAR и BLOB с подтипом text можно указать набор символов в предложении CHARACTER SET. Если набор символов не указан, то по умолчанию принимается тот набор символов, который был указан при создании базы данных. Если же при создании базы данных не был указан набор символов, то при создании домена по умолчанию принимается набор символов NONE. В этом случае данные хранятся и извлекаются, так как они были поданы. В столбец, основанный на таком домене, можно загружать данные в любой кодировке, но невозможно загрузить эти данные в столбец с другой кодировкой. Транслитерация не выполняется между исходными и конечными кодировками, что может приводить к ошибкам.

Необязательное предложение DEFAULT позволяет указать значение по умолчанию для домена. Это значение будет помещено в столбец таблицы, который ссылается на данный домен, при выполнении оператора INSERT, если значение не будет указано для этого столбца. Локальные переменные и аргументы PSQL модулей, которые ссылаются на этот домен, будут проинициализированы значением по умолчанию. В качестве значения по умолчанию может быть литерал совместимый по типу, неизвестное значение NULL и контекстная переменная, тип которой совместим с типом домена.

Предложение NOT NULL запрещает столбцам и переменным, основанным на домене, присваивать значение NULL.

Замечание:

Будьте осторожны при создании домена. Создавая домен можно указать противоречивые ограничения, например, NOT NULL и DEFAULT NULL одновременно.

Необязательное предложение CHECK задаёт ограничение домена. Ограничение домена задаёт условия, которому должны удовлетворять значения столбцов таблицы или переменных, которые ссылаются на данный домен.

Руководство по языку SQL СУБД Firebird

Условие должно быть помещено в круглые скобки. Условие – это логическое выражение, называемое также предикат, которое может возвращать значения TRUE (истина), FALSE (ложь) и UNKNOWN (неизвестно). Условие считается выполненным, если предикат возвращает значение TRUE или UNKNOWN (эквивалент NULL). Если предикат возвращает FALSE, то значение не будет принято.

Ключевое слово VALUE в ограничении домена является заменителем столбца таблицы, который основан на данном домене, или переменной PSQL модуля. Оно содержит значение, присваиваемое переменной или столбцу таблицы. Ключевое слово VALUE может быть использовано в любом месте ограничения CHECK, но обычно его используют в левой части условия.

Необязательное предложение COLLATE позволяет задать порядок сортировки, если домен основан на одном из строковых типов данных (за исключением BLOB). Если порядок сортировки не указан, то по умолчанию принимается порядок сортировки умалчиваемый для указанного набора сортировки при создании домена.

Создать домен может любой пользователь, соединившийся с базой данных.

Примеры:

1. Создание домена, который может принимать значения больше 1000, значением по умолчанию будет число 10000.

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

2. Создание домена, который может принимать значения 'Да' и 'Нет', с набором символов указанным при создании базы данных.

```
CREATE DOMAIN D_BOOLEAN AS  
CHAR(3) CHECK (VALUE IN 'Да', 'Нет');
```

3. Создание домена с набором символов UTF8 и порядком сортировки UNICODE_CI_AI.

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

4. Создание домена типа DATE, который не может принимать значение NULL и в качестве значения по умолчанию использует текущую дату.

```
CREATE DOMAIN D_DATE AS
```

Руководство по языку SQL СУБД Firebird

```
DATE DEFAULT CURRENT_DATE  
NOT NULL;
```

5. Создание домена, определённого как массив из 2 элементов типа NUMERIC(18, 3), нумерация элементов начинается с 1.

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

Замечание:

Вы можете использовать домены определённые как массив только для определения столбцов таблиц. Вы не можете использовать такие домены для определения локальных переменных и аргументов PSQL модулей.

6. Создание домена, элементами которого могут быть только коды стран описанные в таблице COUNTRY.

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

Замечание:

Пример приведён лишь в качестве демонстрации возможности использования предикатов с запросами в условии проверки домена. На практике такие домены создавать не рекомендуется.

См. также: [ALTER DOMAIN](#), [DROP DOMAIN](#)

ALTER DOMAIN

Изменение текущих характеристик домена и его переименование.

Доступно: DSQL, ESQL

Синтаксис:

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

Руководство по языку SQL СУБД Firebird

```
[{TYPE datatype}];
```

```
<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]  
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING]  
  [(size)] [<array_dim>]  
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]  
  [SEGMENT SIZE seglen] [CHARACTER SET charset]  
  | BLOB [(seglen [, subtype_num])]
```

```
<array_dim> ::= [x:y [,x:y ...]]
```

```
<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 <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> ...])
```

}

Таблица 5.5. Параметры оператора ALTER DOMAIN

Аргумент	Описание
name	Имя домена.
new_name	Новое имя домена.
datatype	Тип данных SQL.
literal	Литерал.
context_var	Любая контекстная переменная, тип которой совместим с типом данных домена.
dom_condition	Условие домена.
collation	Порядок сортировки.
array_dim	Размерность массива.
x	Начальный номер элемента в массиве, положительное целое число.
y	Последний номер элемента в массиве, положительное целое число.
precision	Точность. От 1 до 18.
scale	Масштаб. От 0 до 18, должно быть меньше или равно precision.
size	Максимальный размер строки в символах.
charset	Набор символов.
subtype_num	Номер подтипа BLOB.
subtype_name	Мнемоника подтипа BLOB.
seglen	Размер сегмента, не может превышать 65535.
val	Значение.
select_one	Оператор SELECT выбирающий один столбец и возвращающий только одну строку.
select_list	Оператор SELECT выбирающий один столбец и возвращающий ноль и более строк.
select_expr	Оператор SELECT выбирающий несколько столбцов и возвращающий ноль и более строк.
expression	Выражение.
genname	Имя последовательности (генератора).
func	Встроенная или UDF функция.

Описание:

Оператор ALTER DOMAIN изменяет текущие характеристики домена, в том числе и его имя. В одном операторе ALTER DOMAIN можно выполнить любое количество изменений домена.

Предложение TO позволяет переименовать домен. Имя домена можно изменить, если не существует зависимостей от этого домена, т.е. столбцов таблиц, локальных переменных и аргументов процедур, ссылающихся на данный домен.

Руководство по языку SQL СУБД Firebird

Предложение SET DEFAULT позволяет установить новое значение по умолчанию. Если домен уже содержал значение по умолчанию, то установка нового значения по умолчанию не требует предварительного удаления старого.

Предложение DROP DEFAULT удаляет ранее установленное для домена значение по умолчанию. В этом случае значением по умолчанию становится значение NULL.

Предложение ADD [CONSTRAINT] CHECK добавляет условие ограничения домена. Если домен уже содержал ограничение CHECK, то его предварительно необходимо удалить с помощью предложения DROP [CONSTRAINT] CHECK.

Предложение TYPE позволяет изменить тип домена на другой допустимый тип. Не допустимы любые изменения типа, которые могут привести к потере данных. Например, количество символов в новом типе для домена не может быть меньше, чем было установлено ранее. Если домен был объявлен как массив, то изменить ни его тип, ни размерность нельзя.

Замечание:

На данный момент не существует возможности установить и снять ограничение NOT NULL для домена. Кроме того, не существует способа изменить сортировку по умолчанию. В этом случае необходимо удалить домен и пересоздать его с новыми атрибутами.

Изменить описание домена может любой пользователь, соединившийся с базой данных.

Замечание:

При изменении описания домена, существующий PSQL код, может стать некорректным. Информация о том, как это обнаружить, находится в приложении .

Примеры:

1. Изменение значения по умолчанию на новое равное 2000.

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

2. Переименование домена.

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

3. Удаление значения по умолчанию и добавления ограничения для домена.

```
ALTER DOMAIN D_DATE
```

Руководство по языку SQL СУБД Firebird

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

4. Изменение ограничения домена.

```
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. Изменение типа домена, увеличение количества допустимых символов.

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

См. также: [CREATE DOMAIN](#), [DROP DOMAIN](#)

DROP DOMAIN

Удаление существующего домена.

Доступно: DSQL, ESQL

Синтаксис:

```
DROP DOMAIN domain_name;
```

Таблица 5.6. Параметры оператора DROP DOMAIN

Аргумент	Описание
domain_name	Имя домена.

Описание:

Оператор DROP DOMAIN удаляет домен, существующий в базе данных. Невозможно удалить домен, на который ссылаются столбцы таблиц базы данных или если он был задействован в одном из PSQL модулей. Чтобы удалить такой домен необходимо удалить из таблиц все столбцы, ссылающиеся на домен и удалить все ссылки на домен из PSQL модулей.

Удалить домен может любой пользователь, соединившийся с базой данных.

Примеры:

Руководство по языку SQL СУБД Firebird

Удаление домена COUNTRYNAME

```
DROP DOMAIN COUNTRYNAME;
```

См. также: [CREATE DOMAIN](#), [ALTER DOMAIN](#)

TABLE

Firebird - это реляционная СУБД. Данные в таких базах хранятся в таблицах. Таблица – это плоская двумерная структура, содержащая произвольное количество строк (row). Строки таблицы часто называют записями (record). Все строки таблицы имеют одинаковую структуру и состоят из столбцов (column). Столбцы таблицы часто называют полями (fields). Таблица должна иметь хотя бы один столбец. С каждым столбцом связан определённый тип данных SQL.

В данном разделе рассматриваются вопросы создания, модификации и удаления таблиц базы данных.

CREATE TABLE

Создание новой таблицы.

Доступно: DSQL, ESQL

Синтаксис:

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

<col_def> ::= colname
{
    { <datatype> | domainname }
    [DEFAULT {literal | NULL | <context_var>}]
    [NOT NULL]
    [<col_constraint>]
    [COLLATE collation]
} | <col_exp_def>

<col_exp_def> ::= [<datatype>]
    {COMPUTED [BY] | GENERATED ALWAYS AS} (<col_expr>)

<datatype> ::=
    {SMALLINT | INTEGER | BIGINT} [<array_dim>]
    | {FLOAT | DOUBLE PRECISION} [<array_dim>]
```

Руководство по языку SQL СУБД Firebird

```
| {DATE | TIME | TIMESTAMP} [<array_dim>]
| {DECIMAL | NUMERIC} [(precision [, scale])] [<array_dim>]
| {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
  [<array_dim>] [CHARACTER SET charset]
| {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING]
  [(size)] [<array_dim>]
| BLOB [SUB_TYPE {subtype_num | subtype_name}]
  [SEGMENT SIZE seglen] [CHARACTER SET charset]
| BLOB [(seglen [, subtype_num])]

<array_dim> ::= [x:y [,x:y ...]]

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

<tconstraint> ::= [CONSTRAINT constr_name]
{
  UNIQUE (colname [, colname ...]) [<using_index>]
| PRIMARY KEY (colname [, colname ...]) [<using_index>]
| FOREIGN KEY (colname [, colname ...]) REFERENCES
other_table
  [(other_col [, other_col ...])] [<using_index>]
  [ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET
NULL}]
  [ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET
NULL}]
| CHECK (<search_condition>)
}

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

<search_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 <val>
| <val> [NOT] CONTAINING <val>
```

Руководство по языку SQL СУБД Firebird

```

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

```

Таблица 5.7. Параметры оператора CREATE TABLE

Аргумент	Описание
tablename	Имя таблицы, может содержать до 31 символа.
filespec	Спецификация файла (только для внешних таблиц).
col_def	Определение столбца.
colname	Имя столбца таблицы, может содержать до 31 символа.
datatype	Тип данных SQL.
col_expr	Выражение для вычисляемого столбца.
domain_name	Имя домена.
col_constraint	Ограничение столбца.
tconstraint	Ограничение таблицы.
constr_name	Имя ограничения, может содержать до 31 символа.
other_table	Имя таблицы, на которую ссылается внешний ключ.
other_col	Столбец таблицы, на которую ссылается внешний ключ.
literal	Литерал.
context_var	Любая контекстная переменная, тип которой совместим с типом данных столбца.
search_condition	Условие проверки ограничения.
collation	Порядок сортировки.

Руководство по языку SQL СУБД Firebird

Аргумент	Описание
array_dim	Размерность массива.
x	Начальный номер элемента в массиве, положительное целое число.
y	Последний номер элемента в массиве, положительное целое число.
precision	Точность. От 1 до 18
scale	Масштаб. От 0 до 18, должно быть меньше или равно precision.
size	Максимальный размер строки в символах.
charset	Набор символов.
subtype_num	Номер подтипа BLOB.
subtype_name	Мнемоника подтипа BLOB.
seglen	Размер сегмента, не может превышать 65535.
select_one	Оператор SELECT выбирающий один столбец и возвращающий только одну строку.
select_list	Оператор SELECT выбирающий один столбец и возвращающий ноль и более строк.
select_expr	Оператор SELECT выбирающий несколько столбцов и возвращающий ноль и более строк.
experssion	Выражение.
genname	Имя последовательности (генератора).
func	Встроенная или UDF функция.

Описание:

Оператор CREATE TABLE создаёт новую таблицу. Имя таблицы должно быть уникальным среди имён всех таблиц, представлений (VIEWS) и хранимых процедур базы данных.

Таблица может содержать, по меньшей мере, один столбец и произвольное количество ограничений таблицы.

Необязательное предложение EXTERNAL [FILE] указывает, что таблица хранится вне базы данных во внешнем текстовом файле. Столбцы таблицы, хранящейся во внешнем файле, могут быть любого типа за исключением BLOB. Недопустимо также использование массивов с любым типом данных. Над таблицей, хранящейся во внешнем файле, допустимы только операции добавления новых строк (INSERT) и выборки (SELECT) данных. Операции же изменения существующих данных (UPDATE) или удаления строк такой таблицы (DELETE) не могут быть выполнены. Файл с внешней таблицей должен располагаться на устройстве хранения, физически расположенном на сервере, на котором расположена СУБД. Если при обращении к таблице СУБД не находит файла, она его создает. Возможность использования для таблиц внешних файлов зависит от установки значения параметра ExternalFileAccess в файле конфигурации firebird.conf.

Таблица должна содержать не менее одного столбца. Имя столбца должно

Руководство по языку SQL СУБД Firebird

быть уникальным для создаваемой таблицы. Для столбца обязательно должен быть указан либо тип данных, либо имя домена, характеристики которого будут скопированы для столбца, либо должно быть указано, что столбец является вычисляемым.

В качестве типа столбца можно использовать любой тип данных SQL. Для любого типа данных кроме BLOB можно указать размерность массива, если столбец должен быть массивом. Размерность массива указывается в квадратных скобках. Чтобы не перепутать их с символами, означающими необязательные элементы, они выделены жирным шрифтом. При указании размерности массива указываются два числа через двоеточие. Первое число означает начальный номер элемента массива, второе – конечный. Если указано только одно число, то оно означает последний номер в элементе массива, а первым номером считается 1. Для многомерного массива размерности массива перечисляются через запятую.

Для типов CHAR, VARCHAR и BLOB с подтипом text можно указать набор символов в предложении CHARACTER SET. Если набор символов не указан, то по умолчанию принимается тот набор символов, что был указан при создании базы данных. Если же при создании базы данных не был указан набор символов, то по умолчанию принимается набор символов NONE. В этом случае данные хранятся и извлекаются, так как они были поданы. В столбец можно загружать данные в любой кодировке, но невозможно загрузить эти данные в столбец с другой кодировкой. Транслитерация между исходными и конечными кодировками не выполняется, что может приводить к ошибкам.

Необязательное предложение COLLATE позволяет задать порядок сортировки для строковых типов данных (за исключением BLOB). Если порядок сортировки не указан, то по умолчанию принимается порядок сортировки умолчательный для указанного набора сортировки.

Необязательное предложение DEFAULT позволяет указать значение по умолчанию для столбца таблицы. Это значение будет помещено в столбец таблицы при выполнении оператора INSERT, если значение не будет указано для этого столбца. В качестве значения по умолчанию может быть литерал совместимый по типу, неизвестное значение NULL или контекстная переменная, тип которой совместим с типом столбца. Если значение по умолчанию явно не устанавливается, то подразумевается пустое значение, NULL. Использование выражений в значении по умолчанию недопустимо.

Необязательное предложение NOT NULL указывает, что столбцу не может быть присвоено значение NULL.

Для определения столбца, можно воспользоваться ранее описанным доменом. Если определение столбца основано на домене, оно может включать новое значение по умолчанию, дополнительные ограничения CHECK, предложение COLLATE, которые перекрывают значения указанные при определении домена. Определение такого столбца может включать дополнительные ограничения столбца, например NOT NULL, если домен его

Руководство по языку SQL СУБД Firebird

ещё не содержит. Следует обратить внимание на то, что если в определении домена было указано NOT NULL, на уровне столбца невозможно определить допустимость использования в нем значения NULL.

Вычисляемые поля могут быть определены с помощью предложения COMPUTED [BY] или GENERATED ALWAYS AS (согласно стандарту SQL-2003). Они эквивалентны по смыслу. Для вычисляемых полей не требуется описывать тип данных (но допустимо), СУБД вычисляет подходящий тип в результате анализа выражения. В выражении требуется указать корректную операцию для типов данных столбцов, входящих в его состав. При явном указании типа столбца для вычисляемого поля результат вычисления приводится к указанному типу, то есть, например, результат числового выражения можно вывести как строку. Вычисление выражения происходит для каждой строки выбранных данных, если в операторе выборки данных SELECT, присутствует такой столбец.

Замечание:

Вместо использования вычисляемого столбца в ряде случаев имеет смысл использовать обычный столбец, значение которого рассчитывается в триггерах на добавление и обновление данных. Это может снизить производительность вставки/модификации записей, но повысит производительность выборки данных.

Ограничение столбца записывается после определения других характеристик столбца. Существуют четыре вида ограничений столбца:

- первичный ключ (PRIMARY KEY);
- уникальный ключ (UNIQUE);
- внешний ключ (REFERENCES);
- проверочное ограничение столбца (CHECK).

Необязательное предложение CONSTRAINT задаёт имя ограничения. Если имя ограничения не задано, то ограничению будет присвоено имя, автоматически сгенерированное СУБД.

Для первичного ключа (PRIMARY KEY), уникального ключа (UNIQUE) и внешнего ключа (REFERENCES) система автоматически создает соответствующий индекс. Если задано имя ограничения, то автоматически создаваемому индексу будет присвоено это имя (при отсутствии предложения USING). Предложение USING позволяет задать определённое пользователем имя автоматически создаваемого индекса, и опционально определить, какой это будет индекс - по возрастанию (по умолчанию) или по убыванию.

Ограничение первичного ключа PRIMARY KEY строится на поле с заданным ограничением NOT NULL и требует уникальности значений столбца. Таблица может иметь только один первичный ключ.

Руководство по языку SQL СУБД Firebird

Ограничение уникального ключа UNIQUE задает для значений столбца требование уникальности содержимого. В отличие от первичного ключа, в таких полях допускаются значения NULL. Строк уникального ключа со значением NULL в таблице может быть произвольное количество. Таблица может содержать любое количество уникальных ключей.

Ограничение REFERENCES определяет внешний ключ. После ключевого слова REFERENCES указывается имя родительской таблицы, на первичный или уникальный ключ которой ссылается описываемый внешний ключ. Имя столбца родительской таблицы, являющегося первичным (уникальным) ключом, помещается сразу после имени таблицы и заключается в круглые скобки.

Внешний ключ может иметь пустое значение NULL (если для столбца не задано ограничение NOT NULL) или же он должен ссылаться на первичный или уникальный ключ другой или той же самой таблицы (родительской таблицы). Для обеспечения дополнительной целостности данных можно указать необязательные опции ON DELETE и ON UPDATE, которые обеспечат согласованность данных между родительскими и дочерними таблицами по заданным правилам.

Предложение ON UPDATE определяет, что произойдет с записями подчиненной таблицы при изменении значения первичного/уникального ключа в строке главной таблицы.

Предложение ON DELETE определяет, что произойдет с записями подчиненной таблицы при удалении соответствующей строки главной таблицы.

Для обеспечения ссылочной целостности внешнего ключа, когда изменяется или удаляется значение связанного первичного или уникального ключа, могут быть выполнены следующие действия:

- NO ACTION – не будет выполнено никаких действий; в клиентской программе должны быть предприняты специальные меры по поддержанию ссылочной целостности данных, иначе обновление/удаление не будет произведено и будет выдано соответствующее сообщение об ошибке;
- CASCADE – при изменении или удалении значения первичного ключа над значением внешнего ключа будут произведены те же действия. При выполнении удаления строки в главной таблице в подчиненной таблице должны быть удалены все записи, имеющие те же значения внешнего ключа, что и значение первичного (уникального) ключа удаленной строки главной таблицы. При выполнении обновления записи главной таблицы в подчиненной таблице должны быть изменены все значения внешнего ключа, имеющие те же значения, что и значение первичного (уникального) ключа изменяемой строки главной таблицы;
- SET DEFAULT – значения внешнего ключа всех соответствующих строк в подчиненной таблице устанавливаются в значение по умолчанию, заданное в предложении DEFAULT для этого столбца;
- SET NULL – значения внешнего ключа всех соответствующих строк в подчиненной таблице устанавливаются в пустое значение NULL.

Руководство по языку SQL СУБД Firebird

Ограничение CHECK задаёт условие, которому должны удовлетворять значения, помещаемые в данный столбец. Условие – это логическое выражение, называемое также предикат, которое может возвращать значения TRUE (истина), FALSE (ложь) и UNKNOWN (неизвестно). Условие считается выполненным, если предикат возвращает значение TRUE или UNKNOWN (эквивалент NULL). Если предикат возвращает FALSE, то значение не будет принято. Это условие используется при добавлении в таблицу новой строки (оператор INSERT) и при изменении существующего значения столбца таблицы (оператор UPDATE), а также операторов, в которых может произойти одно из этих действий (UPDATE OR INSERT, MERGE).

При использовании предложения CHECK для столбца, базирующегося на домене, следует помнить, что выражение в CHECK лишь дополняет условие проверки, которое может уже быть определено в домене.

Помимо ограничений столбцов, можно также задать ограничения таблицы. Ограничения таблицы являются более универсальным способом записи ограничений, поскольку позволяют ограничить более чем для одного столбца таблицы. Существуют четыре вида ограничений таблицы:

- первичный ключ (PRIMARY KEY);
- уникальный ключ (UNIQUE);
- внешний ключ (FOREIGN KEY);
- проверочное ограничение для строки таблицы (CHECK).

Необязательное предложение CONSTRAINT задаёт имя ограничения таблицы. Если имя ограничения не задано, то ограничению будет присвоено имя, автоматически сгенерированное системой.

Для первичного ключа (PRIMARY KEY), уникального ключа (UNIQUE) и внешнего ключа (FOREIGN KEY) система автоматически создает соответствующий индекс. Если задано имя ограничения, то автоматически создаваемому индексу будет присвоено это имя (при отсутствии предложения USING). Предложение USING позволяет задать определённое пользователем имя автоматически создаваемого индекса, и опционально определить какой это будет индекс по возрастанию (по умолчанию) или по убыванию.

Ограничение первичного ключа PRIMARY KEY строится на поле или полях, с заданным ограничением NOT NULL и требует уникальности к значению столбца или комбинации значений столбцов. После ключевых слов PRIMARY KEY указывается список столбцов создаваемой таблицы, которые входят в состав первичного ключа. Список заключается в круглые скобки. Таблица может иметь только один первичный ключ.

Ограничение уникального ключа UNIQUE задает для значений столбца или комбинации значений столбцов требование уникальности содержимого. После ключевого слова UNIQUE указывается список столбцов создаваемой таблицы, которые входят в состав уникального ключа. Список заключается в круглые скобки. Таблица может содержать любое количество уникальных ключей. В

Руководство по языку SQL СУБД Firebird

отличие от первичного ключа, в таких полях допускаются значения NULL. Таким образом, можно определить уникальный ключ на столбец, который не имеет ограничения NOT NULL. Для уникальных ключей, содержащих несколько столбцов, логика немного сложнее:

- Во всех столбцах, входящих в уникальный ключ, нет ограничения NOT NULL;
- Разрешено хранение значения NULL в столбцах, входящих в уникальный ключ, в нескольких строках;
- Разрешены строки, имеющие в одном из столбцов уникального ключа значение NULL, а остальные столбцы заполнены значениями и эти значения различны хотя бы в одном из них;
- Запрещены строки, имеющие в одном из столбцов уникального ключа значение NULL, а остальные столбцы заполнены значениями, и эти значения имеют совпадения хотя бы в одном из них.

Замечание:

Для уникальных ключей, содержащих несколько столбцов, допускаются дубликаты значений NULL только тогда, когда они помещаются во все столбцы ограничения. В остальных случаях значения NULL будут рассматриваться как одно из обычных значений (не UNKNOWN).

```
RECREATE TABLE t( x int, y int, z int, unique(z,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 ); -- Разрешено
INSERT INTO t values( NULL, NULL, 1 ); - Запрещено
```

Ограничение FOREIGN KEY определяет внешний ключ. Синтаксис определения внешнего ключа на уровне таблицы отличается от синтаксиса определения внешнего ключа на уровне столбца. После ключевых слов FOREIGN KEY указывается список столбцов создаваемой таблицы, которые входят в состав внешнего ключа. Список заключается в круглые скобки.

После ключевого слова REFERENCES указывается имя родительской таблицы, на первичный или уникальный ключ которой ссылается описываемый внешний ключ. Список имен столбцов родительской таблицы, входящих в состав первичного (уникального) ключа помещается сразу после имени таблицы и заключается в круглые скобки. Структура внешнего ключа дочерней таблицы по количеству столбцов и по типам данных, включая размерность символьных данных, должна полностью соответствовать структуре первичного (уникального) ключа родительской таблицы. Совпадения имен не требуется. Упорядоченность индекса внешнего ключа должна соответствовать упорядоченности индекса первичного (уникального) ключа, на который ссылается данный внешний ключ.

Руководство по языку SQL СУБД Firebird

Для обеспечения дополнительной целостности данных, можно указать необязательные опции `ON DELETE` и `ON UPDATE`, которые обеспечат согласованность данных между родительскими и дочерними таблицами по заданным правилам. Возможные действия при операциях изменения/удаления записи описаны ранее при рассмотрении ограничений столбца.

Ограничение `CHECK` задает условия, которым должны удовлетворять значения столбцов данной таблицы при помещении в таблицу новой строки или при изменении значений отдельных столбцов существующей строки таблицы. Описание ограничения `CHECK` для строк таблицы полностью совпадает с описанием ограничения `CHECK` для столбца таблицы.

Если в операторе создания таблицы указано необязательное предложение `GLOBAL TEMPORARY`, то вместо обычной таблицы будет создана глобальная временная таблица. Глобальные временные таблицы (в дальнейшем сокращённо ГТТ) так же, как и обычные таблицы, являются постоянными метаданными, но данные в них ограничены по времени существования транзакцией (значение по умолчанию) или соединением с БД. Каждая транзакция или соединение имеет свой собственный экземпляр ГТТ с данными, изолированный от всех остальных. Экземпляры создаются только при условии обращения к ГТТ, и данные в ней удаляются при подтверждении транзакции или отключении от БД.

Если в операторе создания глобальной временной таблицы указано необязательное предложение `ON COMMIT DELETE ROWS`, то будет создана ГТТ транзакционного уровня (по умолчанию). При указании предложения `COMMIT PRESERVE ROWS` – будет создана ГТТ уровня соединения с базой данных.

Предложение `EXTERNAL [FILE]` нельзя использовать для глобальной временной таблицы.

Глобальные временные таблицы имеют ряд ограничений:

- ГТТ и обычные таблицы не могут ссылаться друг на друга;
- ГТТ уровня соединения (“`PRESERVE ROWS`”) ГТТ не могут ссылаться на ГТТ транзакционного уровня (“`DELETE ROWS`”);
- Доменные ограничения не могут использоваться в ГТТ;
- Уничтожения экземпляра ГТТ в конце своего жизненного цикла не вызывает срабатывания триггеров до/после удаления.

Создать новую таблицу может любой пользователь, подключённый к базе данных. Пользователь, создавший таблицу, становится её владельцем.

Примеры:

1. Создание таблицы `COUNTRY` с заданием первичного ключа как ограничение столбца.

Руководство по языку SQL СУБД Firebird

```
CREATE TABLE COUNTRY (  
    COUNTRY COUNTRYNAME NOT NULL PRIMARY KEY,  
    CURRENCY VARCHAR(10) NOT NULL);
```

2. Создание таблицы STOCK с заданием именованного первичного ключа на уровне столбца и именованного уникального ключа на уровне таблицы.

```
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. Создание таблицы JOB с ограничениями первичного ключа для трёх столбцов, внешнего ключа на таблицу COUNTRY и CHECK ограничение для строки таблицы. Таблица содержит массив из 5 элементов.

```
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, JOB_COUNTRY),  
    FOREIGN KEY (JOB_COUNTRY) REFERENCES COUNTRY (COUNTRY)  
    ON UPDATE CASCADE  
    ON DELETE SET NULL,  
    CONSTRAINT CHK_SALARY CHECK (MIN_SALARY < MAX_SALARY)  
);
```

4. Создание таблицы PROJECT с ограничениями первичного, внешнего и уникального ключей с заданием для них пользовательских имён индексов с помощью предложения USING.

```
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)
```

Руководство по языку SQL СУБД Firebird

```
USING INDEX IDX_LEADER
);
```

5. Создание таблицы SALARY_HISTORY с двумя вычисляемыми полями. Первое объявлено согласно стандарту SQL-2003, второе – согласно классическому стилю объявления вычисляемых полей в Firebird.

```
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. Создание внешней таблицы EXT_LOG.

```
CREATE TABLE EXT_LOG
EXTERNAL FILE 'log.txt' (
    BYTIME TIMESTAMP,
    AMESSAGE VARCHAR(100)
);
```

7. Создание глобальной временной таблицы уровня соединения.

```
CREATE GLOBAL TEMPORARY TABLE MYCONNGTT (
    ID INTEGER NOT NULL PRIMARY KEY,
    TXT VARCHAR(32),
    TS TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
ON COMMIT PRESERVE ROWS;
```

8. Создание глобальной временной таблицы уровня транзакции ссылающейся внешним ключом на глобальную временную таблицу уровня соединения.

```
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);
```

См. Также: ALTER TABLE, DROP TABLE, CREATE DOMAIN

ALTER TABLE

Изменение структуры таблицы.

Доступно: DSQL, ESQL

Синтаксис:

```
ALTER TABLE tablename
<operation> [, <operation>];

<operation> ::=
{
  ADD <col_def>
| ADD <tconstraint>
| DROP colname
| DROP CONSTRAINT constr_name
| ALTER [COLUMN] colname
  {
    TO new_colname
  | TYPE <datatype>
  | POSITION new_col_position
  | SET DEFAULT {literal | NULL | <context_var>}
  | DROP DEFAULT
  }
| ALTER [COLUMN] gencolname [TYPE <datatype>]
  { GENERATED ALWAYS AS | COMPUTED [BY]} (<col_expr>)
}

<col_def> ::= colname
{
  { <datatype> | domainname }
  [DEFAULT {literal | NULL | <context_var>}]
  [NOT NULL]
  [<col_constraint>]
  [COLLATE collation]
} | <col_exp_def>

<col_exp_def> ::= [<datatype>]
{COMPUTED [BY] | GENERATED ALWAYS AS} (<col_expr>)

<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]
| {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING]
```

Руководство по языку SQL СУБД Firebird

```
[(size)] [<array_dim>]
| BLOB [SUB_TYPE {subtype_num | subtype_name}]
  [SEGMENT SIZE seglen] [CHARACTER SET charset]
| BLOB [(seglen [, subtype_num])]
```

<array_dim> ::= [x:y [,x:y ...]]

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

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

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

<search_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 <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>)

Руководство по языку SQL СУБД Firebird

```

| [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 [[<ind> [, <ind> ...]]]
| literal
| <context_var>
| <expression>
| NULL
| NEXT VALUE FOR gename
| GEN_ID(gename, <val>)
| CAST(<val> AS <datatype>)
| (<select_one>)
| func(<val> [, <val> ...])
}

```

Таблица 5.8. Параметры оператора ALTER TABLE

Аргумент	Описание
tablename	Имя таблицы.
operation	Одна из допустимых операций по изменению структуры таблицы.
col_def	Определение столбца.
colname	Имя столбца таблицы, может содержать до 31 символа.
new_colname	Новое имя столбца таблицы, может содержать до 31 символа.
gencolname	Имя вычисляемого столбца таблицы.
new_col_position	Новая позиция столбца в таблице. Целое число в диапазоне от 1 до количества столбцов таблицы.
datatype	Тип данных SQL.
col_expr	Выражение для вычисляемого столбца.
domain_name	Имя домена.
col_constraint	Ограничение столбца.
tconstraint	Ограничение таблицы.
constr_name	Имя ограничения, может содержать до 31 символа.
other_table	Имя таблицы, на которую ссылается внешний ключ.
other_col	Столбец таблицы, на которую ссылается внешний ключ.
literal	Литерал.

Руководство по языку SQL СУБД Firebird

Аргумент	Описание
context_var	Любая контекстная переменная, тип которой совместим с типом данных столбца.
search_condition	Условие проверки ограничения.
collation	Порядок сортировки.
array_dim	Размерность массива.
x	Начальный номер элемента в массиве, положительное целое число.
y	Последний номер элемента в массиве, положительное целое число.
precision	Точность. От 1 до 18.
scale	Масштаб. От 0 до 18, должно быть меньше или равно precision.
size	Максимальный размер строки в символах.
charset	Набор символов.
subtype_num	Номер подтипа BLOB.
subtype_name	Мнемоника подтипа BLOB.
seglen	Размер сегмента, не может превышать 65535.
select_one	Оператор SELECT выбирающий один столбец и возвращающий только одну строку.
select_list	Оператор SELECT выбирающий один столбец и возвращающий ноль и более строк.
select_expr	Оператор SELECT выбирающий несколько столбцов и возвращающий ноль и более строк.
expression	Выражение.
genname	Имя последовательности (генератора).
func	Встроенная или UDF функция.

Описание:

Оператор ALTER TABLE изменяет структуру существующей таблицы. Одиночный оператор ALTER TABLE позволяет производить множество операций добавления/удаления столбцов и ограничений, а также модификаций столбцов. Список операций выполняемых при модификации таблицы разделяется запятой. Некоторые изменения структуры таблицы увеличивают счётчик форматов, закреплённый за каждой таблицей. Количество форматов для каждой таблицы ограничено значением 255. После того, как счётчик форматов достигнет этого значения, вы не сможете больше менять структуру таблицы. Для сброса счётчика форматов необходимо сделать резервное копирование и восстановление базы данных (утилитой gbak).

Предложение ADD позволяет добавить новый столбец или новое ограничение таблицы. Синтаксис определения столбца и синтаксис описания ограничения таблицы полностью совпадают с синтаксисом, описанным в операторе . При каждом добавлении нового столбца номер формата увеличивается на единицу. Добавление нового ограничения таблицы не влечёт за собой увеличение номера формата.

Замечание:

Будьте аккуратны при добавлении нового столбца с установленным ограничением NOT NULL. Это может привести к нарушению логической целостности данных. При добавлении такого столбца рекомендуется либо указывать ему значение по умолчанию, либо обновлять строки таблицы для присвоения ему значений отличных от NULL.

Предложение DROP удаляет указанный столбец таблицы. Столбец таблицы не может быть удален, если от него существуют зависимости. Другими словами для успешного удаления столбца на него должны отсутствовать ссылки. Ссылки на столбец могут содержаться:

- в ограничениях столбцов или таблицы;
- в индексах;
- в хранимых процедурах и триггерах;
- в представлениях.

При каждом удалении столбца номер формата увеличивается на единицу.

Предложение DROP CONSTRAINT удаляет указанное ограничение столбца или таблицы. Ограничение первичного ключа или уникального ключа не могут быть удалены, если они используются в ограничении внешнего ключа другой таблицы. В этом случае, необходимо удалить ограничение FOREIGN KEY до удаления PRIMARY KEY или UNIQUE ключа, на которые оно ссылается. Удаление ограничения столбца или ограничения таблицы не влечёт за собой увеличение номера формата.

Предложение ALTER [COLUMN] позволяет изменить следующие характеристики существующих столбцов:

- изменение имени (не изменяет номер формата);
- изменение типа данных (увеличивает номер формата на единицу);
- изменение позиции столбца в списке столбцов таблицы (не изменяет номер формата);
- удаление значения по умолчанию столбца (не изменяет номер формата);
- добавление значения по умолчанию столбца (не изменяет номер формата);
- изменение типа и выражения для вычисляемого столбца (не изменяет номер формата).

Ключевое слово TO переименовывает существующий столбец. Новое имя столбца не должно присутствовать в таблице. Невозможно изменение имени столбца, если этот столбец включен в какое-либо ограничение – первичный или уникальный ключ, внешний ключ, ограничение столбца или проверочное ограничение таблицы CHECK. Имя столбца также нельзя изменить, если этот

Руководство по языку SQL СУБД Firebird

столбец таблицы используется в каком-либо триггере, в хранимой процедуре или представлении.

Ключевое слово TYPE изменяет тип существующего столбца на другой допустимый тип. Не допустимы любые изменения типа, которые могут привести к потере данных. Например, количество символов в новом типе для столбца не может быть меньше, чем было установлено ранее. Если столбец был объявлен как массив, то изменить ни его тип, ни размерность нельзя. Нельзя изменить тип данных у столбца, который принимает участие в связке внешний ключ / первичный (уникальный) ключ.

Ключевое слово POSITION изменяет позицию существующего столбца. Позиции столбцов нумеруются с единицы. Если будет задан номер позиции меньше 1, то будет выдано соответствующее сообщение об ошибке. Если будет задан номер позиции, превышающий количество столбцов в таблице, то изменения не будут выполнены, но ни ошибки, ни предупреждения не последуют.

Предложение DROP DEFAULT удаляет значение по умолчанию для столбца таблицы. Если столбец основан на домене со значением по умолчанию – доменное значение перекроет это удаление. Если удаление значения по умолчанию производится над столбцом, у которого нет значения по умолчанию, или чье значение по умолчанию основано на домене, то это приведёт к ошибке выполнения данного оператора.

Предложение SET DEFAULT устанавливает значение по умолчанию для столбца таблицы. Если столбец уже имел значение по умолчанию, то оно будет заменено новым. Значение по умолчанию для столбца всегда перекрывает доменное значение по умолчанию.

Предложения COMPUTED [BY] или GENERATED ALWAYS AS позволяют изменить тип и выражение вычисляемого столбца. Невозможно изменить обычный столбец на вычисляемый и наоборот.

Замечание:

На данный момент не существует возможности установить и снять ограничение NOT NULL для столбца таблицы. Кроме того, не существует способа изменить сортировку по умолчанию.

Изменить структуру таблицы могут:

- владелец таблицы;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация

Руководство по языку SQL СУБД Firebird

(trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

1. Добавление столбца CAPITAL в таблицу COUNTRY.

```
ALTER TABLE COUNTRY  
ADD CAPITAL VARCHAR (25) ;
```

2. Добавление столбца CAPITAL с ограничением уникальности и удаление столбца CURRENCY.

```
ALTER TABLE COUNTRY  
ADD CAPITAL VARCHAR(25) NOT NULL UNIQUE,  
DROP CURRENCY;
```

3. Добавление проверочного ограничения CHK_SALARY и внешнего ключа в таблицу JOB.

```
ALTER TABLE JOB  
ADD CONSTRAINT CHK_SALARY CHECK (MIN_SALARY < MAX_SALARY) ,  
ADD FOREIGN KEY (JOB_COUNTRY) REFERENCES COUNTRY  
(COUNTRY) ;
```

4. Установка для поля MODEL значения по умолчанию, изменение типа столбца ITEMID и переименование столбца MODELNAME.

```
ALTER TABLE STOCK  
ALTER COLUMN MODEL SET DEFAULT 1 ,  
ALTER COLUMN ITEMID TYPE BIGINT,  
ALTER COLUMN MODELNAME TO NAME;
```

5. Изменение вычисляемых столбцов NEW_SALARY и 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);
```

См. также: [CREATE TABLE](#), [DROP TABLE](#), [CREATE DOMAIN](#)

DROP TABLE

Удаление существующей таблицы.

Доступно: DSQL, ESQL

Синтаксис:

```
DROP TABLE tablename;
```

Таблица 5.9. Параметры оператора DROP TABLE

Аргумент	Описание
tablename	Имя таблицы.

Описание:

Оператор DROP TABLE удаляет существующую таблицу. Если таблица имеет зависимости, то удаление не будет произведено. При удалении таблицы будут также удалены все триггеры на её события и индексы, построенные для её полей.

Удалить таблицу могут:

- владелец таблицы;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

Удаление таблицы COUNTRY.

```
DROP TABLE COUNTRY;
```

См. Также: [CREATE TABLE](#), [ALTER TABLE](#), [RECREATE TABLE](#)

RECREATE TABLE

Создание новой таблицы или пересоздание существующей.

Доступно: DSQL

Руководство по языку SQL СУБД Firebird

Синтаксис:

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

Полное описание определений столбцов и ограничений таблицы смотрите в разделе .

Описание:

Создаёт или пересоздаёт таблицу. Если таблица с таким именем уже существует, то оператор RECREATE TABLE попытается удалить её и создать новую. Оператор RECREATE TABLE не выполнится, если существующая таблица имеет зависимости.

Примеры:

Создание или пересоздание таблицы COUNTRY.

```
RECREATE TABLE COUNTRY (
  COUNTRY COUNTRYNAME NOT NULL PRIMARY KEY,
  CURRENCY VARCHAR(10) NOT NULL);
```

См. также: [CREATE TABLE](#), [DROP TABLE](#)

INDEX

Индекс (index) – объект базы данных, предназначенный для ускорения выборки данных из таблицы и/или для ускорения упорядочения результатов выборки данных из таблицы. Кроме того, индексы используются для обеспечения ограничений целостности – PRIMARY KEY, FOREIGN KEY, UNIQUE.

В данном разделе описываются вопросы создания индексов, перевода их в активное/неактивное состояние, удаление индексов и сбор статистики (пересчёт селективности) для индексов.

CREATE INDEX

Создание индекса для таблицы.

Доступно: DSQL, ESQL.

Синтаксис:

Руководство по языку SQL СУБД Firebird

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]] INDEX indexname  
ON tablename {(col [, col ...]) | COMPUTED BY (<expression>)};
```

Таблица 5.10. Параметры оператора CREATE INDEX

Аргумент	Описание
<i>indexname</i>	Имя индекса. Может содержать до 31 символа.
<i>tablename</i>	Имя таблицы, для которой строится индекс
<i>col</i>	Столбец таблицы. В качестве столбцов не могут быть использованы поля типа BLOB, ARRAY и вычисляемые поля.
<i>expression</i>	Выражение, содержащее столбцы таблицы

Описание:

Оператор CREATE INDEX создаёт индекс для таблицы, который может быть использован для ускорения поиска, сортировки и/или группирования. Кроме того, индекс может быть использован при определении ограничений, таких как первичный ключ, внешний ключ или ограничения уникальности. Индекс может быть построен на столбцах любого типа кроме BLOB и массивов. Имя индекса должно быть уникальным среди всех имён индексов.

Примечание:

При добавлении ограничений первичного ключа, внешнего ключа или ограничения уникальности будет неявно создан одноимённый индекс. Так, например, при выполнении следующего оператора будет неявно создан индекс PK_COUNTRY.

```
ALTER TABLE COUNTRY ADD CONSTRAINT PK_COUNTRY  
PRIMARY KEY (ID);
```

Если при создании индекса указано ключевое слово UNIQUE, то индекс называется уникальным. Уникальные индексы не могут содержать дубликаты значений ключей, но могут содержать дубликаты значения NULL в соответствии со стандартом SQL-99 (в том числе и в многосегментном индексе).

Ключевое слово ASCENDING (сокращённо ASC), обозначает, что ключи индекса расположены по возрастанию значений. Такое расположение ключей используется по умолчанию.

Ключевое слово DESCENDING (сокращённо DESC), обозначает, что ключи индекса расположены по убыванию значений.

Для каждой таблицы максимально возможное количество индексов ограничено и зависит от размера страницы и количества столбцов в индексе.

Таблица 5.11. Число индексов и количество столбцов

Размер страницы	Число индексов в зависимости от кол-ва столбцов в индексе		
	1	2	3
1024	50	35	27
2048	101	72	56
4096	203	145	113
8192	408	291	227
16384	818	584	454

Максимальная длина ключа индекса ограничена $\frac{1}{4}$ размера страницы.

Максимальная длина индексируемой строки на 9 байтов меньше, чем максимальная длина ключа. Максимальная длина индексируемой строки зависит от размера страницы и набора символов.

Таблица 5.12. Длина индексируемой строки и набор символов

Размер страницы	Максимальная длина индексируемой строки для набора символов			
	1 байт/символ	2 байта/символ	3 байта/символ	4 байта/символ
1024	247	123	82	61
2048	503	251	167	125
4096	1015	507	338	253
8192	2039	1019	679	509
16384	4087	2043	1362	1021

Вычисляемый индекс. При создании индекса вместо одного или нескольких столбцов вы также можете указать одно выражение, используя предложение COMPUTED BY. Вычисляемые индексы используются в запросах, в которых условие в предложениях WHERE, ORDER BY или GROUP BY в точности совпадает с выражением в определении индекса. Выражение в вычисляемом индексе может использовать несколько столбцов таблицы.

Создать новый индекс могут:

- владелец таблицы;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);

Руководство по языку SQL СУБД Firebird

- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

1. Создание индекса для столбца UPDATER_ID таблицы SALARY_HISTORY

```
CREATE INDEX IDX_UPDATER ON SALARY_HISTORY (UPDATER_ID);
```

2. Создание индекса с сортировкой ключей по убыванию для столбца CHANGE_DATE таблицы SALARY_HISTORY

```
CREATE DESCENDING INDEX IDX_CHANGE  
ON SALARY_HISTORY (CHANGE_DATE);
```

3. Создание многосегментного индекса для столбцов ORDER_STATUS, PAID таблицы SALES

```
CREATE INDEX IDX_SALESTAT ON SALES (ORDER_STATUS, PAID);
```

4. Создание индекса, не допускающего дубликаты значений, для столбца NAME таблицы COUNTRY

```
CREATE UNIQUE INDEX UNQ_COUNTRY_NAME ON COUNTRY (NAME);
```

5. Создание вычисляемого индекса для таблицы PERSONS

```
CREATE INDEX IDX_NAME_UPPER ON PERSONS  
COMPUTED BY (UPPER (NAME));
```

Такой индекс может быть использован для регистронезависимого поиска

```
SELECT *  
FROM PERSONS  
WHERE UPPER (NAME) STARTING WITH UPPER ('Iv');
```

См. также: [ALTER INDEX](#), [DROP INDEX](#)

ALTER INDEX

Перевод индекса в активное/неактивное состояние.

Доступно: DSQL, ESQL.

Синтаксис:

```
ALTER INDEX indexname {ACTIVE | INACTIVE};
```

Таблица 5.13. Параметры оператора ALTER INDEX

Аргумент	Описание
<code>indexname</code>	Имя индекса.
<code>ACTIVE</code>	Перевод неактивного индекса в активное состояние.
<code>INACTIVE</code>	Перевод активного индекса в неактивное состояние.

Описание:

Оператор ALTER INDEX переводит индекс в активное/неактивное состояние. Возможность изменения структуры и порядка сортировки ключей этот оператор не предусматривает.

Перевод индекса в неактивное состояние по своему действию похоже на команду DROP INDEX за исключением того, что определение индекса сохраняется в базе данных. При переводе индекса из неактивного состояния в активное система заново создаёт индекс.

Перевод индекса в неактивное состояние может быть полезен при массовой вставке, модификации или удалении записей из таблицы, для которой этот индекс построен.

Замечание:

Оператор ALTER INDEX ACTIVE перестраивает индекс при любом состоянии индекса. Таким образом, эту команду можно использовать для перестройки индексов, автоматически созданных для ограничений PRIMARY KEY, FOREIGN KEY, UNIQUE, для которых выполнение оператора ALTER INDEX INACTIVE невозможно.

Перевод индекса в активное/неактивное состояние могут:

- владелец таблицы, для которой построен индекс;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

1. Перевод индекса IDX_UPDATER в неактивное состояние

Руководство по языку SQL СУБД Firebird

```
ALTER INDEX IDX_UPDATER INACTIVE;
```

2. Возврат индекса IDX_UPDATER в активное состояние

```
ALTER INDEX IDX_UPDATER ACTIVE;
```

См. также: [CREATE INDEX](#), [DROP INDEX](#), [SET STATISTICS](#)

DROP INDEX

Удаление индекса из базы данных.

Доступно: DSQL, ESQL.

Синтаксис:

```
DROP INDEX indexname;
```

Таблица 5.14. Параметры оператора DROP INDEX

Аргумент	Описание
indexname	Имя индекса.

Описание:

Оператор DROP INDEX удаляет существующий индекс из базы данных. При наличии зависимостей для существующего индекса (если он используется в ограничении) удаление не будет выполнено.

Индекс могут удалить:

- владелец таблицы, для которой построен индекс;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

Удаление индекса IDX_UPDATER

```
DROP INDEX IDX_UPDATER;
```

См. также: [CREATE INDEX](#), [ALTER INDEX](#)

SET STATISTICS

Пересчёт селективности индекса.

Доступно: DSQL, ESQL.

Синтаксис:

```
SET STATISTICS INDEX indexname;
```

Таблица 5.15. Параметры оператора SET STATISTICS

Аргумент	Описание
indexname	Имя индекса

Описание:

Оператор SET STATISTICS пересчитывает значение селективности для указанного индекса.

Селективность (избирательность) индекса – это оценочное количество строк, которые могут быть выбраны при поиске по каждому значению индекса. Уникальный индекс имеет максимальную селективность, поскольку при его использовании невозможно выбрать более одной строки для каждого значения ключа индекса. Актуальность селективности индекса важна для выбора наиболее оптимального плана выполнения запросов оптимизатором.

Пересчёт селективности индекса может потребоваться после массовой вставки, модификации или удаления большого количества записей из таблицы, поскольку она становится неактуальной.

Примечание:

Отметим, что в Firebird статистика индексов автоматически не пересчитывается ни после массовых изменений данных, ни при каких либо других условиях. При создании (CREATE) или его активации (ALTER INDEX ACTIVE) статистика индекса полностью соответствует его содержанию.

Пересчитать селективность индекса могут:

- владелец таблицы, для которой построен индекс;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная

Руководство по языку SQL СУБД Firebird

авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

Пересчёт селективности индекса IDX_UPDATER

```
SET STATISTICS INDEX IDX_UPDATER;
```

См. также: [CREATE INDEX](#), [ALTER INDEX](#)

VIEW

Представление (view) – виртуальная таблица, которая по своей сути является именованным запросом SELECT выборки данных произвольной сложности. Выборка данных может осуществляться из одной и более таблиц, других представлений, а также селективных хранимых процедур. В отличие от обычных таблиц реляционных баз данных, представление не является самостоятельным набором данных, хранящимся в базе данных. Результат в виде набора данных динамически создается при обращении к представлению.

CREATE VIEW

Создание нового представления.

Доступно: DSQL

Синтаксис:

```
CREATE VIEW viewname [<full_column_list>]  
AS <select_statement>  
[WITH CHECK OPTION];
```

<full_column_list> ::= (colname [, colname ...])

Таблица 5.16. Параметры оператора CREATE VIEW

Аргумент	Описание
viewname	Имя представления. Может содержать до 31 символа.
select_statement	Оператор SELECT.
full_column_list	Список столбцов представления.
colname	Имя столбца представления. Дубликаты имён столбцов не позволяют.

Описание:

Руководство по языку SQL СУБД Firebird

Оператор `CREATE VIEW` создаёт новое представление. Имя представления должно быть уникальным среди имён всех представлений, таблиц и хранимых процедур базы данных.

После имени создаваемого представления может идти список имён столбцов, получаемых в результате обращения к представлению. Имена в списке могут быть никак не связаны с именами столбцов базовых таблиц. При этом их количество должно точно соответствовать количеству столбцов в списке выбора главного оператора `SELECT` представления. Если список столбцов представления отсутствует, то будут использоваться имена столбцов базовых таблиц или псевдонимов (алиасов) полей оператора `SELECT`.

Примечания:

- Если также существует полный список столбцов, то задание псевдонимов не имеет смысла, поскольку они будут переопределены именами из списка столбцов;
- Полный список столбцов, используемых в представлениях, обязателен для оператора `SELECT`, содержащего выражения на основе столбцов или идентичные имена столбцов. Теперь вы можете не указывать полный список столбцов при условии, что вы укажете их в операторе `SELECT`.

Представление может быть изменяемым и неизменяемым. Если представление изменяемое, то данные, полученные при обращении к такому представлению, можно изменить при помощи DML операторов `INSERT`, `UPDATE`, `DELETE`, `UPDATE OR INSERT`, `MERGE`. Изменения, выполненные над представлением, отражаются в таблице, из которой происходит выборка данных. Неизменяемое представление можно сделать изменяемым при помощи вспомогательных триггеров. Если на обновляемом представлении будет построен один или несколько триггеров, то изменения не будут автоматически попадать в таблицу – либо это делает триггер, либо запись не происходит.

Для того, чтобы представление было изменяемым, необходимо выполнение следующих условий:

- оператор выборки `SELECT` обращается только к одной таблице или одному изменяемому представлению;
- оператор выборки `SELECT` не должен обращаться к хранимым процедурам;
- все столбцы базовой таблицы или изменяемого представления, которые не присутствуют в данном представлении, допускают значение `NULL`;
- оператор выборки `SELECT` не содержит полей определённых через подзапросы или другие выражения;
- оператор выборки `SELECT` не содержит полей определённых через агрегатные функции `MIN`, `MAX`, `AVG`, `SUM`, `COUNT`, `LIST`;

Руководство по языку SQL СУБД Firebird

- оператор выборки SELECT не содержит предложений ORDER BY, GROUP BY, HAVING;
- оператор выборки SELECT не содержит ключевого слова DISTINCT и ограничений количества строк ROWS, FIRST, SKIP.

Необязательное предложение WITH CHECK OPTIONS задаёт для изменяемого представления требования проверки вновь введённых или модифицируемых данных условию, указанному в предложении WHERE оператора выборки SELECT. При попытке вставки новой записи или модификации записи проверяется, выполняется ли для этой записи условие в предложении WHERE, если условие не выполняется, то вставка/модификация не выполняется и будет выдано соответствующее диагностическое сообщение.

Предложение WITH CHECK OPTION может задаваться в операторе создания представления только в том случае, если в главном операторе SELECT представления указано предложение WHERE. Иначе будет выдано сообщение об ошибке.

Создать новое представление может любой пользователь, подключённый к базе данных, если он имеет привилегии на чтение (SELECT) данных из базовых таблиц и представлений, и привилегии на выполнение (EXECUTE) используемых селективных хранимых процедур. Если же представление изменяемое, то пользователю необходимы привилегии ALL к базовым таблицам. Пользователь, создавший представление, становится его владельцем.

Примеры:

1. Создание представления возвращающего столбцы JOB_CODE и JOB_TITLE только для тех работ, где MAX_SALARY меньше \$15000.

```
CREATE VIEW ENTRY_LEVEL_JOBS AS  
SELECT JOB_CODE, JOB_TITLE  
FROM JOB  
WHERE MAX_SALARY < 15000;
```

2. Создание представления возвращающего столбцы JOB_CODE и JOB_TITLE только для тех работ, где MAX_SALARY меньше \$15000. При вставке новой записи или изменении существующей будет осуществляться проверка условия MAX_SALARY < 15000, если условие не выполняется, то вставка/изменение будет отвергнуто.

```
CREATE VIEW ENTRY_LEVEL_JOBS AS  
SELECT JOB_CODE, JOB_TITLE  
FROM JOB  
WHERE MAX_SALARY < 15000  
WITH CHECK OPTIONS;
```

3. Создание представления с использованием списка столбцов.

Руководство по языку SQL СУБД Firebird

```
CREATE VIEW PRICE_WITH_MARKUP (  
    CODE_PRICE,  
    COST,  
    COST_WITH_MARKUP  
) AS  
SELECT  
    CODE_PRICE,  
    COST,  
    COST * 1.1  
FROM PRICE;
```

4. Создание представления с использованием псевдонимов для полей оператора SELECT (тот же результат что и в 3).

```
CREATE VIEW PRICE_WITH_MARKUP AS  
SELECT  
    CODE_PRICE,  
    COST,  
    COST * 1.1 AS COST_WITH_MARKUP  
FROM PRICE;
```

5. Создание необновляемого представления на основе двух таблиц и хранимой процедуры.

```
CREATE VIEW GOODS_PRICE  
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;
```

См. Также: ALTER VIEW, CREATE OR ALTER VIEW, RECREATE VIEW, DROP VIEW

ALTER VIEW

Изменение существующего представления.

Доступно: DSQL

Синтаксис:

```
ALTER VIEW viewname [<full_column_list>]  
AS <select_statement>
```

Руководство по языку SQL СУБД Firebird

[WITH CHECK OPTION];

<full_column_list> ::= (colname [, colname ...])

Таблица 5.17. Параметры оператора ALTER VIEW

Аргумент	Описание
viewname	Имя представления. Может содержать до 31 символа.
select_statement	Оператор SELECT.
full_column_list	Список столбцов представления.
colname	Имя столбца.

Описание:

Оператор ALTER VIEW изменяет определение существующего представления, существующие права на представления и зависимости при этом сохраняются. Синтаксис оператора ALTER VIEW полностью аналогичен синтаксису оператора CREATE VIEW.

Предупреждение:

Будьте осторожны при изменении количества столбцов представления. Существующий код приложения может стать неработоспособным. Кроме того, PSQL модули, использующие изменённое представление, могут стать некорректными. Информация о том, как это обнаружить, находится в приложении .

Изменить представление могут:

- владелец представления;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

Изменение представления PRICE_WITH_MARKUP.

```
ALTER VIEW PRICE_WITH_MARKUP (  
    CODE_PRICE,  
    COST,  
    COST_WITH_MARKUP  
) AS
```

Руководство по языку SQL СУБД Firebird

```
SELECT
  CODE_PRICE,
  COST,
  COST * 1.15
FROM PRICE;
```

См. Также: [CREATE VIEW](#), [CREATE OR ALTER VIEW](#), [RECREATE VIEW](#)

CREATE OR ALTER VIEW

Создание нового или изменение существующего представления.

Доступно: DSQL

Синтаксис:

```
CREATE OR ALTER VIEW viewname [<full_column_list>]
AS <select_statement>
[WITH CHECK OPTION];
```

<full_column_list> ::= (colname [, colname ...])

Таблица 5.18. Параметры оператора CREATE OR ALTER VIEW

Аргумент	Описание
viewname	Имя представления. Может содержать до 31 символа.
select_statement	Оператор SELECT.
full_column_list	Список столбцов представления.
colname	Имя столбца.

Описание:

Оператор CREATE OR ALTER VIEW создаёт представление, если оно не существует. В противном случае он изменит представление с сохранением существующих зависимостей.

Примеры:

Создание нового или изменение существующего представления PRICE_WITH_MARKUP.

```
CREATE OR ALTER VIEW PRICE_WITH_MARKUP (
  CODE_PRICE,
  COST,
  COST_WITH_MARKUP
) AS
SELECT
```

Руководство по языку SQL СУБД Firebird

```
CODE_PRICE,  
COST,  
COST * 1.15  
FROM PRICE;
```

См. Также: [CREATE VIEW](#), [ALTER VIEW](#), [RECREATE VIEW](#)

DROP VIEW

Удаление существующего представления.

Доступно: DSQL

Синтаксис:

```
DROP VIEW viewname;
```

Таблица 5.19. Параметры оператора DROP VIEW

Аргумент	Описание
viewname	Имя представления.

Описание:

Оператор DROP VIEW удаляет существующее представление. Если представление имеет зависимости, то удаление не будет произведено.

Удалить представление могут:

- владелец представления;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

Удаление представления PRICE_WITH_MARKUP.

```
DROP VIEW PRICE_WITH_MARKUP;
```

См. Также: [CREATE VIEW](#), [RECREATE VIEW](#)

RECREATE VIEW

Создание нового или пересоздание существующего представления.

Доступно: DSQL

Синтаксис:

```
RECREATE VIEW viewname [<full_column_list>]
AS <select_statement>
[WITH CHECK OPTION];
```

<full_column_list> ::= (colname [, colname ...])

Таблица 5.20. Параметры оператора RECREATE VIEW

Аргумент	Описание
viewname	Имя представления. Может содержать до 31 символа.
select_statement	Оператор SELECT.
full_column_list	Список столбцов представления.
colname	Имя столбца.

Описание:

Создаёт или пересоздаёт представление. Если представление с таким именем уже существует, то оператор RECREATE VIEW попытается удалить его и создать новое. Оператор RECREATE VIEW не выполнится, если существующее представление имеет зависимости.

Примеры:

Создание нового или пересоздание существующего представления PRICE_WITH_MARKUP.

```
RECREATE VIEW PRICE_WITH_MARKUP (
    CODE_PRICE,
    COST,
    COST_WITH_MARKUP
) AS
SELECT
    CODE_PRICE,
    COST,
    COST * 1.15
FROM PRICE;
```

См. также: [CREATE VIEW](#), [DROP VIEW](#), [CREATE OR ALTER VIEW](#)

TRIGGER

Триггер (trigger) – это хранимая процедура особого типа, которая не вызывается непосредственно, а исполнение которой обусловлено наступлением одного из событий, относящегося к одной конкретной таблице (представлению), или наступлению одного из событий базы данных.

CREATE TRIGGER

Создание нового триггера.

Доступно: DSQL, ESQL

Синтаксис:

```
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
```

Руководство по языку SQL СУБД Firebird

```
| TRANSACTION ROLLBACK
}
```

```
<declarations> ::= {<declare_var> | <declare_cursor>};
  [{<declare_var> | <declare_cursor>}; ...]
```

Таблица 5.21. Параметры оператора CREATE TRIGGER

Аргумент	Описание
trigname	Имя триггера. Может содержать до 31 символа.
relation_trigger_legacy	Объявление табличного триггера (унаследованное).
relation_trigger_sql2003	Объявление табличного триггера согласно стандарту SQL-2003.
database_trigger	Объявление триггера базы данных.
tablename	Имя таблицы.
viewname	Имя представления.
mutation_list	Список событий таблицы.
number	Порядок срабатывания триггера. От 0 до 32767.
db_event	Событие соединения или транзакции.
declarations	Секция объявления локальных переменных и именованных курсоров.
declare_var	Объявление локальной переменной.
declare_cursor	Объявление именованного курсора.
PSQL_statments	Операторы языка PSQL.

Описание:

Оператор CREATE TRIGGER создаёт новый триггер. Имя триггера должно быть уникальным среди имён всех триггеров.

Триггер может быть создан как для события (или событий) таблицы (представления), так и для события базы данных. Табличные триггеры выполняются на уровне строки (записи) каждый раз, когда изменяется образ строки.

Триггер может быть в одном из двух состояний активном (ACTIVE) или неактивном (INACTIVE). Запускаются только активные триггеры. По умолчанию триггеры создаются в активном состоянии.

Объявление табличного триггера существует в двух вариантах: «Legacy» и согласно стандарту SQL-2003. В настоящее время рекомендуется пользоваться последним.

Если триггер создаётся для таблицы (представления), то для него необходимо указать фазу события и событие (или события).

Ключевое слово BEFORE означает, что триггер вызывается до наступления соответствующего события (событий, если их указано несколько),

Руководство по языку SQL СУБД Firebird

AFTER – после наступления события (событий).

Для триггера может быть указано одно из событий таблицы (представления) – INSERT (добавление), UPDATE (изменение), DELETE (удаление) – или несколько событий, разделённых ключевым словом OR, при которых вызывается триггер. При создании триггера каждый тип события на его срабатывание (INSERT, UPDATE или DELETE) не должен упоминаться более одного раза.

Ключевое слово POSITION позволяет задать порядок, в котором будут выполняться триггеры с одинаковой фазой и событием (или группы событий). По умолчанию позиция равна 0. Если позиции для триггеров не заданы или несколько триггеров имеют одно и то же значение позиции, то такие триггеры будут выполняться в алфавитном порядке их имен.

В необязательной секции *declarations* описаны локальные переменные триггера и именованные курсоры. Подробнее в см. в разделе [DECLARE VARIABLE](#).

После секции объявления локальных переменных и именованных курсоров в теле триггера, следует блок PSQL операторов, заключённых в операторные скобки BEGIN и END.

Триггеры для событий таблицы (представления) могут создать:

- владелец таблицы (представления);
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

1. Создание триггера в «Legacy» стиле, срабатывающего до события вставки новой записи в таблицу CUSTOMER.

```
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
```

2. Создание триггера, срабатывающего до события вставки новой записи в таблицу CUSTOMER, согласно стандарту SQL-2003.

Руководство по языку SQL СУБД Firebird

```
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
```

3. Создание триггера, срабатывающего после вставки, обновления или удаления записи.

```
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
```

Триггер может быть создан для одного из событий базы данных:

- CONNECT (соединение с базой данных);
- DISCONNECT (отсоединение от базы данных);
- TRANSACTION START (старт транзакции);
- TRANSACTION COMMIT (подтверждение транзакции);
- TRANSACTION ROLLBACK (откат транзакции).

Указать для триггера несколько событий базы данных невозможно.

Обработка исключений в триггерах для событий базы данных имеет следующие особенности:

- Триггера на события CONNECT и DISCONNECT выполняются в специально созданной для этого транзакции. Если при обработке триггера не было вызвано исключение, то транзакция подтверждается. Не перехваченные исключения откатят транзакцию и:
 - в случае триггера на событие CONNECT соединение разрывается, а исключения возвращается клиенту;
 - для триггера на событие DISCONNECT соединение разрывается, как это и предусмотрено, но исключения не возвращается клиенту.

Руководство по языку SQL СУБД Firebird

- Триггера на событие TRANSACTION срабатывают при старте транзакции, её подтверждении или отмене. Не перехваченные исключения обрабатываются в зависимости от типа события TRANSACTION:
 - для события START исключение возвращается клиенту, а транзакция отменяется;
 - для события COMMIT исключение возвращается клиенту, действия, выполненные триггером, и транзакция отменяются;
 - для события ROLLBACK исключение не возвращается клиенту, а транзакция, как и предусмотрено, отменяется.
- В случае двухфазных транзакций триггера на событие TRANSACTION START срабатывают в фазе подготовки (prepare), а не в фазе commit .

Из вышеизложенного следует, что нет прямого способа узнать, какой триггер (DISCONNECT или ROLLBACK) вызвал исключение. Также ясно, что вы не сможете подключиться к базе данных в случае исключения в триггере на событие CONNECT, а также отменяется старт транзакции при исключении в триггере на событие TRANSACTION START. В обоих случаях база данных эффективно блокируется, в то время как вы собирались работать с ней.

Примечание:

Триггеры для событий базы данных DISCONNECT и ROLLBACK не будут вызваны при отключении клиентов через таблицы мониторинга (DELETE FROM MON\$ATTACHMENTS).

Примечание:

В некоторые утилиты командной строки Firebird были добавлены новые ключи для отключения триггеров на базу данных:

```
gbak -nodbtriggers
isql -nodbtriggers
nbackup -T
```

Эти ключи могут использоваться только SYSDBA или владельцем базы данных.

Триггеры для событий базы данных могут создать:

- владелец базы данных;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Руководство по языку SQL СУБД Firebird

Примеры:

1. Создание триггера на событие подключения к базе данных, который ведёт протокол входа пользователей в систему. Триггер создан в неактивном состоянии.

```
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. Создание триггера на событие подключения к базе данных, который запрещает вход всем пользователям, за исключением SYSDBA, вне рабочее время.

```
CREATE EXCEPTION E_INCORRECT_WORKTIME 'Рабочий день еще не
начался';

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
```

См. Также: ALTER TRIGGER, CREATE OR ALTER TRIGGER, RECREATE TRIGGER, DROP TRIGGER

ALTER TRIGGER

Изменение существующего триггера.

Доступно: DSQL, ESQL

Синтаксис:

```
ALTER TRIGGER trigname
[ACTIVE | INACTIVE]
```

Руководство по языку SQL СУБД Firebird

```
[{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
}

<declarations> ::= {<declare_var> | <declare_cursor>};
[{{<declare_var> | <declare_cursor>}; ...}]
```

Таблица 5.22. Параметры оператора ALTER TRIGGER

Аргумент	Описание
trigname	Имя триггера.
mutation_list	Список мутаций.
number	Порядок срабатывания триггера.
declarations	Секция объявления локальных переменных и именованных курсоров.
declare_var	Объявление локальной переменной.
declare_cursor	Объявление именованного курсора.
PSQL_statments	Операторы языка PSQL.

Описание:

Оператор ALTER TRIGGER позволяет изменять заголовок и/или тело триггера.

В операторе изменения триггера можно изменить состояние активности, фазу события и событие (события) таблицы (представления), позицию триггера и его тело. Триггеры на событие (события) таблицы (представления) не могут быть изменены в триггеры на событие базы данных и наоборот. Если какой-либо элемент не указан, то он остаётся без изменений.

Руководство по языку SQL СУБД Firebird

Ключевое слово BEFORE означает, что триггер вызывается до наступления соответствующего события (событий, если их указано несколько), AFTER – после наступления события (событий).

Для триггера может быть указано одно из событий таблицы (представления) – INSERT (добавление), UPDATE (изменение), DELETE (удаление) – или несколько событий, разделённых ключевым словом OR, при которых вызывается триггер. При создании триггера каждый тип события на его срабатывание (INSERT, UPDATE или DELETE) не должен упоминаться более одного раза.

Ключевое слово POSITION позволяет задать порядок, в котором будут выполняться триггеры с одинаковой фазой и событием (или группы событий). По умолчанию позиция равна 0. Если позиции для триггеров не заданы или несколько триггеров имеют одно и то же значение позиции, то такие триггеры будут выполняться в алфавитном порядке их имен.

Триггеры для событий таблицы (представления) могут быть изменены:

- владельцем таблицы (представления);
- пользователем SYSDBA;
- любым пользователем, подключенным с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователем операционной системы root (Linux);
- администратором Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Триггеры для событий базы данных могут быть изменены:

- владельцем базы данных;
- пользователем SYSDBA;
- любой пользователем, подключенным с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователем операционной системы root (Linux);
- администратором Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

1. Отключение (перевод в неактивное состояние) триггера set_cust_no.

```
ALTER TRIGGER set_cust_no INACTIVE;
```

2. Изменение позиции триггера set_cust_no.

```
ALTER TRIGGER set_cust_no POSITION 14;
```

Руководство по языку SQL СУБД Firebird

3. Перевод триггера TR_CUST_LOG в неактивное состояние и изменение списка событий.

```
ALTER TRIGGER TR_CUST_LOG
INACTIVE AFTER INSERT OR UPDATE;
```

4. Перевод триггера tr_log_connect в активное состояние, изменение его позиции и тела триггера.

```
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
```

См. Также: [CREATE TRIGGER](#), [CREATE OR ALTER TRIGGER](#)

CREATE OR ALTER TRIGGER

Создание нового или изменение существующего триггера.

Доступно: DSQL

Синтаксис:

```
CREATE OR ALTER TRIGGER triname
{ <relation_trigger_legacy>
| <relation_trigger_sql2003>
| <database_trigger> }
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END
```

Полное описание синтаксиса оператора см. в разделе [CREATE TRIGGER](#).

Руководство по языку SQL СУБД Firebird

Описание:

Оператор CREATE OR ALTER TRIGGER создаёт новый триггер, если он не существует, или изменяет и перекомпилирует его в противном случае, при этом существующие права и зависимости сохраняются.

Примеры:

Создание нового триггера, если он не существует или его изменение в противном случае.

```
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
```

См. Также: [CREATE TRIGGER](#), [ALTER TRIGGER](#), [RECREATE TRIGGER](#)

DROP TRIGGER

Удаление существующего триггера.

Доступно: DSQL, ESQL

Синтаксис:

```
DROP TRIGGER trigname;
```

Таблица 5.23. Параметры оператора DROP TRIGGER

Аргумент	Описание
trigname	Имя триггера.

Описание:

Оператор DROP TRIGGER удаляет существующий триггер.

Триггеры для событий таблицы (представления) могут быть удалены:

- владельцем таблицы (представления);
- пользователем SYSDBA;
- любым пользователем, подключенным с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователем операционной системы root (Linux);

Руководство по языку SQL СУБД Firebird

- администратором Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Триггеры для событий базы данных могут быть удалены:

- владельцем базы данных;
- пользователем SYSDBA;
- любой пользователем, подключенным с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователем операционной системы root (Linux);
- администратором Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

Удаление триггера set_cust_no.

```
DROP TRIGGER set_cust_no;
```

См. также: [CREATE TRIGGER](#), [RECREATE TRIGGER](#)

RECREATE TRIGGER

Создание нового или пересоздание существующего триггера.

Доступно: DSQL

Синтаксис:

```
RECREATE TRIGGER trigname
{ <relation_trigger_legacy>
| <relation_trigger_sql2003>
| <database_trigger> }
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END
```

Полное описание синтаксиса оператора см. [CREATE TRIGGER](#).

Описание:

Оператор RECREATE TRIGGER создаёт новый триггер, если триггер с указанным именем не существует, в противном случае оператор RECREATE

Руководство по языку SQL СУБД Firebird

TRIGGER попытается удалить его и создать новый.

Примеры:

Создание или пересоздание триггера 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
```

См. также: [CREATE TRIGGER](#), [DROP TRIGGER](#), [CREATE OR ALTER TRIGGER](#)

PROCEDURE

Хранимая процедура (ХП) – это программный модуль, который может быть вызван с клиента, из другой процедуры, выполняемого блока (executable block) или триггера. Хранимые процедуры, исполняемые блоки и триггеры пишутся на процедурном языке SQL (PSQL). Большинство операторов SQL доступно и в PSQL, иногда с ограничениями или расширениями. Заметными исключениями являются DDL и операторы управления транзакциями.

Хранимые процедуры могут принимать и возвращать множество параметров.

CREATE PROCEDURE

Создание новой хранимой процедуры.

Доступно: DSQL, ESQL

Синтаксис:

```
CREATE PROCEDURE procname
[(<inparam> [, <inparam> ...])]
[RETURNS (<outparam> [, <outparam> ...])]
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END

<inparam> ::= <param_decl> [{= | DEFAULT} <value>]
```

Руководство по языку SQL СУБД Firebird

```

<outparam> ::= <param_decl>

<value> ::= {literal | NULL | context_var}

<param_decl> ::= paramname <type> [NOT NULL]
[COLLATE collation]

<type> ::=
    <datatype>
    | [TYPE OF] domain
    | TYPE OF COLUMN rel.col

<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])]

<declarations> ::= {<declare_var> | <declare_cursor>};
[{{<declare_var> | <declare_cursor>}}; ...

```

Таблица 5.24. Параметры оператора CREATE PROCEDURE

Аргумент	Описание
procname	Имя хранимой процедуры. Может содержать до 31 символа.
inparam	Описание входного параметра.
outparam	Описание выходного параметра.
declarations	Секция объявления локальных переменных и именованных курсоров.
declare_var	Объявление локальной переменной.
declare_cursor	Объявление именованного курсора.
PSQL_statments	Операторы языка PSQL.
literal	Литерал.
context_var	Любая контекстная переменная, тип которой совместим с типом параметра.
paramname	Имя входного или выходного параметра процедуры. Может содержать до 31 символа. Имя параметра должно быть уникальным среди входных и выходных параметров процедуры, а

Руководство по языку SQL СУБД Firebird

	также её локальных переменных.
datatype	Тип данных SQL.
collation	Порядок сортировки.
domain	Домен.
rel	Имя таблицы или представления.
col	Имя столбца таблицы или представления.
precision	Точность. От 1 до 18.
scale	Масштаб. От 0 до 18, должно быть меньше или равно precision.
size	Максимальный размер строки в символах.
charset	Набор символов.
subtype_num	Номер подтипа BLOB.
subtype_name	Мнемоника подтипа BLOB.
seqlen	Размер сегмента, не может превышать 65535.

Описание:

Оператор CREATE PROCEDURE создаёт новую хранимую процедуру. Имя хранимой процедуры должно быть уникальным среди имён всех хранимых процедур, таблиц и представлений базы данных.

Входные параметры передаются в процедуру по значению, то есть любые изменения входных параметров внутри процедуры никак не повлияет на значения этих параметров в вызывающей программе. Входные параметры могут иметь значение по умолчанию. Параметры, для которых заданы значения, должны располагаться в конце списка параметров. Если входной параметр основан на домене, то значение по умолчанию, указанное для параметра, перекрывает значение по умолчанию указанное при описании домена.

Необязательное предложение RETURNS позволяет задать список выходных параметров хранимой процедуры.

У каждого параметра указывается тип данных. Кроме того, для параметра можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL. Для параметра строкового типа существует возможность задать порядок сортировки с помощью предложения COLLATE.

В качестве типа параметра можно указать имя домена. В этом случае, параметр будет наследовать все характеристики домена. Если перед названием домена дополнительно используется предложение "TYPE OF", то используется только тип данных домена – не проверяется (не используется) его ограничение (если оно есть в домене) на NOT NULL, CHECK ограничения и/или значения по умолчанию. Если домен текстового типа, то всегда используется его набор символов и порядок сортировки.

Входные и выходные параметры можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение TYPE OF COLUMN, после которого указывается имя таблиц или

Руководство по языку SQL СУБД Firebird

представления и через точку имя столбца. При использовании TYPE OF COLUMN наследуется только тип данных, а в случае строковых типов ещё и набор символов и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

В необязательной секции *declarations* описаны локальные переменные процедуры и именованные курсоры. Подробнее см. в разделе [DECLARE VARIABLE](#).

После секции объявления локальных переменных и именованных курсоров в теле хранимой процедуры, следует блок PSQL операторов, заключённых в операторные скобки BEGIN и END.

Создать новую хранимую процедуру может любой пользователь, подключённый к базе данных. Пользователь, создавший хранимую процедуру, становится её владельцем.

Примеры:

Создание хранимой процедуры осуществляющей вставку записи в таблицу BREED и возвращающей код вставленной записи.

```
CREATE PROCEDURE ADD_BREED (  
    NAME D_BREEDNAME, /* Наследуются характеристики домена */  
    NAME_EN TYPE OF D_BREEDNAME, /* Наследуется только тип  
домена */  
    SHORTNAME TYPE OF COLUMN BREED.SHORTNAME, /* Наследуется  
тип столбца таблицы */  
    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
```

См. Также: [CREATE OR ALTER PROCEDURE](#), [ALTER PROCEDURE](#), [RECREATE PROCEDURE](#), [DROP PROCEDURE](#)

ALTER PROCEDURE

Изменение существующей хранимой процедуры.

Доступно: DSQL, ESQL

Синтаксис:

```
ALTER PROCEDURE procname
[(inparam [, inparam ...])]
[RETURNS (outparam [, outparam ...])]
AS
[declarations]
BEGIN
[PSQL_statements]
END
```

inparam ::= *param_decl* [{= | DEFAULT} *value*]

outparam ::= *param_decl*

param_decl ::= *paramname* *type* [NOT NULL]
[COLLATE *collation*]

type ::=
 datatype
| [TYPE OF] *domain*
| TYPE OF COLUMN *rel.col*

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*])]

declarations ::= {*declare_var* | *declare_cursor*};
[*declare_var* | *declare_cursor*]; ...]

Таблица 5.25. Параметры оператора ALTER PROCEDURE

Аргумент	Описание
<i>procname</i>	Имя хранимой процедуры. Может содержать до

Руководство по языку SQL СУБД Firebird

	31 символа.
inparam	Описание входного параметра.
outparam	Описание выходного параметра.
declarations	Секция объявления локальных переменных и именованных курсоров.
declare_var	Объявление локальной переменной.
declare_cursor	Объявление именованного курсора.
PSQL_statments	Операторы языка PSQL.
literal	Литерал.
context_var	Любая контекстная переменная, тип которой совместим с типом параметра.
paramname	Имя входного или выходного параметра процедуры. Может содержать до 31 символа.
datatype	Тип данных SQL.
collation	Порядок сортировки.
domain	Домен.
rel	Имя таблицы или представления.
col	Имя столбца таблицы или представления.
precision	Точность. От 1 до 18.
scale	Масштаб. От 0 до 18, должно быть меньше или равно precision.
size	Максимальный размер строки в символах.
charset	Набор символов.
subtype_num	Номер подтипа BLOB.
subtype_name	Мнемоника подтипа BLOB.
seqlen	Размер сегмента, не может превышать 65535.

Описание:

Оператор ALTER PROCEDURE позволяет изменять состав и характеристики входных и выходных параметров, локальных переменных, именованных курсоров и тело хранимой процедуры. После выполнения существующие привилегии и зависимости сохраняются.

Предупреждение:

Будьте осторожны при изменении количества и типов входных и выходных параметров хранимых процедур. Существующий код приложения может стать неработоспособным из-за того, что формат вызова процедуры несовместим с новым описанием параметров. Кроме того, PSQL модули, использующие изменённую хранимую процедуру, могут стать некорректными. Информация о том, как это обнаружить, находится в разделе .

Изменить хранимую процедуру могут:

- владелец хранимой процедуры;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна

Руководство по языку SQL СУБД Firebird

- быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

Модификация хранимой процедуры 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
```

См. также: [CREATE PROCEDURE](#), [CREATE OR ALTER PROCEDURE](#), [RECREATE PROCEDURE](#), [DROP PROCEDURE](#)

CREATE OR ALTER PROCEDURE

Создание новой или изменение существующей хранимой процедуры.

Доступно: DSQL

Синтаксис:

```
CREATE OR ALTER PROCEDURE procname
[(inparam [, inparam ...])]
[RETURNS (outparam [, outparam ...])]
AS
[declarations]
BEGIN
[PSQL_statements]
END
```

Руководство по языку SQL СУБД Firebird

Полное описание синтаксиса оператора см. [CREATE PROCEDURE](#)

Описание:

Оператор CREATE OR ALTER PROCEDURE создаёт новую или изменяет существующую хранимую процедуру. Если хранимая процедура не существует, то она будет создана с использованием предложения CREATE PROCEDURE. Если она уже существует, то она будет изменена и откомпилирована, при этом существующие привилегии и зависимости сохраняются.

Примеры:

Создание или изменение процедуры GET_EMP_PROJ.

```
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
```

См. также: [CREATE PROCEDURE](#), [ALTER PROCEDURE](#), [RECREATE PROCEDURE](#)

DROP PROCEDURE

Удаление существующей хранимой процедуры.

Доступно: DSQL, ESQL

Синтаксис:

```
DROP PROCEDURE procname;
```

Таблица 5.26. Параметры оператора DROP PROCEDURE

Аргумент	Описание
procname	Имя хранимой процедуры.

Руководство по языку SQL СУБД Firebird

Описание:

Оператор DROP PROCEDURE удаляет существующую хранимую процедуру. Если от хранимой процедуры существуют зависимости, то при попытке удаления такой процедуру будет выдана соответствующая ошибка.

Удалить хранимую процедуру могут:

- владелец хранимой процедуры;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Пример:

Удаление хранимой процедуры GET_EMP_PROJ.

```
DROP PROCEDURE GET_EMP_PROJ;
```

См. также: [CREATE PROCEDURE](#), [RECREATE PROCEDURE](#)

RECREATE PROCEDURE

Создание новой или пересоздание существующей хранимой процедуры.

Доступно: DSQL

Синтаксис:

```
RECREATE PROCEDURE procname
[(<inparam> [, <inparam> ...])]
[RETURNS (<outparam> [, <outparam> ...])]
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END
```

Полное описание синтаксиса оператора см. В разделе .

Описание:

Руководство по языку SQL СУБД Firebird

Оператор **RECREATE PROCEDURE** создаёт новую или пересоздаёт существующую хранимую процедуру. Если процедура с таким именем уже существует, то оператор попытается удалить её и создать новую процедуру. Пересоздать процедуру невозможно, если у существующей процедуры имеются зависимости. После пересоздания процедуры привилегии на выполнение хранимой процедуры и привилегии самой хранимой процедуры не сохраняются.

Примеры:

Создание новой или пересоздание существующей процедуры
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
```

См. также: [CREATE PROCEDURE](#), [DROP PROCEDURE](#), [CREATE OR ALTER PROCEDURE](#)

EXTERNAL FUNCTION

Функции, определяемые пользователем (User Defined Function) – это программы, написанные на любом языке программирования, и хранящиеся в динамических библиотеках. Такие функции называют так же внешними функциями. Внешние функции существенно расширяют возможности SQL по обработке данных. Для того, чтобы функции были доступны в базе данных, их необходимо объявить с помощью оператора **DECLARE EXTERNAL FUNCTION**. После объявления функции содержащая её библиотека будет загружаться при первом обращении к любой из функций, включённой в библиотеку.

DECLARE EXTERNAL FUNCTION

Объявление в базе данных функции определённой пользователем (UDF).

Доступно: DSQL, ESQL

Синтаксис:

```

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]
                  | CSTRING(length)
    
```

Таблица 5.27. Параметры оператора DECLARE EXTERNAL FUNCTION

Аргумент	Описание
<code>funcname</code>	Имя функции в базе данных. Может содержать до 31 символа.
<code>entry_point</code>	Имя экспортируемой функции (точка входа).
<code>library_name</code>	Имя модуля, в котором расположена функция.
<code>sqltype</code>	Тип данных SQL. Не может быть массивом или элементом массива.
<code>length</code>	Максимальная длина нуль терминальной строки. Указывается в байтах.
<code>param_num</code>	Номер входного параметра, который будет возвращён функцией.

Описание:

Оператор DECLARE EXTERNAL FUNCTION делает доступным функцию, определённую пользователем (UDF), в базе данных. Имя внешней функции должно быть уникальным среди всех имён функций. Оно может отличаться от имени функции указанной в качестве точки входа.

Входные параметры функции перечисляются через запятую сразу после имени функции. Для каждого параметра указывается SQL тип данных. Массивы не могут использоваться в качестве параметров функций. Помимо SQL типов можно указать тип CSTRING. В этом случае параметр является нуль терминальной строкой с максимальной длиной `length` байт. По умолчанию входные параметры передаются по ссылке. Если указано предложение BY DESCRIPTOR, то входной параметр передаётся по дескриптору. Передача параметра по дескриптору облегчает обработку значений NULL.

Обязательное предложение RETURNS описывает выходной параметр возвращаемый функцией. Функция всегда возвращает только один параметр. Выходной параметр может быть любым SQL типом (кроме массива и элемента массива) или нуль терминальной строкой (CSTRING). Выходной параметр может быть передан по ссылке, по дескриптору или по значению. По умолчанию

Руководство по языку SQL СУБД Firebird

выходной параметр передаётся по ссылке. Если указано предложение BY DESCRIPTOR, то выходной параметр передаётся по дескриптору. Если указано предложение BY VALUE, то выходной параметр передаётся по значению.

Ключевое слово PARAMETER указывает, что функция возвращает значение из параметра с номером *param_num*. Такая необходимость возникает, если необходимо возвращать значение типа BLOB.

Ключевое слово FREE_IT означает, что память, выделенная для хранения возвращаемого значения, будет освобождена после завершения выполнения функции. Применяется только в том случае, если эта память в UDF выделялась динамически. В такой UDF память должна выделяться при помощи функции *ib_util_malloc* из модуля *ib_util*. Это необходимо для совместимости функций выделения и освобождения памяти используемого в коде Firebird и коде UDF.

Предложение ENTRY_POINT указывает имя точки входа (имя экспортируемой функции) в модуле.

Предложение MODULE_NAME задаёт имя модуля, в котором находится экспортируемая функция. В ссылке на модуль может отсутствовать полный путь и расширение файла. Это позволяет легче переносить базу данных между различными платформами. По умолчанию динамические библиотеки пользовательских функций должны располагаться в папке UDF корневого каталога сервера. Параметр UDFAccess в файле *firebird.conf* позволяет изменить ограничения доступа к библиотекам внешних функций.

Объявить внешнюю функцию (UDF) может любой пользователь, подключённый к базе данных.

Примеры:

1. Объявление внешней функции *addDate* расположенной в модуле *fbudf*. Входные и выходные параметры передаются по ссылке.

```
DECLARE EXTERNAL FUNCTION addDay
TIMESTAMP, INT
RETURNS TIMESTAMP
ENTRY_POINT 'addDay' MODULE_NAME 'fbudf';
```

2. Объявление внешней функции *invl* расположенной в модуле *fbudf*. Входные и выходные параметры передаются по дескриптору.

```
DECLARE EXTERNAL FUNCTION invl
INT BY DESCRIPTOR, INT BY DESCRIPTOR
RETURNS INT BY DESCRIPTOR
ENTRY_POINT 'idNvl' MODULE_NAME 'fbudf';
```

3. Объявление внешней функции *isLeapYear* расположенной в модуле *fbudf*.

Руководство по языку SQL СУБД Firebird

Входной параметр передаётся по ссылке, а выходной передаётся по значению.

```
DECLARE EXTERNAL FUNCTION isLeapYear
TIMESTAMP
RETURNS INT BY VALUE
ENTRY_POINT 'isLeapYear' MODULE_NAME 'fbudf';
```

4. Объявление внешней функции i64Truncate расположенной в модуле fbudf. Входной параметр и выходной параметр передаются по дескриптору. В качестве возвращаемого значения используется второй параметр функции.

```
DECLARE EXTERNAL FUNCTION i64Truncate
NUMERIC(18) BY DESCRIPTOR, NUMERIC(18) BY DESCRIPTOR
RETURNS PARAMETER 2
ENTRY_POINT 'fbtruncate' MODULE_NAME 'fbudf';
```

См. также: [ALTER EXTERNAL FUNCTION](#), [DROP EXTERNAL FUNCTION](#)

ALTER EXTERNAL FUNCTION

Изменение точки входа и/или имени модуля для функции определённой пользователем (UDF).

Доступно: DSQL

Синтаксис:

```
ALTER EXTERNAL FUNCTION funcname
{
    [ENTRY_POINT 'new_entry_point']
    [MODULE_NAME 'new_library_name'] };
```

Таблица 5.28. Параметры оператора ALTER EXTERNAL FUNCTION

Аргумент	Описание
funcname	Имя функции в базе данных.
new_entry_point	Новое имя экспортируемой функции (точки входа).
new_library_name	Новое имя модуля, в котором расположена функция.

Описание:

Оператор ALTER EXTERNAL FUNCTION изменяет точку вход и/или имя модуля для функции определённой пользователем (UDF). При этом существующие зависимости сохраняются.

Руководство по языку SQL СУБД Firebird

Предложение ENTRY_POINT позволяет указать новую точку входа (имя экспортируемой функции).

Предложение MODULE_NAME позволяет указать новое имя модуля, в котором расположена экспортируемая функция.

Изменить точку входа и имя модуля может любой пользователь, подключённый к базе данных.

Примеры:

1. Изменение точки входа для внешней функции

```
ALTER EXTERNAL FUNCTION invl ENTRY_POINT 'intNvl';
```

2. Изменение имени модуля для внешней функции

```
ALTER EXTERNAL FUNCTION invl MODULE_NAME 'fbudf2';
```

См. Также: [DECLARE EXTERNAL FUNCTION](#), [DROP EXTERNAL FUNCTION](#)

DROP EXTERNAL FUNCTION

Удаление объявления функции определённой пользователем (UDF) из базы данных.

Доступно: DSQL, ESQL

Синтаксис:

```
DROP EXTERNAL FUNCTION funcname;
```

Таблица 5.29. Параметры оператора DROP EXTERNAL FUNCTION

Аргумент	Описание
funcname	Имя функции в базе данных.

Описание:

Оператор DROP EXTERNAL FUNCTION удаляет объявление функции определённой пользователем из базы данных. Если есть зависимости от внешней функции, то удаления не произойдёт и будет выдана соответствующая ошибка.

Удалить объявление внешней функции может любой пользователь, подключённый к базе данных.

Примеры:

Руководство по языку SQL СУБД Firebird

Удаление объявления функции addDay.

```
DECLARE EXTERNAL FUNCTION addDay;
```

См. также: [DECLARE EXTERNAL FUNCTION](#)

FILTER

BLOB фильтр – объект базы данных, являющийся по сути специальным видом внешних функций с единственным назначением: получение объекта BLOB одного формата и преобразования его в объект BLOB другого формата. Форматы объектов BLOB задаются с помощью подтипов BLOB. Подробнее о подтипах BLOB см. В разделе . Внешние функции для преобразования BLOB типов хранятся в динамических библиотеках и загружаются по необходимости.

DECLARE FILTER

Объявление в базе данных BLOB фильтра.

Доступно: DSQL, ESQL

Синтаксис:

```
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
```

Таблица 5.30. Параметры оператора DECLARE FILTER

Аргумент	Описание
<i>filtername</i>	Имя фильтра. Может содержать до 31 символа.
<i>sub_type</i>	Подтип BLOB.
<i>number</i>	Номер подтипа BLOB.
<i>mnemonic</i>	Мнемоника подтипа BLOB.
<i>function_name</i>	Имя экспортируемой функции (точка входа).
<i>library_name</i>	Имя модуля, в котором расположен фильтр.
<i>user_defined</i>	Определяемая пользователем мнемоника подтипа BLOB.

Описание:

Оператор DECLARE FILTER делает доступным BLOB фильтр в базе

Руководство по языку SQL СУБД Firebird

данных. Имя BLOB фильтра должно быть уникальным среди имён BLOB фильтров.

Предложение `INPUT_TYPE` идентифицирует тип преобразуемого объекта (подтип BLOB). Предложение `OUTPUT_TYPE` идентифицирует тип создаваемого объекта. Подтип может быть задан в виде номера подтипа или мнемоники подтипа. Пользовательские подтипы должны быть представлены отрицательными числами (от -1 до -32768). Не допускается создание двух и более фильтров BLOB с одинаковыми комбинациями входных и выходных типов. Объявление фильтра с уже существующими комбинациями входных и выходных типов BLOB приведёт к ошибке.

Примечание:

Если вы хотите определить мнемоники для собственных подтипов BLOB, вы можете добавить их в системную таблицу `RDB$TYPES`, как показано ниже. После подтверждения транзакции мнемоники могут быть использованы для декларации при создании новых фильтров.

```
INSERT INTO RDB$TYPES (RDB$FIELD_NAME, RDB$TYPE, RDB$TYPE_NAME)
VALUES ('RDB$FIELD_SUB_TYPE', -33, 'MIDI');
```

Значение поля `rdb$field_name` всегда должно быть `'RDB$FIELD_SUB_TYPE'`. Если вы определяете мнемоники в верхнем регистре, то можете использовать их без учета регистра и без кавычек при объявлении фильтра.

Предложение `ENTRTY_POINT` указывает имя точки входа (имя экспортируемой функции) в модуле.

Предложение `MODULE_NAME` задаёт имя модуля, в котором находится экспортируемая функция. По умолчанию модули должны располагаться в папке UDF корневого каталога сервера. Параметр `UDFAccess` в файле `firebird.conf` позволяет изменить ограничения доступа к библиотекам фильтрам.

Создать BLOB фильтр может любой пользователь, подключённый к базе данных.

Примеры:

1. Создание BLOB фильтра с использованием номеров подтипов.

```
DECLARE FILTER DESC_FILTER
INPUT_TYPE 1
OUTPUT_TYPE -4
ENTRY_POINT 'desc_filter'
MODULE_NAME 'FILTERLIB';
```

2. Создание BLOB фильтра с использованием мнемоник подтипов.

```
DECLARE FILTER FUNNEL  
INPUT_TYPE blr OUTPUT_TYPE text  
ENTRY_POINT 'blr2asc' MODULE_NAME 'myfilterlib';
```

См. также: [DROP FILTER](#)

DROP FILTER

Удаление объявления BLOB фильтра.

Доступно: DSQL, ESQL

Синтаксис:

```
DROP FILTER filtername;
```

Таблица 5.31. Параметры оператора DROP FILTER

Аргумент	Описание
<i>filtername</i>	Имя фильтра.

Описание:

Оператор DROP FILTER удаляет объявление BLOB фильтра из базы данных. Удаление BLOB фильтра из базы данных делает его не доступным из базы данных, при этом динамическая библиотека, в которой расположена функция преобразования, остаётся не тронутой.

Удалить объявление BLOB фильтра может любой пользователь, подключённый к базе данных.

Примеры:

Удаление BLOB фильтра.

```
DROP FILTER DESC_FILTER;
```

См. также: [DECLARE FILTER](#)

SEQUENCE (GENERATOR)

Последовательность (sequence) или **генератор** (generator) – объект базы данных, предназначенный для получения уникального числового значения. Термин последовательность является SQL-совместимым. Ранее в Interbase и Firebird последовательности называли генераторами.

Руководство по языку SQL СУБД Firebird

Независимо от диалекта базы данных последовательности (или генераторы) всегда хранятся как 64-битные целые значения. Однако отметим, что:

- Если клиент использует 1 диалект, то сервер передает ему значения последовательности, усеченные до 32-битного значения;
- Если значение последовательности передаются в 32-разрядное поле или переменную, то до тех пор, пока текущее значение последовательности не вышло за границы для 32-битного числа, ошибок не будет. В момент выхода значения последовательности за этот диапазон база данных 3-го диалекта выдаст сообщение об ошибке, а база данных 1-го диалекта будет молча обрезать значения (что также может привести к ошибке — например, если поле, заполняемое генератором, является первичным или уникальным).

В данном разделе описываются вопросы создания, модификации (установка значения последовательности) и удаления последовательностей.

CREATE SEQUENCE (GENERATOR)

Создание новой последовательности (генератора).

Доступно: DSQL, ESQL

Синтаксис:

```
CREATE {SEQUENCE | GENERATOR} seq_name;
```

Таблица 5.32. Параметры оператора CREATE SEQUENCE (GENERATOR)

Аргумент	Описание
seq_name	Имя последовательности (генератора). Может содержать до 31 символа.

Описание:

Оператор CREATE SEQUENCE создаёт новую последовательность. Слова SEQUENCE и GENERATOR являются синонимами. Вы можете использовать любое из них, но рекомендуется использовать SEQUENCE.

В момент создания последовательности ей устанавливается значение равное 0. В дальнейшем при обращении к конструкции NEXT VALUE FOR её значение увеличивается на единицу. Значение последовательности изменяется также при обращении к функции GEN_ID, где в качестве параметра указывается имя последовательности и значение приращения.

Создать последовательность (генератор) может любой пользователь, присоединившийся к базе данных.

Руководство по языку SQL СУБД Firebird

Примеры:

Создание последовательности EMP_NO_GEN.

```
CREATE SEQUENCE EMP_NO_GEN;
```

См. Также: ALTER SEQUENCE, SET GENERATOR, DROP SEQUENCE (GENERATOR)

ALTER SEQUENCE

Устанавливает значение последовательности или генератора в заданное значение.

Доступно: DSQL

Синтаксис:

```
CREATE SEQUENCE seq_name RESTART WITH new_val;
```

Таблица 5.32. Параметры оператора ALTER SEQUENCE

Аргумент	Описание
seq_name	Имя последовательности (генератора).
new_val	Новое значение последовательности (генератора). 64 битное целое в диапазоне от -2^{63} до $2^{63}+1$.

Описание:

Оператор ALTER SEQUENCE устанавливает значение последовательности или генератора в заданное значение.

Предупреждение:

Неосторожное использование оператора ALTER SEQUENCE (изменение значения последовательности или генератора) может привести к нарушению логической целостности данных.

Установить значение последовательности (генератора) может любой пользователь, присоединившийся к базе данных.

Примеры:

Установка для последовательности EMP_NO_GEN значения 145.

Руководство по языку SQL СУБД Firebird

```
ALTER SEQUENCE EMP_NO_GEN RESTART WITH 145;
```

Примечание:

То же самое можно сделать используя оператор

```
SET GENERATOR EMP_NO_GEN TO 145;
```

См. Также: [SET GENERATOR](#), [CREATE SEQUENCE \(GENERATOR\)](#), [DROP SEQUENCE \(GENERATOR\)](#)

SET GENERATOR

Устанавливает значение последовательности или генератора в заданное значение.

Доступно: DSQL, ESQL

Синтаксис:

```
SET GENERATOR seq_name TO new_val;
```

Таблица 5.33. Параметры оператора SET GENERATOR

Аргумент	Описание
seq_name	Имя последовательности (генератора).
new_val	Новое значение последовательности (генератора). 64 битное целое в диапазоне от -2^{63} до $2^{63}+1$.

Описание:

Оператор SET GENERATOR устанавливает значение последовательности или генератора в заданное значение.

Замечание:

Оператор SET GENERATOR считается устаревшим и оставлен ради обратной совместимости. В настоящее время вместо него рекомендуется использовать эквивалентный оператор ALTER SEQUENCE.

Установить значение последовательности (генератора) может любой пользователь, присоединившийся к базе данных.

Предупреждение:

Неосторожное использование оператора SET GENERATOR (изменение значения последовательности или генератора) может привести к потере

логической целостности данных.

Примеры:

Установка для последовательности EMP_NO_GEN значения 145.

```
SET GENERATOR EMP_NO_GEN TO 145;
```

Примечание:

То же самое можно сделать используя оператор

```
ALTER SEQUENCE EMP_NO_GEN RESTART WITH 145;
```

См. Также: ALTER SEQUENCE, CREATE SEQUENCE (GENERATOR), DROP SEQUENCE (GENERATOR)

DROP SEQUENCE (GENERATOR)

Удаление последовательности (генератора).

Доступно: DSQL, ESQL

Синтаксис:

```
DROP {SEQUENCE | GENERATOR} seq_name;
```

Таблица 5.34. Параметры оператора DROP SEQUENCE (GENERATOR)

Аргумент	Описание
seq_name	Имя последовательности (генератора).

Описание:

Оператор DROP SEQUENCE удаляет существующую последовательность (генератор). Слова SEQUENCE и GENERATOR являются синонимами. Вы можете использовать любое из них, но рекомендуется использовать SEQUENCE. При наличии зависимостей для существующей последовательности (генератора) удаления не будет выполнено.

Удалить последовательность может любой пользователь, подключённый к базе данных.

Примеры:

Удаление последовательности EMP_NO_GEN.

```
DROP SEQUENCE EMP_NO_GEN;
```

Руководство по языку SQL СУБД Firebird

См. Также: CREATE SEQUENCE (GENERATOR), ALTER SEQUENCE, SET GENERATOR

EXCEPTION

Пользовательское исключение (exception) – объект базы данных, описывающий сообщение об ошибке. Исключение можно вызывать и обрабатывать в PSQL коде.

В данном разделе описываются операторы создания, модификации и удаления исключений.

CREATE EXCEPTION

Создание пользовательского исключения.

Доступно: DSQL, ESQL

Синтаксис:

```
CREATE EXCEPTION exception_name 'message';
```

Таблица 5.35. Параметры оператора CREATE EXCEPTION

Аргумент	Описание
exception_name	Имя исключения. Максимальная длина 31 символ.
message	Сообщение об ошибке. Максимальная длина ограничена 1021 символом.

Описание:

Оператор CREATE EXCEPTION создаёт новое пользовательское исключение. Исключение должно отсутствовать в базе данных, иначе будет выдана соответствующая ошибка.

Создать исключение может любой пользователь, соединившийся с базой данных.

Примеры:

Создание пользовательского исключения E_LARGE_VALUE.

```
CREATE EXCEPTION E_LARGE_VALUE 'Значение превышает предельно допустимое';
```

Руководство по языку SQL СУБД Firebird

См. также: ALTER EXCEPTION, CREATE OR ALTER EXCEPTION, DROP EXCEPTION, RECREATE EXCEPTION

ALTER EXCEPTION

Изменение текста сообщения пользовательского исключения.

Доступно: DSQL, ESQL

Синтаксис:

```
ALTER EXCEPTION exception_name 'message';
```

Таблица 5.36. Параметры оператора ALTER EXCEPTION

Аргумент	Описание
exception_name	Имя исключения. Максимальная длина 31 символ.
message	Сообщение об ошибке. Максимальная длина ограничена 1021 символом.

Описание:

Оператор ALTER EXCEPTION изменяет текст сообщения пользовательского исключения.

Изменять текст сообщения пользовательского исключения может любой пользователь, соединившийся с базой данных.

Примеры:

Изменение текста сообщения для пользовательского исключения E_LARGE_VALUE.

```
ALTER EXCEPTION E_LARGE_VALUE 'Значение превышает максимально допустимое';
```

См. также: CREATE EXCEPTION, CREATE OR ALTER EXCEPTION, DROP EXCEPTION, RECREATE EXCEPTION

CREATE OR ALTER EXCEPTION

Создание нового или изменение существующего исключения.

Доступно: DSQL

Синтаксис:

Руководство по языку SQL СУБД Firebird

```
CREATE OR ALTER EXCEPTION exception_name 'message';
```

Таблица 5.37. Параметры оператора CREATE OR ALTER EXCEPTION

Аргумент	Описание
<code>exception_name</code>	Имя исключения. Максимальная длина 31 символ.
<code>message</code>	Сообщение об ошибке. Максимальная длина ограничена 1021 символом.

Описание:

Если исключения не существует, то оно будет создано. Уже существующее исключение будет изменено, при этом существующие зависимости исключения будут сохранены.

Примеры:

Изменение текста сообщения для пользовательского исключения E_LARGE_VALUE, если исключения не существует, то оно будет создано.

```
CREATE OR ALTER EXCEPTION E_LARGE_VALUE 'Значение превышает максимально допустимое';
```

См. также: [CREATE EXCEPTION](#), [ALTER EXCEPTION](#), [RECREATE EXCEPTION](#)

DROP EXCEPTION

Удаление пользовательского исключения.

Доступно: DSQL

Синтаксис:

```
DROP EXCEPTION exception_name;
```

Таблица 5.38. Параметры оператора DROP EXCEPTION

Аргумент	Описание
<code>exception_name</code>	Имя исключения.

Описание:

Оператор DROP EXCEPTION удаляет пользовательское исключение. При наличии зависимостей для существующего исключения удаления не будет выполнено.

Руководство по языку SQL СУБД Firebird

Удалить пользовательское исключение может любой пользователь, соединившийся с базой данных.

Примеры:

Удалить пользовательское исключение E_LARGE_VALUE.

См. также: [CREATE EXCEPTION](#), [RECREATE EXCEPTION](#)

RECREATE EXCEPTION

Создание или пересоздание пользовательского исключения.

Доступно: DSQL

Синтаксис:

```
RECREATE EXCEPTION exception_name 'message';
```

Таблица 5.39. Параметры оператора RECREATE EXCEPTION

Аргумент	Описание
exception_name	Имя исключения. Максимальная длина 31 символ.
message	Сообщение об ошибке. Максимальная длина ограничена 1021 символом.

Описание:

Оператор RECREATE EXCEPTION создаёт или пересоздаёт пользовательское исключение. Если исключение с таким именем уже существует, то оператор RECREATE EXCEPTION попытается удалить его и создать новое исключение. При наличии зависимостей для существующего исключения оператор RECREATE EXCEPTION не выполнится.

Примеры:

Пересоздание пользовательского исключения E_LARGE_VALUE.

```
RECREATE EXCEPTION E_LARGE_VALUE 'Значение превышает  
максимально допустимое';
```

См. также: [CREATE EXCEPTION](#), [DROP EXCEPTION](#), [CREATE OR ALTER EXCEPTION](#)

COLLATION

CREATE COLLATION

Добавление новой сортировки.

Доступно: DSQL

Синтаксис:

```
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
```

Таблица 5.40. Параметры оператора CREATE COLLATION

Аргумент	Описание
collname	Имя сортировки. Максимальная длина 31 символ.
charset	Набор символов.
basecoll	Базовая сортировка.
extname	Имя сортировки из конфигурационного файла. Чувствительно к регистру.

Описание:

Оператор CREATE COLLATION добавляет новую сортировку для указанного набора символов. Имя сортировки должно быть уникальным среди всех имён сортировок.

Необязательное предложение FROM указывает сортировку, на основе которой будет создана новая сортировка. Такая сортировка должна уже присутствовать в базе данных. Если указано ключевое слово EXTERNAL, то будет осуществлён поиск сортировки из файла *\$fbroot/intl/fbintl.conf*, при этом «external» имя должно в точности соответствовать имени в конфигурационном файле (чувствительно к регистру).

При отсутствии предложения FROM Firebird ищет в конфигурационном файле *fbintl.conf* подкаталога *intl* корневой директории сервера сортировку с именем, указанным сразу после CREATE COLLATION.

При создании сортировки можно указать учитываются ли конечные пробелы при сравнении. Если указана опция NO PAD, то конечные пробелы при

Руководство по языку SQL СУБД Firebird

сравнении не учитываются. Если указана опция PAD SPACE, то конечные пробелы при сравнении учитываются.

Необязательное предложение CASE позволяет указать будет ли сравнение чувствительно к регистру.

Необязательное предложение ACCENT позволяет указать будет ли сравнение чувствительно к акцентированным буквам (например «е» и «ё»).

В операторе CREATE COLLATION можно также указать специфичные атрибуты для сортировки. Ниже в таблице 5.41 приведён список доступных специфичных атрибутов. Не все атрибуты применимы ко всем сортировкам. Если атрибут не применим к сортировке, но указан при её создании, то это не вызовет ошибки. Имена специфичных атрибутов чувствительны к регистру.

«1 bpc» в таблице указывает на то, что атрибут действителен для сортировок наборов символов, использующих 1 байт на символ (так называемый узкий набор символов), а «UNI» - для юникодных сортировок.

Таблица 5.41. Список доступных специфичных атрибутов COLLATION

Имя	Значение	Валидность	Комментарий
DISABLE-COMPRESSIONS	0, 1	1 bpc	Отключает сжатия (иначе сокращения). Сжатия заставляют определённые символьные последовательности быть сортированными как атомарные модули, например, испанские с+h как единственный символ ch.
DISABLE-EXPANSIONS	0, 1	1 bpc	Отключение расширений. Расширения позволяют рассматривать определённые символы (например, лигатуры или гласные умляуты) как последовательности символов и соответственно сортировать.
ICU-VERSION	<i>default</i> или <i>M.m</i>	UNI	Задаёт версию библиотеки ICU для использования. Допустимые значения определены в соответствующих элементах <intl_module> в файле intl/fbintl.conf. Формат: либо строка «default» или основной и дополнительный номер версии, как «3.0» (оба без

Руководство по языку SQL СУБД Firebird

Имя	Значение	Валидность	Комментарий
			кавычек).
LOCALE	xx_YY	UNI	Задаёт параметры сортировки языкового стандарта. Требуется полная версия библиотеки ICU. Формат строки: «du_NL» (без кавычек).
MULTI-LEVEL	0, 1	1 bpc	Использование нескольких уровней сортировки.
NUMERIC-SORT	0, 1	UNI	Обрабатывает непрерывные группы десятичных цифр в строке как атомарные модули и сортирует их в числовой последовательности. (известна как естественная сортировка)
SPECIALS-FIRST	0, 1	1 bpc	Сортирует специальные символы (пробелы и т.д.) до буквенно-цифровых символов.

Совет:

Если вы хотите добавить в базу данных новый набор символов с его умалчиваемой сортировкой, то зарегистрируйте и выполните хранимую процедуру `sp_register_character_name(name, max_bytes_per_character)` из подкаталога `misc/intl.sql` установки Firebird. Для нормальной работы с набором символов он должен присутствовать в вашей операционной системе и зарегистрирован в файле `fbintl.conf` поддиректории `intl`.

Создать новую сортировку может любой пользователь, подключённый к базе данных.

Примеры:

1. Создание сортировки с использованием имени, найденном в файла `fbintl.conf` (регистро-чувствительно).

```
CREATE COLLATION ISO8859_1_UNICODE FOR ISO8859_1;
```

2. Создание сортировки с использованием специального (заданного пользователем) названия («external» имя должно в точности соответствовать имени в файле `fbintl.conf`).

```
CREATE COLLATION LAT_UNI
FOR ISO8859_1
FROM EXTERNAL ('ISO8859_1_UNICODE');
```

3. Создание регистр независимой сортировки на основе уже

Руководство по языку SQL СУБД Firebird

присутствующей в базе данных.

```
CREATE COLLATION ES_ES_NOPAD_CI
FOR ISO8859_1
FROM ES_ES
NO PAD
CASE INSENSITIVE;
```

4. Создание регистр независимой сортировки на основе уже присутствующей в базе данных со специфичными атрибутами.

```
CREATE COLLATION ES_ES_CI_COMPR
FOR ISO8859_1
FROM ES_ES
CASE INSENSITIVE
'DISABLE-COMPRESSIONS=0';
```

5. Создание регистронезависимой сортировки по значению чисел (так называемой натуральной сортировки).

```
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 ...);
```

См. также: [DROP COLLATION](#)

DROP COLLATION

Удаление существующей сортировки.

Доступно: DSQL

Синтаксис:

```
DROP COLLATION collname;
```

Таблица 5.42. Параметры оператора DROP COLLATION

Аргумент	Описание
collname	Имя сортировки.

Описание:

Руководство по языку SQL СУБД Firebird

Оператор DROP COLLATION удаляет указанную сортировку. Сортировка должна присутствовать в базе данных, иначе будет выдана соответствующая ошибка.

Совет:

Если вы хотите удалить в базе данных набор символов со всеми его сортировками, то зарегистрируйте и выполните хранимую процедуру `sp_unregister_character_set(name)` из подкаталога `misc/intl.sql` установки Firebird.

Удалить сортировку может любой пользователь, подключенный к базе данных.

Примеры:

Удаление сортировки ES_ES_NOPAD_CI.

```
DROP COLLATION ES_ES_NOPAD_CI;
```

См. также: [CREATE COLLATION](#)

CHARACTER SET

ALTER CHARACTER SET

Установка умолчательной сортировки для набора символов.

Доступно: DSQL

Синтаксис:

```
ALTER CHARACTER SET charset  
SET DEFAULT COLLATION collation;
```

Таблица 5.42. Параметры оператора ALTER CHARACTER SET

Аргумент	Описание
charset	Набор символов
collation	Сортировка

Описание:

Оператор ALTER CHARACTER SET изменяет умолчательную сортировку для указанного набора символов. Это повлияет на использование набора символов в будущем, кроме случаев, когда явно переопределена сортировка

Руководство по языку SQL СУБД Firebird

COLLATE. Сортировка существующих доменов, столбцов и переменных PSQL при этом не будет изменена.

Замечания:

Если умолчательная сортировка была изменена для набора символов базы данных (тот, что был указан при создании базы данных), то также изменяется и умолчательная сортировка для базы данных.

Если умолчательная сортировка была изменена для набора символов, который был указан при подключении, строковые константы будут интерпретироваться в соответствии с новыми параметрами сортировки (если набор символов и/или сортировка не переопределяются).

Примеры:

Установка умолчательной сортировки UNICODE_CI_AI для кодировки UTF8.

```
ALTER CHARACTER SET UTF8 SET DEFAULT COLLATION  
UNICODE_CI_AI;
```

SELECT

Доступен в DSQL, PSQL

Оператор (команда) SELECT извлекает данные из базы данных и передает их в приложение или в вызывающую SQL команду. Данные возвращаются в виде набора *строк* (которых может быть 0 или больше), каждая строка содержит один или более *столбцов* или *полей*. Совокупность возвращаемых строк является *результатирующим набором данных* команды.

Общий синтаксис:

```
[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 <PSQL-varlist>]
```

Следующие части команды SELECT являются обязательными:

- Ключевое слово SELECT, за которым следует список полей. Эта часть определяет что запрашивается из базы данных.
- Ключевое слово FROM, за которым следует объект выборки (например, таблица). Эта часть сообщает серверу, где следует искать запрашиваемые данные.

В простейшей форме SELECT извлекает ряд полей из единственной таблицы, например:

Руководство по языку SQL СУБД Firebird

```
select id, name, address
from contacts
```

Или, для того чтобы извлечь все поля таблицы:

```
select * from contacts
```

На практике команда SELECT обычно выполняется с выражением WHERE, которое ограничивает возвращаемый набор данных. Также, полученный набор данных обычно сортируется с помощью выражения ORDER BY, дополнительно ограничивается (с целью организации постраничного просмотра данных) выражениями FIRST, SKIP или ROWS.

Список полей может содержать различные типы выражений вместо имен полей, а источник необязательно должен быть таблицей или представлением, он так же может быть производной таблицей (derived table), общим табличным выражением (CTE) или селективной хранимой процедурой.

Несколько источников данных могут быть скомбинированы с помощью выражения JOIN, и несколько результирующих наборов данных могут быть скомбинированы с использованием выражения UNION.

В следующих секциях мы подробно рассмотрим все выражения для команды SELECT и их использование.

FIRST, SKIP и ROWS

Доступно в DSQL, PSQL

Синтаксис:

```
SELECT
[FIRST <m>] [SKIP <n>]
FROM ...
...
```

```
<m>, <n> ::= integer literal
           | query parameter
           | (integer-expression)
```

ИЛИ

```
SELECT
...
FROM ...
```

Руководство по языку SQL СУБД Firebird

...
ROWS *m* [TO *n*]

m, n ::= integer-expression

Внимание! *FIRST* и *SKIP* используются только в *Firebird*, они не включены в стандарт SQL. Рекомендуется использовать *ROWS* везде, где это возможно.

Выражение *FIRST* <число записей> ограничивает результирующий набор данным указанным числом записей.

Выражение *SKIP* <число записей> пропускает указанное число записей перед выдачей результирующего набора данных.

Когда эти выражения используются совместно, например *FIRST m SKIP n*, то в результате *n* записей будет пропущено и, из оставшихся, *m* записей будет возвращено в результирующем наборе данных.

FIRST и *SKIP* являются необязательными выражениями.

Особенности использования

Разрешается использовать *SKIP 0* – в этом случае 0 записей будет пропущено.

В случае использования *FIRST 0* будет возвращен пустой набор записей.

Отрицательные значения *FIRST* и *SKIP* вызовут ошибку.

Если указанное в *SKIP* значение превышает размер результирующего набора данных, то вернется пустой набор данных.

Если число записей в наборе данных (или остаток после применения *SKIP*) меньше, чем заданное в *FIRST* значение, то соответственно меньшее количество записей будет возвращено.

Любой аргумент *FIRST* или *SKIP*, который не является целым числом или параметром SQL должен был заключен в круглые скобки. Это, означает, что в случае использования вложенной команды *SELECT* в качестве параметра для *FIRST* или *SKIP*, он должен быть вложен в две пары скобок.

Обратите внимание:

Существует две ошибки при использовании *FIRST* в подзапросах

- Этот запрос

```
DELETE FROM MYTABLE  
WHERE ID IN (SELECT FIRST 10 ID FROM MYTABLE)
```

удалит ВСЕ записи из таблицы.

Подзапрос каждый раз при удалении выбирает 10 строк, удаляет их — и так повторяется до тех пор, пока таблица не станет пустой. Знайте об этом! Или лучше: использовать указание ROWS в операторе DELETE.

- Такие запросы, как

...

```
WHERE F1 IN (SELECT FIRST 5 F2  
          FROM TABLE2  
          ORDER BY 1 DESC)
```

не будут работать, как ожидалось, т.к. оптимизатор сервера преобразует предикат IN в предикат EXISTS, как показано ниже. Очевидно, что в этом случае использование указания FIRST N не имеет никакого смысла:

...

```
WHERE EXISTS (SELECT FIRST 5 F2  
          FROM TABLE2  
          WHERE TABLE2.F2 = TABLE1.F2  
          ORDER BY 1 DESC)
```

Примеры FIRST и SKIP

Следующий запрос вернет первые 10 имен из таблицы PEOPLE (имена также будут отсортированы, см. ниже раздел [Предложение ORDER BY](#)):

```
select first 10 id, name  
from PEOPLE  
order by name asc
```

Следующий запрос вернет все записи из таблицы PEOPLE, за исключением первых 10 имен:

Руководство по языку SQL СУБД Firebird

```
select skip 10 id, name
from People
order by name asc
```

А этот запрос вернет последние 10 записей (обратите внимание на двойные кавычки):

```
select skip ((select count(*) - 10 from People))
id, name
from People
order by name asc
```

Этот запрос вернет строки 81-100 из таблицы PEOPLE:

```
select first 20 skip 80 id, name
from People
order by name asc
```

ROWS

Как было сказано выше, FIRST и SKIP не являются стандартными выражениями SQL. Поэтому, если Вы пишете новый код, лучше использовать ключевое слово ROWS.

В отличие от FIRST и SKIP, выражение ROWS принимает все типы целочисленных (integer) выражений в качестве аргумента – без скобок! Конечно, скобки могут требоваться для правильных вычислений внутри выражения, и вложенный запрос также должен быть обернут в скобки.

Вызов ROWS *m* приведет к возвращению первых *m* записей из набора данных.

В случае указания ROWS *m* TO *n*, то будут возвращены записи с *m* по *n* из набора данных. *Важно: нумерация записей в наборе данных начинается с 1!*

Особенности ROWS:

- Если *m* > общего числа записей в возвращаемом наборе данных, то будет возвращен весь набор данных
- Если *m* = 0, то будет возвращен пустой набор данных
- Если *m* < 0, возникнет ошибка
- Если *m* больше общего количества строк в наборе данных, то будет возвращён пустой набор данных;
- Если число *m* не превышает общего количества строк в наборе данных, а *n* превышает, то выборка ограничивается строками начиная с *m* до конца набора данных;

Руководство по языку SQL СУБД Firebird

- Если $m < 1$ и $n < 1$, то оператор SELECT выдаст ошибку;
- Если $n = m - 1$, то будет возвращён пустой набор данных
- Если $n < m - 1$, то оператор SELECT выдаст ошибку.

В сущности, ROWS заменяет собой нестандартные выражения FIRST и SKIP, за исключением единственного случая, когда указывается только SKIP, т.е. когда возвращается весь набор данных за исключением пропуска указанного числа записей с начала.

Для того, что реализовать такое поведение с помощью ROWS, необходимо указать второй аргумент, заведомо больший, чем размер возвращаемого набора данных. Или запросить число записей в возвращаемом наборе через подзапрос.

Нельзя использовать ROWS вместе с FIRST/SKIP в одном и том же операторе SELECT, но можно использовать разные синтаксисы в разных подзапросах.

При использовании ROWS с выражением UNION, он будет применяться к объединённому набору данных, и должен быть помещен после последнего SELECT.

При необходимости ограничить возвращаемые наборы данных одного или нескольких операторов SELECT внутри UNION, можно воспользоваться следующими вариантами:

- 1) Использовать FIRST/SKIP в этих операторах SELECT. Необходимо помнить, что нельзя локально использовать выражение ORDER BY в SELECT внутри UNION – только глобально, ко всему суммарному набору данных.
- 2) Преобразовать SELECT в производные таблицы с выражениями ROWS.

Примеры с ROWS

Ниже приведены примеры, ранее использованные для демонстрации FIRST/SKIP.

Следующий запрос вернет первые 10 имен из таблицы PEOPLE (имена также будут отсортированы, см ниже раздел [Предложение ORDER BY](#)):

```
select id, name
from People
order by name asc
rows 1 to 10
```

или его эквивалент

```
select id, name
```

Руководство по языку SQL СУБД Firebird

```
from People
order by name asc
rows 10
```

Следующий запрос вернет все записи из таблицы PEOPLE, за исключением первых 10 имен:

```
select id, name
from People
order by name asc
rows 11 to (select count(*) from People)
```

А этот запрос вернет последние 10 записей (обратите внимание на скобки):

```
select id, name
from People
order by name asc
rows (select count(*) - 9 from People)
to (select count(*) from People)
```

Этот запрос вернет строки 81-100 из таблицы PEOPLE:

```
select id, name
from People
order by name asc
rows 81 to 100
```

И FIRST/SKIP, и ROWS могут быть использованы без выражения ORDER BY, хотя это редко имеет смысл, за исключением случая, когда необходимо быстро взглянуть на данные таблицы – получаемые строки при этом будут чаще всего в случайном порядке. В этом случае запрос вроде «SELECT * FROM TABLE1 ROWS 20» вернет 20 первых записей, а не целую таблицу (которая может очень большой).

Список полей SELECT

Список полей содержит одно или более выражений, разделенных запятыми. Результатом каждого выражения является значение соответствующего поля в наборе данных команды SELECT (за исключением выражения «*» (звездочка), которое возвращает все поля отношения).

Синтаксис:

Руководство по языку SQL СУБД Firebird

```
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
                        | "или другие выражения возвращающие
                          единственное значение типа данных
                          Firebird или NULL"
```

qualifier ::= имя таблицы или алиас

collation ::= существующее имя сортировки (только для столбцов символьных типов)

Хорошим тоном является указание полного имени поля вместе с именем алиаса или таблицы/представления/хранимой процедуры, к которой это поле принадлежит.

Указание полного имени становится **обязательным** в случае, если поле с одним и тем же именем находится в более чем одной таблице, участвующей в объединении.

Обратите внимание, что алиасы заменяют оригинальное имя таблицы/представления/ хранимой процедуры: как только определен алиас для соответствующего отношения, использовать оригинальное нельзя.

В начало списка полей могут быть добавлены ключевые слова DISTINCT или ALL.

DISTINCT удаляет дубликаты строк: то есть, если две или более записей содержат одинаковые значения во всех соответствующих полях, только одна из этих строк будет включена в результирующий набор данных.

ALL включает все строки в результирующий набор данных. ALL включено по умолчанию и поэтому редко используется: явное указание поддерживается для совместимости со стандартом SQL.

Выражение COLLATE не изменяет содержимое поля, однако, если указать

Руководство по языку SQL СУБД Firebird

COLLATE для определенного поля, то это может изменить регистр символов или чувствительность поля к акцентам (accent sensitivity), что, в свою очередь, может повлиять на:

- Порядок сортировки, в случае если это поле указано в выражении ORDER BY
- Группировку, в случае если это поле указано в выражении GROUP BY
- Количество возвращаемых строк, если используется DISTINCT

Примеры операторов SELECT с различными типами полей

Простой SELECT использующий только имена полей:

```
select cust_id, cust_name, phone
from customers
where city = 'London'
```

Запрос с конкатенацией и вызовом функции в списке полей:

```
select
  'Mr./Mrs. ' || lastname,
  street,
  zip,
  upper(city)
from contacts
where date_last_purchase(id) = current_date
```

Запрос с двумя подзапросами:

```
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
```

Следующий запрос делает то же самое, что и предыдущий, только с использованием соединения таблиц (JOIN) вместо подзапросов:

```
select
  p.fullname,
```

Руководство по языку SQL СУБД Firebird

```
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
```

Этот запрос использует конструкцию CASE для определения корректного обращения, например, при рассылке сообщений конкретному человеку:

```
select  
  case upper(sex)  
    when 'F' then 'Mrs.'  
    when 'M' then 'Mr.'  
    else ''  
  end as title,  
  lastname,  
  address  
from employees
```

Запрос к хранимой процедуре:

```
select *  
from interesting_transactions(2010, 3, 'S')  
order by amount
```

Выборка полей производной таблицы. Производная таблица – это заключенный в скобки оператор SELECT, результат которого используется в запросе уровнем выше, как будто является обычной таблицей или представлением. Производная таблица выделена синим цветом.

```
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
```

Запрос к контекстной переменной (CURRENT_TIME):

Руководство по языку SQL СУБД Firebird

```
select current_time from rdb$database
```

Для тех, кто не знаком с RDB\$DATABASE: это системная таблица, которая всегда существует во всех базах данных Firebird и всегда содержит только одну строку. И, хотя эта таблица не была создана специально для этой цели, стало распространенной практикой среди разработчиков Firebird выполнять запросы к этой таблице в случае, если нужно выполнить запрос, не привязанный ни к какой таблице, в котором результат получается из выражений, указанных в списке полей оператора SELECT.

Например:

```
select
  power(12, 2) as twelve_squared,
  power(12, 3) as twelve_cubed
from rdb$database
```

И, наконец, пример запроса к самой таблице RDB\$DATABASE, с помощью которого можно получить кодировку по умолчанию данной БД:

```
select rdb$character_set_name from rdb$database
```

Выборка в переменные с помощью INTO

Доступно в PSQL

В PSQL (хранимых процедурах, триггерах и др.) результаты выборки команды SELECT могут быть построчно загружены в локальные переменные (число, порядок и типы локальных переменных должны соответствовать полям SELECT). Часто такая загрузка – единственный способ что-то сделать с возвращаемыми значениями.

Простой оператор SELECT может быть использован в PSQL только если он возвращает единственную строку, то есть, если это запрос типа синглтон (singleton). Для запросов, возвращающих несколько строк, PSQL предлагает использовать оператор [FOR SELECT](#).

Также, PSQL поддерживает оператор DECLARE CURSOR, который связывает именованный курсор с определенной командой SELECT – и этот курсор впоследствии может быть использован для навигации по возвращаемому набору данных.

В PSQL выражение INTO должно появляться в самом конце команды SELECT.

Синтаксис:

Руководство по языку SQL СУБД Firebird

```
SELECT [...] <column-list>
FROM ...
[...]
[INTO <variable-list>]

<variable-list> ::= [:]psqlvar [, [:]psqlvar ...]
```

Обратите внимание, что в PSQL двоеточие перед именами переменных является опциональным!

Примеры:

В PSQL, можно присвоить значения `min_amt`, `avg_amt` и `max_amt` заранее объявленным переменным или выходным параметрам:

```
select
  min(amount),
  avg(cast(amount as float)),
  max(amount)
from orders
where artno = 372218
into min_amt,
      avg_amt,
      max_amt;
```

В данном запросе CAST служит для корректного вычисления среднего значения. Если не привести значение к float, то среднее значение будет обрезано до ближайшего целого значения.

В триггере:

```
select list(name, ', ')
from persons p
where p.id in (new.father, new.mother)
into new.parentnames;
```

Выражение FROM

Выражение FROM определяет источники, из которых будут отображены данные. В простейшей форме, это может быть единственная таблица или представление. Однако источниками также могут быть хранимая процедура, производная таблица или общее табличное выражение (СТЕ). Различные виды источников могут комбинироваться с использованием разнообразных видов соединений (JOIN).

Этот раздел посвящен запрос из единственного источника. Объединения рассматриваются в следующем разделе.

Синтаксис:

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 ...]
```

Примечание

Если вы дадите таблице или представлению псевдоним (алиас), то вы должны везде использовать этот псевдоним, а не имя таблицы, при обращении к именам столбцов.

Корректное использование:

```
SELECT PEARS  
FROM FRUIT
```

```
SELECT FRUIT.PEARS  
FROM FRUIT
```

```
SELECT PEARS  
FROM FRUIT F
```

```
SELECT F.PEARS  
FROM FRUIT F
```

Некорректное использование:

```
SELECT FRUIT.PEARS  
FROM FRUIT F
```

Выборка из селективной хранимой процедуры

Селективная хранимая процедура (т.е. с возможностью выборки) должна удовлетворять следующим условиям:

- Содержать по крайней мере один выходной параметр
- Использовать ключевое слово **SUSPEND** таким образом, чтобы вызывающий запрос могут выбирать выходные строки одну за другой, также как выбираются строки таблицы или представления.

Выходные параметры селективной хранимой процедуры с точки зрения команды **SELECT** соответствуют полям обычной таблицы.

Выборка из хранимой процедуры без входных параметров осуществляется точно так же, как обычная выборка из таблицы:

```
select *  
from suspicious_transactions  
where assignee = 'Dmitrii'
```

Если хранимая процедура требует входные параметры, то они должны быть указаны в скобках после имени процедуры:

```
select name, az, alt  
from visible_stars('Brugge', current_date, '22:30')  
where alt >= 20  
order by az, alt
```

Значения для опциональных параметров (то есть, параметров, для которых определены значения по умолчанию) могут быть указаны или опущены.

Однако, если параметры задаются частично, то пропущенные параметры должны быть к концу перечисления внутри скобок.

Если предположить, что процедура `visible_stars` из предыдущего примера имеет два опциональных параметра `spectral_class` (`varchar(12)`) и `min_magn` (`numeric(3,1)`), то следующие команды будут корректными:

```
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)
```

Руководство по языку SQL СУБД Firebird

```
select name, az, alt
from visible_stars('Brugge', current_date, '22:30', 4.0)
```

А вот этот запрос не будет корректным:

```
select name, az, alt
from visible_stars('Brugge', current_date, 4.0)
```

Алиас для селективной хранимой процедуры указывается после списка параметров:

```
select
  number,
  (select name from contestants c
   where c.number = gw.number)
from get_winners('#34517', 'AMS') gw
```

Если вы указываете поле (выходной параметр) с полным именем процедуры, не включайте в это имя список параметров процедуры:

```
select number,
  (select name from contestants c
   where c.number = get_winners.number)
from get_winners('#34517', 'AMS')
```

Выборка из производной таблицы (derived table)

Производная таблица – это корректная команда SELECT, заключенная в круглые скобки, опционально обозначенная псевдонимом таблицы и псевдонимами полей.

Синтаксис:

```
(select-query)
[[AS] derived-table-alias]
[(<derived-column-aliases>)]

<derived-column-aliases> := column-alias [, column-alias ...]
```

Возвращаемый набор данных такого оператора представляет собой виртуальную таблицу, к которой можно составлять запросы, так как будто это обычная таблица.

Руководство по языку SQL СУБД Firebird

Производная таблица в запросе ниже выводит список имён таблиц в базе данных и количество столбцов в них. Запрос к производной таблице выводит количество полей, и количество таблиц с таким количеством полей.

```
SELECT
  FIELDCOUNT,
  COUNT (RELATION) AS NUM_TABLES
FROM (SELECT
  R.RDB$RELATION_NAME RELATION,
  COUNT (*) AS FIELDCOUNT
FROM RDB$RELATIONS R
JOIN RDB$RELATION_FIELDS RF
  ON RF.RDB$RELATION_NAME = R.RDB$RELATION_NAME
GROUP BY RELATION)
GROUP BY FIELDCOUNT
```

Тривиальный пример, демонстрирующий использование псевдонима производной таблицы и списка псевдонимов столбцов (оба опциональные):

```
SELECT
  DBINFO.DESCR, DBINFO.DEF_CHARSET
FROM (SELECT *
  FROM RDB$DATABASE) DBINFO
  (DESCR, REL_ID, SEC_CLASS, DEF_CHARSET)
```

Примечания:

- Производные таблицы могут быть вложенными;
- Производные таблицы могут быть объединениями и использоваться в объединениях. Они могут содержать агрегатные функции, подзапросы и соединения, и сами по себе могут быть использованы в агрегатных функциях, подзапросах и соединениях. Они также могут быть хранимыми процедурами или запросами из них. Они могут иметь предложения WHERE, ORDER BY и GROUP BY, указания FIRST, SKIP или ROWS и т.д.;
- Каждый столбец в производной таблице должен иметь имя. Если этого нет по своей природе (например, потому что это — константа), то надо в обычном порядке присвоить псевдоним или добавить список псевдонимов столбцов в спецификации производной таблицы;
- Список псевдонимов столбцов опциональный, но если он присутствует, то должен быть полным (т.е. он должен содержать псевдоним для каждого столбца производной таблицы);
- Оптимизатор может обрабатывать производные таблицы очень

Руководство по языку SQL СУБД Firebird

эффективно. Однако если производная таблица включена во внутреннее соединение и содержит подзапрос, то никакой порядок соединения не может быть использован оптимизатором.

Приведём пример того, как использование производных таблиц может упростить решение некоторой задачи.

Предположим, что у нас есть таблица COEFFS, которая содержит коэффициенты для ряда квадратных уравнений, которые мы собираемся решить. Она может быть определена примерно так:

```
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)  
)
```

В зависимости от значений коэффициентов a , b и c , каждое уравнение может иметь ноль, одно или два решения. Мы можем найти эти решения с помощью одноуровневого запроса к таблице COEFFS, однако код такого запроса будет громоздким, а некоторые значения (такие, как дискриминанты) будут вычисляться несколько раз в каждой строке.

Если использовать производную таблицу, то запрос можно сделать гораздо более элегантным:

```
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)
```

Если мы захотим показывать коэффициенты рядом с решениями уравнений, то мы можем модифицировать запрос следующим образом:

```
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)
```

Обратите внимание, что в первом запросе мы назначили алиасы для всех

полей производной таблицы в виде списка после таблицы, а во втором, по мере необходимости, добавляем алиасы внутри запроса производной таблицы. Оба этих метода корректны, так как при правильном применении гарантируют, что каждое поле производной таблицы имеет уникальное имя.

Выборка из общих табличных выражений (CTE)

Общие табличные выражения являются более сложной и более мощной вариацией производных таблиц. CTE состоят из преамбулы, начинающейся с ключевого слова WITH, которая определяет одно или более общих табличных выражений (каждое из которых может иметь список алиасов полей). Основным запросом, который следует за преамбулой, может обращаться к CTE так, как будто обычные таблицы. CTE доступны только в рамках основного запроса.

Подробно CTE описываются в разделе [Общие табличные выражения \("WITH ... AS ... SELECT"\)](#), а здесь приведены лишь некоторые примеры использования.

Следующий запрос представляет наш пример с производной таблицей в варианте для общих табличных выражений:

```
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
```

Это не слишком большое улучшение по сравнению с вариантом с производными таблицами (за исключением того, что вычисления проводятся до основного запроса). Мы можем еще улучшить запрос, исключив двойное вычисление `sqrt(D)` для каждой строки:

```
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  
)
```

Руководство по языку SQL СУБД Firebird

```
select
  iif (D >= 0, (-b - sqrtD) / denom, null) sol_1,
  iif (D > 0, (-b + sqrtD) / denom, null) sol_2
from vars2
```

Текст запроса выглядит более сложным, но он стал более эффективным (предполагая, что исполнение функции SQRT занимает больше времени, чем передача значений переменных b,d и denom через дополнительное CTE).

Конечно, мы могли бы добиться такого результата и с помощью производных таблиц, но это потребовало бы вложить запросы один в другой.

Соединения (JOINS)

Соединения объединяют данные из двух источников в один набор данных. Соединение данных осуществляется для каждой строки и обычно включает в себя проверку условия соединения (join condition) для того, чтобы определить, какие строки должны быть объединены и оказаться в результирующем наборе данных.

Результат соединения также может быть соединен с другим набором данных с помощью следующего соединения.

Существует несколько типов (INNER, OUTER) и классов (квалифицированные, натуральные, и др.) соединений, каждый из которых имеет свой синтаксис и правила.

Синтаксис:

```
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>

Руководство по языку SQL СУБД Firebird

```
| NATURAL [<join-type>] JOIN <source>  
| {CROSS JOIN | ,} <source>
```

<join-type> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]

<join-condition> ::= ON *condition* | USING (*column-list*)

Внутренние (INNER) и внешние (OUTER) соединения

Соединение всегда соединяет строки из двух наборов данных (которые обычно называются «левый» и «правый»). По умолчанию, только строки, которые удовлетворяют условию соединения (те, которым соответствует хотя бы одна строка из другого набора строк согласно применяемому условию) попадают в результирующий набор данных. Такой тип соединения (который является типом по умолчанию) называется внутренним (INNER JOIN).

Предположим, у нас есть 2 таблицы:

Таблица A:

ID	S
87	Just some text
235	Silence

Таблица B

CODE	X
-23	56.7735
87	416.0

Если мы соединим эти таблицы с помощью вот такого запроса:

```
select *  
from A  
join B on A.id = B.code
```

то результат будет:

Руководство по языку SQL СУБД Firebird

ID	S	CODE	X
87	Just some text	87	416.0

То есть, первая строка таблицы A была соединена со второй строкой таблицы B, потому что вместе они удовлетворяют условию соединения «A.id = B.code». Другие строки не имеют соответствия и поэтому не включаются в соединение. Помните, что умолчанию соединение всегда внутреннее (INNER).

Мы можем сделать это явным, указав тип соединения:

```
select *  
from A  
inner join B on A.id = B.code
```

но обычно слово inner опускается.

Разумеется, возможны случаи, когда строке в левом наборе данных соответствует несколько строк в правом наборе данных (или наоборот).

В таких случаях все комбинации включаются в результирующий набор данных, и мы можем получить результат вроде этого:

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

Иногда необходимо включить в результат все записи из левого или правого набора данных, вне зависимости от того, есть ли для них соответствующая запись в парном наборе данных. В этом случае необходимо использовать внешние соединения.

Внешнее левое соединение (LEFT OUTER) включает все записи из левого набора данных, и те записи из правого набора, которые удовлетворяют условию соединения.

Руководство по языку SQL СУБД Firebird

Внешнее правое соединение (RIGHT OUTER) включает все записи из правого набора данных и те записи из левого набора данных, которые удовлетворяют условию соединения.

Полное внешнее соединение (FULL OUTER) включает все записи из обоих наборов данных.

Во всех внешних соединениях, «дыры» (то есть поля набора данных, в котором нет соответствующей записи) заполняются NULL.

Для обозначения внешнего соединения используются ключевые слова LEFT, RIGHT или FULL с необязательным ключевым словом OUTER.

Рассмотрим различные внешние соединения на примере запросов с указанными выше таблицами A и B:

```
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

```
select *  
from A  
full [outer] join B on A.id = B.code
```

ID	S	CODE	X
----	---	------	---

Руководство по языку SQL СУБД Firebird

<null>	<null>	-23	53.7735
87	Just some text	87	416.0
235	Silence	<null>	<null>

Обычные соединения

Явный синтаксис соединения требует указания условия соединения записей. Это условие указывается явно в предложении ON или неявно при помощи предложения USING.

Синтаксис:

```
<qualified-join> ::=  
[<join-type>] JOIN <source> <join-condition>  
  
<join-type> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]  
  
<join-condition> ::= ON condition | USING (column-list)
```

Соединения с явными условиями

В синтаксисе явного соединения есть предложение ON, с условием соединения, в котором может быть указано любое логическое выражение, но, как правило, оно содержит условие сравнения между двумя участвующими источниками.

Довольно часто, это условие – проверка на равенство (или ряд проверок на равенство объединённых оператором AND) использующая оператор «=». Такие соединения называются эквисоединениями. (Примеры в главе Внутренние (INNER) и внешние (OUTER) соединения были эквисоединениями)

Примеры соединений с явными условиями:

```
/*  
* Выборка всех заказчиков из города Детройт, которые  
* сделали покупку.  
*/  
select *  
from customers c  
join sales s on s.cust_id = c.id  
where c.city = 'Detroit'
```

Руководство по языку SQL СУБД Firebird

```
/*
 * Тоже самое, но включает в выборку заказчиков, которые
 * не совершали покупки.
 */
select *
from customers c
left join sales s on s.cust_id = c.id
where c.city = 'Detroit'

/*
 * Для каждого мужчины выбрать женщин, которые выше него.
 * Мужчины, для которых такой женщины не существуют,
 * не будут выключены в выборку.
 */
select
  m.fullname as man,
  f.fullname as woman
from males m
join females f on f.height > m.height

/*
 * Выборка всех учеников, их класса и наставника.
 * Ученики без наставника будут включены в выборку.
 * Ученики без класса не будут включены в выборку.
 */
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
```

Соединения именованными столбцами

Эквисоединения часто сравнивают столбцы, которые имеют одно и то же имя в обеих таблицах. Для таких соединений мы можем использовать второй тип явных соединений, называемый соединения именованными столбцами (Named Columns Joins). Соединение именованными столбцами осуществляются с помощью предложения USING, в котором перечисляются только имена столбцов.

Руководство по языку SQL СУБД Firebird

Таким образом, следующий пример:

```
select *  
from flotsam f  
join jetsam j  
  on f.sea = j.sea  
  and f.ship = j.ship
```

можно переписать так:

```
select *  
from flotsam  
join jetsam using (sea, ship)
```

что значительно короче. Результирующий набор несколько отличается, по крайней мере, при использовании «SELECT *»:

- Результат соединения с явным условием соединения в предложении ON будет содержать каждый из столбцов SEA и SHIP дважды: один раз для таблицы FLOTSAM и один раз для таблицы JETSAM. Очевидно, что они будут иметь они и те же значения.
- Результат соединения именованными столбцами, с помощью предложения USING, будет содержать эти столбцы один раз.

Если вы хотите получить в результате соединения именованными столбцами все столбцы, перепишите запрос следующим образом:

```
select f.*, j.*  
from flotsam f  
join jetsam j using (sea, ship)
```

Для внешних (OUTER) соединений именованными столбцами, существуют дополнительные нюансы, при использовании «SELECT *» или неполного имени столбца:

Если столбец строки из одного источника не имеет совпадений со столбцом строки из другого источника, но в результат всё равно должен быть включен из за инструкций LEFT, RIGHT или FULL, то объединяемый столбец получит не NULL значение. Это достаточно справедливо, но теперь вы не можете сказать из какого набора левого, правого или обоих пришло это значение. Это особенно обманывает, когда значения пришли из правой части набора данных, потому что «*» всегда отображает для комбинированных столбцов значения из левой части набора данных, даже если используется RIGHT соединение.

Является ли это проблемой, зависит от ситуации. Если это так, используйте «a.*, b.*» подход продемонстрированный выше, где a и b имена или алиасы двух источников. Или лучше вообще избегать «*» в серьёзных запросах

Руководство по языку SQL СУБД Firebird

и перечислять все имена столбцов для соединяемых множеств. Такой подход имеет дополнительное преимущество, заставляя вас думать, о том какие данные вы хотите получить и откуда.

Вся ответственность за совместимость типов столбцов между соединяемыми источниками, имена которых перечислены в предложении USING, лежит на вас. Если типы совместимы, но не равны, то движок преобразует их в тип с более широким диапазоном значений перед сравнением. Кроме того, это будет типом данных объединённого столбца, который появится в результирующем наборе, если используются «SELECT *» или неполное имя столбца. Полные имена столбцов всегда будут сохранять свой первоначальный тип данных.

Естественные соединения (Natural Joins)

Взяв за основу соединения именованными столбцами, следующим шагом будет естественное соединение, которое выполняет эквисоединение по всем одноимённым столбцам правой и левой таблицы. Типы данных этих столбцов должны быть совместимыми.

Синтаксис:

```
<natural-join> ::= NATURAL [<join-type>] JOIN <source>
```

```
<join-type> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]
```

Даны две таблицы:

```
create table TA (  
    a bigint,  
    s varchar(12),  
    ins_date date  
);
```

```
create table TB (  
    a bigint,  
    descr varchar(12),  
    x float,  
    ins_date date  
);
```

Естественное соединение таблиц TA и TB будет происходить по столбцам

Руководство по языку SQL СУБД Firebird

а и ins_date и два следующих оператора дадут один и тот же результат:

```
select *
from TA
natural join TB;

select *
from TA
join TB using (a, ins_date);
```

Как и все соединения, естественные соединения являются внутренними соединениями по умолчанию, но вы можете превратить их во внешние соединения, указав LEFT, RIGHT или FULL перед ключевым словом JOIN.

Внимание:

Если в двух исходных таблицах не будут найдены одноимённые столбцы, то будет выполнен CROSS JOIN.

Замечание о равенстве

Оператор «=», который явно используется во многих условиях соединения и неявно в соединениях именованными столбцами и естественных соединениях, только при сравнении значений со значениями. В соответствии со стандартом SQL, NULL не является значением и, следовательно, два значения NULL не равны и ни не равны друг с другом. Если необходимо, чтобы значения NULL соответствовали друг другу при объединении, используйте оператор IS NOT DISTINCT FROM. Этот оператор возвращает истину, если операнды имеют то же значение, или, если оба они равны NULL.

```
select *
from A
join B on A.id is not distinct from B.code
```

Точно так же (крайне редко), в случае соединения по неравенству, используйте оператор IS DISTINCT FROM вместо оператора «<>», если вы хотите чтобы значения NULL отличались от любого значения и два значения NULL считались равными.

```
select *
from A
join B on A.id is distinct from B.code
```

Руководство по языку SQL СУБД Firebird

Это замечание об операторах равенства и неравенства применяется повсюду в СУБД Firebird, не только в условия соединения.

Перекрестное соединение (CROSS JOIN)

Перекрёстное соединение или декартово произведение. Каждая строка левой таблицы соединяется с каждой строкой правой таблицы.

Синтаксис:

```
<cross-join> ::= {CROSS JOIN | ,} <source>
```

Обратите внимание, что синтаксис с использованием запятой является устаревшим. Он поддерживается только для поддержания работоспособности унаследованного программного кода и может быть удалён в будущих версиях.

Перекрёстное соединение двух наборов эквивалентно их соединению по условию тавтологии (условие, которое всегда верно).

Следующие два запроса дадут один и тот же результат:

```
select *  
from TA  
cross join TB;
```

```
select *  
from TA  
join TB on 1 = 1;
```

Перекрестные соединения являются внутренними соединениями, потому что они отбирают строки, для которых есть соответствие – так уж случилось, что каждая строка соответствует! Внешнее перекрестное соединение, если бы оно существовало, ничего не добавило бы к результату, потому что внешние соединения добавляют записи, по которым нет соответствия, а они не существуют в перекрёстном соединении.

Перекрёстные соединения редко полезны, кроме случаев, когда вы хотите получить список всех возможных комбинаций двух или более переменных. Предположим, вы продаете продукт, который поставляется в различных размерах, различных цветов и из различных материалов. Если для каждой переменной значения перечислены в собственной таблице, то этот запрос будет возвращать все комбинации:

```
select
  m.name,
  s.size,
  c.name
from materials m
cross join sizes s
cross join colors c
```

Неоднозначные имена полей в соединениях

Firebird отвергает неполные имена полей в запросе, если эти имена полей существуют в более чем одном наборе данных, участвующих в объединении. Это также верно для внутренних эквисоединений, в которых имена полей фигурируют в предложении ON:

```
select a, b, c
from TA
join TB on TA.a = TB.a
```

Существует одно исключение из этого правила: соединения по именованным столбцам и естественные соединения, которые используют неполное имя поля в процессе подбора, могут использоваться законно. Это же относится и к одноимённым объединяемым столбцам. Для соединений по именованным столбцам эти столбцы должны быть перечислены в предложении USING. Для естественных соединений это столбцы имена которых присутствуют в обеих таблицах. Но снова замечу, что, особенно во внешних соединениях, плоское имя colname является не всегда тем же самым что left.colname или right.colname. Типы данных могут отличаться, и один из полных столбцов может иметь значение NULL, в то время как другой нет. В этом случае значение в объединённом, неполном столбце может замаскировать тот факт, что одно из исходных значений отсутствует.

Соединения с хранимыми процедурами

Если соединение происходит с хранимой процедурой, которая не коррелирована с другими потоками данных через входные параметры, то нет никаких особенностей.

В противном случае, есть одна неприятная особенность. Дело в том, что оптимизатор не в состоянии определить зависимость входных параметров процедуры от прочих потоков:

```
SELECT *
FROM MY_TAB
JOIN MY_PROC (MY_TAB.F) ON 1 = 1
```

Руководство по языку SQL СУБД Firebird

В данном случае процедура будет поставлена вперед, когда из таблицы MY_TAB еще не выбрана ни одна запись. Соответственно, на этапе исполнения будет выдана ошибка `isc_no_cur_rec` (no current record for fetch operation). Обходится данная проблема через явное указание порядка соединения синтаксисом:

```
SELECT *
FROM MY_TAB
LEFT JOIN MY_PROC (MY_TAB.F) ON 1 = 1
```

Теперь таблица всегда будет прочитана перед процедурой, и все будет работать корректно.

Примечание:

Эта ошибка оптимизатора будет исправлена в следующей версии сервера.

Предложение WHERE

Предложение WHERE предназначено для ограничения количества возвращаемых строк, теми которые нас интересуют. Условие после ключевого слова WHERE может быть простым, как проверка «AMOUNT = 3», так и сложным, запутанным выражением, содержащим подзапросы, предикаты, вызовы функций, математические и логические операторы, контекстные переменные и многое другое.

Условие в предложении WHERE часто называют условием поиска, выражением поиска или просто поиск.

В DSQL и ESQL, выражение поиска могут содержать параметры. Это полезно, если запрос должен быть повторен несколько раз с разными значениями входных параметров. В строке SQL запроса, передаваемого на сервер, вопросительные знаки используются как заполнители для параметров. Их называют позиционными параметрами, потому что они не могут сказать ничего кроме как о позиции в строке. Библиотеки доступа часто поддерживают именованные параметры в виде `:id`, `:amount`, `:a` и т.д. Это более удобно для пользователя, библиотека заботится о трансляции именованных параметров в позиционные параметры, прежде чем передать запрос на сервер.

Условие поиска может также содержать локальные (PSQL) или хост (ESQL) имена переменных, предваряемых двоеточием.

Синтаксис:

```
SELECT ...
```

Руководство по языку SQL СУБД Firebird

```
FROM ...  
[...]  
WHERE <search-condition>  
[...]
```

<search-condition> ::= логическое выражение возвращающее TRUE, FALSE и возможно UNKNOWN (NULL)

Только те строки, для которых условие поиска истинно будут включены в результирующий набор. Будьте осторожны с возможными получаемыми значениями NULL: если вы отрицаете выражение, дающее NULL с помощью NOT, то результат такого выражения всё равно будет NULL и строка не пройдёт. Это демонстрируется в одном из ниже приведённых примеров.

Примеры:

```
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');
```

Руководство по языку SQL СУБД Firebird

```
select sum (population)
from towns
where name like '%dam'
and province containing 'land';
```

```
select password
from usertable
where username = current_user;
```

Следующий пример показывает, что может быть, если условие поиска вычисляется как NULL.

Предположим у вас есть таблица, в которой находятся несколько детских имён и количество шариков, которыми они обладают.

CHILD	MARBLES
Anita	23
Bob E.	12
Chris	<null>
Deirdre	1
Eve	17
Fritz	0
Gerry	21
Hadassah	<null>
Isaac	6

Первое, обратите внимание на разницу между NULL и 0. Известно, что Fritz не имеет шариков вовсе, однако неизвестно количество шариков у Chris и Hadassah.

Теперь, если ввести этот SQL оператор:

```
select list(child) from marbletable where marbles > 10
```

вы получите имена Anita, Bob E., Eve и Gerry. Все эти дети имеют более чем 10 шариков.

Руководство по языку SQL СУБД Firebird

Если вы отрицаете выражение:

```
select list(child) from marbletable where not marbles > 10
```

запрос вернёт Deirdre, Fritz и Isaac. Chris и Hadassah не будут включены в выборку, так как не известно 10 у них шариков или меньше. Если вы измените последний запрос так:

```
select list(child) from marbletable where marbles <= 10
```

результат будет тем же самым, поскольку выражение NULL <= 10 даёт UNKNOWN. Это не тоже самое что TRUE, поэтому Chris и Hadassah не отображены. Если вы хотите что бы в списке были перечислены все «бедные» дети, то измените запрос следующим образом:

```
select list(child)  
from marbletable  
where marbles <= 10 or marbles is null
```

Теперь условие поиска становится истинным для Chris и Hadassah, потому что условие «marbles is null» возвращает TRUE в этом случае. Фактически, условие поиска не может быть NULL ни для одного из них.

Наконец, следующие два примера SELECT запросов с параметрами в условии поиска. Как определяются параметры запроса и возможно ли это, зависит от приложения. Обратите внимание, что запросы подобные этим не могут быть выполнены немедленно, они должны быть предварительно подготовлены. После того как параметризированный запрос был подготовлен, пользователь (или вызывающий код) может подставить значения параметров и выполнить его многократно, подставляя перед каждым вызовом новые значения параметров. Как вводятся значения параметров, и проходят ли они предобработку зависит от приложения. В GUI средах пользователь, как правило, вводит значения параметров через одно и более текстовых полей, и щёлкает на кнопку «Execute», «Run» или «Refresh».

```
select name, address, phone  
from stores  
where city = ? and class = ?
```

```
select *  
from pants  
where model = :model and size = :size and color = :col
```

Последний запрос не может быть передан непосредственно к движку сервера, приложение должно преобразовать его в другой формат, отображая именованные параметры на позиционные параметры.

Предложение GROUP BY

Предложение GROUP BY соединяет записи, имеющие одинаковую комбинацию значений полей, указанных в его списке, в одну запись. Агрегатные функции в списке выбора применяются к каждой группе индивидуально, а не для всего набора в целом.

Если список выборки содержит только агрегатные столбцы или столбцы, значения которых не зависят от отдельных строк основного множества, то предложение GROUP BY необязательно. Когда предложение GROUP BY опущено, результирующее множество будет состоять из одной строки (при условии, что хотя бы один агрегатный столбец присутствует).

Если в списке выборки содержатся как агрегатные столбцы, так и столбцы, чьи значения зависят от выбираемых строк, то предложение GROUP BY становится обязательным.

Синтаксис:

```
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
```

```
<non-aggr-expression> ::=
любые не агрегатные выражения, которые не
включены в список выборки, т.е. невыбираемые столбцы
из набора источника или выражения, которые
не зависят от набора данных вообще
```

Общее правило гласит, что каждый не агрегированный столбец в SELECT списке, должен быть так же включён в GROUP BY список. Вы можете это сделать тремя способами:

1. Копировать выражение дословно из списка выбора, например "class" или "D: || upper(doccode)".
2. Указать псевдоним, если он существует.

Руководство по языку SQL СУБД Firebird

3. Задать положение столбца в виде целого числа, которое находится в диапазоне от 1 до количества столбцов в списке SELECT. Целые значения, полученные из выражений, параметров или просто инварианты будут использоваться в качестве таковых в группировке. Они не будут иметь никакого эффекта, поскольку их значение одинаково для каждой строки.

Обратите внимание. Если вы группируете по позиции столбца, выражение соответствующее этой позиции будет скопировано из списка выборки SELECT. Это касается и подзапросов, таким образом, подзапрос будет выполняться, по крайней мере, два раза.

В дополнении к требуемым элементам, список группировки так же может содержать:

- Столбцы исходной таблицы, которые не включены в список выборки SELECT, или неагрегатные выражения, основанные на таких столбцах. Добавление таких столбцов может дополнительно разбить группы. Но так как эти столбцы не в списке выборки SELECT, вы не можете сказать, какому значению столбца соответствует значение агрегированной строки. Таким образом, если вы заинтересованы в этой информации, вы так же должны включить этот столбец или выражение в список выборки SELECT, что возвращает вас к правилу «каждый не агрегированный столбец в списке выборки SELECT должен быть включён в список группировки GROUP BY».
- Выражения, которые не зависят от данных из основного набора, т.е. константы, контекстные переменные, некоррелированные подзапросы, возвращающие единственное значение и т.д. Это упоминается только для полноты картины, т.к. добавление этих элементов является абсолютно бессмысленным, поскольку они не повлияют на группировку вообще. «Безвредные, но бесполезные» элементы так же могут фигурировать в списке выбора SELECT без их копирования в список группировки GROUP BY.

Примеры:

Когда в списке выбора SELECT содержатся только агрегатные столбцы, предложение GROUP BY необязательно:

```
select count (*) , avg (age)
from students
where sex = 'M'
```

Этот запрос вернет одну строку с указанием количества студентов мужского пола и их средний возраст. Добавление выражения, которое не

Руководство по языку SQL СУБД Firebird

зависит от строк таблицы STUDENTS ничего не меняет:

```
select count(*), avg(age), current_date
from students
where sex = 'M'
```

Теперь строка результата будет иметь дополнительный столбец, отображающий текущую дату, но кроме этого, ничего фундаментального не изменилось. Группировка по-прежнему не требуется.

Тем не менее, в обоих приведенных выше примерах это разрешено. Это совершенно справедливо и для запроса:

```
select count(*), avg(age)
from students
where sex = 'M'
group by class
```

и вернет результат для каждого класса, в котором есть мальчики, перечисляя количество мальчиков и их средний возраст в этой конкретном классе. (Если вы также оставите поле CURRENT_DATE, то это значение будет повторяться на каждой строке, что не интересно.)

Этот запрос имеет существенный недостаток, хотя он дает вам информацию о различных классах, но не говорит вам, какая строка к какому классу относится. Для того чтобы получить эту дополнительную часть информации, не агрегатный столбец CLASS должен быть добавлен в список выборки SELECT:

```
select class, count(*), avg(age)
from students
where sex = 'M'
group by class
```

Теперь у нас есть полезный запрос. Обратите внимание, что добавление столбца CLASS делает предложение GROUP BY обязательным. Мы не можем удалить это предложение, мы не можем так же удалить столбец CLASS из списка столбцов.

Результат последнего запроса будет выглядеть примерно так:

CLASS	COUNT	AVG
2A	12	13.5
2B	9	13.9

Руководство по языку SQL СУБД Firebird

3A	11	14.6
3B	12	14.4
...

Заголовки "COUNT" и "AVG" не очень информативны. В простейшем случае вы можете обойти это, но лучше, если мы дадим им значимые имена с помощью псевдонимов:

```
select
  class,
  count(*) as num_boys,
  avg(age) as boys_avg_age
from students
where sex = 'M'
group by class
```

Как вы помните из формального синтаксиса списка столбцов, ключевое слово AS не является обязательным.

Добавление большего не агрегированных (или точнее строчно зависимых) столбцов требуется добавления их в предложения GROUP BY тоже. Например, вы хотите видеть вышеуказанную информацию о девочках то же, и хотите видеть разницу между интернатами и студентами дневного отделения:

```
select
  class,
  sex,
  boarding_type,
  count(*) as number,
  avg(age) as avg_age
from students
group by class, sex, boarding_type
```

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

Руководство по языку SQL СУБД Firebird

2A	M	DAY	5	13.4
2B	F	BOARDING	11	13.7
2B	F	DAY	5	13.7
2B	M	BOARDING	6	13.8
...

Каждая строка в результирующем наборе соответствует одной конкретной комбинации переменных CLASS, SEX и BOARDING_TYPE. Агрегированные результаты – количество и средний возраст – приведены для каждой из конкретизированной группы отдельно. В результате запроса вы не можете увидеть обобщённые результаты для мальчиков отдельно или для студентов дневного отделения отдельно. Таким образом, вы должны найти компромисс. Чем больше вы добавляете неагрегатных столбцов, тем больше вы конкретизируете группы, и тем больше вы упускаете общую картину из виду. Конечно, вы все еще можете получить "большие" агрегаты, с помощью отдельных запросов.

HAVING

Так же, как и предложение WHERE ограничивает строки в наборе данных, теми которые удовлетворяют условию поиска, с той разницей, что предложение HAVING накладывает ограничения на агрегированные строки сгруппированного набора. Предложение HAVING не является обязательным и может быть использовано только в сочетании с предложением GROUP BY.

Условие(я) в предложении HAVING может ссылаться на:

- Любой агрегированный столбец в списке выбора SELECT. Это наиболее широко используемый случай.
- Любое агрегированное выражение, которое не находится в списке выбора SELECT, но разрешено в контексте запроса. Иногда это полезно.
- Любой столбец в списке GROUP BY. Однако, более эффективно фильтровать не агрегированные данные на более ранней стадии в предложении WHERE.
- Любое выражение, значение которого не зависит от содержимого набора данных (например, константа или контекстная переменная). Это допустимо, но совершенно бессмысленно, потому что такое условие, не имеющее никакого отношения к самому набору данных, либо подавит весь набор, либо оставит его не тронутым.

Руководство по языку SQL СУБД Firebird

Предложение HAVING не может содержать:

- Не агрегированные выражения столбца, которые не находятся в списке GROUP BY.
- Позицию столбца. Целое число в предложении HAVING – просто целое число.
- Псевдонимы столбца – даже, если они появляются в предложении GROUP BY!

Примеры:

Перестроим наши ранние примеры. Мы можем использовать предложение HAVING для исключения малых групп студентов:

```
select
  class,
  count(*) as num_boys,
  avg(age) as boys_avg_age
from students
where sex = 'M'
group by class
having count(*) >= 5
```

Выберем только группы, которые имеют минимальный разброс по возрасту 1,2 года:

```
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
```

Обратите внимание, что если вас действительно интересует эта информация, то неплохо бы включить в список выбора min(age) и max(age) или выражение max(age) – min(age).

Следующий запрос отбирает только учеников 3 класса:

```
select
```

Руководство по языку SQL СУБД Firebird

```
class,  
count(*) as num_boys,  
avg(age) as boys_avg_age  
from students  
where sex = 'M'  
group by class  
having class starting with '3'
```

Однако гораздо лучше переместить это условие в предложение WHERE:

```
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
```

Предложение PLAN

Предложение PLAN позволяет пользователю указать свой план выполнения запроса, который перекрывает тот план, который оптимизатор сгенерировал автоматически.

Синтаксис:

```
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>
```

Руководство по языку SQL СУБД Firebird

```
<basic-item> ::= <relation>
                {NATURAL
                | INDEX (<indexlist>)
                | ORDER index [INDEX (<indexlist>)]}
```

```
<relation> ::= table
            | view [table]
```

```
<indexlist> ::= index [, index ...]
```

```
table, view ::= имя или псевдоним (алиас)
```

Каждый раз, когда пользователь отправляет запрос движку Firebird, оптимизатор вычисляет стратегию извлечения данных. Большинство клиентов Firebird имеют возможность отобразить пользователю план извлечения данных. В собственном инструменте isql это делается с помощью команды SET PLAN ON. Если вы хотите только изучить план запроса без его выполнения, то вам необходимо ввести команду SET PLANONLY ON, после чего будут извлекаться планы запросов без их выполнения. Для возврата ISQL в режим выполнения запросов введите команду SET PLANONLY OFF.

В большинстве случаев, вы можете доверять тому, что Firebird выберет наиболее оптимальный план запроса. Однако если ваши запросы очень сложны и вам кажется, что они выполняются не эффективно, вам необходимо посмотреть план запроса и подумать можете ли вы улучшить его.

Простые планы

Простейшие планы состоят только из имени таблицы и следующим за ним метода извлечения. Например, для неотсортированной выборки из единственной таблицы без предложения WHERE:

```
select * from students
plan (students natural)
```

Если есть предложение WHERE вы можете указать индекс, который будет использоваться при нахождении совпадений:

```
select *
from students
where class = '3C'
plan (students index (ix_stud_class))
```

Директива INDEX может использоваться также для условий соединения (которые будут обсуждаться чуть позже). Она содержит список индексов, разделенных запятыми.

Руководство по языку SQL СУБД Firebird

Директива ORDER определяет индекс, который используется при сортировке набора данных, если присутствуют предложения ORDER BY или GROUP BY:

```
select *
from students
plan (students order pk_students)
order by id
```

Инструкции ORDER и INDEX могут быть объединены:

```
select *
from students
where class >= '3'
plan (students order pk_students index (ix_stud_class))
order by id
```

Так же совершенно нормально, если указать для инструкций ORDER и INDEX один и тот же индекс:

```
select *
from students
where class >= '3'
plan (students order ix_stud_class index (ix_stud_class))
order by class
```

Для сортировки наборов данных, когда невозможно использовать индекс (или вы хотите подавить его использование), уберите инструкцию ORDER и предварите выражение плана инструкцией SORT:

```
select *
from students
plan sort (students natural)
order by name
```

Или когда индекс используется для поиска:

```
select *
from students
where class >= '3'
plan sort (students index (ix_stud_class))
order by name
```

Обратите внимание, что инструкция SORT, в отличие от ORDER, находится за пределами скобок. Это отражает тот факт, что строки данных

Руководство по языку SQL СУБД Firebird

извлекаются несортированными и сортируются движком в последствии.

При выборке из представления указывается само представление и участвующее в нём таблица. Например, если у вас есть представление FRESHMEN, которое выбирает только студентов первокурсников:

```
select *  
from freshmen  
plan (freshmen students natural)
```

Или, например:

```
select *  
from freshmen  
where id > 10  
plan sort (freshmen students index (pk_students))  
order by name desc
```

Обратите внимание: если вы назначили псевдоним таблице или представлению, то в предложении PLAN необходимо использовать псевдоним, а не оригинальное имя.

Составные планы

Если вы делаете соединение, то вы можете указать индекс, который будет использоваться для сопоставления. Кроме того, вы должны использовать директиву JOIN для двух потоков в плане:

```
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))
```

То же самое соединение, отсортированное по индексированному столбцу:

```
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
```

И соединение, отсортированное не по индексированному столбцу:

```
select s.id, s.name, s.class, c.mentor  
from students s  
join classes c on c.name = s.class
```

Руководство по языку SQL СУБД Firebird

```
plan join (s order pk_students, c index (pk_classes))  
order by s.id
```

Соединение с добавленным условием поиска:

```
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
```

То же самое, но используется левое внешнее соединение:

```
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
```

Если нет доступных индексов для условия соединения (или вы не хотите его использовать), то план должен сначала отсортировать оба потока по соединяемым столбцам и затем произвести слияние. Это достигается с помощью директив SORT (которую вы уже встречали) и MERGE используемую вместо JOIN.

```
select *  
from students s  
join classes c on c.cookie = s.cookie  
plan merge (sort (c natural), sort (s natural))
```

Добавление предложения ORDER BY означает, что результат слияния также должен быть отсортирован:

```
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
```

И наконец, мы добавляем условие поиска на двух индексированных столбцах таблицы STUDENTS:

```
select *  
from students s  
join classes c on c.cookie = s.cookie
```

Руководство по языку SQL СУБД Firebird

```
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
```

Как следует из формального определения синтаксиса, JOIN и MERGE могут объединять в плане более двух потоков. Кроме того, каждое выражение плана может использоваться в качестве элемента в охватывающем плане. Это означает, что планы некоторых сложных запросов могут иметь различные уровни вложенности.

Наконец, вместо MERGE вы можете писать SORT MERGE. Поскольку это не имеет абсолютно никакого значения и может создать путаницу с «настоящей» директивой SORT (которая действительно имеет значение), то вероятно лучше придерживаться простой директивы MERGE.

UNION

Предложение UNION объединяет два и более набора данных, тем самым увеличивая общее количество строк, но не столбцов. Наборы данных, принимающие участие в UNION, должны иметь одинаковое количество столбцов. Однако, столбцы в соответствующих позициях не обязаны иметь один и тот же тип данных, они могут быть абсолютно не связанными.

По умолчанию, объединение подавляет дубликаты строк. UNION ALL отображает все строки, включая дубликаты. Необязательное ключевое слово DISTINCT делает поведение по умолчанию явным.

Синтаксис:

```
<union> ::= <individual-select>  
          UNION [DISTINCT | ALL]  
          <individual-select>  
          [UNION [DISTINCT | ALL]  
          <individual-select>  
          ...]  
          [<union-wide-clauses>]  
  
<individual-select> ::= SELECT  
                      [FIRST <m>] [SKIP <n>]  
                      [DISTINCT | ALL] <columns>  
                      FROM source [[AS] alias]  
                      [<joins>]  
                      [WHERE <condition>]
```

Руководство по языку SQL СУБД Firebird

```
[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>]
```

Объединения получают имена столбцов из первого запроса на выборку. Если вы хотите дать псевдонимы объединяемым столбцам, то делайте это для списка столбцов в самом верхнем запросе на выборку. Псевдонимы в других участвующих в объединении выборках разрешены, и могут быть даже полезными, но они не будут распространяться на уровне объединения.

Если объединение имеет предложение ORDER BY, то единственными возможными элементами сортировки являются целочисленные литералы, указывающие на позиции столбцов, необязательно сопровождаемые ASC/DESC и/или NULLS FIRST/LAST директивами. Это так же означает, что вы не можете упорядочить объединение ничем, что не является столбцом объединения. (Однако, вы можете завернуть его в производную таблицу, которая даст вам все обычные параметры сортировки.)

Объединения позволены в подзапросах любого вида и могут самостоятельно содержать подзапросы. Они также могут содержать соединения (joins), и могут принимать участие в соединениях, если завернуты в производную таблицу.

Примеры:

Этот запрос представляет информацию из различных музыкальных коллекций в одном наборе данных с помощью объединений:

```
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
```

Если id, title, artist и length – единственные поля во всех участвующих таблицах, то запрос может быть записан так:

Руководство по языку SQL СУБД Firebird

```
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
```

Квалификация “звезд” необходима здесь, потому что они не являются единственным элементом в списке столбцов. Заметьте, что псевдонимы “с” в первой и третьей выборке не кусают друг друга. Они не имеют контекста объединения, а лишь применяются к отдельным запросам на выборку.

Следующий запрос получает имена и телефонные номера переводчиков и корректоров. Те переводчики, которые также работают корректорами, будут отображены только один раз в результирующем наборе, если номера их телефонов одинаковые в обеих таблицах. Тот же результат может быть получен без ключевого слова DISTINCT. Если вместо ключевого слова DISTINCT, будет указано ключевое слово ALL, эти люди будут отображены дважды.

```
select name, phone
from translators
union distinct
select name, telephone
from proofreaders
```

Пример использования UNION в подзапросе:

```
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
```

Предложение ORDER BY

Результат выборки данных при выполнении оператора SELECT по умолчанию никак не упорядочивается (хотя довольно часто происходит упорядочение в хронологическом порядке помещения строк в таблицу операторами INSERT). Предложение ORDER BY позволяет задать необходимый порядок при выборке данных.

Синтаксис:

Руководство по языку SQL СУБД Firebird

```
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}]
```

В предложении через запятую перечисляются столбцы, по которым нужно упорядочить результирующий набор данных. Можно задавать имя столбца, псевдоним, присвоенный столбцу в списке выбора при помощи ключевого слова AS, или порядковый номер столбца в списке выбора. В одном предложении можно для разных столбцов смешивать форму записи. Например, один столбец в списке упорядочивания может быть задан своим именем, а другой порядковым номером.

Примечание

В случае сортировки по номеру столбца для запроса вида "SELECT " сервер раскрывает звёздочку (*) для определения сортируемых столбцов. Однако использование данной особенности в ваших запросах является плохой практикой.

Ключевое слово ASCENDING задает упорядочение по возрастанию значений. Допустимо сокращение ASC. Применяется по умолчанию.

Ключевое слово DESCENDING задает упорядочение по убыванию значений. Допустимо сокращение DESC. В одном предложении упорядочение по одному столбцу может идти по возрастанию значений, а по другому – по убыванию.

Ключевое слово COLLATE позволяет задать порядок сортировки строкового столбца, если нужен порядок, отличный от того, который был установлен для этого столбца (явно при описании столбца или по умолчанию, принятому для соответствующего набора символов).

Ключевое слово NULLS определяет, где в отсортированном наборе данных будут находиться значения NULL соответствующего столбца – в начале выборки (FIRST) или в конце (LAST). По умолчанию принимается NULLS FIRST.

Части выборки SELECT, участвующих в объединении UNION, не могут быть отсортированы с использованием предложения ORDER BY. Однако вы можете достичь желаемого результата с использованием производных таблиц или общих табличных выражений. Предложение ORDER BY, записанное последним в объединении, будет применено ко всей выборке в целом, а не к последней его части. Для объединений, единственными возможными элементами

Руководство по языку SQL СУБД Firebird

сортировки являются целочисленные литералы, указывающие на позиции столбцов, необязательно сопровождаемые ASC/DESC и/или NULLS FIRST/LAST директивами.

Примеры:

В описанном ниже запросе выборка будет отсортирована по возрастанию по столбцам RDB\$CHARACTER_SET_ID, RDB\$COLLATION_ID таблицы 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
```

То же самое, но сортировка производится по псевдонимам столбцов:

```
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
```

В следующем запросе производится сортировка, по номерам столбцов:

```
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
```

Как было выше сказано, такая сортировка тоже допустима, но не рекомендуется:

```
SELECT *
FROM RDB$COLLATIONS
ORDER BY 3, 2
```

В данном запросе сортировка происходит по второму столбцу таблицы BOOKS:

```
SELECT
  BOOKS.*,
  FILMS.DIRECTOR
```

Руководство по языку SQL СУБД Firebird

```
FROM BOOKS, FILMS  
ORDER BY 2
```

Предупреждение.

Обратите внимание на то, что выражения, результатом вычисления которых должны быть целые неотрицательные числа, будут интерпретироваться как номер столбца и вызовут исключение, если они не будут в диапазоне от 1 до числа столбцов.

Пример:

```
SELECT  
  X, Y, NOTE  
FROM PAIRS  
ORDER BY X+Y DESC
```

Примечания:

- Число, возвращаемое функцией или процедурой из UDF или хранимой процедуры, непредсказуемо, независимо от того, определена сортировка самим выражением или номером столбца;
- Только неотрицательные целые числа интерпретируются как номер столбца. Целое число, полученное однократным вычислением выражения или заменой параметра, запоминается как целочисленная постоянная величина, так как это значение одинаково для всех строк.

Сортировка по убыванию значений столбца PROCESS_TIME с размещением значений NULL в начале выборки:

```
SELECT *  
FROM MSG  
ORDER BY PROCESS_TIME DESC NULLS FIRST
```

Сортировка выборки полученной объединением выборок из двух запросов. Выборка сортируется по убыванию значений второго столбца с размещением NULL значений в конце списка и возрастанием значений первого столбца с размещением NULL значений в начале списка.

```
SELECT  
  DOC_NUMBER, DOC_DATE  
FROM PAYORDER  
UNION ALL  
SELECT
```

Руководство по языку SQL СУБД Firebird

```
DOC_NUMBER, DOC_DATE
FROM BUDGORDER
ORDER BY 2 DESC NULLS LAST, 1 ASC NULLS FIRST
```

WITH LOCK

Доступно: DSQL, PSQL

Опция WITH LOCK, обеспечивает возможность ограниченной явной пессимистической блокировки для осторожного использования в затронутых наборах строк:

- a) крайне малой выборки (в идеале из одной строки) и
- b) при контроле из приложения.

Пессимистическая блокировка редко требуется при работе с Firebird. Эту функцию можно использовать только хорошо понимая её последствия. Требуется хорошее знание различных уровней изоляции транзакции. Предложение WITH LOCK доступно для использования в DSQL и PSQL и только для выборки данных (SELECT) из одной таблицы. Предложение WITH LOCK нельзя использовать:

- в подзапросах;
- в запросах с объединением нескольких таблиц (JOIN);
- с оператором DISTINCT, предложением GROUP BY и при использовании любых агрегатных функций;
- при работе с представлениями;
- при выборке данных из селективных хранимых процедур;
- при работе с внешними таблицами.

Синтаксис:

```
SELECT ...
FROM single_table
[WHERE ...]
[FOR UPDATE [OF <column-names>]]
[WITH LOCK]
```

При успешном выполнении предложения WITH LOCK будут заблокированы выбранные строки данных и таким образом запрещён доступ на их изменение в рамках других транзакций до момента завершения Вашей транзакции.

При выборке с использованием предложения FOR UPDATE блокировка применяется к каждой строке, одна за другой, по мере попадания выборки в кэш сервера. Это делает возможным ситуацию, в которой успешная блокировка

Руководство по языку SQL СУБД Firebird

данных *перестает работать* при достижении в выборке строки, заблокированной другой транзакцией.

Сервер, в свою очередь, для каждой записи, подпадающей под явную блокировку, возвращает версию записи, которая является в настоящее время подтвержденной (актуальной), независимо от состояния базы данных, когда был выполнен оператор выборки данных, или исключение при попытке обновления заблокированной записи.

Ожидаемое поведение и сообщения о конфликте зависят от параметров транзакции, определенных в блоке TPB:

Влияние параметров TPB на явную блокировку

Режим TPB	Поведение
isc_tpb_consistency	Явные блокировки переопределяются неявными или явными блокировками табличного уровня и игнорируются
isc_tpb_concurrency + isc_tpb_nowait	При подтверждении изменения записи в транзакции, стартовавшей после транзакции, запустившей явную блокировку, немедленно возникает исключение конфликта обновления
isc_tpb_concurrency + isc_tpb_wait	При подтверждении изменения записи в транзакции, стартовавшей после транзакции, запустившей явную блокировку, немедленно возникает исключение конфликта обновления. Если в активной транзакции идет редактирование записи (с использованием явной блокировки или нормальной оптимистической блокировкой записи), то транзакция, делающая попытку явной блокировки, ожидает окончания транзакции блокирования и, после её завершения, снова пытается получить блокировку записи. Это означает что при изменении версии записи и подтверждении транзакции с блокировкой возникает исключение конфликта обновления.
isc_tpb_read_committed + isc_tpb_nowait	Если есть активная транзакция, редактирующая запись (с использованием явной блокировки или нормальной оптимистической блокировкой записи), то сразу же возникает исключение конфликта обновления.
isc_tpb_read_committed	Если в активной транзакции идет редактирование записи (с использованием явной блокировки или нормальной оптимистической блокировкой записи), то транзакция,

Руководство по языку SQL СУБД Firebird

+ isc_tpb_wait	делающая попытку явной блокировки, ожидает окончания транзакции блокирования и, после её завершения, снова пытается получить блокировку записи. Для этого режима TPВ никогда не возникает конфликта обновления.
----------------	--

Как сервер работает с WITH LOCK

Попытка редактирования записи с помощью оператора UPDATE, заблокированной другой транзакцией, приводит к вызову исключения конфликта обновления или ожиданию завершения блокирующей транзакции - в зависимости от режима TPВ. Поведение сервера здесь такое же, как если бы эта запись уже была изменена блокирующей транзакцией.

Нет никаких специальных кодов gdscode, возвращаемых для конфликтов обновления, связанных с пессимистической блокировкой.

Сервер гарантирует, что все записи, возвращенные явным оператором блокировки, фактически заблокированы и действительно соответствуют условиям поиска, заданным в операторе WHERE, если эти условия не зависят ни от каких других таблиц, не имеется операторов соединения, подзапросов и т.п. Это также гарантирует то, что строки, не попадающие под условия поиска, не будут заблокированы. Это не даёт гарантии, что нет строк, которые попадают под условия поиска, и не заблокированы.

Примечание

Такая ситуация может возникнуть, если в другой, параллельной транзакции подтверждаются изменения в процессе выполнения текущего оператора блокировки.

Сервер блокирует строки по мере их выборки. Это имеет важные последствия, если Вы блокируете сразу несколько строк. Многие методы доступа к базам данных Firebird по умолчанию используют для выборки данных пакеты из нескольких сотен строк (так называемый "буфер выборки"). Большинство компонентов доступа к данным не выделяют строки, содержащиеся в последнем принятом пакете, и для которых произошёл конфликт обновления.

Оptionальное предложение "OF <column-names>"

Предложение FOR UPDATE обеспечивает метод, позволяющий

предотвратить использование буферизованных выборок, с помощью подпункта “OF <column-names>”, что включает позиционированное обновление.

Совет

Кроме того, некоторые компоненты доступа позволяют установить размер буфера выборки и уменьшить его до 1 записи. Это позволяет Вам заблокировать и редактировать строку до выборки и блокировки следующей или обрабатывать ошибки, не отменяя действий Вашей транзакции.

Предостережения при использовании WITH LOCK

- Откат неявной или явной точки сохранения отменяет блокировку записей, которые изменялись в рамках её действий, но ожидающие окончания блокировки транзакции при этом не уведомляются. Приложения не должны зависеть от такого поведения, поскольку в будущем оно может быть изменено;
- Хотя явные блокировки могут использоваться для предотвращения и/или обработки необычных ошибок конфликтов обновления, объем ошибок обновления (deadlock) вырастет, если Вы тщательно не разработаете свою стратегию блокировки и не будете ей строго управлять;
- Большинство приложений не требуют явной блокировки записей. Основными целями явной блокировки являются: 1) предотвращение дорогостоящей обработки ошибок конфликта обновления в сильно загруженных приложениях и 2) для поддержания целостности объектов, отображаемых из реляционной базы данных в кластеризируемой среде. Если использование Вами явной блокировки не подпадает под одну из этих двух категорий, то это является неправильным способом решения задач в Firebird;
- Явная блокировка — это расширенная функция; не злоупотребляйте её использованием! В то время как явная блокировка может быть очень важной для веб-сайтов, обрабатывающих тысячи параллельных пишущих транзакций или для систем типа ERP/CRM, работающих в крупных корпорациях, большинство прикладных программ не требуют её использования.

Примеры использования явной блокировки:

Руководство по языку SQL СУБД Firebird

1. Одиночная:

```
SELECT *  
FROM DOCUMENT  
WHERE DOCUMENT_ID=? WITH LOCK
```

2. Несколько строк с последовательной их обработкой с курсором DSQL:

```
SELECT *  
FROM DOCUMENT  
WHERE PARENT_ID=?  
FOR UPDATE WITH LOCK
```

Общие табличные выражения (“WITH ... AS ... SELECT”)

Доступно: DSQL, PSQL

Описание: Общие табличные выражения (Common Table Expressions), сокращённо CTE, могут быть описаны как виртуальные таблицы или представления, определенных в преамбуле основного запроса, которые участвуют в основном запросе. Основной запрос может ссылаться на любое CTE из определенных в преамбуле, как и при выборке данных из обычных таблиц или представлений. CTE могут быть рекурсивными, т.е. ссылающимися сами на себя, но не могут быть вложенными.

Синтаксис:

```
<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 ...]  
  
<cte-stmt> ::= любой оператор SELECT или UNION
```

Руководство по языку SQL СУБД Firebird

<main-query> ::= основной оператор SELECT, который может ссылаться на любое CTE из найденных в преамбуле.

Пример:

```
WITH DEPT_YEAR_BUDGET AS (  
  SELECT  
    FISCAL_YEAR,  
    DEPT_NO,  
    SUM(PROJECTED_BUDGET) BUDGET  
  FROM PROJ_DEPT_BUDGET  
  GROUP BY FISCAL_YEAR, DEPT_NO  
)  
SELECT  
  D.DEPT_NO,  
  D.DEPARTMENT,  
  DYB_2008.BUDGET 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)
```

Примечания:

- Определение CTE может содержать любые конструкции синтаксиса оператора SELECT, пока не закончится преамбула "WITH ...» (операторы WITH не могут быть вложенными);
- CTE могут использовать друг друга, но ссылки не должны иметь циклов;
- CTE могут быть использованы в любой части главного запроса или другого табличного выражения и сколько угодно раз;
- Основной запрос может ссылаться на CTE несколько раз, но с разными алиасами;
- CTE могут быть использованы в операторах INSERT, UPDATE и DELETE как подзапросы;

Руководство по языку SQL СУБД Firebird

- Если CTE объявлен, то он должен быть обязательно использован;
- CTE могут быть использованы и в PSQL (FOR WITH ... SELECT ... INTO ...):

```
FOR
  WITH MY_RIVERS AS (
    SELECT *
    FROM RIVERS
    WHERE OWNER = 'me')
  SELECT
    NAME,
    LENGTH
  FROM MY_RIVERS
  INTO :RNAME,
       :RLEN
DO
BEGIN
  ...
END
```

Рекурсивные CTE

Рекурсивное (ссылающееся само на себя) CTE это ОБЪЕДИНЕНИЕ, у которого должен быть по крайней мере один не рекурсивный элемент, к которому привязываются остальные элементы объединения. Не рекурсивный элемент помещается в CTE первым. Рекурсивные члены отделяются от не рекурсивных и друг от друга с помощью UNION ALL. Объединение не рекурсивных элементов может быть любого типа.

Рекурсивное CTE требует наличия ключевого слова RECURSIVE справа от WITH. Каждый рекурсивный член объединения может сослаться на себя только один раз и это должно быть сделано в предложении FROM.

Главным преимуществом рекурсивных CTE является то, что они используют гораздо меньше памяти и процессорного времени, чем эквивалентные рекурсивные хранимые процедуры.

Выполнение рекурсивного CTE с точки зрения сервера Firebird можно описать следующим образом:

- Сервер начинает выполнение с не рекурсивного члена;
- Для каждой выбранной строки он начинает выполнять каждый рекурсивный элемент один за другим, используя текущие значения из не рекурсивного члена как параметры;
- Если во время выполнения экземпляр рекурсивного элемента не выдаёт строк, цикл выполнения переходит на предыдущий уровень и получает

Руководство по языку SQL СУБД Firebird

следующую строку от внешнего для него набора данных.

Важно отметить, что если CTE объявлена, то где-то ниже она обязательно должна быть использована, иначе будет ошибка вида «CTE "AAA" is not used in query».

Пример рекурсивного CTE:

```
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
```

Руководство по языку SQL СУБД Firebird

```
(DYB_2009.FISCAL_YEAR = 2009)
```

Следующий пример выводит родословную лошади. Основное отличие состоит в том, что рекурсия идёт сразу по двум веткам дерева родословной.

```
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  
    -- ограничение глубины рекурсии  
    PEDIGREE.DEPTH < :MAX_DEPTH  
UNION ALL  
SELECT  
    HORSE.CODE_HORSE,  
    HORSE.CODE_FATHER,  
    HORSE.CODE_MOTHER,  
    HORSE.NAME,
```

Руководство по языку SQL СУБД Firebird

```
'M' || PEDIGREE.MARK,  
PEDIGREE.DEPTH + 1  
FROM  
HORSE  
JOIN PEDIGREE  
  ON HORSE.CODE_HORSE = PEDIGREE.CODE_MOTHER  
WHERE  
  -- ограничение глубины рекурсии  
  PEDIGREE.DEPTH < :MAX_DEPTH  
)  
SELECT  
  CODE_HORSE,  
  NAME,  
  MARK,  
  DEPTH  
FROM  
  PEDIGREE
```

Примечания для рекурсивного CTE:

- В рекурсивных членах объединения не разрешается использовать агрегаты (DISTINCT, GROUP BY, HAVING) и агрегатные функции (SUM, COUNT, MAX и т.п.);
- Рекурсивная ссылка не может быть участником внешнего объединения OUTER JOIN;
- Максимальная глубина рекурсии составляет 1024.
- Рекурсивный член не может быть представлен в виде производной таблицы

CASE

Доступно: DSQL, PSQL

Описание: Оператор CASE возвращает только одно значение из нескольких возможных. Есть два синтаксических варианта:

- Простой CASE, сравнимый с Pascal case или C switch;
- Поисковый CASE, который работает как серия операторов “if ... else if ... else if”.

Простой CASE

Синтаксис:

```
CASE <test-expr>
```

Руководство по языку SQL СУБД Firebird

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

При использовании этого варианта <test-expr> сравнивается с <expr> 1, <expr> 2 и т.д. до тех пор, пока не будет найдено совпадение, и тогда возвращается соответствующий результат. Если совпадений не найдено, то возвращается defaultresult из ветви ELSE. Если нет совпадений и ветвь ELSE отсутствует, возвращается значение NULL.

Совпадение эквивалентно оператору «=», т.е. если <test-expr> имеет значение NULL, то он не соответствует ни одному из <expr>, даже тем, которые имеют значение NULL.

Полученные результаты не должны быть буквальным значением: они могут быть полями или именами переменных, сложными выражениями, или иметь значение NULL.

Сокращённый вид простого оператора CASE используется в функции DECODE, доступной начиная с версии Firebird 2.1.

Пример:

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

Поисковый CASE

Синтаксис:

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

Здесь <bool_expr> выражение, которое даёт тройной логический результат:

Руководство по языку SQL СУБД Firebird

TRUE, FALSE или NULL. Первое выражение, возвращающее TRUE, определяет результат. Если нет выражений, возвращающих TRUE, то в качестве результата берётся defaultresult из ветви ELSE. Если нет выражений, возвращающих TRUE, и ветвь ELSE отсутствует, результатом будет NULL.

Как и в простом операторе CASE, результаты не должны быть буквальным значением: они могут быть полями или именами переменных, сложными выражениями, или иметь значение NULL.

Пример:

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

UPDATE

Доступно: DSQL, ESQL, PSQL

Описание:

Изменяет значения в одной таблице или в большем количестве таблиц в представлении. Изменяемые столбцы указываются в части SET; для ограничения набора изменяемых строк используются WHERE и/или ROWS. Если WHERE или ROWS отсутствуют, то обновляются все записи таблицы.

Синтаксис:

```
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 <values> [INTO <variables>]]
```

target ::= Таблица или обновляемое представление

<m>, <n> ::= Любое выражение типа integer.

<values> ::= value_expression [, value_expression ...]

<variables> ::= [:]varname [, [:]varname ...]

Ограничения:

- WHERE CURRENT OF используется только в PSQL, т.к. в DSQL нет оператора DSQL для создания курсора.
- Один столбец может быть использован только один раз в конструкции SET
- Подвыражение "INTO <variables>" доступно только в PSQL
- У столбцов, возвращаемых в контекстные переменные NEW в триггере, перед именем не должно использоваться двоеточие.

Использование псевдонимов

Если вы назначаете псевдоним таблице или представлению, вы обязаны использовать псевдоним для уточнения столбцов таблицы.

Примеры

Правильно:

Руководство по языку SQL СУБД Firebird

```
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 ...
```

Неправильно:

```
update Fruit F set Fruit.soort = 'pisang' where ...
```

SET

Указывает значения, присваиваемые столбцам. Столбцы и их значения перечисляются через запятую. Слева имя столбца, и справа значение или выражение.

Строковые литералы могут предваряться именем набора символов, для того чтобы движок понимал, как интерпретировать данные.

Примеры:

```
update addresses
set city = 'Saint Petersburg', citycode = 'PET'
where city = 'Leningrad'
```

```
update employees
set salary = 2.5 * salary
where title = 'CEO'
```

```
update People
set name = _ISO8859_1 'Hans-Jörg Schäfer'
-- обратите внимание на префикс '_'
where id = 53662
```

Вполне нормально использовать имена столбцов в выражениях справа. При этом, использоваться будет всегда *старое* значение столбца, даже если присваивание этому столбцу уже произошло ранее в перечислении SET. Вот пример

Данные в таблице TSET:

```
A B
----
1 0
```

Руководство по языку SQL СУБД Firebird

2 0

Оператор

```
update tset set a = 5, b = a
```

Поменяет значения на

```
A B
----
5 1
5 2
```

Обратите внимание, что старые значения (1 и 2) используются для обновления столбца b, даже после того как столбцу a было назначено новое значение (5).

Замечание

Так было не всегда. До версии 2.5, столбцы сразу получали новые значения. Это являлось нестандартным поведением, и поэтому было изменено в версии 2.5.

Однако, для восстановления совместимости со старыми версиями в *firebird.conf* существует параметр *OldSetClauseSemantics*, который, будучи установленным в 1, восстанавливает старое поведение. Этот параметр в будущем будет удален.

WHERE

Ограничивает набор обновляемых записей заданным условием, или – в PSQL – до текущей строки именованного курсора.

Примеры

```
update People
set firstname = 'Boris'
where lastname = 'Johnson'
```

ORDER BY и ROWS

Имеют смысл только вместе. Однако, их можно использовать по отдельности.

Руководство по языку SQL СУБД Firebird

При одном аргументе m , ROWS ограничивает update первыми m строками.

При двух аргументах m и n , ROWS ограничивает update до строк от m до n включительно. Оба аргумента – целочисленные, и начинаются с 1.

Пример

```
-- дать надбавку 20ти сотрудникам с наименьшей зарплатой
update employees
set salary = salary + 50
order by salary asc
rows 20
```

При использовании ROWS с одним аргументом:

- Если $m >$ количества обрабатываемых строк, то обновляется весь набор строк
- Если $m = 0$, ни одна строка не обновляется
- Если $m < 0$, выдается ошибка

При использовании ROWS с двумя аргументами:

- Если $m >$ количества обрабатываемых строк, ни одна строка не обновляется
- Если n больше количества строк, то обновляются строки от m до конца набора
- Если $m < 1$ или $n < 1$, выдается ошибка
- Если $n = m - 1$, ни одна строка не обновляется
- Если $n < m - 1$, выдается ошибка

RETURNING

Оператор UPDATE, обновляющий как минимум одну строку, может включать RETURNING для возврата значений из обновляемой строки. В RETURNING могут включаться любые строки, необязательно только те, которые обновляются.

Возвращаемые значения содержат изменения, произведенные в триггерах BEFORE, но не в триггерах AFTER. OLD.fieldname и NEW.fieldname могут быть использованы в качестве имен столбцов. Если OLD. Или NEW. не указано, возвращаются новые значения столбцов (NEW.).

Пример:

```
update Scholars
set firstname = 'Hugh', lastname = 'Pickering'
where firstname = 'Henry' and lastname = 'Higgins'
returning id, old.lastname, new.lastname
```

Руководство по языку SQL СУБД Firebird

В DSQL оператор с RETURNING всегда возвращает только одну строку. Если записи не были обновлены оператором, то возвращаемые значения содержат NULL. Это поведение может быть изменено в последующих версиях Firebird.

В PSQL, если записи не было обновлены, ничего не возвращается, и переменные, указанные в RETURNING, сохраняют свои прежние значения.

Обновление столбцов BLOB

Обновление столбцов BLOB всегда полностью меняет их содержимое. Даже идентификатор BLOB (ID), который является ссылкой на данные BLOB и хранится в столбце, меняется. BLOB могут быть изменены, если

1. Клиентское приложение меняет BLOB посредством Firebird API. В этом случае все зависит от приложения, и не рассматривается в этом руководстве.
2. При конкатенации двух строк (не BLOB со строкой) новое строковое значение не может превышать 32767 байт.
3. Если источником данных является столбец BLOB или выражение, возвращающее BLOB

INSERT

Доступно в DSQL, PSQL

Описание: Добавляет строки в таблицу, или в одну или более таблиц представления. Если значения столбцов указаны в разделе VALUES, то будет вставлена одна строка. Значения столбцов также могут быть получены из оператора SELECT, в этом случае может быть вставлено от нуля и более строк. В случае DEFAULT VALUES, значения можно не указывать, и вставлена будет одна строка.

Синтаксис:

```
INSERT
  INTO {tablename | viewname}
  {DEFAULT VALUES | [(column_list)] <value_source>}
  [RETURNING <value_list> [INTO <variables>]]

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

<value_source> ::= VALUES (<value_list>) | <select_stmt>

<value_list> ::= value_expression [, value_expression ...]

<variables> ::= [:]varname [, [:]varname ...]
```

Руководство по языку SQL СУБД Firebird

`<select_stmt>` ::= оператор SELECT или UNION, результат которого соответствует вставляемым столбцам

Ограничения

- У столбцов, возвращаемых в контекстные переменные NEW в триггере, перед именем не должно использоваться двоеточие.
- Один столбец не может быть указан более одного раза в списке столбцов.

UPDATE OR INSERT

Доступно в DSQL, PSQL

UPDATE OR INSERT вставляет или обновляет одну или более существующих записей. Производимое действие зависит от значений столбцов в конструкции MATCHING (или, если она не указана, то от значений столбцов первичного ключа - PK). Если найдены записи, совпадающие с указанными значениями, то они обновляются. Если нет, то вставляется новая запись.

Совпадением считается полное совпадение значений столбцов MATCHING или PK. Совпадение проверяется с использованием IS NOT DISTINCT, поэтому null совпадает с null.

Синтаксис:

```
UPDATE OR INSERT INTO {tablename | viewname} [(<columns>)]  
VALUES (<values>)  
[MATCHING (<columns>)]  
[RETURNING <values> [INTO <variables>]]
```

```
<columns> ::= colname [, colname ...]
```

```
<values> ::= value [, value ...]
```

```
<variables> ::= [:]varname [, [:]varname ...]
```

Ограничения

- Если у таблицы нет первичного ключа, то указание MATCHING считается обязательным.
- Если список MATCHING совпадает со списком столбцов update/insert, то каждый столбец может быть упомянут только один раз.
- Конструкция "INTO `<variables>`" доступна только в PSQL
- У столбцов, возвращаемых в контекстные переменные NEW в триггере, перед именем не должно использоваться двоеточие.

Руководство по языку SQL СУБД Firebird

Пример:

```
update or insert into Cows (Name, Number, Location)
values ('Suzy Creamcheese', 3278823, 'Green Pastures')
matching (Number)
returning rec_id into :id;
```

RETURNING

Если указано, то может содержать любые столбцы, указанные в операторе, или другие столбцы и выражения. Возвращаемые значения содержат все изменения, произведенные в триггерах BEFORE, но не в триггерах AFTER. *OLD.fieldname* и *NEW.fieldname* могут быть использованы в качестве возвращаемых значений. Для обычных имен столбцов возвращаются новые значения.

В DSQL, оператор с RETURNING всегда возвращает только одну строку. Если указан RETURNING, и обнаружено более одной совпадающей строки, выдается сообщение об ошибке. Это поведение может быть изменено в последующих версиях Firebird.

MERGE

Доступно в DSQL, PSQL

Производит слияние записей источника в целевую таблицу (или обновляемое представление). Источником может быть таблица, представление, или «все то, к чему вы можете выполнить SELECT». Каждая запись источника используется для обновления одной или более записей цели, или вставки записи в целевую таблицу. Условие обычно содержит сравнение столбцов в таблицах источника и цели.

Синтаксис:

```
MERGE INTO target [[AS] target-alias]
USING source [[AS] source-alias]
ON condition
WHEN MATCHED THEN
    UPDATE SET colname = value [, colname = value ...]
WHEN NOT MATCHED THEN
    INSERT [(columns)] VALUES (values)
```

target ::= a table or updatable view

*source ::= a table, GTT, view, selectable SP,
derived table or CTE*

Руководство по языку SQL СУБД Firebird

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

<values> ::= value [, value ...]

Замечание: Допускается использование только одного WHEN

Примеры

```
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)
```

Замечание

WHEN NOT MATCHED должно быть видимо с точки зрения *источника* (таблицы, которая указана в USING). Так сделано потому, что если запись источника не имеет совпадения с записью цели, то выполняется INSERT. Разумеется, если записи в цели не соответствует запись в источнике, то никакие действия не производятся.

Предупреждение

Если условие WHEN MATCHED присутствует, и несколько записей совпадают с записями в целевой таблице, UPDATE выполнится для всех совпадающих записей источника, и каждое последующее обновление перезапишет предыдущее. Это нестандартное поведение: стандарт SQL-2003 требует, чтобы в этой ситуации выдавалось исключение (ошибка).

DELETE

Доступно в DSQL, PSQL

Описание. DELETE удаляет строки из таблицы или из одной и более таблиц представления. Условия WHERE и ROWS могут ограничить количество

Руководство по языку SQL СУБД Firebird

удаляемых строк. Если не указано WHERE или ROWS, DELETE удалит все столбцы таблицы.

Синтаксис:

```
DELETE
FROM {tablename | viewname} [[AS] alias]
[WHERE {search-conditions | CURRENT OF cursorname}]
[PLAN plan_items]
[ORDER BY sort_items]
[ROWS <m> [TO <n>]]
[RETURNING <values> [INTO <variables>]]
```

<m>, <n> ::= Любое целочисленное (integer) выражение.

<values> ::= value_expression [, value_expression ...]

<variables> ::= [:]varname [, [:]varname ...]

Ограничения

- В чистом DSQL выражение WHERE CURRENT OF не имеет смысла, т.к. в DSQL нет оператора для создания курсора.
- Конструкция "INTO <variables>" доступна только в PSQL
- У столбцов, возвращаемых в контекстные переменные NEW в триггере, перед именем не должно использоваться двоеточие.

Псевдонимы

Псевдоним представляет таблицу во всем операторе. Как только алиас указан, он должен использоваться для всех столбцов, или не использоваться полностью.

Примеры:

Правильно:

```
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'
```

Неправильно:

```
delete from Cities C where Cities.name starting 'Alex'
```

Руководство по языку SQL СУБД Firebird

WHERE

Условие WHERE ограничивает набор удаляемых строк. Удаляются только те, которые удовлетворяют условию поиска, или – только в PSQL – текущей строке именованного курсора.

Примеры:

```
delete from People  
where firstname <> 'Boris' and lastname <> 'Johnson'
```

```
delete from Cities  
where current of Cur_Cities; -- только в PSQL
```

Удаление с помощью WHERE CURRENT OF называется *позиционированным удалением* (positioned delete), потому что удаляется запись в текущей позиции. Удаление при помощи «WHERE условие» называется *поисковым удалением* (searched delete), поскольку движок ищет записи, соответствующие условию.

PLAN

Доступно в DSQL, PSQL

Конструкция PLAN позволяет вручную указать план для оптимизатора.

Пример:

```
delete from Submissions  
where date_entered < '1-Jan-2002'  
plan (Submissions index ix_subm_date)
```

ORDER BY

Доступно в DSQL, PSQL

ORDER BY упорядочивает набор перед его удалением. Имеет смысл только в комбинации с ROWS, но допустим и без ROWS.

Примеры:

Удаление самой старой покупки

```
delete from Purchases order by date rows 1
```

Удаление заказов для 10 клиентов с самыми большими номерами

Руководство по языку SQL СУБД Firebird

```
delete from Sales order by custno desc rows 1 to 10
```

Удаляет все записи из sales, поскольку не указано ROWS

```
delete from Sales order by custno desc
```

RETURNING

Доступно в DSQL, PSQL

Оператор DELETE, удаляющий *не более одной строки*, может содержать конструкцию RETURNING для возвращения значений удаляемой строки. В RETURNING могут быть указаны любые столбцы, не обязательно все, а также другие столбцы и выражения.

У столбцов, возвращаемых в контекстные переменные NEW в триггере, перед именем не должно использоваться двоеточие.

Примеры

```
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;
```

```
delete from TempSales ts
where ts.id = tempid
returning ts.qty into new.qty; -- не ":new.qty"
```

Замечания

- В DSQL, оператор с RETURNING *всегда* возвращает только одну строку. Если записи не были удалены, то возвращаемые столбцы содержат NULL. Это поведение может быть изменено в последующих версиях Firebird.
- В PSQL, если строка не была удалена, ничего не возвращается, и целевые переменные сохраняют свои значения.

ROWS

Доступно в DSQL, PSQL

Ограничивает количество удаляемых строк.

Руководство по языку SQL СУБД Firebird

Синтаксис:

ROWS $\langle m \rangle$ [TO $\langle n \rangle$]

$\langle m \rangle$, $\langle n \rangle$::= Любое целочисленное (integer) выражение

При одном аргументе m , удаляются первые m строк. Порядок строк без ORDER BY не определен (случаен).

Замечания:

- Если $m >$ общего числа строк в наборе, то весь набор удаляется
- Если $m = 0$, то удаление не происходит
- Если $m < 0$, то выдается сообщение об ошибке

Если указаны аргументы m и n , удаление ограничено количеством строк от m до n , включительно. Нумерация строк от 1.

Замечания по использованию двух аргументов:

- Если $m >$ общего числа строк в наборе, ни одна строка не удаляется
- Если $m > 0$ и $\leq n$ числа строк в наборе, а n вне этих значений, то удаляются строки от m до конца набора.
- Если $m < 1$ или $n < 1$, выдается сообщение об ошибке
- Если $n = m - 1$, ни одна строка не удаляется
- Если $n < m - 1$, выдается сообщение об ошибке

Примеры:

Удаляет одну запись «с конца», т.е. от Z...

```
delete from popgroups order by name desc rows 1
```

Удаляет пять самых старых групп

```
delete popgroups order by formed rows 5
```

Сортировка (ORDER BY) не указана, поэтому будут удалены 8 обнаруженных записей, начиная с пятой.

```
delete from popgroups rows 5 to 12
```

EXECUTE BLOCK

Доступен в DSQL

Руководство по языку SQL СУБД Firebird

Описание: Выполняет блок PSQL кода, так как будто это хранимая процедура, возможно, с входными и выходными параметрами и локальными переменными. Это позволяет пользователю выполнять "на лету" PSQL в контексте DSQL.

Общий синтаксис:

```
EXECUTE BLOCK [(<inparams>)]
[RETURNS (<outparams>)]
AS
[<declarations>]
BEGIN
[<PSQL statements>]
END

<inparams> ::= <param_decl> = ? [, <inparams> ]

<outparams> ::= <param_decl> [, <outparams>]

<param_decl> ::= paramname <type> [NOT NULL] [COLLATE
collation]

<type> ::= sql_datatype
        | [TYPE OF] domain
        | TYPE OF COLUMN rel.col

<declarations> ::= См. DECLARE VARIABLE

<PSQL statements> ::= См. главу Операторы PSQL
```

Примеры:

Этот пример вводит цифры от 0 до 127 и соответствующие им ASCII символы в таблицу ASCIITABLE:

```
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
end
```

Руководство по языку SQL СУБД Firebird

Следующий пример вычисляет среднее геометрическое двух чисел и возвращает его пользователю:

```
execute block (  
    x double precision = ?,  
    y double precision = ?)  
returns (gmean double precision)  
as  
begin  
    gmean = sqrt(x*y);  
    suspend;  
end
```

Поскольку этот блок имеет входные параметры, он должен быть предварительно подготовлен. После чего можно установить параметры и выполнить блок. Как это будет сделано, и вообще возможно ли это сделать, зависит от клиентского программного обеспечения. Смотрите примечания ниже.

Наш последний пример принимает два целочисленных значений, `smallest` и `largest`. Для всех чисел в диапазоне `smallest..largest`, блок выводит само число, его квадрат, куб и четвертую степень.

```
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
```

Опять же, как вы можете установить значения параметров, зависит от программного обеспечения клиента.

Руководство по языку SQL СУБД Firebird

Входные и выходные параметры

Выполнение блока без входных параметров должно быть возможным с любым клиентом Firebird, который позволяет пользователю вводить свои собственные DSQL операторы. Если есть входные параметры, все становится сложнее: эти параметры должны получить свои значения после подготовки оператора, но перед его выполнением. Это требует специальных возможностей, которыми располагает не каждое клиентское приложение. (Например, ISQL такой возможности не предлагает).

Сервер принимает только вопросительные знаки ("?") в качестве заполнителей для входных значений, а не ":a", ":MyParam" и т.д., или литеральные значения. Клиентское программное обеспечение может поддерживать форму ":xxx», в этом случае будет произведена предварительная обработка запроса перед отправкой его на сервер.

Если блок имеет выходные параметры, вы должны использовать SUSPEND, иначе ничего не будет возвращено.

Выходные данные всегда возвращаются в виде набора данных, так же как и в случае с оператором SELECT. Вы не можете использовать RETURNING_VALUES или выполнить блок, вернув значения в некоторые переменные, используя INTO, даже если возвращается всего одна строка.

Для получения дополнительной информации о параметрах и объявлениях переменных, [TYPE OF] domain, TYPE OF COLUMN и т.д. обратитесь к главе PSQL подглаве DECLARE VARIABLE в частности.

Терминатор оператора

Некоторые клиенты, особенно те, что позволяет пользователю отослать несколько операторов сразу, могут потребовать от вас, окружить оператор EXECUTE BLOCK строками SET TERM, как в этом примере:

```
set term #;
execute block (...)
as
begin
  statement1;
  statement2;
end
#
set term ;#
```

В качестве примера, в клиенте isql СУБД Firebird необходимо установить терминатор отличный от «;», прежде чем ввести оператор EXECUTE BLOCK. Если этого не сделать, ISQL попытается выполнить часть которая была напечатана до того момента, как вы нажали клавишу Enter после строки с точкой запятой «;».

EXECUTE PROCEDURE

Доступен в ESQL, DSQL, PSQL

Описание: Выполняет хранимую процедуру (SP), необязательно берёт входные параметры и/или возвращает выходные значения.

Общий синтаксис:

```
EXECUTE PROCEDURE
[TRANSACTION П]
procname
    [<in_item> [, <in_item> ...]]
    | ((<in_item> [, <in_item> ...]))
[RETURNING_VALUES <out_item> [, <out_item> ...]]
```

<in_item> ::= <inparam> [<nullind>]

<out_item> ::= <outvar> [<nullind>]

<inparam> ::= an expression evaluating to
the declared parameter type

<outvar> ::= a host language or PSQL variable
to receive the return value

<nullind> ::= [INDICATOR]:host_lang_intvar

Примечания:

- В ESQL, входные параметры должны быть литералами или языковыми переменными хоста. Для выходных параметров, переменные хоста должны быть указаны в предложении RETURNING_VALUES. NULL индикаторы должны быть хост переменными целого типа. Значения меньше нуля указывают NULL, ноль и больше нуля не NULL (что означает, что соответствующее значение присутствует в соответствующем параметре).
- В PSQL, входные параметрами могут быть любые выражения, значения которых ожидаемого типа. Для выходных параметров, локальные переменные должны быть указаны в предложении RETURNING_VALUES.
- В DSQL, входные параметрами могут быть любые выражения, значения которых ожидаемого типа. Обработка выходных параметров зависит от клиентского программного обеспечения.
- В PSQL и DSQL, NULL индикаторы не требуются. Значения NULL

Руководство по языку SQL СУБД Firebird

передаются через сами параметры ввода / вывода.

- Предложение TRANSACTION не поддерживается в PSQL.
- В ESQL, имена переменных, используемых в качестве параметров или выходных переменных должны предваряться двоеточием (":"). В PSQL двоеточие, как правило, не обязательно, но оно запрещено для триггерных контекстных переменных OLD и NEW.

Примеры:

В PSQL (с опциональными двоеточиями):

```
execute procedure MakeFullName  
  :FirstName, :MiddleName, :LastName  
returning_values :FullName;
```

То же вызов в ESQL (с обязательными двоеточиями):

```
exec sql  
execute procedure MakeFullName  
  :FirstName, :MiddleName, :LastName  
returning_values :FullName;
```

... и в утилите командной строки ISQL (с литералами в качестве параметров):

```
execute procedure MakeFullName  
  'J', 'Edgar', 'Hoover';
```

Примечание: В ISQL, не используется RETURNING_VALUES. Любые выходные значения отображаются автоматически.

Наконец, пример с выражениями в качестве параметров:

```
execute procedure MakeFullName  
  'Mr./Mrs. ' || FirstName, MiddleName, upper(LastName)  
returning_values FullName;
```

Глава 7

Операторы PSQL

Procedural SQL (PSQL) – процедурное расширение языка SQL. Это подмножество языка используется для написания хранимых процедур, триггеров и PSQL блоков.

Это расширение содержит все основные конструкции классических языков программирования. Кроме того, в него входят немного модифицированные DML операторы (SELECT, INSERT, UPDATE, DELETE и др.). Процедурное расширение может содержать объявления локальных переменных и курсоров, операторы присваивания, условные операторы, операторы циклов, выброса пользовательского исключений, средства для обработки ошибок, отправки сообщений (событий) клиентским программам. Кроме того, в триггерах доступны специфичные контекстные переменные, такие как NEW и OLD. В PSQL не допустимы операторы модификации метаданных (DDL операторы).

Если DML операторы (SELECT, INSERT, UPDATE, DELETE и др.) содержат именованные параметры, то они должны быть предварительно объявлены как переменные в операторе DECLARE [VARIABLE] или доступны во входных или выходных параметрах PSQL модуля. Перед именованными параметрами необходимо ставить двоеточие «:». Однако в предложении INTO символ «:» не обязателен. То же самое касается оператора выполнения хранимой процедуры EXECUTE PROCEDURE.

Хранимые процедуры выполняются в контексте той транзакции, в которой они были запущены. Триггеры выполняются в контексте транзакции, в которой выполнялась операция манипулирования данными, в результате чего был автоматически запущен триггер. Для триггеров на событие базы данных запускается отдельная транзакция. В PSQL не допустимы операторы старта и завершения транзакций, но существует возможность запуска оператора или блока операторов в автономной транзакции.

В синтаксисе PSQL модулей можно выделить заголовок и тело. Заголовок содержит имя модуля и описание локальных переменных. Для хранимых процедур и PSQL блоков заголовок может содержать описание входных и выходных параметров. Для триггеров в заголовке также указывается событие и его фаза, при котором триггер будет вызван автоматически. Тело PSQL модуля представляет собой блок операторов, содержащий описание выполняемых программой действий. Блок операторов заключается в операторные скобки BEGIN и END. В самих программах возможно присутствие произвольного количества блоков, как последовательных, так и вложенных друг в друга. Все операторы за исключением блоков BEGIN ... END отделяются друг от друга точкой с запятой. Никакой другой символ не является допустимым терминатором операторов PSQL.

Хранимые процедуры

Хранимая процедура является программой, хранящейся в области метаданных базы данных и выполняющейся на стороне сервера. К хранимой процедуре могут обращаться хранимые процедуры (в том числе и сама к себе), триггеры и клиентские программы. При обращении хранимой процедуре самой к себе такая хранимая процедура называется рекурсивной.

Хранимые процедуры имеют следующие преимущества:

1. Модульность

Приложения, работающие с одной и той же базой данных, могут использовать одну и ту же хранимую процедуру, тем самым уменьшив размер кода приложения и устранив дублирование кода.

2. Упрощение поддержки приложений

При изменении хранимой процедуры, изменения отражаются сразу во всех приложениях, без необходимости их перекомпиляции.

3. Увеличение производительности

Поскольку хранимые процедуры выполняются на стороне сервера, а не клиента, то это уменьшает сетевой трафик, что повышает производительность.

Существуют два вида хранимых процедур – выполнимые хранимые процедуры (executed stored procedures) и процедуры выбора (selected stored procedures).

Выполняемые процедуры данных, осуществляют обработку данных, находящихся в базе данных. Эти процедуры могут получать входные параметры и возвращать выходные. Такие процедуры выполняются с помощью оператора **EXECUTE PROCEDURE**

Хранимые процедуры выбора обычно осуществляют выборку данных из базы данных, возвращает при этом произвольное количество строк. Такие процедуры позволяют получать довольно сложные наборы данных, которые зачастую невозможно или весьма затруднительно получить с помощью обычных SELECT запросов. Процедуры выбора могут иметь входные параметры. Значение каждой очередной прочитанной строки возвращается вызвавшей программе в выходных параметрах. Для временной приостановки выполнения такой процедуры и передачи выбранных данных, вызвавшей программе, в хранимой процедуре используется оператор . Обращение к хранимой процедуре выбора осуществляется при помощи оператора SELECT (см. [Выборка из селективной хранимой процедуры](#)).

Создание хранимой процедуры

Синтаксис создание выполняемых хранимых процедур и процедур выбора ничем не отличается.

Синтаксис (не полный):

Руководство по языку SQL СУБД Firebird

```
CREATE PROCEDURE procname
[(<inparam> [, <inparam> ...])]
[RETURNS (<outparam> [, <outparam> ...])]
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END
```

Заголовок хранимой процедуры обязательно содержит имя процедуры, которое должно быть уникальным среди имён хранимых процедур, таблиц и представлений. В нём так же может быть описано некоторое количество входных и выходных параметров.

Тело хранимой процедуры может содержать объявление локальных переменных и курсоров. Оно также содержит блок операторов PSQL, который может быть пустым.

Более подробная информация приведена в главе Операторы DDL (CREATE PROCEDURE).

Изменение хранимой процедуры

В существующих хранимых процедурах можно изменять набор входных и выходных параметров и тело процедуры.

Синтаксис (не полный):

```
ALTER PROCEDURE procname
[(<inparam> [, <inparam> ...])]
[RETURNS (<outparam> [, <outparam> ...])]
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END
```

Более подробная информация приведена в главе Операторы DDL (ALTER PROCEDURE).

Удаление хранимой процедуры

Для удаления хранимых процедур используется оператор DROP PROCEDURE.

Синтаксис (полный):

Руководство по языку SQL СУБД Firebird

```
DROP PROCEDURE procname;
```

Более подробная информация приведена в главе [Операторы DDL \(DROP PROCEDURE\)](#).

Хранимые функции

В Firebird (до версии 3.0) нет PSQL функций. Однако вы можете использовать запрос или подзапрос к хранимым процедурам выбора для их замены.

Пример:

```
SELECT  
    PSQL_FUNC(T.col1, T.col2) AS col3,  
    col3  
FROM T
```

Можно заменить на

```
SELECT  
    (SELECT output_column FROM PSQL_PROC(T.col1)) AS col3,  
    col2  
FROM T
```

Или

```
SELECT  
    output_column AS col3,  
    col2,  
FROM T  
LEFT JOIN PSQL_PROC(T.col1)
```

PSQL блоки

Для выполнения из декларативного SQL (DSQL) некоторых императивных действий используются анонимные (безымянные) PSQL блоки. Заголовок анонимного PSQL блока опционально может содержать входные и выходные параметры. Тело анонимного PSQL блока может содержать объявление локальных переменных и блок PSQL операторов. Анонимный PSQL блок не может обращаться сам к себе.

Как и хранимые процедуры анонимные PSQL блоки могут использоваться для обработки данных или для осуществления выборки из базы данных.

Синтаксис (не полный):

Руководство по языку SQL СУБД Firebird

```
EXECUTE BLOCK
[(<inparam> = ? [, <inparam> = ? ...])]
[RETURNS (<outparam> [, <outparam> ...])]
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END
```

Таблица 7.1. Параметры PSQL блока

Аргумент	Описание
<i>inparam</i>	Описание входного параметра.
<i>outparam</i>	Описание выходного параметра.
<i>declarations</i>	Секция объявления локальных переменных и именованных курсоров.
<i>PSQL_statments</i>	Операторы языка PSQL.

Подробнее смотреть [EXECUTE BLOCK](#).

Триггеры

Триггер является программой, которая хранится в области метаданных базы данных и выполняется на стороне сервера. Напрямую обращение к триггеру невозможно. Он вызывается автоматически при наступлении одного или нескольких событий, относящихся к одной конкретной таблице (к представлению), или при наступлении одного из событий базы данных. Триггер, вызываемый при наступлении события таблицы, связан с одной таблицей или представлением, с одним или более событиями для этой таблицы или представления (добавление, изменение или удаление данных) и ровно с одной фазой такого события (до наступления события или после этого). Триггер выполняется в контексте той транзакции, в контексте которой выполнялась программа, вызвавшая соответствующее событие. Исключением являются триггеры, реагирующие на события базы данных. Для некоторых из них запускается транзакция по умолчанию.

Существует шесть вариантов соотношения событие-фаза для таблицы (представления):

- до добавления новой строки (BEFORE INSERT);
- после добавления новой строки (AFTER INSERT);
- до изменения строки (BEFORE UPDATE);
- после изменения строки (AFTER UPDATE);
- до удаления строки (BEFORE DELETE);
- после удаления строки (AFTER DELETE).

Существует возможность создавать триггеры, вызываемые автоматически

Руководство по языку SQL СУБД Firebird

для одной таблицы (представления), для одной фазы и одного события, а также для одной фазы и нескольких событий.

Если для одной таблицы (представления), одного события и одной фазы существует несколько триггеров, то можно задать порядок их выполнения, указав позицию триггера в этой цепочке.

Для триггеров существуют специфические контекстные переменные OLD и NEW. Более правильное название этих ключевых слов – префиксы имен столбцов. В триггерах можно обращаться к значению любого столбца таблицы (представления) до его изменения в клиентской программе (для этого перед именем столбца помещается ключевое слово OLD и точка) и после изменения (перед именем столбца помещается NEW и точка).

Контекстная переменная OLD (префикс имени столбца) для всех видов триггеров является переменной только для чтения. Она недоступна в триггерах, вызываемых при добавлении данных, независимо от фазы события.

Контекстная переменная NEW в триггерах для фазы события после (AFTER) также является переменной только для чтения. Она недоступна в триггерах для события удаления данных.

Триггер, связанный с событиями базы данных, может вызываться при следующих событиях:

- при соединении с базой данных (CONNECT); перед выполнением триггера автоматически запускается транзакция по умолчанию;
- при отсоединении от базы данных (DISCONNECT); перед выполнением триггера запускается транзакция по умолчанию;
- при старте транзакции (TRANSACTION START); триггер выполняется в контексте этой транзакции;
- при подтверждении транзакции (TRANSACTION COMMIT); триггер выполняется в контексте этой транзакции;
- при отмене транзакции (TRANSACTION ROLLBACK); триггер выполняется в контексте транзакции.

Создание триггера

Синтаксис:

```
CREATE TRIGGER trigname
{ <relation_trigger_legacy>
| <relation_trigger_sql2003>
| <database_trigger> }
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
```

Руководство по языку SQL СУБД Firebird

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  
}
```

Заголовок триггера обязательно содержит имя триггера, которое должно быть уникальным среди имён триггеров, и событие при котором срабатывает триггер. Если триггер создаётся для события таблицы, то необходимо также указать фазу события и имя таблицы.

Тело триггера может содержать объявление локальных переменных и курсоров. Оно также содержит блок операторов PSQL, который может быть пустым.

Более подробная информация приведена в главе Операторы DDL (CREATE TRIGGER).

Изменение триггера

В существующих триггерах можно изменить состояние активности, фазу события и событие (события) таблицы (представления), позицию триггера и его тело. Триггеры на событие (события) таблицы (представления) не могут быть изменены в триггеры на событие базы данных и наоборот. Если какой-либо

элемент не указан, то он остаётся без изменений.

Синтаксис:

```
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
}

Более подробная информация приведена в главе Операторы DDL (ALTER TRIGGER).

Удаление триггера

Для удаления триггера используется оператор DROP TRIGGER.

Синтаксис (полный):

```
DROP TRIGGER trigname;
```

Более подробная информация приведена в главе Операторы DDL (DROP TRIGGER).

SET TERM

Изменение терминатора

Доступно: ISQL

Синтаксис:

```
SET TERM <new_terminator><old_terminator>
```

Таблица 7.2. Параметры оператора SET TERM

Аргумент	Описание
new_terminator	Новый терминатор.
old_terminator	Старый терминатор.

Описание:

При написании триггеров и хранимых процедур в текстах скриптов, создающих требуемые программные объекты базы данных, во избежание двусмысленности относительно использования символа завершения операторов (по нормам SQL это точка с запятой) применяется оператор SET TERM, который, строго говоря, не является оператором SQL. При помощи этого оператора перед началом создания триггера или хранимой процедуры задается символ или строка символов, являющийся завершающим в конце текста триггера или хранимой процедуры. После описания текста соответствующего программного объекта при помощи того же оператора SET TERM значение терминатора возвращается к обычному варианту - точка с запятой.

Альтернативный терминатор может быть любой произвольной строкой символов за исключением точки с запятой, пробела и апострофа. Если вы используете буквенный символ, то он будет чувствителен к регистру.

Примеры:

Задание в качестве альтернативного терминатора символа «^».

```
SET TERM ^;
```

```
CREATE OR ALTER PROCEDURE SHIP_ORDER (  
    PO_NUM CHAR(8))  
AS  
BEGIN  
    /* Тело хранимой процедуры */  
END^
```

```
/* Другие хранимые процедуры и триггеры */
```

```
SET TERM ; ^
```

```
/* Другие операторы DDL */
```

Операторы языка PSQL

Оператор присваивания

Присваивание переменной значения.

Доступно: PSQL

Синтаксис:

```
varname = <value_expr>
```

Таблица 7.3. Оператор присваивания

Аргумент	Описание
varname	Имя локальной переменной или параметра процедуры.
value_expr	Выражение.

Описание:

Оператор присваивания устанавливает переменной значение SQL выражения. Выражением может быть любое правильное выражение SQL. Оно может содержать литералы, имена внутренних переменных, арифметические, логические и строковые операции, обращения к встроенным функциям и к функциям, определенным пользователем (UDF).

Примеры:

Примеры использования оператора присваивания.

```
CREATE PROCEDURE MYPROC (  
    a INTEGER,  
    b INTEGER,  
    name VARCHAR (30)  
)  
RETURNS (  
    c INTEGER,  
    str VARCHAR(100))  
AS  
BEGIN  
    -- присваиваем константу  
    c = 0;
```

Руководство по языку SQL СУБД Firebird

```
str = '';  
SUSPEND;  
-- присваиваем значения выражений  
c = a + b;  
str = name || CAST(b AS VARCHAR(10));  
SUSPEND;  
-- присваиваем значение выражения  
-- построенного с использованием запроса  
c = (SELECT 1 FROM rdb$database);  
-- присваиваем значение из контекстной переменной  
str = CURRENT_USER;  
SUSPEND;  
END
```

См. также: [DECLARE VARIABLE](#)

DECLARE VARIABLE

Объявление локальной переменной или курсора.

Доступно: PSQL

Синтаксис:

```
DECLARE [VARIABLE]  
{ varname <type> [NOT NULL] [COLLATE collation]  
  [{= | DEFAULT} <value>] }  
| { cursorname CURSOR FOR (<select>)}  
  
<value> ::= {literal | NULL | context_var}  
  
<type> ::=  
<datatype>  
| [TYPE OF] domain  
| TYPE OF COLUMN rel.col  
  
<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])]
```

Таблица 7.4. DECLARE VARIABLE

Аргумент	Описание
varname	Имя локальной переменной.
cursorname	Имя курсора.
literal	Литерал.
context_var	Любая контекстная переменная, тип которой совместим с типом локальной переменной.
datatype	Тип данных SQL.
collation	Порядок сортировки.
domain	Домен.
rel	Имя таблицы или представления.
col	Имя столбца таблицы или представления.
precision	Точность. От 1 до 18.
scale	Масштаб. От 0 до 18, должно быть меньше или равно precision.
size	Максимальный размер строки в символах.
charset	Набор символов.
subtype_num	Номер подтипа BLOB.
subtype_name	Мнемоника подтипа BLOB.
seqlen	Размер сегмента, не может превышать 65535.
select	Оператор SELECT.

Описание:

Оператор DECLARE [VARIBALE] объявляет локальную переменную. Ключевое слово VARIABLE можно опустить. В одном операторе разрешено объявлять только одну переменную. В процедурах и триггерах можно объявить произвольное число локальных переменных, используя при этом каждый раз, новый оператор DECLARE [VARAIBLE].

Имя локальной переменной должно быть уникально среди имён локальных переменных, входных и выходных параметров процедуры внутри программного объекта.

В качестве типа данных локальной переменной может быть любой SQL тип, за исключением массивов.

В качестве типа переменной можно указать имя домена. В этом случае, переменная будет наследовать все характеристики домена. Если перед названием домена дополнительно используется предложение "TYPE OF", то используется только тип данных домена – не проверяется (не используется) его ограничение (если оно есть в домене) на NOT NULL, CHECK ограничения и/или значения по умолчанию. Если домен текстового типа, то всегда используется его набор символов и порядок сортировки.

Руководство по языку SQL СУБД Firebird

Локальные переменные можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение TYPE OF COLUMN, после которого указывается имя таблиц или представления и через точку имя столбца. При использовании TYPE OF COLUMN наследуется только тип данных, а в случае строковых типов ещё набор символов и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

Для локальных переменных можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL. Для переменной строкового типа существует возможность задать порядок сортировки с помощью предложения COLLATE.

Локальной переменной можно устанавливать инициализирующее (начальное) значение. Это значение устанавливается с помощью предложения DEFAULT или оператора «=». В качестве значения по умолчанию может быть использовано значение NULL, литерал и любая контекстная переменная совместимая по типу данных.

Переменная может быть курсором, связанным с конкретным набором данных, полученным при помощи оператора SELECT, записанного после ключевых слов CURSOR FOR. Такую переменную ещё называют именованным курсором. В дальнейшем курсор может быть открыт, использоваться для обхода результирующего набора данных, и снова быть закрытым. Также поддерживаются позиционированные обновления и удаления (при использовании WHERE CURRENT OF).

Примечания:

- Предложение "FOR UPDATE" разрешено использовать в операторе SELECT, но оно не требуется для успешного выполнения позиционированного обновления или удаления;
- Удостоверьтесь, что объявленные имена курсоров не совпадают, ни с какими именами, определенными позже в предложениях AS CURSOR;
- Если курсор требуется только для прохода по результирующему набору данных, то практически всегда проще (и менее подвержено ошибкам) использовать оператор **FOR SELECT** с предложением AS CURSOR. Объявленные курсоры должны быть явно открыты, использованы для выборки данных и закрыты. Кроме того, вы должны проверить контекстную переменную row_count после каждой выборки и выйти из цикла, если её значение ноль. Предложение FOR SELECT делает эту проверку автоматически. Однако объявленные курсоры дают большие возможности для контроля над последовательными событиями и позволяют управлять несколькими курсорами параллельно;
- Оператор SELECT может содержать параметры, например: "SELECT NAME || :SFX FROM NAMES WHERE NUMBER = :NUM". Каждый

Руководство по языку SQL СУБД Firebird

параметр должен быть заранее объявлен как переменная PSQL (это касается также входных и выходных параметров). При открытии курсора параметру присваивается текущее значение переменной;

- **Внимание!** Если значение переменной PSQL, используемой в операторе SELECT, изменяется во время выполнения цикла, то её новое значение может (но не всегда) использоваться при выборке следующих строк. Лучше избегать таких ситуаций. Если вам действительно требуется такое поведение, то необходимо тщательно протестировать код и убедиться, что вы точно знаете, как изменения переменной влияют на результаты выборки. Особо отмечу, что поведение может зависеть от плана запроса, в частности, от используемых индексов. В настоящее время нет строгих правил для таких ситуаций, но в новых версиях Firebird это может измениться.

Примеры:

1. Демонстрация различных способов объявления локальных переменных.

```
CREATE OR ALTER PROCEDURE SOME_PROC
AS
  -- Объявление переменной типа INT
  DECLARE I INT;
  -- Объявление переменной типа INT не допускающей
  значение NULL
  DECLARE VARIABLE J INT NOT NULL;
  -- Объявление переменной типа INT со значением по
  умолчанию 0
  DECLARE VARIABLE K INT DEFAULT 0;
  -- Объявление переменной типа INT со значением по
  умолчанию 1
  DECLARE VARIABLE L INT = 1;
  -- Объявление переменной на основе домена COUNTRYNAME
  DECLARE FARM_COUNTRY COUNTRYNAME;
  -- Объявление переменной с типом равным типу домена
  COUNTRYNAME
  DECLARE FROM_COUNTRY TYPE OF COUNTRYNAME;
  -- Объявление переменной с типом столбца CAPITAL таблицы
  COUNTRY
  DECLARE CAPITAL TYPE OF COLUMN COUNTRY.CAPITAL;
BEGIN
  /* Операторы PSQL */
END
```

2. Объявление именованного курсора в триггере.

```
CREATE OR ALTER TRIGGER TBU_STOCK
BEFORE UPDATE ON STOCK
AS
  -- Объявление именованного курсора
```

Руководство по языку SQL СУБД Firebird

```
DECLARE C_COUNTRY CURSOR FOR (  
    SELECT  
        COUNTRY,  
        CAPITAL  
    FROM COUNTRY  
);  
BEGIN  
    /* Операторы PSQL */  
END
```

3. Сборка скриптов создания представлений с помощью PSQL блока с использованием именованных курсоров.

```
EXECUTE BLOCK  
RETURNS (  
    SCRIPT BLOB SUB_TYPE TEXT)  
AS  
-- Объявление локальной переменной  
DECLARE VARIABLE FIELDS VARCHAR(8191);  
-- Объявление локальной переменной с типа, который указан  
-- в типе для домена  
DECLARE VARIABLE FIELD_NAME TYPE OF RDB$FIELD_NAME;  
-- Объявление локальной переменной с использованием домена  
DECLARE VARIABLE RELATION RDB$RELATION_NAME;  
-- Объявление локальной переменной в качестве типа,  
-- которой используется тип столбца таблицы  
DECLARE VARIABLE SOURCE TYPE OF COLUMN  
RDB$RELATIONS.RDB$VIEW_SOURCE;  
-- Объявление именованного курсора  
DECLARE VARIABLE CUR_R CURSOR FOR (  
    SELECT  
        RDB$RELATION_NAME,  
        RDB$VIEW_SOURCE  
    FROM  
        RDB$RELATIONS  
    WHERE  
        RDB$VIEW_SOURCE IS NOT NULL);  
-- Объявление именованного курсора, в котором  
-- используется локальная переменная  
DECLARE CUR_F CURSOR FOR (  
    SELECT  
        RDB$FIELD_NAME  
    FROM  
        RDB$RELATION_FIELDS  
    WHERE  
        -- Важно переменная должна быть объявлена ранее  
        RDB$RELATION_NAME = :RELATION);  
BEGIN  
    OPEN CUR_R;
```

Руководство по языку SQL СУБД Firebird

```
WHILE (1 = 1) DO
BEGIN
    FETCH CUR_R
    INTO :RELATION, :SOURCE;
    IF (ROW_COUNT = 0) THEN
        LEAVE;

    FIELDS = NULL;
    -- Курсор CUR_F будет использовать значение
    -- переменной RELATION инициализированной выше
    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
    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
```

См. также: [Типы и подтипы данных](#), [Работа с доменами](#), [OPEN](#), [FETCH](#), [CLOSE](#)

BEGIN ... END

Блок операторов.

Доступно: PSQL

Синтаксис:

```
<block> ::=  
BEGIN  
    <compound_statement>  
    [<compound_statement> ...]  
END  
  
<compound_statement> ::= {<block> | <statement>;}
```

Описание:

Операторные скобки BEGIN ... END определяют блок операторов, которые выполняются как один оператор. Каждый блок начинается оператором BEGIN и завершается оператором END. Блоки могут быть вложенными, глубина вложения не ограничена. Блоки могут быть пустыми, что позволяет использовать его как заглушку, позволяющую избежать написания фиктивных операторов.

После операторов BEGIN и END точка с запятой не ставится. Однако, утилита командной строки **isql** требует, чтобы после последнего оператора END в определении PSQL модуля следовал символ терминатора, установленного командой SET TERM.

Последний оператор END в триггере завершает работу триггера. Последний оператор END в хранимой процедуре работает в зависимости от типа процедуры:

- В селективной процедуре последний оператор END возвращает управление приложению и устанавливает значение SQLCODE равным 100, что означает, что больше нет строк для извлечения.
- В выполняемой процедуре последний оператор END возвращает управление и текущие значения выходных параметров, если таковые имеются, вызывающему приложению.

Примеры:

Пример процедуры из базы данных employee.fdb, который демонстрирует применение блоков BEGIN...END.

```
SET TERM ^;  
CREATE OR ALTER PROCEDURE DEPT_BUDGET (  
    DNO CHAR(3))  
RETURNS (  
    TOT DECIMAL(12,2))
```

Руководство по языку SQL СУБД Firebird

```
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 ;^
```

См. также: [EXIT](#), [LEAVE](#), [SET TERM](#)

IF ... THEN ... ELSE

Оператор условного перехода.

Доступно: PSQL

Синтаксис:

```
IF (<condition>)  
  THEN <compound_statement>  
  [ELSE <compound_statement>]
```

Таблица 7.5. Параметры IF ... THEN ... ELSE

Аргумент	Описание
condition	Логическое условие возвращающее TRUE, FALSE или UNKNOWN.
compound_statement	Оператор или блок операторов.

Описание:

Оператор условного перехода IF используется для выполнения ветвления процесса обработки данных в PSQL. Если условие возвращает значение TRUE, то выполняется оператор или блок операторов после ключевого слова THEN. Иначе (если условие возвращает FALSE или UNKNOWN) выполняется оператор или блок операторов после ключевого слова ELSE, если оно присутствует. Условие всегда заключается в круглые скобки.

Примечание:

В языке PSQL отсутствует оператор ветвления CASE (SWITCH). Однако в нём доступен поисковый оператор CASE из DSQL.

```
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
```

Пример использования в PSQL.

```
...  
C = CASE  
  WHEN A=2 THEN 1  
  WHEN A=1 THEN 3  
  ELSE 0
```

```
END;
```

```
...
```

Примеры:

Иллюстрация использования оператора IF. Предполагается, что переменные FIRST, LINE2 и LAST были объявлены ранее.

```
. . .  
IF (FIRST IS NOT NULL) THEN  
    LINE2 = FIRST || ' ' || LAST;  
ELSE  
    LINE2 = LAST;  
. . .
```

См. также: [WHILE ... DO](#), [CASE](#)

WHILE ... DO

Оператор цикла.

Доступно: PSQL

Синтаксис:

```
WHILE (<condition>) DO  
<compound_statement>
```

Таблица 7.6. Оператор WHILE ... DO

Аргумент	Описание
condition	Логическое условие возвращающее TRUE, FALSE или UNKNOWN.
compound_statement	Оператор или блок операторов.

Описание:

Оператор WHILE используется для организации циклов в PSQL. Оператор или блок операторов будут выполняться до тех пор, пока условие истинно (возвращает TRUE). Циклы могут быть вложенными, глубина вложения не ограничена.

Примеры:

Процедура расчёта суммы чисел от 1 до I демонстрирует использование оператора цикла.

Руководство по языку SQL СУБД Firebird

```
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
```

При выполнении в ISQL

```
EXECUTE PROCEDURE SUM_INT(4);
```

результат будет следующий

```
S
=====
10
```

См. также: [IF ... THEN ... ELSE](#), [LEAVE](#), [EXIT](#), [FOR SELECT](#), [FOR EXECUTE STATEMENT](#)

LEAVE

Оператор выхода из цикла.

Доступно: PSQL

Синтаксис:

```
[label:]
{FOR <select_stmt> |
WHILE (<condition>)} DO
BEGIN
    ...
    LEAVE [label];
    ...
END
```

Таблица 7.7. Оператор LEAVE

Аргумент	Описание
label	Метка.

Руководство по языку SQL СУБД Firebird

select_stmt	Оператор SELECT.
condition	Логическое условие возвращающее TRUE, FALSE или UNKNOWN.

Описание:

Оператор LEAVE моментально прекращает работу внутреннего цикла операторов WHILE или FOR. С использованием опционального параметра метки. LEAVE также может выйти и из внешних циклов, при этом выполнение кода продолжается с первого оператора, следующего после прекращения блока внешнего цикла.

Примеры:

1. Выход из цикла при возникновении ошибки вставки в таблицу NUMBERS. Код продолжается со строки C = 0.

```
...
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
C = 0;
...
```

2. Пример использования меток в операторе LEAVE. "LEAVE LOOPA" завершает внешний цикл, а "LEAVE LOOPB" - внутренний. Обратите внимание: простого оператора "LEAVE" также было бы достаточно, чтобы завершить внутренний цикл.

```
...
STMT1 = 'SELECT NAME FROM FARMS';
LOOPA:
FOR EXECUTE STATEMENT :STMT1
INTO :FARM DO
BEGIN
    STMT2 = 'SELECT NAME ' || 'FROM ANIMALS WHERE FARM =
    ''';
    LOOPB:
```

Руководство по языку SQL СУБД Firebird

```
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
...
```

См. также: [EXIT](#)

EXIT

Завершение работы программы.

Доступно: PSQL

Синтаксис:

```
EXIT
```

Описание:

Оператор EXIT позволяет из любой точки триггера или хранимой процедуры перейти на последний оператор END, то есть завершить выполнение программы.

Примеры:

Использование оператора EXIT в процедуре выбора.

```
CREATE PROCEDURE GEN_100
RETURNS (
  I INTEGER
)
AS
BEGIN
  I = 1;
  WHILE (1=1) DO
  BEGIN
    SUSPEND;
    IF (I=100) THEN
      EXIT;
```

Руководство по языку SQL СУБД Firebird

```
        I = I + 1;  
    END  
END
```

См. также: [LEAVE](#), [SUSPEND](#)

SUSPEND

Приостановка выполнения хранимой процедуры или PSQL блока и передача вызывающей программе значений выходных аргументов.

Доступно: PSQL

Синтаксис:

```
SUSPEND
```

Описание:

Оператор SUSPEND приостанавливает выполнение хранимой процедуры выбора и передает вызывающей программе значения выходных параметров. Когда вызвавшая программа выполняет после этого оператор FETCH (этот оператор неявно выдается при выполнении оператора SELECT), работа процедуры возобновляется с оператора, следующего непосредственно за оператором SUSPEND.

Если оператор SUSPEND выдается в исполняемой (executable) хранимой процедуре, то это равносильно выполнению оператора EXIT, в результате чего завершается работа процедуры.

Примечание:

Оператор SUSPEND «разрывает» атомарность блока, внутри которого он находится. В случае возникновения ошибки в селективной процедуре, операторы, выполненные после последнего оператора SUSPEND, будут откачены. Операторы, выполненные до последнего оператора SUSPEND, не будут откачены, если не будет выполнен откат транзакции.

Примеры:

Использование оператора SUSPEND в процедуре выбора.

```
CREATE PROCEDURE GEN_100  
RETURNS (  
    I INTEGER  
)  
AS  
BEGIN
```

Руководство по языку SQL СУБД Firebird

```
I = 1;
WHILE (1=1) DO
BEGIN
    SUSPEND;
    IF (I=100) THEN
        EXIT;
    I = I + 1;
END
END
```

См. также: [EXIT](#)

EXECUTE STATEMENT

Выполнение динамически созданных SQL операторов.

Доступно: PSQL

Синтаксис:

```
<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> | <netbeui_hostspec>

<tcpip_hostspec> ::= hostname:
```

Руководство по языку SQL СУБД Firebird

```
<netbeui_hostspec> ::= \\hostname\
```

```
<variables> ::= [:]varname [, [:]varname ...]
```

Таблица 7.8. Оператор EXECUTE STATEMENT

Аргумент	Описание
paramless_stmt	Строки или переменная, содержащая не параметризованный SQL запрос.
stmt_with_params	Строки или переменная, содержащая параметризованный SQL запрос.
paramname	Имя параметра SQL запроса.
value_expr	Выражение.
user	Имя пользователя. Может быть строкой или переменной.
password	Пароль. Может быть строкой или переменной.
role	Роль. Может быть строкой или переменной.
connection_string	Строка соединения. Может быть строкой или переменной.
filepath	Путь к первичному файлу базы данных.
db_alias	Псевдоним базы данных.
hostname	Имя компьютера или IP адрес.
varname	Переменная.

Описание:

Оператор EXECUTE STATEMENT принимает строковый параметр и выполняет его, как будто это оператор DSQL. Если оператор возвращает данные, то с помощью предложения INTO их можно передать в локальные переменные.

В DSQL операторе можно использовать параметры. Параметры могут быть именованными и позиционными (безымянными). Значение должно быть присвоено каждому параметру.

Особенности использования параметризованных операторов:

1. Одновременное использование именованных и позиционных параметров в одном запросе запрещено.
2. Когда у оператора есть параметры, они должны быть помещены в круглые скобки при вызове EXECUTE STATEMENT, независимо от вида их представления: непосредственно в виде строки, как имя переменной или как выражение;
3. Именованным параметрам должно предшествовать двоеточие (:) в самом операторе, но не при присвоении значения параметру;
4. Передача значений безымянным параметрам должна происходить в том же порядке, в каком они встречаются в тексте запроса;

Руководство по языку SQL СУБД Firebird

5. Присвоение значений параметров должно осуществляться при помощи специального оператора «:=», аналогичного оператору присваивания языка Pascal;
6. Каждый именованный параметр может использоваться в операторе несколько раз, но только один раз при присвоении значения;
7. Для позиционных параметров число подставляемых значений должно точно равняться числу параметров (вопросительных знаков) в операторе.

По умолчанию оператор выполняется в контексте текущей транзакции. При использовании предложения WITH AUTONOMOUS TRANSACTION запускается новая транзакция с такими же параметрами, как и текущая. Она будет подтверждена, если оператор выполнен без ошибок и отменена (откачена) в противном случае. С предложением WITH COMMON TRANSACTION по возможности используется текущая транзакция. Если оператор должен работать в отдельном соединении, то используется уже запущенная в этом соединении транзакция (если таковая транзакция имеется). В противном случае стартует новая транзакция с параметрами текущей транзакции. Любые новые транзакции, запущенные в режиме "COMMON", подтверждаются или откатываются вместе с текущей транзакцией.

По умолчанию операторы SQL выполняются с правами текущего пользователя. Спецификация WITH CALLER PRIVILEGES добавляет к ним привилегии для вызова хранимой процедуры или триггера, так же, как если бы оператор выполнялся непосредственно подпрограммой. WITH CALLER PRIVILEGES не имеет никакого эффекта, если также присутствует предложение ON EXTERNAL.

С предложением ON EXTERNAL DATA SOURCE оператор выполняется в отдельном соединении с той же или другой базой данных, возможно даже на другом сервере. Если строка подключения имеет значение NULL или " (пустая строка), предложение ON EXTERNAL считается отсутствующим и оператор выполняется для текущей базы данных. Набор символов, используемый для внешнего соединения, совпадает с используемым набором для текущего соединения. Двухфазные транзакции не поддерживаются.

При выполнении оператора в отдельном соединении используется пул соединений и пул транзакций.

Особенности пула подключений (Connection pooling):

1. Внешние соединения используют по умолчанию предложение WITH COMMON TRANSACTION и остаются открытыми до закрытия текущей транзакции. Они могут быть снова использованы при последующих вызовах оператора EXECUTE STATEMENT, но только если строка подключения точно такая же;
2. Внешние соединения, созданные с использованием предложения WITH AUTONOMOUS TRANSACTION, закрываются после выполнения

оператора;

- Операторы WITH AUTONOMOUS TRANSACTION могут использовать соединения, которые ранее были открыты операторами WITH COMMON TRANSACTION. В этом случае использованное соединение остаётся открытым и после выполнения оператора, т.к. у этого соединения есть по крайней мере одна не закрытая транзакция.

Особенности пула транзакций (Transaction pooling):

- При использовании предложения WITH COMMON TRANSACTION транзакции будут снова использованы как можно дольше. Они будут подтверждаться или откатываться вместе с текущей транзакцией;
- При использовании предложения WITH AUTONOMOUS TRANSACTION всегда запускается новая транзакция. Она будет подтверждена или отменена сразу же после выполнения оператора;

Примечание:

При использовании предложения ON EXTERNAL дополнительное соединение всегда делается через так называемого внешнего провайдера, даже если это соединение к текущей базе данных. Одним из последствий этого является то, что Вы не можете обработать исключение привычными способами. Каждое исключение, вызванное оператором, возвращает eds_connection или eds_statement ошибки. Для обработки исключений в коде PSQL вы должны использовать WHEN GDSCODE eds_connection, WHEN GDSCODE eds_statement или WHEN ANY. Если предложение ON EXTERNAL не используется, то исключения перехватываются в обычном порядке, даже если это дополнительное соединение с текущей базой данных.

Необязательные предложения AS USER, PASSWORD и ROLE позволяют указывать от имени какого пользователя, и с какой ролью будет выполняться SQL оператор. То, как авторизуется пользователь и открыто ли отдельное соединение, зависит от присутствия и значений параметров ON EXTERNAL [DATA SOURCE], AS USER, PASSWORD и ROLE.

- При использовании предложения ON EXTERNAL открывается новое соединение и:
 - Если присутствует, по крайней мере, один из параметров AS USER, PASSWORD и ROLE, то будет предпринята попытка нативной аутентификации с указанными значениями параметров (в зависимости от строки соединения – локально или удалённо). Для недостающих параметров не используются никаких значений по умолчанию;

Руководство по языку SQL СУБД Firebird

- Если все три параметра отсутствуют, и строка подключения не содержит имени сервера (или IP адреса), то новое соединение устанавливается к локальному серверу с пользователем и ролью текущего соединения. Термин 'локальный' означает 'компьютер, где установлен сервер Firebird'. Это совсем не обязательно компьютер клиента;
 - Если все три параметра отсутствуют, но строка подключения содержит имя сервера (или IP адреса), то будет предпринята попытка доверенной (trusted) авторизации к удалённому серверу. Если авторизация прошла, то удаленная операционная система назначит пользователю имя – обычно это учётная запись, под которой работает сервер Firebird.
- Если предложение ON EXTERNAL отсутствует:
 - Если присутствует, по крайней мере, один из параметров AS USER, PASSWORD и ROLE, то будет открыто соединение к текущей базе данных с указанными значениями параметров. Для недостающих параметров не используются никаких значений по умолчанию;
 - Если все три параметра отсутствуют, то оператор выполняется в текущем соединении.

Внимание:

Если значение параметра NULL или "", то весь параметр считается отсутствующим. Кроме того, если параметр считается отсутствующим, то AS USER принимает значение CURRENT_USER, а ROLE – CURRENT_ROLE. Сравнение при авторизации сделано чувствительным к регистру: в большинстве случаев это означает, что имена пользователя и роли должны быть написаны в верхнем регистре.

Предупреждения:

1. Не существует способа проверить синтаксис выполняемого SQL оператора;
2. Нет никаких проверок зависимостей для обнаружения удалённых столбцов в таблице или самой таблицы;
3. Выполнение оператора с помощью оператора EXECUTE STATEMENT значительно медленнее, чем при непосредственном выполнении;
4. Возвращаемые значения строго проверяются на тип данных во избежание непредсказуемых исключений преобразования типа. Например, строка '1234' преобразуется в целое число 1234, а строка 'abc' вызовет ошибку преобразования.

В целом эта функция должна использоваться очень осторожно, а вышеупомянутые факторы всегда должны приниматься во внимание. Если

такого же результата можно достичь с использованием PSQL и/или DSQL, то это всегда предпочтительнее.

См. также: [FOR EXECUTE STATEMENT](#)

FOR SELECT

Цикл по строкам результата выполнения оператора SELECT.

Доступно: PSQL

Синтаксис:

```
FOR <select_stmt> [AS CURSOR cursorname]
DO <compound_statement>
```

Таблица 7.9. Оператор FOR SELECT

Аргумент	Описание
select_stmt	Оператор SELECT.
cursorname	Имя курсора. Должно быть уникальным среди имён курсоров PSQL модуля (хранимой процедуры, триггера или PSQL блока).
compound_statement	Оператор или блок операторов.

Описание:

Оператор SELECT выбирает очередную строку из таблицы (представления, хранимой процедуры выбора), после чего выполняется оператор или блок операторов. В каждой итерации цикла значения полей текущей строки копируются в локальные переменные. Добавление предложения AS CURSOR делает возможным позиционное удаление и обновление данных. Операторы FOR SELECT могут быть вложенными.

Оператор SELECT может содержать именованные параметры, которые должны быть предварительно объявлены в операторе DECLARE VARIABLE, или во входных (выходных) параметрах процедуры (PSQL блока).

Оператор SELECT должен содержать предложение INTO, которое располагается в конце этого оператора. На каждой итерации цикла в список переменных указанных в предложении INTO копируются значения полей текущей строки запроса. Цикл повторяется, пока не будут прочитаны все строки. После этого происходит выход из цикла. Цикл также может быть завершён до прочтения всех строк при использовании оператора LEAVE.

Необязательное предложение AS CURSOR создаёт именованный курсор,

Руководство по языку SQL СУБД Firebird

на который можно сослаться (с использованием предложения WHERE CURRENT OF) внутри оператора или блока операторов следующего после предложения DO, для того чтобы удалить или модифицировать текущую строку.

Примечания:

1. Над курсором, объявленным с помощью предложения AS CURSOR нельзя выполнять операторы OPEN, FETCH и CLOSE;
2. Убедитесь, что имя курсора, определённое здесь, не совпадает ни с какими именами, созданными ранее оператором DECLARE VARIABLE;
3. Предложение FOR UPDATE, разрешённое для использования в операторе SELECT, не является обязательным для успешного выполнения позиционного обновления или добавления.

Примеры:

1. Простой цикл по результатам запроса.

```
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. Вложенный FOR SELECT цикл.

```
CREATE PROCEDURE RELFIELDS
RETURNS (
    RELATION CHAR(32),
    POS INTEGER,
    FIELD CHAR(32))
AS
BEGIN
```

Руководство по языку SQL СУБД Firebird

```
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 = ' ';
-- Для исключения повтора имён таблиц и
представлений
SUSPEND;
END
END
END
```

3. Использование предложения AS CURSOR для позиционного удаления записи.

```
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
-- Позиционное удаление записи
DELETE FROM TOWNS
WHERE CURRENT OF TCUR;
ELSE
SUSPEND;
END
END
```

Руководство по языку SQL СУБД Firebird

См. также: [DECLARE VARIABLE](#), [LEAVE](#), [SELECT](#), [UPDATE](#), [DELETE](#)

FOR EXECUTE STATEMENT

Выполнение динамически созданных SQL операторов с возвратом нескольких строк данных.

Доступно: PSQL

Синтаксис:

```
FOR <execute_statement> DO <compound_statement>
```

```
<execute_statement> ::= оператор
```

Таблица 7.10. Оператор FOR EXECUTE STATEMENT

Аргумент	Описание
execute_statement	Оператор EXECUTE STATEMENT.
compound_statement	Оператор или блок операторов.

Описание:

Оператор FOR EXECUTE STATEMENT используется (по аналогии с конструкцией FOR SELECT) для операторов SELECT или EXECUTE BLOCK, возвращающих более одной строки.

Примеры:

Выполнение динамически собранного селективного запроса, возвращающего набор данных.

```
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
```

Руководство по языку SQL СУБД Firebird

```
IF (:P_ONE_LINE IS NOT NULL) THEN
    LINE = :LINE || :P_ONE_LINE || ' ';
SUSPEND;
END
```

См. также: [EXECUTE STATEMENT](#)

OPEN

Открытие курсора.

Доступно: PSQL

Синтаксис:

```
OPEN cursorname;
```

Таблица 7.11. Оператор OPEN

Аргумент	Описание
cursorname	Имя курсора. Курсор с таким именем должен быть предварительно объявлен с помощью оператора DECLARE VARIABLE.

Описание:

Оператор OPEN открывает ранее объявленный курсор, выполняет объявленный в нём оператор SELECT и получает записи из результирующего набора данных. Оператор OPEN применим только к курсорам, объявленным в операторе DECLARE VARIABLE.

Примечание:

Если в операторе SELECT курсора имеются параметры, то они должны быть объявлены как локальные переменные или входные (выходные) параметры до того как объявлен курсор. При открытии курсора параметру присваивается текущее значение переменной.

Примеры:

1. Пример использования оператора OPEN.

```
SET TERM ^;
```

```
CREATE OR ALTER PROCEDURE GET_RELATIONS_NAMES
RETURNS (
    RNAME CHAR(31)
)
```

Руководство по языку SQL СУБД Firebird

```
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. Сборка скриптов создания представлений с помощью PSQL блока с использованием именованных курсоров.

```
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;
  -- Объявление именованного курсора
  DECLARE VARIABLE CUR_R CURSOR FOR (
    SELECT
      RDB$RELATION_NAME,
      RDB$VIEW_SOURCE
    FROM
      RDB$RELATIONS
    WHERE
      RDB$VIEW_SOURCE IS NOT NULL);
  -- Объявление именованного курсора, в котором
  -- используется локальная переменная
  DECLARE CUR_F CURSOR FOR (
    SELECT
      RDB$FIELD_NAME
    FROM
      RDB$RELATION_FIELDS
    WHERE
      -- Важно переменная должна быть объявлена ранее
      RDB$RELATION_NAME = :RELATION);
BEGIN
```

Руководство по языку SQL СУБД Firebird

```
OPEN CUR_R;
WHILE (1 = 1) DO
BEGIN
    FETCH CUR_R
    INTO :RELATION, :SOURCE;
    IF (ROW_COUNT = 0) THEN
        LEAVE;

    FIELDS = NULL;
    -- Курсор CUR_F будет использовать значение
    -- переменной RELATION инициализированной выше
    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
```

См. также: [DECLARE VARIABLE](#), [FETCH](#), [CLOSE](#)

FETCH

Чтение очередной записи набора данных, связанного с курсором.

Доступно: PSQL

Синтаксис:

```
FETCH cursorname INTO [:]varname [, [:]varname ...];
```

Таблица 7.12. Оператор FETCH

Аргумент	Описание
cursorname	Имя курсора. Курсор с таким именем должен быть предварительно объявлен с помощью оператора DECLARE VARIABLE.
varname	Имя переменной.

Описание:

Оператор **FETCH** выбирает следующую строку данных из результирующего набора данных курсора и присваивает значения столбцов в переменные **PSQL**. Оператор **FETCH** применим только к курсорам, объявленным в операторе **DECLARE VARIABLE**.

Предложение **INTO** помещает данные из текущей строки курсора в **PSQL** переменные.

Для проверки того, что записи набора данных исчерпаны, используется контекстная переменная **ROW_COUNT**, которая возвращает количество считанных оператором строк. Если произошло чтение очередной записи из набора данных, то **ROW_COUNT** равняется единице, иначе нулю.

Примеры:

Пример использования оператора **FETCH**.

```

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 ; ^

См. также: [DECLARE VARIABLE](#), [OPEN](#),

CLOSE

Закрытие курсора.

Доступно: PSQL

Синтаксис:

```
CLOSE cursorname;
```

Таблица 7.13. Оператор CLOSE

Аргумент	Описание
cursorname	Имя открытого курсора. Курсор с таким именем должен быть предварительно объявлен с помощью оператора DECLARE VARIABLE .

Описание:

Оператор CLOSE закрывает открытый курсор. Любые все ещё открытые курсоры будут автоматически закрыты после выполнения кода триггера, хранимой процедуры или выполняемого блока, в пределах кода которого он был открыт. Оператор CLOSE применим только к курсорам, объявленным в операторе [DECLARE VARIABLE](#).

Примеры:

Пример использования оператора CLOSE.

```
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;
```

Руководство по языку SQL СУБД Firebird

```
IF (ROW_COUNT = 0) THEN
    LEAVE;
SUSPEND;
END
CLOSE C;
END^

SET TERM ;^
```

IN AUTONOMOUS TRANSACTION

Выполнение оператора или блока операторов в автономной транзакции.

Доступно: PSQL

Синтаксис:

```
IN AUTONOMOUS TRANSACTION DO <compound_statement>
```

Таблица 7.13. IN AUTONOMOUS TRANSACTION

Аргумент	Описание
compound_statement	Оператор или блок операторов.

Описание:

Оператор `IN AUTONOMOUS TRANSACTION` позволяет выполнить оператор или блок операторов в автономной транзакции. Код, работающий в автономной транзакции, будет подтверждаться сразу же после успешного завершения независимо от состояния родительской транзакции. Это бывает нужно, когда определенные действия не должны быть отменены, даже в случае возникновения ошибки в родительской транзакции.

Автономная транзакция имеет тот же уровень изоляции, что и родительская транзакция. Любое исключение, вызванное или появившееся в блоке кода автономной транзакции, приведет к откату автономной транзакции и отмене всех внесенных изменений. Если код будет выполнен успешно, то автономная транзакция будет подтверждена.

Примеры:

Использование автономной транзакции в триггере на событие подключения к базе данных для регистрации всех попыток соединения, в том числе и неудачных.

```
CREATE TRIGGER TR_CONNECT ON CONNECT
AS
BEGIN
```

Руководство по языку SQL СУБД Firebird

```
-- Все попытки соединения с БД сохраняем в журнал
IN AUTONOMOUS TRANSACTION DO
  INSERT INTO LOG(MSG)
    VALUES ('USER ' || CURRENT_USER || ' CONNECTS. ');
IF (CURRENT_USER IN (SELECT
                        USERNAME
                        FROM
                          BLOCKED_USERS)) THEN

BEGIN
  -- Сохраняем в журнал, что попытка соединения
  -- с БД оказалась неудачной
  -- и отправляем сообщение о событии
  IN AUTONOMOUS TRANSACTION DO
    BEGIN
      INSERT INTO LOG(MSG)
        VALUES ('USER ' || CURRENT_USER || ' REFUSED. ');
      POST_EVENT 'CONNECTION ATTEMPT' || ' BY BLOCKED
USER!';
    END
    -- теперь вызываем исключение
    EXCEPTION EX_BADUSER;
  END
END
```

См. также: [Управление транзакциями](#)

EXCEPTION

Возбуждение пользовательского исключения или повторный вызов исключения.

Доступно: PSQL

Синтаксис:

```
EXCEPTION [exception_name [custom_message]]
```

Таблица 7.14. Аргументы EXCEPTION

Аргумент	Описание
<code>exception_name</code>	Имя исключения.
<code>custom_message</code>	Альтернативный текст сообщения, выдаваемый при возникновении исключения. Максимальная длина текстового сообщения составляет 1021 байт.

Описание:

Руководство по языку SQL СУБД Firebird

Оператор **EXCEPTION** возбуждает пользовательское исключение с указанным именем. При возбуждении исключения можно также указать альтернативный текст сообщения, который заменит текст сообщения заданным при создании исключения.

Исключение может быть обработано в операторе `IF`. Если пользовательское исключение не было обработано в триггере или в хранимой процедуре, то все выполненные действия отменяются, вызвавшая программа получает текст, заданный при создании исключения или альтернативный текст сообщения.

В блоке обработки исключений (и только в нём), вы можете повторно вызвать пойманное исключение или ошибку, вызывая оператор **EXCEPTION** без параметров. Вне блока с исключением такой вызов не имеет никакого эффекта.

Примеры:

1. Вызов исключения по условию в хранимой процедуре SHIP_ORDER.

```
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;

    /* Этот заказ уже отправлен на поставку. */
    IF (ord_stat = 'shipped') THEN
        EXCEPTION order_already_shipped;
    /* Другие операторы */
END
```

2. Вызов исключения по условию с заменой исходного сообщения альтернативным сообщением.

Руководство по языку SQL СУБД Firebird

```
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;

    /* Этот заказ уже отправлен на поставку. */
    IF (ord_stat = 'shipped') THEN
        EXCEPTION order_already_shipped 'Order status is "' ||
ord_stat || '"';
    /* Другие операторы */
END
```

3. Регистрация ошибки в журнале и повторное её возбуждение в блоке WHEN.

```
CREATE PROCEDURE ADD_COUNTRY (
    ACountryName COUNTRYNAME,
    ACurrency VARCHAR(10) )
AS
BEGIN
    INSERT INTO country (country,
                        currency)
    VALUES (:ACountryName,
            :ACurrency);
    WHEN ANY DO
    BEGIN
        -- Записываем ошибку в журнал
        IN AUTONOMOUS TRANSACTION DO
            INSERT INTO ERROR_LOG (PSQL_MODULE,
                                   GDS_CODE,
                                   SQL_CODE,
                                   SQL_STATE)
            VALUES ('ADD_COUNTRY',
```

Руководство по языку SQL СУБД Firebird

```
        GDSCODE,  
        SQLCODE,  
        SQLSTATE) ;  
    -- Повторно возбуждаем ошибку  
    EXCEPTION;  
END  
END
```

См. также: [CREATE EXCEPTION, WHEN ... DO](#)

WHEN ... DO

Обработка ошибок.

Доступно: PSQL

Синтаксис:

```
WHEN {<error> [, <error> ...] | ANY}  
DO <compound_statement>
```

```
<error> ::= {  
    EXCEPTION exception_name  
    | SQLCODE number  
    | GDSCODE errcode}
```

Таблица 7.15. Параметры оператора WHEN ... DO

Аргумент	Описание
exception_name	Имя исключения.
number	Код ошибки SQLCODE.
errcode	Символическое имя ошибки GDSCODE.
compound_statement	Оператор или блок операторов.

Описание:

Оператор WHEN ... DO используется для обработки ошибочных ситуаций и пользовательских исключений. Оператор перехватывает все ошибки и пользовательские исключения, перечисленные после ключевого слова WHEN. Если после ключевого слова WHEN указано ключевое слово ANY, то оператор перехватывает любые ошибки и пользовательские исключения, даже если они уже были обработаны в вышестоящем WHEN блоке.

Оператор WHEN ... DO должен находиться в самом конце блока операторов перед оператором END.

После ключевого слова DO следует оператор или блок операторов, заключенных в операторные скобки BEGIN и END, которые выполняют

Руководство по языку SQL СУБД Firebird

некоторую обработку возникшей ситуации. В этом операторе (блоке операторов) доступны контекстные переменные SQLCODE, GDSCODE, SQLSTATE. Там же разрешен оператор повторного вызова ошибки или исключительной ситуации EXCEPTION (без параметров).

Внимание:

После предложения WHEN GDSCODE вы должны использовать символьные имена — такие, как grant_obj_notfound и т.д. Но в операторе или блоке операторов после предложения DO доступна контекстная переменная GDSCODE, которая содержит целое число. Для сравнения его с определенной ошибкой Вы должны использовать числовое значение, например, 335544551 для grant_obj_notfound.

Оператор WHEN ... DO вызывается только в том случае, если произошло одно из указанных в его условии событий. В случае выполнения оператора (даже если в нем фактически не было выполнено никаких действий) ошибка или пользовательское исключение не прерывает и не отменяет действий триггера или хранимой процедуры, где был выдан этот оператор, работа продолжается, как если бы никаких исключительных ситуаций не было.

Оператор перехватывает ошибки и исключения в текущем блоке операторов. Он также перехватывает подобные ситуации во вложенных блоках, если эти ситуации не были в них обработаны.

Примеры:

1. Замена стандартной ошибки своей.

```
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 'Такая страна уже добавлена!';
END^
SET TERM ^;
```

2. Регистрация ошибки в журнале и повторное её возбуждение в блоке WHEN.

Руководство по языку SQL СУБД Firebird

```
CREATE PROCEDURE ADD_COUNTRY (
    ACountryName COUNTRYNAME,
    ACurrency VARCHAR(10) )
AS
BEGIN
    INSERT INTO country (country,
                        currency)
    VALUES (:ACountryName,
            :ACurrency);
    WHEN ANY DO
    BEGIN
        -- Записываем ошибку в журнал
        IN AUTONOMOUS TRANSACTION DO
            INSERT INTO ERROR_LOG (PSQL_MODULE,
                                   GDS_CODE,
                                   SQL_CODE,
                                   SQL_STATE)
            VALUES ('ADD_COUNTRY',
                    GDSCODE,
                    SQLCODE,
                    SQLSTATE);
        -- Повторно возбуждаем ошибку
    EXCEPTION;
    END
END
```

3. Обработка в одном WHEN ... DO блоке нескольких ошибок

Пример:

```
...
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
...
```

См. также: [EXCEPTION](#), [Коды ошибок SQLSTATE и их описание](#), [Коды ошибок GDSCODE их описание](#), и [SQLCODE, GDSCODE, SQLCODE, SQLSTATE](#)

POST_EVENT

Посылка события (сообщения) клиентским приложениям.

Доступно: PSQL

Синтаксис:

```
POST_EVENT event_name;
```

Таблица 7.16. Параметры оператора POST_EVENT

Аргумент	Описание
event_name	Имя события, ограничено 64 символами.

Описание:

Оператор POST_EVENT сообщает о событии менеджеру событий, который сохраняет его в таблице событий. При подтверждении транзакции менеджер событий информирует приложения, ожидающие это событие.

В качестве имени события может быть использован строковый литерал, переменная или любое правильное SQL выражение.

Примеры:

1. Оповещение приложения о вставке записи в таблицу SALES

```
SET TERM ^;  
CREATE TRIGGER POST_NEW_ORDER FOR SALES  
ACTIVE AFTER INSERT POSITION 0  
AS  
BEGIN  
    POST_EVENT 'new_order';  
END^  
SET TERM ;^
```

Контекстные переменные

CURRENT_CONNECTION

Данная переменная хранит уникальный идентификатор текущего соединения. Тип значения `INTEGER`.

Доступно: DSQL, PSQL

Описание:

Значение переменной хранится в странице заголовка базы и сбрасывается после `restore`. Начиная с версии FB 2.1, переменная увеличивается на единицу при каждом последующем соединении с базой данных. Следовательно, переменная показывает количество подключений произошедших к базе после ее восстановления (или после ее создания).

Пример:

```
SELECT CURRENT_CONNECTION FROM RDB$DATABASE
```

CURRENT_DATE

Переменная возвращает текущую дату сервера. Тип значения `DATE`.

Доступно: DSQL, PSQL, ESQL.

Синтаксис:

```
CURRENT_DATE
```

Пример:

```
CREATE DOMAIN DDATE_DNN AS  
DATE DEFAULT CURRENT_DATE NOT NULL
```

См. также: `'TODAY'`, `CURRENT_TIMESTAMP`, `CURRENT_TIME`.

CURRENT_ROLE

Данная контекстная переменная служит для определения роли, с которой произошло подключение к базе данных. Тип значения `VARCHAR(31)`. В случае если произошло подключение без указания роли, переменная принимает

Руководство по языку SQL СУБД Firebird

значение NONE.

Доступно: DSQL, PSQL.

Пример:

```
SELECT CURRENT_ROLE FROM RDB$DATABASE
```

Примечание:

Такое же значение можно будет получить и в результате выполнения запроса:

```
SELECT RDB$GET_CONTEXT ('SYSTEM', 'CURRENT_ROLE')  
FROM RDB$DATABASE;
```

См. также: [RDB\\$GET_CONTEXT \(\)](#).

CURRENT_TIME

Данная переменная возвращает текущее время сервера. Тип значения TIME.

В Firebird до версии 2.0 дробная часть всегда была '.0000', то есть была точность до **секунды**. Начиная с FB 2.0, вы можете определять точность в запросе значения этой переменной. Значения по умолчанию – 0, то есть точность до 1-й секунды.

Доступно: DSQL, PSQL, ESQL. В ESQL не поддерживается параметр <точность>.

Синтаксис:

```
CURRENT_TIME [ (<точность>) ],  
где <точность> ::= 0|1|2|3
```

Пример:

```
SELECT CURRENT_TIME(2) FROM RDB$DATABASE;  
-- результат будет (например) 23:35:33.1200
```

В блоке кода PSQL (процедура, триггер, исполняемый блок) значение CURRENT_TIME не меняется по мере выполнения. При вызове вложенного кода, значение также **не изменится** и будет равно значению в коде самого верхнего уровня. Для определения реального времени используйте другие переменные, например NOW (с полным приведением типа данных).

См. также: ['NOW'](#), [CURRENT_TIMESTAMP](#), [CURRENT_TIME](#),

[CURRENT_DATE](#).

CURRENT_TIMESTAMP

Переменная возвращает текущую дату и время сервера. Тип значения `TIMESTAMP`.

В версиях Firebird до 2.0 дробная часть всегда была `'.0000'`. Начиная с версии 2.0 можно указать требуемую точность для данной переменной. Значением по умолчанию является 3 цифры после запятой (точность до одной миллисекунды).

Доступно: DSQL, PSQL, ESQL. В ESQL не поддерживается параметр `<точность>`.

Синтаксис:

```
CURRENT_TIMESTAMP [ (<точность> ) ],  
где <точность> ::= 0|1|2|3
```

В модуле PSQL значение `CURRENT_TIMESTAMP` остается постоянной до момента полного исполнения кода независимо от того, когда Вы его считаете. При вызове вложенного кода, значение также **не изменится** и будет равно значению в коде самого верхнего уровня. Для определения реального времени используйте другие переменные, например `NOW` (с полным приведением типа данных).

См. также: ['NOW'](#), [CURRENT_TIME](#), [CURRENT_DATE](#).

CURRENT_TRANSACTION

Данная переменная содержит уникальный номер текущей транзакции. Тип значения `INTEGER`.

Доступно: DSQL, PSQL.

Примеры:

```
SELECT CURRENT_TRANSACTION FROM RDB$DATABASE;
```

```
NEW.TRANS_ID = CURRENT_TRANSACTION;
```

Значение `CURRENT_TRANSACTION` хранится в странице заголовка базы данных и сбрасывается в 0 после восстановления (или создания базы). Оно увеличивается при старте новой транзакции.

Примечание:

Такое же значение можно будет получить и в результате выполнения запроса:

```
SELECT RDB$GET_CONTEXT ('SYSTEM', 'CURRENT_ROLE')
FROM RDB$DATABASE;
```

См. также: [RDB\\$GET_CONTEXT \(\)](#).

CURRENT_USER

Переменная содержит имя текущего подключенного пользователя базы данных. Тип значения VARCHAR(31).

Доступно: DSQL, PSQL.

Пример:

```
NEW.ADDED_BY = CURRENT_USER;
```

См. также: [USER](#)

DELETING

Контекстная переменная DELETING доступна только в коде триггеров. Тип значения логический (boolean).

Может использоваться для триггеров на различные события (несколько типов событий) и показывает, что триггер сработал при выполнении операции DELETE.

Доступно: PSQL.

Пример:

```
...
IF (DELETING) THEN
BEGIN
  INSERT INTO REMOVED_CARS (
    ID, MAKE, MODEL, REMOVED)
  VALUES (
    OLD.ID, OLD.MAKE, OLD.MODEL, CURRENT_TIMESTAMP);
END
...
```

См. также: [INSERTING](#), [UPDATING](#).

GDSCODE

В блоке обработки ошибок "WHEN ... DO" контекстная переменная GDSCODE содержит числовое представление текущего кода ошибки Firebird. До версии Firebird 2.0 GDSCODE можно было получить только с использованием конструкции WHEN GDSCODE. Теперь эту контекстную переменную можно также использовать в блоках WHEN ANY, WHEN SQLCODE и WHEN EXCEPTION при условии, что код ошибки соответствует коду ошибки Firebird. Вне обработчика ошибок GDSCODE всегда равен 0. Вне PSQL GDSCODE не существует вообще. Тип значения INTEGER.

Доступно: PSQL

Пример:

```
...
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
...
```

Обратите внимание, пожалуйста: после, WHEN GDSCODE Вы должны использовать *символьные имена* — такие, как grant_obj_notfound и т.д. Но контекстная переменная GDSCODE - *целое число*. Для сравнения его с определенной ошибкой Вы должны использовать числовое значение, например, 335544551 для grant_obj_notfound.

См. также: [SQLCODE](#), [SQLSTATE](#).

INSERTING

Контекстная переменная INSERTING доступна только коде триггеров.

Тип значения логический (boolean).

Может использоваться для триггеров на различные события (несколько типов событий) и показывает, что триггер сработал при выполнении операции INSERT.

Руководство по языку SQL СУБД Firebird

Доступно: PSQL.

Пример:

```
...
IF (INSERTING OR UPDATING) THEN
BEGIN
  IF (NEW.SERIAL_NUM IS NULL) THEN
    NEW.SERIAL_NUM = GEN_ID (GEN_SERIALIZS, 1);
END
...
```

См. Также: [DELETING](#), [UPDATING](#).

NEW

Контекстная переменная `NEW` доступна только в коде триггеров. Значение `NEW` содержит новое значение данных, которое возникло в базе во время операции обновления или вставки.

Тип значения зависит от столбца, который она обозначает в коде триггера.

Начиная с версии Firebird 2.0 в `AFTER` триггерах переменная доступна только для чтения.

Доступно: PSQL.

Замечание:

Для триггеров на несколько типов событий переменная `NEW` доступна всегда. Однако в случае если триггер сработал на операцию удаления, то для него новая версия данных не имеет смысла. В этой ситуации чтение переменной `NEW` всегда вернет `NULL`.

Внимание:

Попытка записи в переменную `NEW` в `AFTER` триггере вызовет исключение в коде.

См. Также: [OLD](#).

'NOW'

Не является переменной, а является строковым литералом. При использовании преобразования типов данных, например, с помощью функции `CAST()` в тип даты/времени позволяет получить текущую дату и/или время. Значение «после запятой» у переменной показывают число миллисекунд.

Руководство по языку SQL СУБД Firebird

Начиная с версии Firebird 2.0 точность составляет 3 знака после запятой (миллисекунды). Написание 'NOW' зависит от регистра, при преобразовании в дату функция игнорирует все пробелы слева и справа от слова.

Тип возвращаемого значения CHAR(3).

Доступно: DSQL, PSQL, ESQL.

Примеры:

```
select cast('Now' as date) from rdb$database;  
-- возвратит, например 2014-10-03
```

```
select cast('now' as time) from rdb$database;  
-- возвратит, например 14:20:19.6170
```

```
select cast('NOW' as timestamp) from rdb$database;  
-- возвратит, например 2014-10-03 14:20:19.6170
```

Примечание:

Поскольку 'NOW' всегда возвращает актуальные значения даты и времени при использовании CAST() для приведения типов данных она может использоваться для измерения временных интервалов и скорости выполнения кода в процедурах, триггерах и блоках кода PSQL. Будьте внимательны при использовании 'NOW', т.к. использование сокращенного преобразования типов 'NOW' оценивается во время синтаксического анализа, а затем время ее «замораживается», даже при многократном выполнении кода.

См. Также: CURRENT_DATE, CURRENT_TIME, 'TODAY', 'YESTERDAY', CURRENT_TIMESTAMP.

OLD

Контекстная переменная OLD доступна только коде триггеров. Значение, содержащееся в OLD, обозначает прошлое значение данных, которое было в базе до операции изменения или удаления.

Тип значения зависит от столбца, который она обозначает в коде триггера.

Начиная с версии Firebird 2.0, переменная доступна только для чтения.

Доступно: PSQL.

Замечание:

Для триггеров, работающих на несколько типов событий, введенных в Firebird

1.5, значения для переменной `OLD` всегда возможны. Однако для триггеров, сработавших на вставку записи, значение данной переменной не имеет смысла, поэтому в этой ситуации чтение `OLD` возвратит `NULL`, а попытка записи в нее вызовет исключение в коде.

См. также: [NEW](#).

ROW_COUNT

Контекстная переменная `ROW_COUNT` содержит число строк, затронутых последним оператором DML (`INSERT`, `UPDATE`, `DELETE`, `SELECT` или `FETCH`) в текущем триггере, хранимой процедуре или исполняемом блоке.

Тип значения `INTEGER`.

Доступно: `PSQL`.

Пример:

```
update Figures set Number = 0 where id = :id;
if (row_count = 0) then
  insert into Figures (id, Number) values (:id, 0);
```

Поведение с `SELECT` и `FETCH`:

- После выполнения `singleton SELECT` запроса (запроса, который может вернуть не более одной строки данных), `ROW_COUNT` равна 1, если была получена строка данных и 0 в противном случае;
- В цикле `FOR SELECT` переменная `ROW_COUNT` увеличивается на каждой итерации (начиная с 0 в качестве первого значения);
- После выборки (`FETCH`) из курсора, `ROW_COUNT` равна 1, если была получена строка данных и 0 в противном случае. Выборка нескольких записей из одного курсора *не* увеличивает `ROW_COUNT` после 1;
- В Firebird 1.5 `ROW_COUNT` всегда равна 0 после выполнения любого оператора `SELECT`.

Внимание!

Переменная `ROW_COUNT` не может быть использована для определения количества строк, затронутых при выполнении операторов `EXECUTE STATEMENT` или `EXECUTE PROCEDURE`. Для оператора `MERGE` переменная `ROW_COUNT` будет содержать 0 или 1, даже если было затронуто более записей.

SQLCODE

В блоках обработки ошибок `"WHEN ... DO"` контекстная переменная

Руководство по языку SQL СУБД Firebird

SQLCODE содержит текущий код ошибки SQL. До Firebird 2.0 значение SQLCODE можно было получить только в блоках обработки ошибок WHEN SQLCODE и WHEN ANY. Теперь она может быть отлична от нуля в блоках WHEN GDSCODE и WHEN EXCEPTION при условии, что ошибка, вызвавшее срабатывание блока, соответствует коду ошибки SQL. Вне обработчиков ошибок SQLCODE всегда равен 0, а вне PSQL не существует вообще.

Тип значения INTEGER.

Доступно: PSQL. Добавлено в версии Firebird 1.5. Изменена в версии 2.0.
Устарела в 2.5.1.

Пример:

```
WHEN ANY DO
BEGIN
  IF (SQLCODE <> 0) THEN
    MSG = 'Обнаружена ошибка SQL!';
  ELSE
    MSG = 'Ошибки нет!';
  EXCEPTION EX_CUSTOM MSG;
END
```

См. Также: [GDSCODE](#), [SQLSTATE](#).

SQLSTATE

В блоках обработки ошибок "WHEN ... DO" контекстная переменная SQLSTATE переменная содержит 5 символов SQL-2003-совместимого кода состояния, переданного оператором, вызвавшим ошибку. Вне обработчиков ошибок SQLSTATE всегда равен '00000', а вне PSQL не существует вообще.

Тип значения CHAR(5).

Доступно: PSQL. Добавлено в версии 2.5.1.

Пример:

```
WHEN ANY DO
BEGIN
  MSG = CASE SQLSTATE
    WHEN '22003' THEN
      'Число вышло за пределы диапазона!'
    WHEN '22012' THEN
      'Деление на ноль!'
    WHEN '23000' THEN
      'Нарушение ограничения целостности!'
```

Руководство по языку SQL СУБД Firebird

```
ELSE 'Ошибок нет! SQLSTATE = ' || SQLSTATE;  
END;  
EXCEPTION EX_CUSTOM MSG;  
END
```

Примечания:

- SQLSTATE предназначен для замены SQLCODE. Последняя, в настоящее время устарела и будет удалена в будущих версиях Firebird;
- Firebird не поддерживает (пока) синтаксис "WHEN SQLSTATE ... DO". Используйте обработчик блока WHEN ANY для проверки значения переменной SQLSTATE;
- Любой код SQLSTATE состоит из двух символов класса и трёх символов подкласса. Класс 00 (успешное выполнение), 01 (предупреждение) и 02 (нет данных) представляют собой *условия завершения*. Каждый код статуса вне этих классов является *исключением*. Поскольку классы 00, 01 и 02 не вызывают ошибку, они никогда не будут обнаруживаться в переменной SQLSTATE.

Полный список кодов SQLSTATE приведён в Приложении 2 **Ошибка!**
источник ссылки не найден..

См. Также: [GDSCODE](#), [SQLCODE](#).

'TODAY'

Не является переменной, а является строковым литералом. При использовании преобразования типов данных, например, с помощью функции CAST() в тип даты/времени позволяет получить текущую дату. Написание 'TODAY' не зависит от регистра, при преобразовании в дату функция игнорирует все пробелы слева и справа от слова.

Тип возвращаемого значения CHAR(5).

Доступно: DSQL, PSQL, ESQL.

Примеры:

```
select cast('Today' as date) from rdb$database;  
-- возвратит, например 2014-10-03  
select cast('TODAY' as timestamp) from rdb$database;  
-- возвратит, например 2014-10-03 00:00:00.0000  
  
-- при использовании короткого, «C-Style» преобразования:  
select date 'Today' from rdb$database;  
select timestamp 'TODAY' from rdb$database;
```

См. также: 'TOMORROW', 'YESTERDAY', 'NOW'.

Руководство по языку SQL СУБД Firebird

'TOMORROW'

Не является переменной, а является строковым литералом. При использовании преобразования, например, с помощью функции `CAST()` в тип даты / времени позволяет получить дату, следующую за текущей.

Тип возвращаемого значения `CHAR(8)`.

Доступно: DSQL, PSQL, ESQL.

Примеры:

```
select cast('Tomorrow' as date) from rdb$database;
-- возвратит, например 2014-10-04

select cast('TOMORROW' as timestamp) from rdb$database;
-- возвратит, например 2014-10-04 00:00:00.0000

-- при использовании короткого, «C-Style» преобразования:
select date 'Tomorrow' from rdb$database;
```

См. также: ['TODAY'](#), ['YESTERDAY'](#), ['NOW'](#).

UPDATING

Контекстная переменная `UPDATING` доступна только коде триггеров.

Тип значения логический (`boolean`).

Доступно: PSQL. Добавлена в версии 1.5.

Может использоваться для триггеров на различные события (несколько типов событий) и показывает, что триггер сработал при выполнении операции `UPDATE`.

Доступна для PSQL. Переменная добавлена в версии 1.5.

См. Также: [DELETING](#), [INSERTING](#).

'YESTERDAY'

Не является переменной, а является строковым литералом. При использовании преобразования, например, с помощью функции `CAST()` в тип даты / времени позволяет получить дату, которая была день назад.

Руководство по языку SQL СУБД Firebird

Тип возвращаемого значения CHAR (9).

Доступно: DSQL, PSQL, ESQL.

Примеры:

```
select cast('Yesterday' as date) from rdb$database;  
--возвратит, например 2014-10-04
```

```
-- при использовании короткого, «C-Style» преобразования  
select date 'Yesterday' from rdb$database;  
select timestamp 'YESTERDAY' from rdb$database;
```

См. также: ['TODAY'](#), ['TOMORROW'](#), ['YESTERDAY'](#).

USER

Переменная содержит имя текущего подключенного пользователя базы данных.

Тип значения VARCHAR (31).

Доступно: DSQL, PSQL.

Пример:

```
NEW.CUR_USER = USER;
```

См. также: [CURRENT_USER](#)

Функции для работы с контекстными переменными

Доступны: DSQL, PSQL.

Функции `RDB$GET_CONTEXT()` и `RDB$SET_CONTEXT()` используются, соответственно для чтения и для записи (там, где это возможно) значений контекстных переменных. Эти функции были добавлены в Firebird в версии 2.0. В настоящее время они реализованы в виде встроенных UDF (то есть для их вызова не требуется писать в базе их объявление!).

RDB\$GET_CONTEXT ()

Функция возвращает значение контекстной переменной из одного из пространства имен – `SYSTEM`, `USER_SESSION` или `USER_TRANSACTION`.

Руководство по языку SQL СУБД Firebird

Тип возвращаемого значения VARCHAR (255).

Синтаксис записи:

```
RDB$GET_CONTEXT ('<name_space>', '<varname>')
```

```
<name_space> ::= SYSTEM | USER_SESSION | USER_TRANSACTION
```

```
<varname> ::= имя переменной.
```

Имя переменной – это строка, зависящая от регистра максимальной длиной 80 символов. Пространство имен SYSTEM – только для чтения.

Пространства имен USER_SESSION и USER_TRANSACTION – изначально пусты и пользователь сам создает переменные и наполняет их при помощи функции RDB\$SET_CONTEXT.

Имена переменных доступных только для чтения из пространства SYSTEM приведены в таблице 8.1.

Таблица 8.1. Переменные пространства SYSTEM.

Переменная	Описание	Примечание
DB_NAME	Полный путь к базе данных или алиас к базе данных, из строки подключения к базе данных	
NETWORK_PROTOCOL	Протокол, используемый для соединения с базой данных	Возможные значения: «TCPv4», «WNET», «XNET», NULL
CLIENT_ADDRESS	Для TCPv4 – IP адрес, для XNET – локальный ID процесса. Для остальных случаев NULL	
CURRENT_USER	Глобальная переменная CURRENT_USER	Смотри
CURRENT_ROLE	Глобальная переменная CURRENT_ROLE	Смотри
SESSION_ID	Глобальная переменная CURRENT_CONNECTION	Смотри CURRENT_CONNECTION
TRANSACTION_ID	Глобальная переменная CURRENT_TRANSACTION	Смотри
ISOLATION_LEVEL	Уровень изоляции текущей транзакции - CURRENT_TRANSACTION	Значения: 'READ_COMMITED', 'SNAPSHOT' или 'CONSISTENCY'

Руководство по языку SQL СУБД Firebird

ENGINE_VERSION	Версия сервера Firebird.	Добавлено в версии 2.1
CLIENT_PID	PID процесса на клиентском компьютере	Добавлено в версии 2.5.3
CLIENT_PROCESS	Полный путь к клиентскому приложению, подключившемуся к базе данных. Позволяет не использовать системную таблицу MON\$ATTACHMENTS (поле MON\$REMOTE_PROCESS)	Добавлено в версии 2.5.3
LOCK_TIMEOUT	время ожидания транзакцией высвобождения ресурса при блокировке, в секундах	Добавлено в версии 2.5.3
READ_ONLY	отображает является ли транзакция, транзакцией только для чтения FALSE для R/W транзакций TRUE для Readonly	Добавлено в версии 2.5.3

Если запрашиваемая функцией переменная существует в указанном пространстве имен, то будет возвращено ее значение в виде строки VARCHAR(255). При обращении к несуществующей переменной в пространстве 'SYSTEM' возникает ошибка, если такое происходит с другими пространствами имен – функция возвращает NULL.

Еще раз обратите внимание на то, что пространства имен и имена переменных регистрочувствительны, должны быть непустыми строками, и заключены в кавычки!

Пример:

```
NEW.USER_ADR = RDB$GET_CONTEXT ('SYSTEM', 'CLIENT_ADDRESS');
```

RDB\$SET_CONTEXT()

Функция создает, устанавливает значение или обнуляет переменную в одном из используемых пользователем пространстве имен: USER_SESSION или USER_TRANSACTION.

Тип возвращаемого значения INTEGER.

Функция возвращает 1, если переменная уже существовала до вызова и 0, если не существовала. Для удаления переменной надо установить её значение

Руководство по языку SQL СУБД Firebird

в NULL. Если данное пространство имен не существует, то функция вернёт ошибку. Пространство имен и имя переменной зависят от регистра, должны быть не пустыми строками, и заключены в кавычки.

Синтаксис записи:

```
RDB$SET_CONTEXT ('<name_space>', '<varname>', <value>|NULL)
```

```
<name_space> ::= USER_SESSION | USER_TRANSACTION
```

```
<varname> ::= имя переменной (регистрозависимое)
```

```
<value> ::= данные любого типа при условии, что их можно  
привести к типу VARCHAR(255).
```

Примеры:

```
SELECT RDB$SET_CONTEXT ('USER_SESSION', 'DEBUGL', 3)  
FROM RDB$DATABASE;
```

```
-- в PSQL доступен такой синтаксис  
RDB$SET_CONTEXT ('USER_SESSION', 'RECORDSFOUND',  
                RECCOUNTER);
```

```
SELECT RDB$SET_CONTEXT ('USER_TRANSACTION', 'SAVEPOINTS',  
                        'YES')  
FROM RDB$DATABASE;
```

Примеры работы с функциями для контекстных переменных:

```
SET TERM ^;  
create procedure set_context (User_ID varchar(40),  
                             Trn_ID integer) as  
begin  
  RDB$SET_CONTEXT ('USER_TRANSACTION', 'Trn_ID', Trn_ID);  
  RDB$SET_CONTEXT ('USER_TRANSACTION', 'User_ID', User_ID);  
end^  
SET TERM ;^
```

```
create table journal (  
  jrn_id integer not null primary key,  
  jrn_lastuser varchar(40),  
  jrn_lastaddr varchar(255),  
  jrn_lasttran integer  
);
```

```
SET TERM ^;  
CREATE TRIGGER UI_JOURNAL
```

Руководство по языку SQL СУБД Firebird

```
FOR JOURNAL BEFORE INSERT OR UPDATE
as
begin
  new.jrn_lastuser = rdb$get_context('USER_TRANSACTION',
                                     'User_ID');
  new.jrn_lastaddr = rdb$get_context('SYSTEM',
                                     'CLIENT_ADDRESS');
  new.jrn_lasttran = rdb$get_context('USER_TRANSACTION',
                                     'Trn_ID');

end^
SET TERM ;^

execute procedure set_context('skidder', 1);

insert into journal(jrn_id) values(0);

commit;
```

Примечания:

Пространство имен SYSTEM доступно только для чтения!

- Максимальное число переменных в рамках одного соединения равно **1000**;
- Все переменные в пространстве имён USER_TRANSACTION сохраняются при ROLLBACK RETAIN или ROLLBACK TO SAVEPOINT, независимо от того, в какой точке во время выполнения транзакции они были установлены.

Скалярные функции

Важно:

Если в Вашей базе данных имя декларированной внешней функции совпадает с именем встроенной, то **будет вызываться внешняя функция**. Для того, чтобы была доступна внутренняя функция, необходимо удалить или изменить внешнюю функцию (UDF).

ABS ()

Функция возвращает абсолютное значение аргумента. Тип возвращаемого значения числовой.

Синтаксис:

Руководство по языку SQL СУБД Firebird

ABS (< число >)

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

ACOS ()

Функция возвращает арккосинус аргумента. Тип возвращаемого значения DOUBLE PRECISION.

Синтаксис:

ACOS (< число >)

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

В случае если аргумент функции вне границы диапазона [-1, 1], то функция вернет неопределенное значения NaN.

См. Также: [COS \(\)](#).

ASCII_CHAR ()

Функция возвращает ASCII символ соответствующий номеру, переданному в качестве аргумента. Тип возвращаемого значения [VAR]CHAR (1) CHARSET NONE .

Синтаксис:

ASCII_CHAR (< код >)

где < код > – целое число в диапазоне [0 .. 255]

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

ASCII_VAL ()

Функция возвращает ASCII код символа, переданного в качестве аргумента. Тип возвращаемого значения SMALLINT.

Синтаксис:

ASCII_VAL (< ch >)

где < ch > – строка типа данных [VAR]CHAR или текстовый BLOB максимального размера 32767 байт.

Руководство по языку SQL СУБД Firebird

- Если строка содержит более одного символа, то возвращается код первого символа строки
- Если строка пустая, возвращается ноль
- Если аргумент `NULL`, то возвращаемое значение также `NULL`

Функция добавлена в Firebird в версии 2.1.4. Доступна для DSQL, PSQL.

ASIN ()

Функция возвращает арксинус аргумента. Тип возвращаемого значения `DOUBLE PRECISION`.

Синтаксис:

`ASIN (< число >)`

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

В случае если аргумент функции вне границы диапазона `[-1, 1]`, то функция вернет неопределенное значения `NaN`.

См. Также: [SIN \(\)](#).

ATAN ()

Функция возвращает арктангенс аргумента. Тип возвращаемого значения `DOUBLE PRECISION`.

Синтаксис:

`ATAN (< число >)`

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

Функция возвращает угол в диапазоне `< - π/2 и π/2 >` радиан.

См. Также: [ATAN2 \(\)](#), [TAN \(\)](#).

ATAN2 ()

Функция возвращает угол как *отношение* синуса к косинусу, аргументы, у которых задаются этими двумя параметрами, а *знаки* синуса и косинуса соответствуют знакам параметров. Это позволяет получать результаты по всей окружности, включая углы `- π/2` и `π/2`. Тип возвращаемого значения `DOUBLE PRECISION`.

Руководство по языку SQL СУБД Firebird

Синтаксис:

ATAN2 (y, x)

- Результат - угол в диапазоне $[-\pi, \pi]$ радиан;
- Если x отрицательный, то при нулевом значении y результат равен π , а при значении -0 равен $-\pi$;
- Если и y и x равны 0, то результат бессмыслен.

Примечания:

- Полностью эквивалентное описание этой функции следующее: ATAN2 (y, x) является углом между положительной осью X и линией от начала координат до точки (x, y). Это также делает очевидным, что значение ATAN2 (0, 0) не определено;
- Если x больше, чем 0, ATAN2 (y, x) совпадает с ATAN (y/x);
- Если известны синус и косинус угла, то ATAN2 (SIN, COS) возвращает угол.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

BIN_AND ()

Функция возвращает результат побитовой операции AND (И) аргументов. Тип возвращаемого значения INTEGER или BIGINT.

Синтаксис:

BIN_AND (number [, number ...])

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

BIN_OR ()

Функция возвращает результат побитовой операции OR (ИЛИ) аргументов. Тип возвращаемого значения INTEGER или BIGINT.

Синтаксис:

BIN_OR (number [, number ...])

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

BIN_SHL ()

Функция возвращает первый параметр, побитно смещенный влево на значение второго параметра. Тип возвращаемого значения `BIGINT`.

Синтаксис:

```
BIN_SHL (number, shift)
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

BIN_SHR ()

Функция возвращает первый параметр, побитно смещенный вправо на значение второго параметра. Тип возвращаемого значения `BIGINT`.

Синтаксис:

```
BIN_SHR (number, shift)
```

- Выполняемая операция является арифметическим сдвигом вправо (SAR), а это означает, что знак первого операнда всегда сохраняется.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

BIN_XOR ()

Функция возвращает результат побитовой операции XOR аргументов. Тип возвращаемого значения `INTEGER` или `BIGINT`.

Синтаксис:

```
BIN_XOR (number [, number ...])
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

BIT_LENGTH ()

Функция возвращает длину входной строки в битах. Тип возвращаемого значения `INTEGER`. Для многобайтных наборов символов результат может быть в 8 раз меньше, чем количество символов в "формальном" числе байт на символ, записанном в `RDB$CHARACTER_SETS`.

Синтаксис:

Руководство по языку SQL СУБД Firebird

`BIN_LENGTH (< строка >)`

С параметрами типа `CHAR` эта функция берет во внимание всю формальную строковую длину (например, объявленная длина поля или переменной). Если Вы хотите получить "логическую" длину в битах, не считая пробелов, то перед передачей аргумента в `BIN_LENGTH` надо выполнить над ним операцию `RIGHT TRIM`.

Примеры:

```
SELECT BIT_LENGTH ('Hello!') FROM RDB$DATABASE
-- возвращает 48
```

```
SELECT BIT_LENGTH (_ISO8859_1 'Grüß Di!')
FROM RDB$DATABASE
-- возвращает 64: каждый, и ü, и ß занимают один байт в
ISO8859_1
```

```
SELECT BIT_LENGTH (
    CAST (_ISO8859_1 'Grüß di!' AS VARCHAR (24)
    CHARACTER SET UTF8))
FROM RDB$DATABASE
-- возвращает 80: каждый, и ü, и ß занимают по два байта в
UTF8
```

```
SELECT BIT_LENGTH (
    CAST (_ISO8859_1 'Grüß di!' AS CHAR (24)
    CHARACTER SET UTF8))
FROM RDB$DATABASE
-- возвращает 208: размер всех 24 позиций CHAR и два из них
16-битные
```

Функция добавлена в Firebird в версии 2.0. Доступна для DSQL, PSQL.

Начиная с версии Firebird 2.1, функция полностью поддерживает текстовые BLOB любой размерности и с любым набором символов.

См. также: [OCTET_LENGTH \(\)](#), [CHAR_LENGTH \(\)](#), [CHARACTER_LENGTH \(\)](#)

CAST ()

Функция служит для явного преобразования данных из одного типа данных в другой тип данных или домен.

Синтаксис:

```
CAST (<значение> | NULL AS <тип данных>),
```

Руководство по языку SQL СУБД Firebird

где <тип данных> ::= *sql_datatype*
| [TYPE OF] <домен>
| TYPE OF COLUMN *relname.colname*

Функция добавлена в IB. Доработки в версиях Firebird 2.0, 2.1, 2.5. Доступна для DSQL, ESQL, PSQL.

Начиная с версии Firebird 2.1, функция полностью поддерживает преобразование как «из» так и «в» BLOB.

См. также: [Явное преобразование типов данных.](#)

CEIL (), CEILING ()

Функция возвращает наименьшее целое число, большее или равное аргументу. Тип возвращаемого значения BIGINT или DOUBLE PRECISION.

Синтаксис:

```
CEIL[ING] (number)
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [FLOOR \(\), TRUNC \(\)](#).

CHAR_LENGTH (), CHARACTER_LENGTH ()

Функция возвращает длину (в символах) строки, переданной в качестве аргумента. Тип возвращаемого значения INTEGER.

Синтаксис:

```
CHAR_LENGTH (< строка >)  
CHARACTER_LENGTH (< строка >)
```

Функция добавлена в Firebird в версии 2.0. Доступна для DSQL, PSQL.

Примечание:

С параметрами типа CHAR эта функция берет во внимание всю формальную строковую длину (например, объявленная длина поля или переменной). Если Вы хотите получить "логическую" длину без учёта пробелов, то перед передачей аргумента в CHAR[ACTER]_LENGTH надо выполнить над ним операцию RIGHT TRIM.

Руководство по языку SQL СУБД Firebird

В версии Firebird 2.1 добавлена полная поддержка текстовых BLOB любых размеров и с любым набором символов.

См. также: [OCTET_LENGTH \(\)](#), [BIT_LENGTH \(\)](#).

CHAR_TO_UUID ()

Функция преобразует читабельную 36-ти символьную символику UUID к соответствующему 16-ти байтовому значению UUID. Тип возвращаемого значения CHAR(16) CHARACTER SET OCTETS.

Синтаксис:

```
CHAR_TO_UUID (< ascii_uuid >)
```

Функция добавлена в Firebird в версии 2.5. Доступна для DSQL, PSQL.

См. также: [GEN_UUID \(\)](#), [UUID_TO_CHAR \(\)](#).

COALESCE ()

Функция возвращает первый NOT NULL аргумент из двух и более переданных. Если все аргументы имеют значение NULL, то и результат будет NULL.

Синтаксис:

```
COALESCE (<exp1>, <exp2> [, <expN> ...])
```

Примеры:

```
select coalesce (sum (q), 0) from bills where...  
-- в случае получения при суммировании NULL, вернет 0.
```

```
select id, coalesce (Nickname, Name, 'не определено')  
from Person;
```

Функция добавлена в Firebird в версии 1.5. Доступна для DSQL, PSQL.

См. также: [IIF \(\)](#), [NULLIF \(\)](#), [CASE](#)

COS ()

Функция возвращает косинус угла. Аргумент должен быть задан в радианах. Тип возвращаемого значения DOUBLE PRECISION.

Руководство по языку SQL СУБД Firebird

Синтаксис:

```
COS (< угол >)
```

Любой NOT NULL результат находится в диапазоне [-1, 1].

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [ACOS \(\)](#).

COSH ()

Функция возвращает гиперболический косинус аргумента. Тип возвращаемого значения DOUBLE PRECISION.

Синтаксис:

```
COSH (< число >)
```

Любой NOT NULL результат находится в диапазоне [1, +∞].

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

COT ()

Функция возвращает косинус угла. Аргумент должен быть задан в радианах. Тип возвращаемого значения DOUBLE PRECISION.

Синтаксис:

```
COT (< угол >)
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

DATEADD ()

Функция позволяет добавить заданное число лет, месяцев, недель, часов, минут, секунд, миллисекунд к заданному значению даты/времени. Тип возвращаемого значения DATE, TIME или TIMESTAMP.

Синтаксис:

```
DATEADD (<args>)
```

```
<args> ::= <amount> <unit> TO <datetime>  
         | <unit>, <amount>, <datetime>
```

Руководство по языку SQL СУБД Firebird

<amount> ::= целое выражение (отрицательное вычитается)

<unit> ::= YEAR | MONTH | WEEK | DAY
| HOUR | MINUTE | SECOND | MILLISECOND

<datetime> ::= DATE, TIME или TIMESTAMP выражение

Тип результата определяется третьим аргументом функции.

- С аргументом типа `TIMESTAMP` и `DATE` можно использовать любую составляющую даты/времени (<unit>) (до Firebird 2.5 для типа данных `DATE` было запрещено использовать приращения меньше дня);
- Для типа данных `TIME` разрешается использовать только `HOUR`, `MINUTE`, `SECOND` и `MILLISECOND`.

Примеры:

```
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')
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

Параметр `WEEK` – неделя добавлен в версии 2.5.

См. также: [DATEDIFF \(\)](#)

DATEDIFF ()

Функция возвращает количество лет, месяцев, недель, дней, часов, минут, секунд или миллисекунд между двумя значениями даты/времени. Тип возвращаемого значения `BIGINT`.

Синтаксис:

```
DATEDIFF (<args>)
```

```
<args> ::= <unit> FROM <moment1> TO <moment2>
| <unit>, <moment1>, <moment2>
```

```
<unit> ::= YEAR | MONTH | WEEK | DAY
| HOUR | MINUTE | SECOND | MILLISECOND
```

Руководство по языку SQL СУБД Firebird

<momentN> := <datetime>

<datetime> ::= DATE, TIME или TIMESTAMP выражение

- Параметры DATE и TIMESTAMP могут использоваться совместно. Совместное использование типа TIME с типами DATE и TIMESTAMP не разрешается;
- С аргументом типа TIMESTAMP и DATE можно использовать любую составляющую даты/времени (<unit>) (До Firebird 2.5 для типа данных DATE было запрещено использовать приращения меньше дня);
- Для типа данных TIME разрешается использовать только HOUR, MINUTE, SECOND и MILLISECOND.

Примечания по работе функции:

- Функция DATEDIFF не проверяет разницу в более мелких составляющих даты/времени, чем задана в первом аргументе (<unit>). В результате получаем:
- DATEDIFF (YEAR, DATE '1-JAN-2009', DATE '31-DEC-2009') = 0,
НО
- DATEDIFF (YEAR, DATE '31-DEC-2009', DATE '1-JAN-2010') = 1
- Однако для *более мелких* составляющих даты/времени имеем:
- DATEDIFF (DAY, DATE '26-JUN-1908', DATE '11-SEP-1973') = 23818
- DATEDIFF (DAY, DATE '30-NOV-1971', DATE '8-JAN-1972') = 39
- Отрицательное значение функции говорит о том, что дата/время в <moment2> меньше, чем в <moment1>.

Примеры:

```
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))
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

Параметр WEEK – неделя добавлен в версии 2.5.

См. также: [DATEADD \(\)](#)

DECODE ()

Данная функция эквивалентна конструкции «Простой CASE», в которой заданное выражение сравнивается с другими выражениями до нахождения совпадения. Результатом является значение, указанное после выражения, с которым найдено совпадение. Если совпадений не найдено, то возвращается значение по умолчанию (если оно, конечно, задано — в противном случае возвращается NULL).

Синтаксис:

```
DECODE (<test - expr>,  
        <expr >, result  
        [, <expr >, result ...]  
        [, defaultresult])
```

Эквивалентная конструкция CASE

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

Пример:

```
select name, age,  
       decode (upper (sex),  
              'M', 'М',  
              'F', 'Ж',  
              'не указано'),  
       UID  
from people
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [CASE](#)

EXP ()

Функция возвращает значение натуральной экспоненты, $e^{\langle \text{число} \rangle}$. Тип возвращаемого значения DOUBLE PRECISION.

Синтаксис:

Руководство по языку SQL СУБД Firebird

EXP (< число >)

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: LN ()

EXTRACT ()

Функция служит для извлечения составляющих даты и времени из типов данных DATE, TIME и TIMESTAMP. Тип возвращаемого значения SMALLINT или NUMERIC.

Синтаксис:

```
EXTRACT (<part> FROM <datetime>)
```

```
<part> ::= YEAR | MONTH | WEEK  
         | DAY | WEEKDAY | YEARDAY  
         | HOUR | MINUTE | SECOND | MILLISECOND
```

```
<datetime> ::= выражение с типом данных DATE, TIME или  
             TIMESTAMP
```

Типы и диапазоны результатов оператора EXTRACT

Составляющая даты/времени	Тип	Диапазон	Комментарий
YEAR	SMALLINT	1–9999	
MONTH	SMALLINT	1–12	
WEEK	SMALLINT	1–53	
DAY	SMALLINT	1–31	
WEEKDAY	SMALLINT	0–6	0 = Воскресенье
YEARDAY	SMALLINT	0–365	0 = 1 января
HOUR	SMALLINT	0–23	
MINUTE	SMALLINT	0–59	
SECOND	NUMERIC (9, 4)	0.0000–59.9999	Включает в себя миллисекунды
MILLISECOND	NUMERIC (9, 1)	0.0–999.9	Возвращает неправильный результат в версиях Firebird 2.1 и 2.1.1

Замечание:

Если составляющая даты/времени не присутствует в аргументе дата/время, например `SECOND` в аргументе с типом `DATE` или `YEAR` в `TIME`, то функция вызовет ошибку.

Параметр `WEEK`:

Начиная с версии Firebird 2.1 можно извлекать номер недели из аргумента с типом данных `DATE` или `TIMESTAMP`. В соответствии со стандартом ISO-8601 неделя начинается с понедельника и всегда включает в себя 7 дней. Первой неделей года является первая неделя, у которой в ней больше дней в новом году (по крайней мере, 4): дни 1-3 могут принадлежать предыдущей неделе (52 или 53) прошлого года. По аналогии дни 1-3 текущего года могут принадлежать 1 неделе следующего года.

Пример:

```
/* получить по дате номер квартала */
SELECT (EXTRACT(MONTH FROM CURRENT_TIMESTAMP) - 1) / 3 + 1
FROM RDB$DATABASE
```

Функция добавлена в Interbase 6.0. Доступна для DSQL, PSQL, ESQL.

Начиная с версии Firebird 2.1 доступна опция `WEEK`, с версии Firebird 2.1.2, доступна опция `MILLISECOND`, она возвращает значения типа `NUMERIC(9, 1)`.

См. также: [Типы данных для работы с датой и временем](#)

FLOOR ()

Функция возвращает целое число, меньшее или равное аргументу. Тип возвращаемого значения `BIGINT` или `DOUBLE PRECISION`.

Синтаксис:

```
FLOOR (< число >)
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [CEIL \(\)](#), [CEILING \(\)](#), [TRUNC \(\)](#).

GEN_ID ()

Функция увеличивает значение генератора или последовательности и

Руководство по языку SQL СУБД Firebird

возвращает новое значение. Тип возвращаемого значения BIGINT.

Синтаксис:

```
GEN_ID (< имя генератора >, < шаг >)
```

где < шаг > - целое число.

Примечание:

Если значение параметра < шаг > меньше нуля, произойдет уменьшение значения генератора. Внимание! Следует быть крайне аккуратным при таких манипуляциях в базе данных, они могут привести к потере целостности данных. Если < шаг > равен 0, функция не будет ничего делать со значением генератора и вернет его текущее значение.

Пример:

```
NEW.ID = GEN_ID (GEN_TABLE_ID, 1);
```

Функция добавлена в Interbase. Доступна для DSQL, PSQL, ESQL.

В Firebird 2.0 для получения следующего значения последовательности / генератора стало доступно использование оператора NEXT VALUE FOR.

См. также: [NEXT VALUE FOR](#), [SEQUENCE \(GENERATOR\)](#), [ALTER SEQUENCE](#), [SET GENERATOR](#).

GEN_UUID ()

Функция возвращает универсальный уникальный идентификатор ID в виде 16-байтной строки символов. До версии Firebird 2.5.2 это была полностью случайная строка, что не отвечало требованиям стандарта RFC-4122. Начиная с версии 2.5.2, функция возвращает строку UUID 4-ой версии, где несколько битов зарезервированы, а остальные являются случайными. Тип возвращаемого значения CHAR(16) CHARACTER SET OCTETS.

Синтаксис:

```
GEN_UUID ()
```

Примеры:

```
SELECT GEN_UUID () FROM RDB$DATABASE
```

```
-- результат (до версии Firebird 2.5.2):
```

Руководство по языку SQL СУБД Firebird

017347BFE212B2479C00FA4323B36320 (16-байтная строка)

-- результат (начиная с версии Firebird 2.5.2):

XXXXXXXX-XXXX-4XXX-YXXX-XXXXXXXXXXXX

где 4 это номер версии, а Y может принимать значение 8, 9, A или B .

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [CHAR_TO_UUID \(\)](#), [UUID_TO_CHAR \(\)](#).

HASH ()

Функция возвращает хэш-значение входной строки. Эта функция полностью поддерживает текстовые BLOB любой длины и с любым набором символов. Тип возвращаемого значения BIGINT.

Синтаксис:

HASH(< строка >)

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

IIF ()

Функция имеет 3 аргумента: если первый аргумент является истиной, то результатом будет второй аргумент, иначе – третий аргумент.

Синтаксис:

IIF (<условие>, <результат при Истина>, <результат при Ложь>)

<условие> ::= булевское выражение

Функция является упрощенной формой CASE, ее можно заменить на

```
CASE
  WHEN <условие>
  THEN <результат при Истина>
  ELSE <результат при Ложь>
END
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [DECODE \(\)](#), [CASE](#).

LEFT ()

Функция возвращает левую часть строки, количество возвращаемых символов определяется вторым параметром. Тип возвращаемого значения VARCHAR или BLOB.

Синтаксис:

```
LEFT (< строка >, < число >)
```

- Функция поддерживает текстовые блоки любой длины и с любыми наборами символов;
- Если строковый аргумент BLOB, результатом будет BLOB, в противном случае результатом будет VARCHAR(N), при этом N – будет равно длине строкового параметра;
- Если числовой параметр превысит длину текста, результатом будет текст.
- В случае если числовой параметр не будет *целым* числом, к нему применится банковское округление: то есть
0,5 станет 0,
1,5 станет 2,
2,5 станет 2,
3,5 станет 4 , ...
и так далее.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [RIGHT \(\)](#), [SUBSTRING \(\)](#).

LN ()

Функция возвращает натуральный логарифм аргумента. Тип возвращаемого значения DOUBLE PRECISION.

Синтаксис:

```
LN (< число >)
```

Примечание:

В случае если передан отрицательный или нулевой аргумент функция вернет ошибку.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [EXP \(\)](#)

LOG ()

Функция возвращает логарифм y (второй аргумент) по основанию x (первый аргумент). Тип возвращаемого значения `DOUBLE PRECISION`.

Синтаксис:

```
LOG (x, y)
```

- Если один из аргументов меньше или равен 0, то возникает ошибка. До версии Firebird 2,5 это приведет к выдаче NaN (Not-a-Number — не число), \pm INF (infinity - бесконечность) или 0 в зависимости от точного значения аргументов;
- Если оба аргумента равны 1, то результатом функции будет NaN;
- Если $x = 1$ и $y < 1$, то результатом функции будет $-INF (-\infty)$;
- Если $x = 1$ и $y > 1$, то результатом функции будет $+INF (+\infty)$.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL. Изменена в Firebird 2.5.

LOG10 ()

Функция возвращает десятичный логарифм аргумента. Тип возвращаемого значения `DOUBLE PRECISION`.

Синтаксис:

```
LOG10 (< число >)
```

Примечание:

Если входной аргумент отрицательный или равен 0, возникает ошибка. До версии Firebird 2,5 это приведет к выдаче NaN и $-INF (-\infty)$, соответственно.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL. Изменена в Firebird 2.5.

LOWER ()

Функция возвращает входную строку в нижнем регистре. Тип возвращаемого значения `VARCHAR` или `BLOB`.

Синтаксис:

```
LOWER (<строка>)
```

Руководство по языку SQL СУБД Firebird

Результат работы функции зависит от набора символов. Например, для набора символов ASCII и NONE только ASCII приводятся к нижнему регистру. Строки с набором символов OCTETS возвращаются без изменений. Начиная с версии Firebird 2.1, функция стала полностью поддерживать текстовые BLOB любого размера и с любым набором символов.

Функция модифицирована в Firebird в версии 2.1. Доступна для DSQL, PSQL, ESQL.

См. также: [UPPER \(\)](#).

LPAD ()

Функция дополняет слева входную строку пробелами или определенной пользователем строкой до заданной длины. Тип возвращаемого значения VARCHAR или BLOB.

Синтаксис:

```
LPAD (< строка >, <длина строки> [, < заполнение >])
```

Функция полностью поддерживает текстовые BLOB любой длины и с любым набором символов.

Если входная строка имеет тип BLOB, то результат также будет BLOB, в противном случае результат будет VARCHAR(<длина строки>).

Если аргумент < заполнение> задан, но равен пустой строке ("), то дополнения строки не происходит! В случае если <длина строки> меньше длины входной строки, то в результате происходит ее усечение до длины <длина строки>, даже если параметр < заполнение > равен пустой строке.

Примеры:

```
select lpad('ABC', 5) as lp,  
       lpad('ABC', 5, '*') as lp_  
from rdb$database;
```

```
-- вернет 'He'  
lpad('Hello', 2, '_')
```

```
-- вернет ' ab-abHello'  
lpad('Hello', 10, 'ab-')
```

Примечание:

Руководство по языку SQL СУБД Firebird

При использовании BLOB в параметрах функции может потребоваться загрузить объект полностью в память. При больших объемах BLOB могут наблюдаться потери производительности.

Функция добавлена в Firebird в версии 2.1.4. Доступна для DSQL, PSQL.

См. также: [RPAD \(\)](#).

MAXVALUE ()

Функция возвращает максимальное значение из входного списка чисел, строк или параметров с типом данных DATE/TIME/TIMESTAMP.

Синтаксис:

```
MAXVALUE (param, [ param ...])
```

В случае если один и более из входных параметров будет NULL, то результатом функции будет NULL, в отличие от агрегатной функции MAX.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [MINVALUE \(\)](#).

MINVALUE ()

Функция возвращает минимальное значение из входного списка чисел, строк или параметров с типом данных DATE/TIME/TIMESTAMP.

Синтаксис:

```
MINVALUE (param, [ param ...])
```

В случае если один и более из входных параметров будет NULL, то результатом функции будет NULL, в отличие от агрегатной функции MIN.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [MAXVALUE \(\)](#).

MOD ()

Функция возвращает остаток от целочисленного деления. Тип возвращаемого значения INTEGER или BIGINT.

Руководство по языку SQL СУБД Firebird

Синтаксис:

```
MOD (a, b)
```

- Вещественные числа округляются до выполнения деления. Например, результатом `7.5 mod 2.5` будет `2` (`8 mod 3`), а не `0`.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

NULLIF ()

Функция возвращает значение первого аргумента, если он не равен второму. В случае равенства аргументов возвращается NULL.

Синтаксис:

```
NULLIF (param1, param2)
```

Пример:

```
select avg(nullif(weight, -1)) from cargo;
/* запрос вернет среднее значение поля weight по таблице,
за исключением строк, где он не указан (равен -1). Если бы
не было этой функции простой оператор avg(weight) вернул бы
некорректное значение */
```

Функция добавлена в Firebird в версии 1.5. Доступна для DSQL, PSQL.

См. также: [COALESCE \(\)](#), [DECODE \(\)](#), [IIF \(\)](#), [CASE](#)

ОСТЕТ_LENGTH ()

Функция возвращает количество **байт** занимаемое строкой. Тип возвращаемого значения INTEGER.

Синтаксис:

```
ОСТЕТ_LENGTH (< строка >)
```

Внимание!

Следует помнить, что не во всех наборах символов количество байт занимаемых строкой равно количеству символов.

При работе с параметрами типа `CHAR` функция возвращает значение всей формальной строковой длины. Для того чтобы узнать «логическую» длину строки в байтах, то перед передачей аргумента функции следует применить

RIGHT TRIM.

Функция полностью поддерживает BLOB любой длины и с любым набором символов.

Функция добавлена в Firebird в версии 2.0. Доступна для DSQL, PSQL.

В версии 2.1 добавлена поддержка BLOB.

См. также: [CHAR_LENGTH \(\)](#), [CHARACTER_LENGTH \(\)](#), [BIT_LENGTH \(\)](#).

OVERLAY ()

Функция предназначена для замены части строки другой строкой. Тип возвращаемого значения VARCHAR или BLOB.

Синтаксис:

```
OVERLAY (< строка > PLACING < чем > FROM <pos> [FOR  
<длина>])
```

По умолчанию число удаляемых из строки символов равняется длине заменяемой строки. Дополнительный четвертый параметр позволяет пользователю задать своё число символов, которые будут удалены.

- Функция полностью поддерживает тестовые BLOB с любым набором символов и любой длины;
- Если входная строка имеет тип BLOB, то и результат будет иметь тип BLOB. В противном случае тип результата будет VARCHAR(*n*), где *n* является суммой длин параметров <строка> и <чем>;
- Как и во всех строковых функциях SQL параметр <pos> является определяющим;
- Если <pos> больше длины строки, то <чем> помещается сразу после окончания строки;
- Если число символов от pos до конца строки меньше, чем длина <чем> (или, чем параметр <длина>, если он задан), то строка усекается до значения pos и <чем> помещается после него;
- При нулевом параметре <длина> (FOR 0) <чем> просто вставляется в строку, начиная с позиции <pos>;
- Если любой из параметров имеет значение NULL, то и результат будет NULL;
- Если параметры <pos> и <длина> не являются целым числом, то используется банковское округление (до четного): 0.5 становится 0, 1.5 становится 2, 2.5 становится 2, 3.5 становится 4 и т.д.
- При использовании BLOB функции может потребоваться загрузить весь объект в память. При больших размерах BLOB это может повлиять на

Руководство по языку SQL СУБД Firebird

производительность.

Примеры:

```
overlay ('Goodbye' placing 'Hello' from 2)
-- Результат: 'Ghelloe'

overlay ('Goodbye' placing 'Hello' from 5)
-- Результат: 'GoodHello'

overlay ('Goodbye' placing 'Hello' from 8)
-- Результат: 'GoodbyeHello'

overlay ('Goodbye' placing 'Hello' from 20)
-- Результат: 'GoodbyeHello'

overlay ('Goodbye' placing 'Hello' from 2 for 0)
-- Результат: 'GHellooodbye'

overlay ('Goodbye' placing 'Hello' from 2 for 3)
-- Результат: 'GHellobye'

overlay ('Goodbye' placing 'Hello' from 2 for 6)
-- Результат: 'GHello'

overlay ('Goodbye' placing 'Hello' from 2 for 9)
-- Результат: 'Ghello'

overlay ('Goodbye' placing '' from 4)
-- Результат: 'Goodbye'

overlay ('Goodbye' placing '' from 4 for 3)
-- Результат: 'Gooe'

overlay ('Goodbye' placing '' from 4 for 20)
-- Результат: 'Goo'

overlay ('' placing 'Hello' from 4)
-- Результат: 'Hello'

overlay ('' placing 'Hello' from 4 for 0)
-- Результат: 'Hello'

overlay ('' placing 'Hello' from 4 for 20)
-- Результат: 'Hello'
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

Руководство по языку SQL СУБД Firebird

См. также: `SUBSTRING ()`, `REPLACE ()`

PI ()

Функция возвращает число π . Тип возвращаемого значения `DOUBLE PRECISION`.

Синтаксис:

```
PI ()
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

POSITION ()

Функция возвращает позицию первого вхождения подстроки в строку. Отсчет начинается с 1. Третий аргумент (опциональный) задает позицию в строке, с которой начинается поиск подстроки, тем самым игнорирую любые вхождения подстроки в строку до этой позиции. Если совпадение не найдено, функция возвращает 0. Тип возвращаемого значения `INTEGER`.

Синтаксис:

```
POSITION (<подстрока> IN <строка>)  
| POSITION (<подстрока>, <строка> [, <старт>])
```

- Опциональный третий параметр поддерживается только вторым вариантом синтаксиса (синтаксис с запятой);
- Пустую строку, функция считает подстрокой любой строки. Поэтому при входном параметре `<подстрока>`, равном " (пустая строка), и при параметре `string`, отличном от `NULL`, результатом будет:
 - 1, если параметр `<старт>` не задан;
 - `<старт>`, если `<старт>` не превышает длину параметра `string`;
 - 0, если `<старт>` превышает длину параметра `string`.

Примеры:

```
POSITION ('be' in 'To be or not to be') -- Результат: 4  
POSITION ('be', 'To be or not to be') -- Результат: 4  
POSITION ('be', 'To be or not to be', 4) -- Результат: 4  
POSITION ('be', 'To be or not to be', 8) -- Результат: 17
```

Функция добавлена в Firebird в версии 2.1.4. Доступна для DSQL, PSQL.

Руководство по языку SQL СУБД Firebird

См. также: [SUBSTRING \(\)](#).

POWER ()

Функция возвращает результат возведения числа <аргумент> в <степень>. Тип возвращаемого значения `DOUBLE PRECISION`.

Синтаксис:

```
POWER (<аргумент>, <степень>)
```

Если <аргумент> меньше нуля, возникает ошибка.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

RAND ()

Функция возвращает псевдослучайное число в интервале от 0 до 1. Тип возвращаемого значения `DOUBLE PRECISION`.

Синтаксис:

```
RAND ()
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

REPLACE ()

Функция заменяет в строке **все** вхождения одной строки на другую строку.

Синтаксис:

```
REPLACE (<строка>, <что меняем>, <на что меняем>)
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [OVERLAY \(\)](#), [SUBSTRING \(\)](#), [POSITION \(\)](#), [CHAR_LENGTH \(\)](#), [CHARACTER_LENGTH \(\)](#).

REVERSE ()

Функция возвратит строку перевернутую «задом наперед». Тип возвращаемого значения `VARCHAR`.

Синтаксис:

Руководство по языку SQL СУБД Firebird

REPLACE (<строка>)

Пример:

```
REVERSE ('spoonful')      -- Результат: 'lufnoops'
```

Примечание:

Данная функция очень удобна, если вам предстоит работать (сортировать или группировать информацию) которая находится в окончаниях строк. Пример такой информации - доменные имена.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

RIGHT ()

Функция возвращает конечную (правую) часть входной строки. Длина возвращаемой подстроки определяется вторым параметром. Тип возвращаемого значения VARCHAR или BLOB.

Синтаксис:

```
RIGHT (< строка >, <длина>)
```

Пример:

```
select right('ABC', 1) from rdb$database;  
-- результат C
```

- Функция поддерживает текстовые BLOB любой длины, в том числе и с многобайтными наборами символов;
- Если строковый параметр типа BLOB, то и результат будет типа BLOB. В противном случае результат будет типа VARCHAR(n), где n равно длине строкового параметра;
- Если числовой параметр больше длины строкового параметра, то результатом будет строковый параметр;
- Если числовой параметр не является целым числом, то используется банковское округление («до четного»):
0.5 становится 0,
1.5 становится 2,
2.5 становится 2,
3.5 становится 4 и т.д.

Примечание:

Руководство по языку SQL СУБД Firebird

При использовании BLOB в параметрах функции может потребоваться загрузить объект полностью в память. При больших объемах BLOB могут наблюдаться потери производительности.

Функция добавлена в Firebird в версии 2.1.4. Доступна для DSQL, PSQL.

См. также: [LEFT \(\)](#), [SUBSTRING \(\)](#).

ROUND ()

Функция округляет число до ближайшего целого числа. Если дробная часть равна 0.5, то округление до ближайшего большего целого числа для положительных чисел и до ближайшего меньшего для отрицательных чисел. С дополнительным опциональным параметром <знаков> число может быть округлено до одной из степеней числа 10 (десятки, сотни, десятые части, сотые части и т.д.) вместо просто целого числа. Тип возвращаемого значения INTEGER, масштабируемый BIGINT, DOUBLE.

Синтаксис:

```
ROUND (<число> [, <знаков>])
```

где <знаков> - целое число, определяющее число десятичных разрядов, к которым должен быть проведено округление, например:

- 2 для округления к самому близкому кратному 0.01 числу
- 1 для округления к самому близкому кратному 0.1 числу
- 0 для округления к самому близкому целому числу
- 1 для округления к самому близкому кратному 10 числу
- 2 для округления к самому близкому кратному 100 числу

- Если используется параметр <знаков>, то результат имеет такой же масштаб, как и первый параметр <число>, например:

- ROUND(123.654, 1) Результат: 123.700 (а не 123.7)
- ROUND(8341.7, -3) Результат: 8000.0 (а не 8000)
- ROUND(45.1212, 0) Результат: 45.0000 (а не 45)

В противном случае (параметр *scale* не задан) результат будет:

- ROUND(45.1212) Результат: 45

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

Руководство по языку SQL СУБД Firebird

RPAD ()

Функция дополняет **справа** входную строку пробелами или определенной пользователем строкой до заданной длины. Тип возвращаемого значения VARCHAR или BLOB.

Синтаксис:

```
RPAD (< строка >, <длина строки> [, < заполнение >])
```

Функция полностью поддерживает текстовые BLOB любой длины и с любым набором символов.

Если входная строка имеет тип BLOB, то результат также будет BLOB, в противном случае результат будет VARCHAR(<длина строки>).

Если аргумент < заполнение> задан, но равен пустой строке ("), то дополнения строки не происходит! В случае если <длина строки> меньше длины входной строки, то в результате происходит ее усечение до длины <длина строки>, даже если параметр < заполнение > равен пустой строке.

Примеры:

```
RPAD ('Hello', 12)           -- Результат: 'Hello '
RPAD ('Hello', 12, '-')     -- Результат: 'Hello-----'
RPAD ('Hello', 12, '')      -- Результат: 'Hello'
RPAD ('Hello', 12, 'abc')   -- Результат: 'Helloabcabca'
RPAD ('Hello', 12, 'abcdefghij')-- Результат: 'Helloabcdefgh'
RPAD ('Hello', 2)          -- Результат: 'He'
RPAD ('Hello', 2, '-')     -- Результат: 'He'
RPAD ('Hello', 2, '')      -- Результат: 'He'
```

Примечание:

При использовании BLOB в параметрах функции может потребоваться загрузить объект полностью в память. При больших объемах BLOB могут наблюдаться потери производительности.

Функция добавлена в Firebird в версии 2.1.4. Доступна для DSQL, PSQL.

См. также: [LPAD \(\)](#).

SIGN ()

Функция возвращает знак входного параметра. Тип возвращаемого значения INTEGER, масштабируемый SMALLINT.

Руководство по языку SQL СУБД Firebird

Синтаксис:

```
SIGN (< число >)
```

Таблица результатов:

Результат	Значение
-1	число меньше нуля
0	число равно нулю
1	число больше нуля

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

SIN ()

Функция возвращает синус угла. Аргумент должен быть занят в радианах. Тип возвращаемого значения `DOUBLE PRECISION`.

Синтаксис:

```
SIN (< угол >)
```

Любой `NOT-NULL` результат находится в диапазоне `[-1, 1]`.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [ASIN \(\)](#).

SINH ()

Функция возвращает гиперболический синус аргумента. Тип возвращаемого значения `DOUBLE PRECISION`.

Синтаксис:

```
SINH (< число >)
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

SQRT ()

Функция возвращает квадратный корень аргумента. Тип возвращаемого значения `DOUBLE PRECISION`.

Синтаксис:

Руководство по языку SQL СУБД Firebird

SQRT (<число>)

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

SUBSTRING ()

Функция возвращает подстроку строки <строка>, начиная с заданной позиции <старт> до конца строки или указанной длины. Тип возвращаемого значения VARCHAR или BLOB.

Синтаксис:

```
SUBSTRING (<строка> FROM <старт> [FOR <длина>])
```

Начиная с версии Firebird 2.0 параметры функции могут быть любым допустимым целочисленным выражением.

Начиная с версии Firebird 2.1 функция полностью поддерживает двоичные и текстовые BLOB любой длины и с любым набором символов. Если параметр <строка> имеет тип BLOB, то и результат будет иметь тип. Для любых других типов результатом будет тип VARCHAR (n). Для входного параметра <строка>, не являющегося BLOB, длина результата функции всегда будет равна длине <строка>, независимо от значений параметров <старт> и <длина>.

Если любой из входных параметров имеет значение NULL, то и результат тоже будет иметь значение NULL.

При использовании BLOB в параметрах функции может потребоваться загрузить объект в память полностью. При больших объемах BLOB могут наблюдаться потери производительности.

Пример:

```
SUBSTRING('Привет!' from 4 for 3)
-- вернет подстроку 'вет'
```

Функция добавлена в Firebird в версии 1.0. Дорабатывалась в версиях 2.0, 2.1, 2.1.5, 2.5.1. Доступна для DSQL, PSQL.

См. также: [POSITION \(\)](#), [LEFT \(\)](#), [RIGHT \(\)](#), [CHAR_LENGTH \(\)](#), [CHARACTER_LENGTH \(\)](#).

TAN ()

Функция возвращает тангенс угла. Аргумент должен быть занят в

Руководство по языку SQL СУБД Firebird

радианах. Тип возвращаемого значения `DOUBLE PRECISION`.

Синтаксис:

```
TAN (< угол >)
```

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [ATAN \(\)](#), [ATAN2 \(\)](#).

TANH ()

Функция возвращает гиперболический тангенс числа. Тип возвращаемого значения `DOUBLE PRECISION`.

Синтаксис:

```
TANH (< число >)
```

Любой `NOT-NULL` результат находится в диапазоне `[-1, 1]`.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

TRIM ()

Функция удаляет начальные и /или конечные пробелы (или текст согласно настройкам) из входной строки. Тип возвращаемого значения `VARCHAR` или `BLOB`.

Синтаксис:

```
TRIM ([<опции>] <строка>)
```

где <опции> ::= {[где] [что]} FROM

где ::= BOTH | LEADING | TRAILING -- по умолчанию BOTH

что ::= подстрока, которую надо удалить (неоднократно, если таких вхождений несколько) из входной строки <строка> в её начале и/или конце. По умолчанию является пробелом (' ').

Спецификация опций функции

BOTH	с обеих сторон строки (по умолчанию)
LEADING	с начала строки
TRAILING	с конца строки

Руководство по языку SQL СУБД Firebird

Примеры:

```
SELECT TRIM (' Waste no space ')
FROM RDB$DATABASE -- Результат: 'Waste no space'
```

```
SELECT TRIM (LEADING FROM ' Waste no space ')
FROM RDB$DATABASE -- Результат: 'Waste no space '
```

```
SELECT TRIM (LEADING '.' from ' Waste no space ')
FROM RDB$DATABASE -- Результат: ' Waste no space '
```

```
SELECT TRIM (TRAILING '!' from 'Help!!!!')
FROM RDB$DATABASE -- Результат: 'Help'
```

```
SELECT TRIM ('la' FROM 'lalala I love you Ella')
FROM RDB$DATABASE -- Результат: ' I love you El'
```

Примечания:

- Если входной параметр <строка> имеет тип BLOB, то и результат будет иметь тип BLOB. В противном случае результат будет иметь тип VARCHAR(*n*), где *n* является длиной параметра <строка>;
- Подстрока для удаления, если она, конечно, задана, не должна иметь длину больше, чем 32767 байта. Однако при повторениях подстроки в начале и/или конце входного параметра <строка> общее число удаляемых байтов может быть гораздо больше.

При использовании BLOB в параметрах функции может потребоваться загрузить объект в память полностью. При больших объемах BLOB могут наблюдаться потери производительности.

Функция добавлена в Firebird в версии 2.0. Дорабатывалась в версии 2.1. Доступна для DSQL, PSQL.

См. также: [OVERLAY \(\)](#), [REPLACE \(\)](#).

TRUNC ()

Функция возвращает целую часть числа. Тип возвращаемого значения INTEGER, масштабируемый BIGINT или DOUBLE.

Синтаксис записи

```
TRUNC (<число>[, <масштаб>])
```

где <масштаб> - целое число, определяющее число десятичных

Руководство по языку SQL СУБД Firebird

разрядов, к которым должен быть проведено округление, например:
2 для приведения к кратному 0.01 числу
1 для приведения к кратному 0.1 числу
0 для приведения к целому числу
-1 для приведения к кратному 10 числу
-2 для приведения к кратному 100 числу.

Примечания:

- Если используется параметр <масштаб>, то результат имеет такой же масштаб, как и первый параметр <число>

например:

- TRUNC (789.2225, 2) Результат: 789.2200 (а не 789.22)
- TRUNC (345.4, -2) Результат: 300.0 (а не 300)
- TRUNC (-163.41, 0) Результат: -163.00 (а не -163)

В противном случае (параметр <масштаб> не задан) результат будет:

- TRUNC (-163.41) Результат: -163

Внимание! Функция всегда **увеличивает** отрицательные числа, поскольку она обрезает дробную часть.

Функция добавлена в Firebird в версии 2.1. Доступна для DSQL, PSQL.

См. также: [CEIL \(\)](#), [CEILING \(\)](#), [FLOOR \(\)](#).

UPPER ()

Функция возвращает входную строку в верхнем регистре. Точный результат зависит от набора символов входной строки. Например, для наборов символов NONE и ASCII только ASCII символы переводятся в верхний регистр; для OCTETS — вся входная строка возвращается без изменений. Тип возвращаемого значения VARCHAR или BLOB.

Синтаксис:

```
UPPER (<строка>)
```

Пример:

```
SELECT UPPER(_ISO8859_1 'Débâcle' COLLATE FR_FR)  
FROM RDB$DATABASE  
/* Результат: 'DEBACLE', в соответствии с французскими  
правилами приведения в верхний регистр */
```

Руководство по языку SQL СУБД Firebird

Начиная с версии Firebird 2.1 полностью поддерживаются текстовые BLOB любой длины и с любым набором символов.

Функция добавлена в IB. Дорабатывалась в версии 2.1. Доступна для DSQL, PSQL, ESQL.

См. также: [LOWER \(\)](#).

UUID_TO_CHAR ()

Функция конвертирует 16-ти байтный UUID в его 36-ти знаковое ASCII представление. Тип возвращаемого значения CHAR (36).

Синтаксис:

```
UUID_TO_CHAR (<uuid>)
```

Пример:

```
SELECT UUID_TO_CHAR (GEN_UUID ()) FROM RDB$DATABASE;
```

Функция добавлена в Firebird в версии 2.5. Доступна для DSQL, PSQL.

См. также: [GEN_UUID \(\)](#), [CHAR_TO_UUID \(\)](#).

Агрегатные функции

Агрегатными называют функции, которые работают по группе записей. Их часто применяют в комбинации с предложением GROUP BY конструкции SELECT. Работая с группой значений, функции учитывают только те из них, которые не являются NULL. Агрегатные функции доступны, если не указано иное, в DSQL, ESQL, PSQL.

AVG ()

Функция возвращает среднее значение для группы.

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

Синтаксис:

```
AVG (<выражение>)
```

COUNT()

Функция возвращает количество значений в группе, которые не являются NULL.

Тип возвращаемого значения INTEGER.

Синтаксис:

```
COUNT ([ALL | DISTINCT] <выражение>)
```

При указании DISTINCT из выборки устраняются дубликаты, ALL является значением по умолчанию для всех выборки значений не NULL.

Для пустой выборки данных или если при выборке окажутся одни значения, содержащие NULL, функция возвратит значение равное 0.

LIST ()

Функция возвращает строку, содержащую значения элементов выборки, которые не равны NULL. При пустой выборке функция возвратит NULL. Тип возвращаемого значения текстовый BLOB за исключением тех случаев, когда выражением являются BLOB других подтипов.

Синтаксис:

```
LIST ([ALL | DISTINCT] <выражение> [, <разделитель>])
```

Разделителем элементов строки по умолчанию является запятая. Начиная с версии FB 2.5 появилась возможность явно указать разделитель; <разделитель> может быть любой строкой.

ALL является опцией по умолчанию. При ней обрабатываются все значения в выборке, не содержащие NULL. При указании DISTINCT из выборки устраняются дубликаты.

Значения <выражение> и <разделитель> поддерживают тип данных BLOB любого размера и набора символов. Поля типа дата / время и числовые перед проведением операции конкатенации преобразуются в строки.

Порядок вывода списка значений не определен.

Пример:

```
select list (display_name, ';' ' ) from GR_WORK;
```

Доступно в DSQL, PSQL.

MAX ()

Функция возвращает максимальный элемент выборки, которые не равны NULL. При пустой выборке, или при выборке из одних NULL функция возвратит NULL. Если аргумент функции строка, то функция вернет значение, которое окажется последним в сортировке при применении активного COLLATE.

Синтаксис:

```
MAX (<выражение>)
```

Начиная с версии FB 2.1 , функция полностью поддерживает текстовые BLOB любого размера и набора символов.

См. также: [MIN \(\)](#).

MIN ()

Функция возвращает минимальный элемент выборки, которые не равны NULL. При пустой выборке, или при выборке из одних NULL функция возвратит NULL. Если аргумент функции строка, то функция вернет значение, которое окажется первым в сортировке при применении активного COLLATE.

Синтаксис:

```
MIN (<выражение>)
```

Начиная с версии FB 2.1 , функция полностью поддерживает текстовые BLOB любого размера и набора символов.

См. также: [MAX \(\)](#).

SUM ()

Функция возвращает сумму элементов выборки, которые не равны NULL. При пустой выборке, или при выборке из одних NULL функция возвратит NULL.

Синтаксис:

```
SUM ([ALL | DISTINCT] <выражение>)
```

ALL является опцией по умолчанию. При ней обрабатываются все значения из выборки, не содержащие NULL. При указании DISTINCT из выборки устраняются дубликаты, после осуществляется подсчет.

<выражение> может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип

данных.

Пример:

```
select
  dept_no,
  sum (salary),
  avg (salary),
  min (salary),
  max (salary),
  count (salary)
from employee
group by dept_no
```

RELEASE SAVEPOINT

Описание: Удаляет именованную точку сохранения, освобождая все связанные с ней ресурсы.

Синтаксис:

```
RELEASE SAVEPOINT name [ONLY]
```

Без использования предложения ONLY удаляются также все точки сохранения, создаваемые после указанной.

Более подробное описание точек сохранения приведено в разделе [SAVEPOINT](#).

ROLLBACK

Синтаксис:

```
ROLLBACK [WORK] [TRANSACTION tr_name]  
[RETAIN [SNAPSHOT] | TO [SAVEPOINT] sp_name | RELEASE]
```

- Предложение TRANSACTION доступно только в Embedded SQL;
- Предложение RELEASE доступно только в ESQL и не рекомендуется для использования;
- Предложения RETAIN и TO доступны только в DSQL.

ROLLBACK RETAIN

Отменяет все изменения в базе данных, выполненные в рамках транзакции, при этом не закрывая её. Пользовательские переменные, заданные с помощью функции RDB\$SET_CONTEXT () остаются неизменными.

Синтаксис:

```
ROLLBACK [WORK] RETAIN [SNAPSHOT]
```

Примечание:

Функциональные возможности, предоставляемые ROLLBACK RETAIN, присутствуют начиная с InterBase 6, но единственным способом получить доступ к ним был через вызов API функции `isc_rollback_retaining ()`.

ROLLBACK TO SAVEPOINT

Описание: Отменяет все изменения, произошедшие в рамках транзакции, начиная с созданной точки сохранения (SAVEPOINT).

Синтаксис:

```
ROLLBACK [WORK] TO [SAVEPOINT] name
```

Оператор ROLLBACK TO SAVEPOINT выполняет следующие операции:

- Все изменения в базе данных, выполненные в рамках транзакции начиная с созданной точки сохранения, отменяются. Пользовательские переменные, заданные с помощью функции RDB\$SET_CONTEXT () остаются неизменными;
- Все точки сохранения, создаваемые после названной, уничтожаются. Все более ранние точки сохранения, как сама точка сохранения, остаются. Это означает, что можно откатываться к той же точке сохранения несколько раз;
- Все явные и неявные заблокированные записи, начиная с точки сохранения, освобождаются. Другие транзакции, запросившие ранее доступ к строкам, заблокированным после точки сохранения, должны продолжать ожидать, пока транзакция не фиксируется или откатывается. Другие транзакции, которые ещё не запрашивали доступ к этим строкам, могут запросить и сразу же получить доступ к разблокированным строкам.

Более подробное описание точек сохранения приведено в разделе [SAVEPOINT](#).

SAVEPOINT

Описание: Создает SQL 99 совместимую точку сохранения, к которой можно позже откатывать работу с базой данных, не отменяя все действия, выполненные с момента старта транзакции. Механизмы точки сохранения также известны под термином "вложенные транзакции" ("nested transactions").

Синтаксис:

```
SAVEPOINT <name>
```

<name> ::= выбранный пользователем идентификатор для точки сохранения, уникальный для стартовавшей транзакции

Если имя точки сохранения уже существует в рамках транзакции, то существующая точка сохранения будет удалена, и создается новая с тем же именем.

Руководство по языку SQL СУБД Firebird

Для отката изменений к точке сохранения используйте оператор:

```
ROLLBACK [WORK] TO [SAVEPOINT] name
```

Оператор ROLLBACK TO SAVEPOINT выполняет следующие операции:

- Все изменения в базе данных, выполненные в рамках транзакции начиная с созданной точки сохранения, отменяются. Пользовательские переменные, заданные с помощью функции RDB\$SET_CONTEXT () остаются неизменными;
- Все точки сохранения, создаваемые после названной, уничтожаются. Все более ранние точки сохранения, как сама точка сохранения, остаются. Это означает, что можно откатываться к той же точке сохранения несколько раз;
- Все явные и неявные заблокированные записи, начиная с точки сохранения, освобождаются. Другие транзакции, запросившие ранее доступ к строкам, заблокированным после точки сохранения, должны продолжать ожидать, пока транзакция не фиксируется или откатывается. Другие транзакции, которые ещё не запрашивали доступ к этим строкам, могут запросить и сразу же получить доступ к разблокированным строкам.

Внутренний механизм точек сохранения может использовать большие объемы памяти, особенно если Вы обновляете одни и те же записи многократно в одной транзакции. Если точка сохранения уже не нужна, но Вы еще не готовы закончить транзакцию, то можно её удалить, тем самым освобождая ресурсы:

```
RELEASE SAVEPOINT name [ONLY]
```

Без использования предложения ONLY удаляются также все точки сохранения, создаваемые после указанной.

Примеры:

Пример DSQL сессии с использованием точек сохранения:

```
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; -- возвращает пустую строку
```

Руководство по языку SQL СУБД Firebird

```
ROLLBACK TO Y;  
SELECT * FROM TEST; -- возвращает две строки  
ROLLBACK;  
SELECT * FROM TEST; -- возвращает одну строку
```

Внутренние точки сохранения

По умолчанию сервер использует автоматическую системную точку сохранения уровня транзакции для выполнения её отката. При выполнении оператора ROLLBACK, все изменения, выполненные в транзакции, откатываются до системной точки сохранения и после этого транзакция подтверждается.

Когда объем изменений, выполняемых под системной точкой сохранения уровня транзакции, становится большим (затрагивается порядка 50000 записей) сервер освобождает системную точку сохранения и, при необходимости отката транзакции, использует механизм TIP.

Совет:

Если Вы ожидаете, что объем изменений в транзакции будет большим, то можно задать опцию NO AUTO UNDO в операторе SET TRANSACTION, или – если используется API – установить флаг TPB isc_tpb_no_auto_undo. В обоих вариантах предотвращается создание системной точки сохранения уровня транзакции.

Точки сохранения и PSQL

Использование операторов управления транзакциями в PSQL не разрешается, так как это нарушит атомарность оператора, вызывающего процедуру. Но Firebird поддерживает вызов и обработку исключений в PSQL, так, чтобы действия, выполняемые в хранимых процедурах и триггерах, могли быть выборочно отменены без полного отката всех действий в них. Внутренне автоматические точки сохранения используются для:

- отмены всех действий внутри блока BEGIN ... END, где происходит исключение;
- отмены всех действий, выполняемых в SP/триггере (или, в случае селективной SP, всех действий, выполненных с момента последнего оператора SUSPEND), если они завершаются преждевременно из-за непредусмотренной ошибки или исключения.

Каждый блок обработки исключений PSQL также ограничен автоматическими точками сохранения сервера.

SET TRANSACTION

Описание: Задаёт параметры транзакции и стартует её.

Синтаксис:

```
SET TRANSACTION
    [NAME hostvar]
    [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 ...]

- Опция NAME доступна только в ESQL. При этом должна быть объявлена и инициализирована переменная базового языка. Без опции NAME оператор SET TRANSACTION применяется к транзакции по умолчанию;
- Опция USING также доступна только в ESQL. Она задаёт базы данных, к которым транзакция может получить доступ;
- Опции IGNORE LIMBO и LOCK TIMEOUT не доступны в ESQL;
- Опции IGNORE LIMBO и NO WAIT являются взаимоисключающими;
- Опции по умолчанию для транзакции: READ WRITE + WAIT + SNAPSHOT.

IGNORE LIMBO

Описание: С этой опцией игнорируются записи, создаваемые “потерянными” (т.е. не завершёнными) транзакциями (limbo transaction). Транзакция считается “потерянной”, если не завершён второй этап двухфазного подтверждения (two-phase commit).

Примечание:

Параметр IGNORE LIMBO аналогичен параметру isc_tpb_ignore_limbo TPB, доступного в API со времен InterBase - в основном используются утилитой командной строки gfix.

LOCK TIMEOUT

Описание: Эта опция доступна только вместе при использовании WAIT. Она принимает неотрицательное целое число в качестве параметра, задавая максимальное количество секунд, в течение которых транзакция должна ожидать при возникновении конфликта блокировки. Если время ожидания прошло, а блокировка ещё не снята, то выдается сообщение об ошибке.

Примечание:

Это совершенно новая опция, добавленная в Firebird 2. Её эквивалент в API - новый параметр TPB isc_tpb_lock_timeout.

NO AUTO UNDO

При использовании опции NO AUTO UNDO не ведётся журнал, используемый для отмены изменений в случае отката транзакции. При откате транзакции в конечном счете сборка мусора выполняется в рамках других транзакций.

Эта опция может быть полезна при выполнении транзакции, в рамках которой производится много отдельных операторов, изменяющих данные, и при этом есть уверенность, что эта транзакция будет чаще всего завершаться успешно, а не откатываться.

Для транзакций, в рамках которых не выполняется никаких изменений, опция NO AUTO UNDO игнорируется

Примечание:

Опция NO AUTO UNDO имеет свой аналог в API - параметр TPB isc_tpb_no_auto_undo, доступный начиная с InterBase.

Глава 10

Безопасность

Базы данных, как и данные, хранимые в файлах базы данных, должны быть защищены. Firebird обеспечивает двухуровневую защиту данных – аутентификация пользователя на уровне сервера и привилегии на уровне базы данных. В данной главе рассказывается, каким образом управлять безопасностью вашей базы данных на каждом из уровней.

Безопасность всей базы данных зависит от проверки подлинности идентификатора пользователя. Информация о пользователях, зарегистрированных для конкретного сервера Firebird, хранится в особой базе данных безопасности (security database) – security2.fdb.

Имя пользователя может состоять максимум из 31 символа, при этом их регистр не учитывается. Пароль может состоять максимум из 8 символов (если ввести больше, лишние символы игнорируются), регистр - учитывается.

Встроенная версия сервера (embedded), не использует базу данных безопасности. Имя пользователя, указанное при подключении (не происходит аутентификация по паролю) используется для контроля доступа к объектам базы данных. Если при подключении был указан пользователь SYSDBA (или владелец базы данных), то он получит неограниченный доступ к базе данных.

В Firebird существует специальная учётная запись SYSDBA, которая существует вне всех ограничений безопасности и имеет полный доступ ко всем базам данных сервера. По умолчанию пароль SYSDBA – masterkey (точнее masterke, т.к. длина пароля ограничена 8 символами).

В Unix-подобных системах, Firebird будет трактовать пользователя Unix точно так же как и пользователя, хранящегося в собственной базе данных безопасности Firebird, до тех пор, пока сервер видит клиента в качестве доверенного хоста. Учетные записи пользователя должны существовать как на клиенте, так и на сервере. Для того чтобы установить доверительные отношения с хостом клиента, необходимо на сервере занести соответствующую запись в файл /etc/hosts.equiv или /etc/gds_hosts.equiv. В файле hosts.equiv прописываются доверительные отношения на уровне операционных систем, которые, соответственно, распространяются на все сервисы (например, rlogin, rsh, rcp). В файле gds_hosts.equiv устанавливаются доверительные отношения между хостами, только для Firebird. Формат записи идентичен для обоих файлов, и выглядит следующим образом:

```
hostname [username]
```

В Unix системах пользователь root может выступать в роли SYSDBA. Firebird в этом случае будет трактовать имя пользователя root как SYSDBA, и вы будете иметь доступ ко всем базам данных сервера.

Руководство по языку SQL СУБД Firebird

В операционных системах семейства Windows NT вы также можете пользоваться учётными записями ОС. Для этого необходимо включить разрешение на использование доверительной аутентификации (*Trusted Authentication*). Для этого необходимо установить параметр Authentication в файле конфигурации firebird.conf в значение Trusted или Mixed.

Администраторы операционной системы Windows автоматически не получают права SYSDBA при подключении к базе данных (если, конечно, разрешена доверенная авторизация). Имеют ли администраторы автоматические права SYSDBA зависит от установки значения флага AUTO ADMIN MAPPING.

Как уже отмечалось, в Firebird реализована двухуровневая модель безопасности. На первом уровне осуществляется аутентификация пользователя в момент подключения к базе данных, при этом используется база данных безопасности. Второй уровень реализуется уже на уровне самой базы данных. Все привилегии по доступу к объектам базы данных хранятся в самой базе. Авторизованный пользователь не имеет никаких привилегий до тех пор, пока какие либо права не будут предоставлены ему явно. При создании объекта только его создатель и SYSDBA имеет привилегии на него и может назначать привилегии другим пользователям, ролям или объектам.

Примечание:

К сожалению, вопросы создания новых объектов базы данных не контролируются в настоящий момент (Firebird 2.5.3). Любой авторизованный пользователь, имеющий доступ к базе данных, может создать любой допустимый объект базы данных.

Пользователь, создавший объект базы данных, становится его владельцем. Только владелец объекта базы данных и пользователи с административными привилегиями могут изменять или удалять объект базы данных.

Примечание:

К сожалению, в настоящий момент не все объекты базы данных ассоциированы с владельцем, что делает их уязвимыми (позволяет изменять и удалять любым пользователям). Это относится к доменам, функциям (UDF), BLOB фильтрам, генераторам (последовательностям) и исключениям.

SYSDBA или владелец объекта могут выдавать привилегии другим пользователям, в том числе и привилегии на право выдачи привилегий другим пользователям. Собственно сам процесс раздачи привилегий на уровне SQL реализуется двумя операторами: **GRANT**, **REVOKE**.

Роль (role) – объект базы данных, представляющий набор привилегий. Роли реализуют концепцию управления безопасностью на групповом уровне.

Операторы управления пользователями

В данном разделе описываются операторы создания, модификации и удаления учётных пользователей Firebird средствами операторов SQL. Такая возможность предоставлена следующим пользователям:

- SYSDBA;
- Любому пользователю, имеющему права на роль RDB\$ADMIN в базе данных пользователей и права на ту же роль для базы данных в активном подключении (пользователь должен подключаться к базе данных с ролью RDB\$ADMIN);
- При включённом флаге AUTO ADMIN MAPPING в базе данных пользователей (security2.fdb) – любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации – trusted authentication) без указания роли. При этом не важно, включен или выключен флаг AUTO ADMIN MAPPING в самой базе данных.

Непривилегированные пользователи могут использовать только оператор ALTER USER для изменения собственной учётной записи.

CREATE USER

Создание учётной записи пользователя Firebird.

Доступно: DSQL

Синтаксис:

```
CREATE USER username PASSWORD 'password'  
  [FIRSTNAME 'firstname']  
  [MIDDLENAME 'middlename']  
  [LASTNAME 'lastname']  
  [GRANT ADMIN ROLE];
```

Аргумент	Описание
<code>username</code>	Имя пользователя. Максимальная длина 31 символ.
<code>password</code>	Пароль пользователя. Может включать в себя до 32 символов, однако только первые 8 имеют значение. Чувствительно к регистру.
<code>firstname</code>	Вспомогательная информация: имя пользователя. Максимальная длина 32 символа.
<code>middlename</code>	Вспомогательная информация: «второе имя» (отчество, «имя отца») пользователя. Максимальная длина 32 символа.
<code>lastname</code>	Вспомогательная информация: фамилия пользователя. Максимальная длина 32 символа.

Руководство по языку SQL СУБД Firebird

Описание:

Оператор `CREATE USER` создаёт учётную запись пользователя Firebird. Пользователь должен отсутствовать в Firebird иначе будет выдано соответствующее сообщение об ошибке.

Предложение `PASSWORD` задаёт пароль пользователя. Необязательные предложения `FIRSTNAME`, `MIDDLENAME` и `LASTNAME` задают дополнительные атрибуты пользователя, такие как имя пользователя (имя человека), отчество и фамилия соответственно.

Если указана опция `GRANT ADMIN ROLE`, то новая учётная запись пользователя создаётся с правами роли `RDB$ADMIN` в базе данных пользователей (`security2.fdb`). Это позволяет вновь созданному пользователю управлять учётными записями пользователей, но не даёт ему специальных полномочий в обычных базах данных.

Для создания учётной записи пользователя текущий пользователь должен обладать административными привилегиями.

Примеры:

1. Создание пользователя с именем `bigshot`.

```
CREATE USER bigshot PASSWORD 'buckshot';
```

2. Создание пользователя `John` с дополнительными атрибутами (именем и фамилией).

```
CREATE USER john PASSWORD 'fYe_3Ksw'  
FIRSTNAME 'John'  
LASTNAME 'Doe';
```

3. Создание пользователя `superuser` с возможностью управления пользователями.

```
CREATE USER superuser PASSWORD 'kMn8Kjh'  
GRANT ADMIN ROLE;
```

Смотреть также [ALTER USER](#), [DROP USER](#)

ALTER USER

Изменение учётной записи пользователя Firebird.

Доступно: DSQL

Руководство по языку SQL СУБД Firebird

Синтаксис:

```
ALTER USER username
{
  [SET]
  [PASSWORD 'password']
  [FIRSTNAME 'firstname']
  [MIDDLENAME 'middlename']
  [LASTNAME 'lastname']
}
[ {GRANT | REVOKE} ADMIN ROLE ] ;
```

Аргумент	Описание
username	Имя пользователя.
password	Пароль пользователя. Может включать в себя до 32 символов, однако только первые 8 имеют значение. Чувствительно к регистру.
firstname	Вспомогательная информация: имя пользователя. Максимальная длина 32 символа.
middlename	Вспомогательная информация: «второе имя» (отчество, «имя отца») пользователя. Максимальная длина 32 символа.
lastname	Вспомогательная информация: фамилия пользователя. Максимальная длина 32 символа.

Описание:

Оператор ALTER USER изменяет данные учётной записи пользователя Firebird. В операторе ALTER USER должен присутствовать хотя бы одно из необязательных предложений.

Необязательное предложение PASSWORD задаёт новый пароль пользователя. Необязательные предложения FIRSTNAME, MIDDLENAME и LASTNAME позволяют изменить дополнительные атрибуты пользователя, такие как имя пользователя (имя человека), отчество и фамилия соответственно.

Предложение GRANT ADMIN ROLE предоставляет указанному пользователю привилегии роли RDB\$ADMIN в базе данных пользователей (security2.fdb). Это позволяет указанному пользователю управлять учётными записями пользователей, но не даёт ему специальных полномочий в обычных базах данных.

Предложение REVOKE ADMIN ROLE отбирает у указанного пользователя привилегии роли RDB\$ADMIN в базе данных пользователей (security2.fdb). Это запрещает указанному пользователю управлять учётными записями пользователей.

Для модификации чужой учётной записи пользователя текущий пользователь должен обладать административными привилегиями. Свои собственные учётные записи могут изменять любые пользователи, однако это не относится к опциям GRANT/REVOKE ADMIN ROLE для изменения которых,

Руководство по языку SQL СУБД Firebird

необходимы административные привилегии

Примеры:

1. Изменение пароля пользователя bobby и выдача ему привилегии управления пользователями.

```
ALTER USER bobby PASSWORD '67-UiT_G8'  
GRANT ADMIN ROLE;
```

2. Изменение дополнительных атрибутов (имени и фамилии) пользователя dan.

```
ALTER USER dan  
FIRSTNAME 'No_Jack'  
LASTNAME 'Kennedy';
```

3. Отбор привилегии управления пользователями у пользователя dumbbell.

```
ALTER USER dumbbell  
DROP ADMIN ROLE;
```

Смотреть также [CREATE USER](#), [DROP USER](#)

DROP USER

Удаление учётной записи пользователя Firebird.

Доступно: DSQL

Синтаксис:

```
DROP USER username;
```

Аргумент	Описание
username	Имя пользователя.

Описание:

Оператор DROP USER удаляет учётную запись пользователя Firebird.

Для удаления учётной записи пользователя текущий пользователь должен обладать административными привилегиями.

Примеры:

Руководство по языку SQL СУБД Firebird

Удаление пользователя bobby.

```
DROP USER bobby;
```

Смотреть также [CREATE USER](#), [ALTER USER](#)

Операторы управления ролями

В данном разделе рассматриваются вопросы создания и удаления ролей, а также разрешения или запрещения автоматического предоставления роли RDB\$ADMIN администраторам Windows (Trusted Authentication).

CREATE ROLE

Создание новой роли.

Доступно: DSQL, ESQL

Синтаксис:

```
CREATE ROLE rolename;
```

Аргумент	Описание
rolename	Имя роли. Максимальная длина 31 символ.

Описание:

Оператор CREATE ROLE создаёт новую роль. Имя роли должно быть уникальным среди имён ролей.

Предупреждение:

Желательно также чтобы имя роли было уникальным не только среди имён ролей, но и среди имён пользователей. Если вы создадите роль с тем же именем существующего пользователя, то такой пользователь не сможет подключиться к базе данных.

Создать роль может любой пользователь, подключённый к базе данных. Пользователь, создавший роль, становится её владельцем.

Примеры:

Создание роли SELLERS.

```
CREATE ROLE SELLERS;
```

Смотреть также [DROP ROLE](#), [GRANT](#), [REVOKE](#)

ALTER ROLE

Разрешает или запрещает автоматическое предоставление роли RDB\$ADMIN администраторам Windows, если используется доверительная авторизация (Trusted Authentication).

Доступно: DSQL

Синтаксис:

```
ALTER ROLE RDB$ADMIN {SET | DROP} AUTO ADMIN MAPPING;
```

Описание:

Оператор ALTER ROLE RDB\$ADMIN разрешает или запрещает автоматическое предоставление роли RDB\$ADMIN администраторам Windows в текущей базе данных, если используется доверительная авторизация (Trusted Authentication). По умолчанию автоматическое предоставление роли RDB\$ADMIN отключено.

Замечание:

Нет никаких операторов SQL, чтобы включить или выключить флаг AUTO ADMIN MAPPING в базе данных пользователей. Для этого можно использовать только утилиту командной строки gsec:

```
gsec -mapping set  
gsec -mapping drop
```

Этот оператор может быть выполнен пользователями с достаточными правами, а именно:

- SYSDBA;
- Любой пользователь, имеющий права на роль RDB\$ADMIN и указавший её при соединении с базой данных;
- Администраторы Windows, если разрешено автоматическое предоставление роли RDB\$ADMIN (при использовании Trusted Authentication).

Примеры:

1. Разрешение администратор Windows автоматически предоставлять роль RDB\$ADMIN.

```
ALTER ROLE RDB$ADMIN SET AUTO ADMIN MAPPING;
```

Руководство по языку SQL СУБД Firebird

2. Запретить администратор Windows автоматически предоставлять роль RDB\$ADMIN.

```
ALTER ROLE RDB$ADMIN DROP AUTO ADMIN MAPPING;
```

Смотреть также [Роль RDB\\$ADMIN](#)

DROP ROLE

Удаление существующей роли.

Доступно: SQL, ESQL

Синтаксис:

```
DROP ROLE rolename;
```

Аргумент	Описание
<i>rolename</i>	Имя роли.

Описание:

Оператор DROP ROLE удаляет существующую роль. При удалении роли все привилегии, предоставленные этой роли, отменяются.

Удалить роль могут:

- владелец роли;
- пользователь SYSDBA;
- любой пользователь, подключенный с ролью RDB\$ADMIN (роль должна быть назначена пользователю);
- пользователь операционной системы root (Linux);
- администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB\$ADMIN администраторам Windows.

Примеры:

Удаление роли SELLERS.

```
DROP ROLE SELLERS;
```

Смотреть также [CREATE ROLE](#), [GRANT](#), [REVOKE](#)

Операторы назначения привилегий

GRANT

Предоставление привилегий или назначение ролей.

Доступно: DSQL

Синтаксис:

```
GRANT {
  <privileges> ON [TABLE] {tablename | viewname}
  TO { <object_list>
      | <user_list> [WITH GRANT OPTION]
      | GROUP Unix_group}
  | EXECUTE ON PROCEDURE procname
  TO {<object_list> | <user_list>}
  | <role_granted> TO {PUBLIC | <role_grantee_list> [WITH ADMIN
OPTION]}
} [{GRANTED BY | AS} [USER] grantor];

<privileges> ::= ALL [PRIVILEGES] | <privilege_list>

<privilege_list> ::= {
  SELECT
  | DELETE
  | INSERT
  | UPDATE [(col [, col ...])]
  | REFERENCES [(col [, col ...])]
} [, <privilege_list> ...]

<object_list> ::= {
  PROCEDURE procname
  | TRIGGER trigname
  | VIEW viewname
  | PUBLIC
} [, <object_list> ...]

<user_list> ::= {
  [USER] username
  | [ROLE] rolename
  | Unix_user
} [, <user_list> ...]
```

Руководство по языку SQL СУБД Firebird

`<role_granted> ::= rolename [, rolename ...]`

`<role_grantee_list> ::= [USER] username [, [USER] username ...]`

Аргумент	Описание
tablename	Имя таблицы, к которой должно быть применена привилегия.
viewname	Имя представления, к которому должно быть применена привилегия.
procname	Имя хранимой процедуры, для которой должна быть выдана привилегия EXECUTE или которой будут даны привилегии.
col	Столбец таблицы, к которому должна быть применена привилегия.
Unix_group	Имя группы пользователей в операционных системах семейства UNIX.
username	Имя пользователя, для которого выдаются привилегии или которому назначается роль.
rolename	Имя роли.
trigname	Имя триггера.
grantor	Пользователь от имени, которого предоставляются привилегии.

Описание:

Оператор GRANT предоставляет одну или несколько привилегий для объектов базы данных пользователям, ролям, хранимым процедурам, триггерам и представлениям.

Авторизованный пользователь не имеет никаких привилегий до тех пор, пока какие либо права не будут предоставлены ему явно. При создании объекта только его создатель и SYSDBA имеет привилегии на него и может назначать привилегии другим пользователям, ролям или объектам.

Для таблиц существуют следующие привилегии:

Привилегия	Описание
ALL	Объединяет привилегии SELECT, INSERT, UPDATE, DELETE и REFERENCES.
SELECT	Разрешает выборку данных из таблицы или представления.
INSERT	Разрешает добавлять записи в таблицу или представление.
UPDATE	Разрешает изменять записи в таблице или представлении. Можно указать ограничения, чтобы можно было изменять только указанные столбцы.
DELETE	Разрешает удалять записи из таблицы или представления.
REFERENCES	Разрешает ссылаться на указанные столбцы внешним ключом. Необходимо указать для столбцов, на которых построен первичный ключ таблицы, если на неё есть ссылка внешним ключом другой таблицы.

Для выполнения хранимых процедур существует отдельная привилегия EXECUTE. Она позволяет выполнять хранимые процедуры и делать выборку данных из процедур выбора (с помощью оператора SELECT).

Руководство по языку SQL СУБД Firebird

В предложении TO указывается список пользователей, ролей и объектов базы данных (процедур, триггеров и представлений) для которых будут выданы перечисленные привилегии. Необязательные предложения USER и ROLE позволяют уточнить, кому именно выдаётся привилегия. Если ключевое слово USER или ROLE не указано, то сервер проверяет, существует ли роль с данным именем, если таковой не существует, то привилегии назначаются пользователю. Существование пользователя, которому выдаются права, не проверяются при выполнении оператора GRANT. Если привилегия выдаётся объекту базы данных, то необходимо обязательно указывать тип объекта.

Замечание:

Несмотря на то, что ключевые слова USER и ROLE не обязательные, желательно использовать их, чтобы избежать путаницы.

В SQL существует специальный пользователь PUBLIC, представляющий всех пользователей. Если какая-то операция разрешена пользователю PUBLIC, значит, любой аутентифицированный пользователь может выполнить эту операцию над указанным объектом.

Замечание:

Если привилегии назначены пользователю PUBLIC, то отозваны они должны быть у пользователя PUBLIC.

Необязательное предложение WITH GRANT OPTION позволяет пользователям, указанным в списке пользователей, передавать другим пользователям привилегии указанные в списке привилегий.

Другое назначение оператора GRANT заключается в назначении ролей для группы перечисленных пользователей. В этом случае после предложения GRANT следует список ролей, которые будут назначены списку пользователей, указанному после предложения TO.

Необязательное предложение WITH ADMIN OPTION позволяет пользователям, указанным в списке пользователей, назначать роли из списка ролей, указанных в списке ролей, другим пользователям.

При предоставлении прав в базе данных в качестве лица, предоставившего эти права, обычно записывается текущий пользователь. Используя предложение GRANTED BY можно предоставлять права от имени другого пользователя. При использовании оператора REVOKE после GRANTED BY права будут удалены только в том случае, если они были зарегистрированы от удаляющего пользователя. Для облегчения миграции из некоторых других реляционных СУБД нестандартное предложение AS поддерживается как синоним оператора GRANTED BY.

Предложение GRANTED BY может использовать:

Руководство по языку SQL СУБД Firebird

- Владелец базы данных;
- SYSDBA;
- Любой пользователь, имеющий права на роль RDB\$ADMIN и указавший её при соединении с базой данных;
- При использовании флага AUTO ADMIN MAPPING - любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации - trusted authentication), даже без указания роли.

Даже владелец роли не может использовать GRANTED BY, если он не находится в вышеупомянутом списке.

Примеры:

1. Назначить привилегии на чтение и вставку в таблицу SALES для пользователя ALEX

```
GRANT SELECT, INSERT ON TABLE SALES  
TO USER ALEX;
```

2. Назначить привилегии на чтение таблицы CUSTOMER ролям MANAGER, ENGINEER и пользователю IVAN

```
GRANT SELECT ON TABLE CUSTOMER  
TO ROLE MANAGER, ROLE ENGINEER, USER IVAN;
```

3. Назначить все привилегии на таблицу CUSTOMER для роли ADMINISTRATOR с возможностью передать свои привилегии другим пользователям или ролям

```
GRANT ALL ON TABLE CUSTOMER  
TO ROLE ADMINISTRATOR WITH GRANT OPTION;
```

4. Назначить привилегии на чтение таблицы COUNTRY всем пользователям, а также возможность ссылаться на столбец NAME таблицы COUNTRY

```
GRANT SELECT, REFERENCES (NAME) ON TABLE COUNTRY  
TO PUBLIC;
```

5. Назначить привилегии на чтение таблицы EMPLOYEE для пользователя IVAN от имени пользователя ALEX

```
GRANT SELECT ON TABLE EMPLOYEE  
TO USER IVAN GRANTED BY ALEX;
```

6. Назначить привилегии на обновление столбцов FIRST_NAME и LAST_NAME таблицы EMPLOYEE для пользователя IVAN

Руководство по языку SQL СУБД Firebird

```
GRANT UPDATE (FIRST_NAME, LAST_NAME) ON TABLE EMPLOYEE  
TO USER IVAN;
```

7. Назначить привилегии на вставку записей в таблицу EMPLOYEE_PROJECT для процедуры ADD_EMP_PROJ

```
GRANT INSERT ON EMPLOYEE_PROJECT  
TO PROCEDURE ADD_EMP_PROJ;
```

8. Назначить привилегии на выполнение процедуры ADD_EMP_PROJ для роли MANAGER

```
GRANT EXECUTE ON PROCEDURE ADD_EMP_PROJ  
TO ROLE MANAGER;
```

9. Назначить пользователю IVAN роли DIRECTOR и MANAGER

```
GRANT DIRECTOR, MANAGER  
TO USER IVAN;
```

10. Назначить пользователю ALEX роль MANAGER с возможностью назначать эту роль другим пользователям

```
GRANT MANAGER  
TO USER IVAN WITH ADMIN OPTION;
```

Смотреть также [REVOKE](#)

REVOKE

Отмена привилегий или отбор ролей.

Доступно: DSQL

Синтаксис:

```
REVOKE {  
  [GRANT OPTION FOR] <privileges>  
  ON [TABLE] {tablename | viewname}  
  FROM { <object_list>  
        | <user_list>  
        | GROUP Unix_group}  
  | EXECUTE ON PROCEDURE procname  
  FROM { <object_list>
```

Руководство по языку SQL СУБД Firebird

```

    | <user_list>
    | GROUP Unix_group}
| [ADMIN OPTION FOR] <role_grantee_list> FROM {PUBLIC |
<role_grantee_list>}
} [{GRANTED BY | AS} [USER] grantor]
| ALL ON ALL FROM <user_list>;

```

```
<privileges> ::= ALL [PRIVILEGES] | <privilege_list>
```

```
<privilege_list> ::= {
  SELECT
  | DELETE
  | INSERT
  | UPDATE [(col [,col ...])]
  | REFERENCES [(col [,col ...])]
  } [, <privilege_list> ...]

```

```
<object_list> ::= {
  PROCEDURE procname
  | TRIGGER trigrname
  | VIEW viewname
  | PUBLIC
  } [, <object_list> ...]

```

```
<user_list> ::= {
  [USER] username
  | [ROLE] rolename
  | Unix_user
  } [, <user_list> ...]

```

```
<role_granted> ::= rolename [, rolename ...]
```

```
<role_grantee_list> ::= [USER] username [, [USER] username ...]
```

Аргумент	Описание
tablename	Имя таблицы, у которой должна быть отозвана привилегия.
viewname	Имя представления, у которого должна быть отозвана привилегия.
procname	Имя хранимой процедуры, для которой должна быть отозвана привилегия EXECUTE или у которой должны быть отозваны привилегии.
trigrname	Имя триггера.
col	Столбец таблицы, у которого должна быть отозвана привилегия.
username	Имя пользователя, у которого отзываются привилегии или у которого отнимается роль.
rolename	Имя роли.
Unix_group	Имя группы пользователей в операционных системах семейства UNIX.
grantor	Пользователь от имени, которого отзываются привилегии.

Руководство по языку SQL СУБД Firebird

Описание:

Оператор REVOKE отменяет привилегии для пользователей, ролей, хранимых процедур, триггеров и представлений выданные оператором GRANT.

У таблиц можно отобразить следующие привилегии:

Привилегии	Описание
ALL	Объединяет привилегии SELECT, INSERT, UPDATE, DELETE и REFERENCES. Отменяет сразу все привилегии для таблицы или представления
SELECT	Отменяет привилегию на чтение из таблицы или представления
INSERT	Отменяет привилегию на добавление записи в таблицу или представление
UPDATE	Отменяет привилегию на изменение записи в таблице или представлении.
DELETE	Отменяет привилегии на удаление записи из таблицы или представления
REFERENCES	Отменяет возможность ссылаться на указанные столбцы внешним ключом.

У хранимых процедур можно отобразить привилегию EXECUTE. Она позволяет выполнять хранимые процедуры и делать выборку данных из процедур выбора (с помощью оператора SELECT).

В предложении FROM указывается список пользователей, ролей и объектов базы данных (процедур, триггеров и представлений) у которых будут отняты перечисленные привилегии. Необязательные предложения USER и ROLE позволяют уточнить, у кого именно выдаётся привилегия. Если ключевое слово USER или ROLE не указано, то сервер проверяет, существует ли роль с данным именем, если таковой не существует, то привилегии отбираются у пользователя. Если привилегия отбирается у объекта базы данных, то необходимо обязательно указывать тип объекта.

Только пользователь, который назначил привилегию, может удалить её. Если привилегии были назначены специальному пользователю PUBLIC, то отменять привилегии необходимо для пользователя PUBLIC. Специальный пользователь PUBLIC используется, когда необходимо предоставить привилегии сразу всем пользователям. Однако не следует рассматривать PUBLIC как группу пользователей.

Необязательное предложение GRANT OPTION FOR отменяет для соответствующего пользователя или роли право предоставления другим пользователям или ролям привилегии к таблицам, представлениям, триггерам, хранимым процедурам.

Другое назначение оператора REVOKE в отборе назначенных группе пользователей ролей оператором GRANT. В этом случае после предложения REVOKE следует список ролей, которые будут отозваны у списка

Руководство по языку SQL СУБД Firebird

пользователей, указанных после предложения FROM.

Необязательное предложение ADMIN OPTION FOR отменяет ранее предоставленную административную опцию (право на передачу предоставленной пользователю роли другим) из грантополучателей, не отменяя прав на роль. В одном операторе могут быть обработаны несколько ролей и/или грантополучателей.

При предоставлении прав в базе данных в качестве лица, предоставившего эти права, обычно записывается текущий пользователь. Используя предложение GRANTED BY можно предоставлять права от имени другого пользователя. При использовании оператора REVOKE после GRANTED BY права будут удалены только в том случае, если они были зарегистрированы от удаляющего пользователя. Для облегчения миграции из некоторых других реляционных СУБД нестандартное предложение AS поддерживается как синоним оператора GRANTED BY.

Предложение GRANTED BY может использовать:

- Владелец базы данных;
- SYSDBA;
- Любой пользователь, имеющий права на роль RDB\$ADMIN и указавший её при соединении с базой данных;
- При использовании флага AUTO ADMIN MAPPING - любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации - trusted authentication), даже без указания роли.

Даже владелец роли не может использовать GRANTED BY, если он не находится в вышеупомянутом списке.

Если после ключевого слова REVOKE указано предложение ALL ON ALL, то это позволяет отменить все привилегии (включая роли) на всех объектах от одного или более пользователей и/или ролей. Это - быстрый способ "очистить" (отобрать) права, когда пользователю должен быть заблокирован доступ к базе данных.

Примечания:

- Когда оператор REVOKE ALL ON ALL вызывается привилегированным пользователем (владельцем базы данных, SYSDBA или любым пользователем, у которого CURRENT_ROLE - RDB\$ADMIN), удаляются все права независимо от того, кто их предоставил. В противном случае удаляются только права, предоставленные текущим пользователем;
- Не поддерживается предложение GRANTED BY;
- Этот оператор не удаляет флаг пользователя, давшего права на хранимые процедуры, триггеры или представлений (права на такие объекты конечно удаляются).

Примеры:

Руководство по языку SQL СУБД Firebird

1. Отобрать привилегии на чтение и вставку в таблицу SALES для пользователя ALEX

```
REVOKE SELECT, INSERT ON TABLE SALES FROM USER ALEX;
```

2. Отобрать привилегии на чтение таблицы CUSTOMER ролям MANAGER, ENGINEER и пользователю IVAN

```
REVOKE SELECT ON TABLE CUSTOMER  
FROM ROLE MANAGER, ROLE ENGINEER, USER IVAN;
```

3. Отменяет возможность передавать любую из привилегии на таблицу CUSTOMER другим пользователям или ролям у роли ADMINISTRATOR

```
REVOKE GRANT OPTION FOR ALL ON TABLE CUSTOMER  
FROM ROLE ADMINISTRATOR;
```

4. Отозвать привилегии на чтение таблицы COUNTRY и возможность ссылаться на столбец NAME таблицы COUNTRY у любого пользователя (специального пользователя PUBLIC).

```
REVOKE SELECT, REFERENCES (NAME) ON TABLE COUNTRY  
FROM PUBLIC;
```

5. Отозвать привилегии на чтение таблицы EMPLOYEE для пользователя IVAN, которые были выданы от пользователя ALEX

```
REVOKE SELECT ON TABLE EMPLOYEE  
FROM USER IVAN GRANTED BY ALEX;
```

6. Отозвать привилегии на обновление столбцов FIRST_NAME и LAST_NAME таблицы EMPLOYEE для пользователя IVAN

```
REVOKE UPDATE (FIRST_NAME, LAST_NAME) ON TABLE EMPLOYEE  
FROM USER IVAN;
```

7. Отозвать привилегии на вставку записей в таблицу EMPLOYEE_PROJECT у процедуры ADD_EMP_PROJ

```
REVOKE INSERT ON EMPLOYEE_PROJECT  
FROM PROCEDURE ADD_EMP_PROJ;
```

11. Отозвать привилегии на выполнение процедуры ADD_EMP_PROJ у роли MANAGER

```
REVOKE EXECUTE ON PROCEDURE ADD_EMP_PROJ  
FROM ROLE MANAGER;
```

Руководство по языку SQL СУБД Firebird

12. Назначить пользователю IVAN роли DIRECTOR и MANAGER

```
REVOKE DIRECTOR, MANAGER FROM USER IVAN;
```

13. Отобрать у пользователя ALEX возможность назначать роль MANAGER другим пользователям

```
REVOKE ADMIN OPTION FOR MANAGER FROM USER IVAN;
```

14. Отобрать у пользователя IVAN все привилегии (включая роли) на все объекты

```
REVOKE ALL ON ALL FROM IVAN;
```

После выполнения этой команды у пользователя IVAN нет вообще никаких прав.

Смотреть также [GRANT](#)

Роль RDB\$ADMIN

Системная роль RDB\$ADMIN, присутствует в каждой базе данных. Предоставление пользователю роли RDB\$ADMIN в базе данных дает ему права SYSDBA, но только в этой базе данных. В обычной базе данных это означает полный контроль над всеми объектами. В базе данных пользователей это означает возможность создавать, изменять и удалять учетные записи пользователей. В обоих случаях пользователь с правами RDB\$ADMIN роли может всегда передавать эту роль другим. Другими словами, "WITH ADMIN OPTION" уже встроен в эту роль и эту опцию можно не указывать.

Предоставление роли RDB\$ADMIN в обычной базе данных

Для предоставления и удаления роли RDB\$ADMIN в обычной базе данных используются операторы GRANT и REVOKE, как и для назначения и отмены остальных ролей.

Синтаксис:

```
GRANT RDB$ADMIN TO username
```

```
REVOKE RDB$ADMIN FROM username
```

Аргумент	Описание
----------	----------

Руководство по языку SQL СУБД Firebird

username	Имя пользователя, которому назначается роль RDB\$ADMIN или у которого она отбирается.
----------	---

Права на роль RDB\$ADMIN могут давать:

- Владелец базы данных;
- SYSDBA;
- Любой пользователь, имеющий права на роль RDB\$ADMIN и указавший её при соединении с базой данных;
- При использовании флага AUTO ADMIN MAPPING - любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации - trusted authentication), даже без указания роли.

Смотреть также [GRANT](#), [REVOKE](#)

Использование роли RDB\$ADMIN в обычной базе данных

Для использования прав роли RDB\$ADMIN пользователь просто указывает её при соединении с базой данных.

Предоставление роли RDB\$ADMIN в базе данных пользователей

Так как никто не может соединиться с базой данных пользователей, то операторы GRANT и REVOKE здесь не могут использоваться. Вместо этого роль RDB\$ADMIN предоставляют и удаляют SQL командами управления пользователями: CREATE USER и ALTER USER, в которых указываются специальные опции GRANT ADMIN ROLE и REVOKE ADMIN ROLE.

Синтаксис:

```
CREATE USER newuser
PASSWORD 'password'
GRANT ADMIN ROLE
```

```
ALTER USER existinguser
GRANT ADMIN ROLE
```

```
ALTER USER existinguser
REVOKE ADMIN ROLE
```

Аргумент	Описание
newuser	Имя вновь создаваемого пользователя. Максимальная длина 31 символ.
existinguser	Имя существующего пользователя.

Руководство по языку SQL СУБД Firebird

password	Пароль пользователя. Может включать в себя до 32 символов, однако только первые 8 имеют значение. Чувствительно к регистру.
----------	---

Пожалуйста, помните, что GRANT ADMIN ROLE и REVOKE ADMIN ROLE это не операторы GRANT и REVOKE. Это параметры для CREATE USER и ALTER USER.

Права на роль RDB\$ADMIN могут давать:

- SYSDBA;
- Любой пользователь, имеющий права на роль RDB\$ADMIN в базе данных пользователей и указавший её при соединении с базой данных (или во время работы с утилитой gsec);
- При использовании флага AUTO ADMIN MAPPING - любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации - trusted authentication), даже без указания роли.

Смотреть также [CREATE USER](#), [ALTER USER](#)

Использование роли RDB\$ADMIN в базе данных пользователей

Для управления учетными записями пользователей через SQL пользователь, имеющий права на роль RDB\$ADMIN, должен подключиться к базе данных с этой ролью. Так как к базе данных пользователей не имеет права соединяться никто, то пользователь должен подключиться к обычной базе данных, где он также имеет права на роль RDB\$ADMIN. Он определяет роль при соединении с обычной базой данных и может в ней выполнить любой SQL запрос. Это не самое элегантное решение, но это единственный способ управлять пользователями через SQL запросы. Если нет обычной базы данных, где у пользователя есть права на роль RDB\$ADMIN, то управление учётными записями посредством SQL запросов недоступно.

AUTO ADMIN MAPPING

Операционная система: только Windows.

Администраторы операционной системы Windows автоматически не получают права SYSDBA при подключении к базе данных (если, конечно, разрешена доверенная авторизация). Имеют ли администраторы автоматические права SYSDBA зависит от установки значения флага AUTO ADMIN MAPPING. Это флаг в каждой из баз данных, который по умолчанию выключен. Если флаг AUTO ADMIN MAPPING включен, то он действует, когда администратор Windows: а) подключается с помощью доверенной аутентификации, и б) не определяет никакой роли при подключении. После успешного “auto admin” подключения текущей ролью будет являться RDB\$ADMIN.

Включение и выключение флага в обычной базе данных

Включение и выключение флага AUTO ADMIN MAPPING в обычной базе данных осуществляется следующим образом:

```
ALTER ROLE RDB$ADMIN SET AUTO ADMIN MAPPING
```

```
ALTER ROLE RDB$ADMIN DROP AUTO ADMIN MAPPING
```

Эти операторы могут быть выполнены пользователями с достаточными правами, а именно:

- SYSDBA;
- Любой пользователь, имеющий права на роль RDB\$ADMIN в базе данных пользователей и указавший её при соединении с базой данных (или во время работы с утилитой gsec);
- При использовании флага AUTO ADMIN MAPPING - любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации - trusted authentication), даже без указания роли. В обычных базах данных состояние флага AUTO ADMIN MAPPING проверяется только во время подключения. Если подключение производилось с ролью RDB\$ADMIN, то даже при удалении прав на неё права сохраняются до момента отключения от базы данных. Аналогично, включение флага AUTO ADMIN MAPPING не изменит текущую роль на RDB\$ADMIN для администраторов, которые уже были подключены к базе данных.

Смотреть также [ALTER ROLE](#)

Включение и выключение флага в базе данных пользователей

Нет никаких операторов SQL, чтобы включить или выключить флаг AUTO ADMIN MAPPING в базе данных пользователей. Для этого можно использовать только утилиту командной строки gsec:

```
gsec -mapping set
```

```
gsec -mapping drop
```

При этом в командной строке можно использовать и другие параметры, например, -user and -pass, или -trusted.

Включение/выключение флага AUTO ADMIN MAPPING разрешено выполнять только:

Руководство по языку SQL СУБД Firebird

- SYSDBA;
- При включенном флаге AUTO ADMIN MAPPING - любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации - trusted authentication) без указания роли.

В отличие от случая с обычными базами данных пользователи, подключённые к базе данных с ролью RDB\$ADMIN, не могут включить или выключить флаг AUTO ADMIN MAPPING в базе данных пользователей. Также заметьте, что администратор операционной системы Windows может только выключить флаг AUTO ADMIN MAPPING. При выключении флага он отключает сам механизм, который предоставлял ему доступ и, таким образом, он не будет в состоянии включить AUTO ADMIN MAPPING. Даже в интерактивном gsec сеансе новая установка флага сразу вступает в силу.

Приложения

Приложение 1. Поле **RDB\$VALID_BLR**

В версии Firebird 2.1 в системные таблицы RDB\$PROCEDURES и RDB\$TRIGGERS было добавлено поле RDB\$VALID_BLR. Целью его введения была установка флага о возможной не валидности модуля PSQL (процедуры или триггера) при изменении доменов или столбцов таблиц, от которых он зависит. При возникновении описанной выше ситуации флаг поле RDB\$VALID_BLR устанавливается в 0 для процедур и/или триггеров, код которых возможно является не валидным.

Нижеприведённый запрос находит процедуры и триггеры, зависящие от определенного домена (в примере это домен 'MYDOMAIN'), и выводит информацию о состоянии поля RDB\$VALID_BLR:

```
SELECT * FROM (
  SELECT 'Procedure',
         RDB$PROCEDURE_NAME, RDB$VALID_BLR
  FROM RDB$PROCEDURES
  UNION
  SELECT 'Trigger',
         RDB$TRIGGER_NAME, RDB$VALID_BLR
  FROM RDB$TRIGGERS) (TYPE, NAME, VALID)
WHERE EXISTS
  (SELECT *
   FROM RDB$DEPENDENCIES
   WHERE
     RDB$DEPENDENT_NAME = NAME AND
     RDB$DEPENDENT_ON_NAME = 'MYDOMAIN')
```

/*
 Замените MYDOMAIN фактическим именем проверяемого домена. Используйте заглавные буквы, если домен создавался нечувствительным к регистру — в противном случае используйте точное написание имени домена с учётом регистра
*/

Следующий запрос находит процедуры и триггеры, зависящие от определенного столбца таблицы (в примере это столбец 'MYCOLUMN' таблицы 'MYTABLE'), и выводит информацию о состоянии поля RDB\$VALID_BLR:

Руководство по языку SQL СУБД Firebird

```
SELECT * FROM (
  SELECT 'Procedure',
         RDB$PROCEDURE_NAME, RDB$VALID_BLR
  FROM RDB$PROCEDURES
  UNION
  SELECT 'Trigger',
         RDB$TRIGGER_NAME, RDB$VALID_BLR
  FROM RDB$TRIGGERS) (TYPE, NAME, VALID)
WHERE EXISTS
  (SELECT *
   FROM RDB$DEPENDENCIES
   WHERE RDB$DEPENDENT_NAME = NAME
        AND RDB$DEPENDENT_ON_NAME = 'MYTABLE'
  AND RDB$FIELD_NAME = 'MYCOLUMN')
/*
  Замените MYTABLE и MYCOLUMN фактическими именами
  проверяемой таблицы и её столбца. Используйте
заглавные
  буквы, если таблица и её столбец создавались
  нечувствительными к регистру — в противном случае
  используйте точное написание имени таблицы и её
столбца
  с учётом регистра
*/
```

К сожалению, не все случаи не валидности кода PSQL будут отражены в поле RDB\$VALID_BLR. Поэтому после изменения домена или столбца таблицы желательно тщательно проанализировать все процедуры и триггеры, о которых сообщают вышеупомянутые запросы - даже те, которые имеют 1 в столбце "RDB\$VALID_BLR".

Обратите внимание на то, что для модулей PSQL, наследованных от более ранних версий Firebird (включая многие системные триггеры, даже если база данных создавалась под версией Firebird 2.1 или выше), поле RDB\$VALID_BLR имеет значение NULL. Это *не означает*, что их BLR является недействительным.

Команды утилиты командной строки isql SHOW PROCEDURES и SHOW TRIGGERS при выводе информации отмечают звездочкой модули, у которых поле RDB\$VALID_BLR равно 0. Команды SHOW PROCEDURE PROCNAME и SHOW TRIGGER TRIGNAME, выводящие на экран код PSQL модуля, не сигнализируют пользователя о недопустимом BLR.

Приложение 2. Обработка ошибок, коды и сообщения

В данном приложении будут рассмотрены вопросы, касающиеся модулей PSQL, а конкретно, что происходит, если в результате работы происходит **исключение**, вопросы перехвата исключения и их обработки в выполняемом коде.

Доступно: PSQL

Типы исключений

При работе кода PSQL возможны следующие типы исключений:

- *Ошибки SQL* – сообщения SQL, при которых в системе взводится признак ошибки.
- *Внутренние ошибки Firebird*, которые имеют отношение к конкурирующему взаимодействию, данным, метаданным и условиям окружения.
- *Пользовательские исключения*, которые декларируются в базе как постоянные объекты и вызываются в коде PSQL для того, чтобы код, например, соответствовал определенным бизнес правилам.

В коде PSQL исключения обрабатываются при помощи оператора `WHEN`. Если исключение будет обработано в вашем коде, то вы обеспечите исправление или обход ошибки и позволите продолжить выполнение, - то клиенту **не** возвращается никакого сообщения об исключении.

Исключение приводит к прекращению выполнения в блоке. Вместо того чтобы передать выполнение на конечный оператор `END`, теперь процедура отыскивает уровни во вложенных блоках, начиная с блока где была вызвана ошибка, и переходит на внешние блоки, чтобы найти код обработчика, который «знает» о таком исключении. Она отыскивает первый оператор `WHEN`, который может обработать эту ошибку.

Пользовательские исключения не должны дублировать работу внутренне определенных исключений. Нужно определять в коде пользовательские исключения для использования там, где мы хотим выявлять ошибочные ситуации, которые нарушают *наши* бизнес – правила.

Смотрите также контекстные переменные `GDSCODE`, `SQLCODE`, `SQLSTATE`, таблицы со значениями контекстных переменных для различных видов системных исключений: [Коды ошибок SQLSTATE и их описание](#), [Коды ошибок GDSCODE их описание](#), и `SQLCODE`; `WHEN ... DO`, `CREATE EXCEPTION`, `EXCEPTION`.

Руководство по языку SQL СУБД Firebird

Системные исключения

Исключение представляет собой сообщение, которое генерируется, когда возникает ошибка.

Все обрабатываемые СУБД исключения имеют заранее определенные числовые значения для контекстных переменных и связанные с ними тексты сообщений. Сообщения об ошибке написаны по умолчанию на английском языке. Существуют и локализованные сборки СУБД, в которых сообщения об ошибках переведены на другие языки.

Примечание:

В селективных хранимых процедурах выходные строки, которые уже были получены клиентом в предыдущих циклах `FOR SELECT ... DO ... SUSPEND`, остаются доступными для клиента в случае если по мере выборки потом возникло исключение!

Смотрите также [Коды ошибок SQLSTATE](#) и их описание, [Коды ошибок GDSCODE](#) их описание, и [SQLCODE](#)

Пользовательские исключения. Создание

В Firebird существует простой синтаксис DDL, для создания пользовательских исключений, с текстами сообщений до 78 символов. Начиная с Firebird версии 1.5, вы можете динамически модифицировать текст сообщения в зависимости от контекста. Имя исключения должно быть уникальным в базе данных среди исключений, а в Диалекте 3 – может быть заключено в кавычки; в этом случае имя исключения будет чувствительно к регистру.

Синтаксис:

```
CREATE EXCEPTION [<имя исключения> [пользовательское  
сообщение]]
```

<имя исключения> ::= идентификатор длиной до 31 символа

пользовательское сообщение ::= текстовое сообщение,
заключенное в апострофы.

Текстовое сообщение идет в наборе символов NONE.

Примеры:

```
CREATE EXCEPTION ERROR_REFIN_RATE 'Пересечение диапазона  
ставок рефинансирования';
```

Руководство по языку SQL СУБД Firebird

Пример вызова исключением с динамически формируемым текстом:

```
...
EXCEPTION EX_BAD_TYPE 'Неверный тип записи с id ' ||
new.id;
...
```

Пользовательские исключения. Изменение и удаление

Изменением и/или удалением пользовательских исключений в Firebird может заниматься их владелец или пользователь **SYSDBA**.

Если исключение используется только в хранимых процедурах, вы можете изменять или удалять его в любое время. Если исключение используется в триггере, вы можете только его изменять и изменять только его текст сообщения.

Исходя из описанного выше, возьмите себе за правило удалять исключение из кода хранимых процедур при удалении исключения. Иначе при выполнении таких процедур возникнет ошибка выполнения по причине отсутствия пользовательского исключения в базе.

Примеры:

```
DROP EXCEPTION ERROR_REFIN_RATE;
```

```
ALTER EXCEPTION ERROR_REFIN_RATE 'Пересечение диапазона';
```

Совет:

В скриптах на обновление системы группируйте операторы **CREATE EXCEPTION** вместе для удобства работы и документирования. Рекомендуется использовать в наименовании исключений систему префиксов в соответствии с категориями пользовательских исключений.

Коды ошибок SQLSTATE и их описание

В данной главе приведены коды ошибок для контекстной переменной **SQLSTATE** и их описания. Коды ошибок **SQLSTATE** построены следующим образом: пяти символьный код ошибки состоит из SQL класса ошибки (2 символа) и SQL подкласса (3 символа).

В большинстве случаев, коды ошибок **SQLCODE** не соотносятся с кодами **SQLSTATE** «один в один». Коды ошибок **SQLSTATE** соответствуют SQL стандарту. *В то же время SQLCODE использовались много лет и в настоящий момент считаются устаревшими. В следующих версиях поддержка SQLCODE может полностью прекратиться.*

Руководство по языку SQL СУБД Firebird

Код SQLSTATE	Связанное сообщение	Примечание
<i>SQLCLASS 00 (Success)</i>		
0	Success	Успех
<i>SQLCLASS 01 (Warning)</i>		Класс 01 (предупреждения)
1000	General Warning	Общее предупреждение
1001	Cursor operation conflict	Конфликт при операции с курсором
1002	Disconnect error	Ошибка, связанная с разъединением
1003	NULL value eliminated in set function	Значение NULL устраняется в определении функции
1004	String data, right-truncated	Строковые данные, обрезание справа
1005	Insufficient item descriptor areas	Недостаточно элементов в области дескрипторов
1006	Privilege not revoked	Привилегии не отозваны
1007	Privilege not granted	Привилегии не розданы
1008	Implicit zero-bit padding	Неявное обрезание нулевого бита
1100	Statement reset to unprepared	Выражение сброшено в состояние unprepared
1101	Ongoing transaction has been committed	Текущая транзакция завершена COMMIT
1102	Ongoing transaction has been rolled back	Текущая транзакция завершена ROLLED BACK
<i>SQLCLASS 02 (No Data)</i>		Класс ошибок 02; (Нет данных)
2000	No data found or no rows affected	Данные не обнаружены или не доступны строки.

Руководство по языку SQL СУБД Firebird

<i>SQLCLASS 07 (Dynamic SQL error)</i>		Класс ошибок 07 (Ошибки DSQL)
7000	Dynamic SQL error	Ошибка DSQL
7001	Wrong number of input parameters	Неверное число входных параметров
7002	Wrong number of output parameters	Неверное число выходных параметров
7003	Cursor specification cannot be executed	Определение курсора не может быть выполнено
7004	USING clause required for dynamic parameters	Для динамического параметра требуется кляуза USING
7005	Prepared statement not a cursor-specification	Подготовленное предложение не является курсоро - специфичным
7006	Restricted data type attribute violation	Исключение по причине запрещенного типа данных для атрибута
7007	USING clause required for result fields	Для возвращаемого поля требуется кляуза USING
7008	Invalid descriptor count	Неверный счетчик дескрипторов
7009	Invalid descriptor index	Неверный индекс дескриптора
<i>SQLCLASS 08 (Connection Exception)</i>		Класс ошибок 08 (Исключения коннекта)
8001	Client unable to establish connection	Клиент не может установить соединение
8002	Connection name in use	Имя коннекта уже используется
8003	Connection does not exist	Соединение не существует
8004	Server rejected the connection	Сервер отверг подключение
8006	Connection failure	Ошибка при подключении
8007	Transaction resolution unknown	Неизвестно разрешение на

Руководство по языку SQL СУБД Firebird

		транзакцию
<i>SQLCLASS 0A (Feature Not Supported)</i>		Класс ошибок 0A (Не поддерживается)
<i>0A000</i>	Feature Not Supported	Конструкция не поддерживается
<i>SQLCLASS 0B (Invalid Transaction Initiation)</i>		Класс ошибок 0A (неверная инициализация транзакции)
<i>0B000</i>	Invalid transaction initiation	Неверная инициализация транзакции
<i>SQLCLASS 0L (Invalid Grantor)</i>		Класс ошибок 0L (неверный grantor)
<i>0L000</i>	Invalid grantor	Неверный grantor
<i>SQLCLASS 0P (Invalid Role Specification)</i>		Класс ошибок 0P (неверная спецификация РОЛИ)
<i>0P000</i>	Invalid role specification	Неверная спецификация РОЛИ
<i>SQLCLASS 0U (Attempt to Assign to Non-Updatable Column)</i>		Класс ошибок 0U (попытка доступа к не обновляемому столбцу)
<i>0U000</i>	Attempt to assign to non-updatable column	Попытка доступа к не обновляемому столбцу
<i>SQLCLASS 0V (Attempt to Assign to Ordering Column)</i>		Класс ошибок 0V (попытка доступа к сортируемому столбцу)
<i>0V000</i>	Attempt to assign to Ordering column	Попытка доступа к сортируемому столбцу
<i>SQLCLASS 20 (Case Not Found For Case Statement)</i>		Класс 20 (не обнаружен CASE для предложения CASE)
<i>20000</i>	Case not found for case statement	Не обнаружен CASE для предложения CASE
<i>SQLCLASS 21 (Cardinality Violation)</i>		Класс 21 (Нарушения определения)
<i>21000</i>	Cardinality violation	Нарушения определения

Руководство по языку SQL СУБД Firebird

21S01	Insert value list does not match column list	Вставляемое значения не соответствует списку столбцов
21S02	Degree of derived table does not match column list	Состояние производной таблицы не соответствует списку столбцов
<i>SQLCLASS 22 (Data Exception)</i>		Класс ошибок 22 (исключения, вызванные данными)
22000	Data exception	Исключения данных
22001	String data, right truncation	Строковые данные, усекание справа
22002	Null value, no indicator parameter	Значение NULL, параметр не обозначен
22003	Numeric value out of range	Числовое значение вышло за предел достижимого
22004	Null value not allowed	Значение NULL не допустимо
2205	Error in assignment	Ошибка присваивания
2206	Null value in field reference	Значение NULL в поле ссылки
2207	Invalid datetime format	Неверный формат даты/времени
22008	Datetime field overflow	Переполнение в поле даты/времени
22009	Invalid time zone displacement value	Неверная временная зона, неверное значение
2200A	Null value in reference target	Значение NULL в целевой ссылке
2200B	Escape character conflict	Конфликт символа управления
2200C	Invalid use of escape character	Неверное использование управляющего символа
2200D	Invalid escape octet	Неверный октет для управляющего символа
2200E	Null value in array target	Значение NULL в массиве назначения

Руководство по языку SQL СУБД Firebird

2200F	Zero-length character string	Нулевая длина строки символов
2200G	Most specific type mismatch	Наиболее определенное несоответствие типов
22010	Invalid indicator parameter value	Неверный индикатор значения параметра
22011	Substring error	Ошибка подстроки
22012	Division by zero	Деление на ноль
22014	Invalid update value	Неверное значение в операции update
22015	Interval field overflow	Переполнение интервала в поле
22018	Invalid character value for cast	Неверный символ для преобразования типов
22019	Invalid escape character	Неверный символ управления
2201B	Invalid regular expression	Неверное регулярное выражение
2201C	Null row not permitted in table	Строка содержащая NULL не допустима для таблицы
22020	Invalid limit value	Неверное значение лимита
22021	Character not in repertoire	Символ вне диапазона
22022	Indicator overflow	Переполнение индикатора
22023	Invalid parameter value	Неверное значение параметра
22024	Character string not properly terminated	Символьная строка имеет некорректный замыкающий символ
22025	Invalid escape sequence	Неверная управляющая последовательность
22026	String data, length mismatch	Строковые данные, длина неверная
22027	Trim error	Ошибка операции TRIM

Руководство по языку SQL СУБД Firebird

22028	Row already exists	Строка уже существует
2202D	Null instance used in mutator function	NULL экземпляр используется для мутирующей функции
2202E	Array element error	Ошибка элемента массива
2202F	Array data, right truncation	Данные массива, обрезание справа
<i>SQLCLASS 23 (Integrity Constraint Violation)</i>		Класс ошибок 23 (Нарушение целостности констрейна)
23000	Integrity constraint violation	Нарушение целостности констрейна
<i>SQLCLASS 24 (Invalid Cursor State)</i>		Класс ошибок 24 (неверное состояние курсора)
24000	Invalid cursor state	Неверное состояние курсора
24504	The cursor identified in the UPDATE, DELETE, SET, or GET statement is not positioned on a row	Курсор определенный для UPDATE, DELETE, SET или GET операции не позиционирован по строке
<i>SQLCLASS 25 (Invalid Transaction State)</i>		Класс ошибок 25 (неверное состояние транзакции)
25000	Invalid transaction state	Неверное состояние транзакции
25	xxxx	Класс ошибок 25
25S01	Transaction state	Состояние транзакции
25S02	Transaction is still active	Транзакция до сих пор активная
25S03	Transaction is rolled back	Транзакция откатена
<i>SQLCLASS 26 (Invalid SQL Statement Name)</i>		Класс ошибок 26 (неверное имя SQL предложения)
26000	Invalid SQL statement name	Неверное имя SQL предложения
<i>SQLCLASS 27 (Triggered Data Change Violation)</i>		Класс ошибок 27 (ошибки изменения данных триггером)
27000	Triggered data change violation	Ошибки изменения данных

Руководство по языку SQL СУБД Firebird

		триггером
<i>SQLCLASS 28 (Invalid Authorization Specification)</i>		Класс ошибок 28 (неверная спецификация авторизации)
28000	Invalid authorization specification	Неверная спецификация авторизации
<i>SQLCLASS 2B (Dependent Privilege Descriptors Still Exist)</i>		Класс ошибок 2B (зависимые описания привилегий еще существуют)
2B000	Dependent privilege descriptors still exist	Зависимые описания привилегий еще существуют
<i>SQLCLASS 2C (Invalid Character Set Name)</i>		Класс ошибок 2C (неверное имя таблицы символов)
2C000	Invalid character set name	Неверное имя таблицы символов
<i>SQLCLASS 2D (Invalid Transaction Termination)</i>		Класс ошибок 2D (неверное завершение транзакции)
2D000	Invalid transaction termination	Неверное завершение транзакции
<i>SQLCLASS 2E (Invalid Connection Name)</i>		Класс ошибок 2E (неверное имя соединения)
2,00E+00	Invalid connection name	Неверное имя соединения
<i>SQLCLASS 2F (SQL Routine Exception)</i>		Класс ошибок 2F (процедурные исключения SQL)
2F000	SQL routine exception	Процедурное исключение SQL
2F002	Modifying SQL-data not permitted	На модификацию SQL данных нет доступа
2F003	Prohibited SQL-statement attempted	Запрещенное SQL предложение встречается
2F004	Reading SQL-data not permitted	Нет доступа на чтение SQL данных
2F005	Function executed no return statement	Исполняемая функция не имеет возвращаемого выражения

Руководство по языку SQL СУБД Firebird

<i>SQLCLASS 33 (Invalid SQL Descriptor Name)</i>		Класс ошибок 33 (неверное имя SQL описания)
33000	Invalid SQL descriptor name	Неверное имя SQL описания
<i>SQLCLASS 34 (Invalid Cursor Name)</i>		Класс ошибок 34 (неверное имя курсора)
34000	Invalid cursor name	Неверное имя курсора
<i>SQLCLASS 35 (Invalid Condition Number)</i>		Класс ошибок 35 (неверный номер условия)
35000	Invalid condition number	Неверный номер условия
<i>SQLCLASS 36 (Cursor Sensitivity Exception)</i>		Класс ошибок 36 (ошибка восприятия курсора)
36001	Request rejected	Запрос отвергнут
36002	Request failed	Запрос ошибочный
<i>SQLCLASS 37 (Invalid Identifier)</i>		Класс ошибок 37 (неверный идентификатор)
37000	Invalid identifier	Неверный идентификатор
37001	Identifier too long	Идентификатор слишком длинный
<i>SQLCLASS 38 (External Routine Exception)</i>		Класс ошибок 38 (ошибка внешних процедур)
38000	External routine exception	Ошибка внешних процедур
<i>SQLCLASS 39 (External Routine Invocation Exception)</i>		Класс ошибок 39 (ошибка вызова внешних процедур)
39000	External routine invocation exception	Ошибка вызова внешних процедур
<i>SQLCLASS 3B (Invalid Save Point)</i>		Класс ошибок 3B (неверный Save Point)
3B000	Invalid save point	Неверный Save Point
<i>SQLCLASS 3C (Ambiguous Cursor Name)</i>		Класс ошибок 3C (имя курсора не

Руководство по языку SQL СУБД Firebird

		однозначное)
3C000	Ambiguous cursor name	Имя курсора неоднозначное
<i>SQLCLASS 3D (Invalid Catalog Name)</i>		Класс ошибок 3D (неверное имя каталога)
3D000	Invalid catalog name	Неверное имя каталога
3D001	Catalog name not found	Каталог с таким именем не обнаружен
<i>SQLCLASS 3F (Invalid Schema Name)</i>		Класс ошибок 3F (неверное имя схемы)
3F000	Invalid schema name	Неверное имя схемы
<i>SQLCLASS 40 (Transaction Rollback)</i>		Класс ошибок 40 (откат транзакции)
40000	Ongoing transaction has been rolled back	Текущая транзакция была откатена
40001	Serialization failure	Отказ сериализации
40002	Transaction integrity constraint violation	Нарушение условия целостности транзакции
40003	Statement completion unknown	Неизвестно состояние завершения транзакции
<i>SQLCLASS 42 (Syntax Error or Access Violation)</i>		Класс ошибок 42 (синтаксическая ошибка или ошибка доступа)
42000	Syntax error or access violation	Синтаксическая ошибка или ошибка доступа
42702	Ambiguous column reference	Неоднозначная ссылка на столбец
42725	Ambiguous function reference	Неоднозначная ссылка на функцию
42818	The operands of an operator or function are not compatible	Операнды оператора или функции являются не совместимыми
42S01	Base table or view already exists	Таблица в базе или view уже существует

Руководство по языку SQL СУБД Firebird

42S02	Base table or view not found	Таблица в базе или view не найдены
42S11	Index already exists	Индекс уже существует
42S12	Index not found	Индекс не найден
42S21	Column already exists	Столбец уже существует
42S22	Column not found	Столбец не найден
<i>SQLCLASS 44 (With Check Option Violation)</i>		Класс ошибок 44 (нарушение опции WITH CHECK)
44000	WITH CHECK OPTION Violation	Нарушение опции WITH CHECK
<i>SQLCLASS 45 (Unhandled User-defined Exception)</i>		Класс ошибок 44 (неизвестное исключение определенное пользователем)
45000	Unhandled user-defined exception	Неизвестное исключение определенное пользователем
<i>SQLCLASS 54 (Program Limit Exceeded)</i>		Класс ошибок 54 (превышены ограничения программы)
54000	Program limit exceeded	Превышены ограничения программы
54001	Statement too complex	Выражение слишком сложное
54011	Too many columns	Слишком много столбцов
54023	Too many arguments	Слишком много аргументов
<i>SQLCLASS HY (CLI-specific Condition)</i>		Класс ошибок HY (условия CLI-specific)
HY000	CLI-specific condition	Условия CLI-specific
HY001	Memory allocation error	Ошибка выделения памяти
HY003	Invalid data type in application descriptor	Неверный тип данных в дескрипторе приложения

Руководство по языку SQL СУБД Firebird

<i>HY004</i>	Invalid data type	Неверный тип данных
<i>HY007</i>	Associated statement is not prepared	Связанный оператор не подготовлен
<i>HY008</i>	Operation canceled	Операция отменена
<i>HY009</i>	Invalid use of null pointer	Неправильное использование нулевого указателя
<i>HY010</i>	Function sequence error	Ошибка последовательности функций
<i>HY011</i>	Attribute cannot be set now	Атрибут не может быть установлен сейчас
<i>HY012</i>	Invalid transaction operation code	Неверный код транзакции операции
<i>HY013</i>	Memory management error	Ошибка управления памятью
<i>HY014</i>	Limit on the number of handles exceeded	Достигнут лимит числа указателей
<i>HY015</i>	No cursor name available	Недоступен курсор без имени
<i>HY016</i>	Cannot modify an implementation row descriptor	Невозможно изменить реализацию дескриптора строки
<i>HY017</i>	Invalid use of an automatically allocated descriptor handle	Неверное использование автоматически выделяемого дескриптора указателей
<i>HY018</i>	Server declined the cancellation request	Сервер отклонил запрос на отмену
<i>HY019</i>	Non-string data cannot be sent in pieces	Не строковые данные не могут быть отправлены по кускам
<i>HY020</i>	Attempt to concatenate a null value	Попытка конкатенации значения NULL
<i>HY021</i>	Inconsistent descriptor information	Противоречивая информация о дескрипторе
<i>HY024</i>	Invalid attribute value	Неверное значение атрибута
<i>HY055</i>	Non-string data cannot be used with string routine	Не строковые данные не могут быть использованы со строковой

Руководство по языку SQL СУБД Firebird

		процедурой
<i>HY090</i>	Invalid string length or buffer length	Неверная длина строки или длина буфера
<i>HY091</i>	Invalid descriptor field identifier	Неверный дескриптор идентификатора поля
<i>HY092</i>	Invalid attribute identifier	Неверный идентификатор атрибута
<i>HY095</i>	Invalid FunctionId specified	Неверное указание ID функции
<i>HY096</i>	Invalid information type	Неверный тип информации
<i>HY097</i>	Column type out of range	Тип столбца вне диапазона
<i>HY098</i>	Scope out of range	Определение вне диапазона
<i>HY099</i>	Nullable type out of range	Типы с допустимыми NULL вне диапазона
<i>HY100</i>	Uniqueness option type out of range	Тип опции «уникальность» вне диапазона
<i>HY101</i>	Accuracy option type out of range	Тип опции «точность» вне диапазона
<i>HY103</i>	Invalid retrieval code	Неверный код поиска
<i>HY104</i>	Invalid LengthPrecision value	Неверное значение длина/точность
<i>HY105</i>	Invalid parameter type	Неверный тип параметра
<i>HY106</i>	Invalid fetch orientation	Неверное направление для fetch
<i>HY107</i>	Row value out of range	Значение строки вне диапазона
<i>HY109</i>	Invalid cursor position	Неверная позиция курсора
<i>HY110</i>	Invalid driver completion	Неверное код завершения драйвера
<i>HY111</i>	Invalid bookmark value	Неверное значение метки bookmark
<i>HYC00</i>	Optional feature not implemented	Опциональная функция не реализована

Руководство по языку SQL СУБД Firebird

<i>HYT00</i>	Timeout expired	Достигнут тайм-аут
<i>HYT01</i>	Connection timeout expired	Достигнут тайм-аут соединения
<i>SQLCLASS XX (Internal Error)</i>		SQLCLASS XX (внутренние ошибки)
<i>XX000</i>	Internal error	Внутренняя ошибка
<i>XX001</i>	Data corrupted	Данные разрушены
<i>XX002</i>	Index corrupted	Индекс разрушен

Коды ошибок GDSCODE их описание, и SQLCODE

Таблица ошибок содержит числовое и символьное значения GDSCODE, текст сообщения об ошибке и описание ошибки. Также приводится SQLCODE ошибки.

В настоящее время SQLCODE использовались много лет и СЕЙЧАС считаются устаревшим. В следующих версиях поддержка SQLCODE может полностью прекратиться.

Данная таблица взята из документации к Firebird 2.1.

SQLCODE	GDSCODE	SYMBOL	ТЕХТ описание и перевод сообщения
101	335544366	Segment	Segment buffer length shorter than expected Длина буфера сегмента меньше, чем ожидается
100	335544338	from_no_match	No match for first value expression Нет соответствия для первого значения выражения
100	335544354	no_record	Invalid database key Неверный ключ базы данных
100	335544367	segstr_eof	Attempted retrieval of more segments than exist Попытка обращения к сегменту

Руководство по языку SQL СУБД Firebird

			большему чем их существует	
100	335544374	stream_eof	Attempt to fetch past the last record in a record stream Попытка получения в потоке записей записи, следующей за последней	
0	335741039	gfix_opt_SQL_dialect	-sql_dialect	set database dialect n
0	335544875	bad_debug_format	Bad debug info format Неверный формат отладочной информации	
-84	335544554	nonsql_security_rel	Table/procedure has non-SQL security class defined Для таблицы/процедуры определен HE-SQL класс безопасности	
-84	335544555	nonsql_security fld	Column has non-SQL security class defined Для столбца определен HE-SQL класс безопасности	
-84	335544668	dsq procedure_use_err	Procedure @1 does not return any values Процедура <строка> не возвращает никакого значения	
-85	335544747	usrname_too_long	The username entered is too long. Maximum length is 31 bytes. Введенной имя пользователя очень длинное. Максимальная длина 31 байт.	
-85	335544748	password_too_long	The password specified is too long. Maximum length is 8 bytes. Введенный пароль очень длинный. Максимальная длина 8 байт.	
-85	335544749	usrname_required	A username is required for this operation. Для этой операции требуется имя пользователя	
-85	335544750	password_required	A password is required for this operation Для этой операции требуется пароль	
-85	335544751	bad_protocol	The network protocol specified is invalid Указан неверный сетевой протокол	

Руководство по языку SQL СУБД Firebird

-85	335544752	dup_username_found	A duplicate user name was found in the security database В базе данных безопасности обнаружено дублирование имен пользователей
-85	335544753	username_not_found	The user name specified was not found in the security database Указанное имя пользователя не найдено в базе данных безопасности
-85	335544754	error_adding_sec_record	An error occurred while attempting to add the user. Ошибка произошла при попытке добавления пользователя
-85	335544755	error_modifying_sec_record	An error occurred while attempting to modify the user record. Ошибка произошла при попытке редактирования записи о пользователе
-85	335544756	error_deleting_sec_record	An error occurred while attempting to delete the user record. Ошибка произошла при попытке удаления записи о пользователе
-85	335544757	error_updating_sec_db	An error occurred while updating the security database. Ошибка произошла при изменении базы данных безопасности
-103	335544571	dsql_constant_err	Data type for constant unknown Неизвестный тип данных для константы
-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 Точность от 10 до 18 в SQL диалекте 1 изменена для DOUBLE PRECISION до 64-битного масштабируемого целого в диалекте 3
-104	336003077	sql_db_dialect_dtype_unsupport	Database SQL dialect @1 does not support reference to @2 datatype База данных SQL с диалектом <номер> не поддерживает ссылку на

Руководство по языку SQL СУБД Firebird

			<строка> тип данных
-104	336003087	dsql_invalid_label	Label @1 @2 in the current scope Метка <строка> <строка> находится в текущей зоне видимости
-104	336003088	dsql_datatypes_not_comparable	Datatypes @1 are not comparable in expression @2 Тип данных <строка> не сравним в выражении <строка>
-104	335544343	invalid_blr	Invalid request BLR at offset @1 Неверный запрос BLR со смещением <число >
-104	335544390	syntaxerr	BLR syntax error: expected @1 at offset @2, encountered @3 Ошибка синтаксиса BLR: ожидается <строка> по смещению <число >, встречено <число >
-104	335544425	ctxinuse	Context already in use (BLR error) Контекст находится в использовании (ошибка BLR)
-104	335544426	ctxnotdef	Context not defined (BLR error) Контекст не определен (ошибка BLR)
-104	335544429	badparnum	Bad parameter number Неверный номер параметра
-104	335544440	bad_msg_vec	--
-104	335544456	invalid_sdl	Invalid slice description language at offset @1 Неверный фрагмент языка описания по смещению <число>
-104	335544570	dsql_command_err	Invalid command Неверная команда
-104	335544579	dsql_internal_err	Internal error Внутренняя ошибка
-104	335544590	dsql_dup_option	Option specified more than once Режим указан более одного раза

Руководство по языку SQL СУБД Firebird

-104	335544591	dsql_tran_err	Unknown transaction option Неизвестный режим транзакции
-104	335544592	dsql_invalid_array	Invalid array reference Неверная ссылка на массив
-104	335544608	command_end_err	Unexpected end of command Неожиданное завершение команды
-104	335544612	token_err	Token unknown Неизвестный синтаксический элемент
-104	335544634	dsql_token_unk_err	Token unknown- line @1, column @2 Неизвестный синтаксический элемент – строка <число>, символ <число>
-104	335544709	dsql_agg_ref_err	Invalid aggregate reference Неверная ссылка на агрегат
-104	335544714	invalid_array_id	Invalid blob id Неверный идентификатор BLOB
-104	335544730	cse_not_supported	Client/Server Express not supported in this release Client/Server Express не поддерживается в этом релизе
-104	335544743	token_too_long	Token size exceeds limit Размер синтаксического элемента превышает предел
-104	335544763	invalid_string_constant	A string constant is delimited by double quotes Строковая константа определена в кавычках
-104	335544764	transitional_date	DATE must be changed to TIMESTAMP DATE должно измениться TIMESTAMP
-104	335544796	sql_dialect_datatype_unsupport	Client SQL dialect @1 does not support reference to @2 datatype SQL диалект <номер> клиента не поддерживает ссылку на тип данных <строка>
-104	335544798	depend_on_uncommitted_rel	You created an indirect dependency on uncommitted metadata. You must roll back the current transaction. Вы создали непрямую зависимость на неподтвержденные метаданные. Вы

Руководство по языку SQL СУБД Firebird

			должны отменить текущую транзакцию.
-104	335544821	dsql_column_pos_err	Invalid column position used in the @1 clause В предложении <строка> используется неверная позиция столбца
-104	335544822	dsql_agg_where_err	Cannot use an aggregate function in a WHERE clause, use HAVING instead Невозможно использовать агрегатную функцию в предложении WHERE, вместо этого используйте HAVING
-104	335544823	dsql_agg_group_err	Cannot use an aggregate function in a GROUP BY clause Невозможно использовать агрегатную функцию в кляузе GROUP BY
-104	335544824	dsql_agg_column_err	Invalid expression in the @1 (not contained in either an aggregate function or the GROUP BY clause) Неверное выражение в <строка> (ни содержится ни в агрегатной функции ни в кляузе GROUP BY)
-104	335544825	dsql_agg_having_err	Invalid expression in the @1 (neither an aggregate function nor a part of the GROUP BY clause) Неверное выражение в <строка> (не агрегатная функция, ни часть кляузы GROUP BY)
-104	335544826	dsql_agg_nested_err	Nested aggregate functions are not allowed Вложенные агрегатные функции не допустимы
-104	335544849	malformed_string	Malformed string Искаженная (некорректная) строка
-104	335544851	command_end_err2	Unexpected end of command- line @1, column @2 Неожиданный конец команд — линия <число>, колонка <число>
-104	336397215	dsql_max_sort_items	Cannot sort on more than 255 items Не могу сортировать условие более чем из 255 элементов
-104	336397216	dsql_max_group_items	Cannot group on more than 255 items Не могу группировать более чем 255 элементов

Руководство по языку SQL СУБД Firebird

-104	336397217	dsql_conflicting_sort_field	Cannot include the same field (@1.@2) twice in the ORDER BY clause with conflicting sorting options Не могу включить такое же поле (<строка>.<строка>) дважды в клязу ORDER BY с противоречивыми параметрами сортировки
-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 Список столбцов в производной таблице <строка> содержит больше столбцов, чем количество элементов в SELECT
-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 Список столбцов из производной таблицы имеет меньше столбцов, чем количество элементов в SELECT
-104	336397220	dsql_derived_field_unnamed	No column name specified for column number @1 in derived table @2 Нет имени столбца указанного для столбца под номером <число> в производной таблице <строка>
-104	336397221	dsql_derived_field_dup_name	Column @1 was specified multiple times for derived table @2 Столбец <строка> был указан несколько раз для производной таблицы <строка>
-104	336397222	dsql_derived_alias_select	Internal dsql error: alias type expected by pass1_expand_select_node Внутренняя ошибка DSQL: тип всевдоним ожидался pass1_expand_select_node
-104	336397223	dsql_derived_alias_field	Internal dsql error: alias type expected by pass1_field Внутренняя ошибка DSQL: тип всевдоним ожидался pass1_field
-104	336397224	dsql_auto_field_bad_pos	Internal dsql error: column position out of range in pass1_union_auto_cast Внутренняя ошибка DSQL: позиция столбца вышла из диапазона в pass1_union_auto_cast
-104	336397225	dsql_cte_wrong_reference	Recursive CTE member (@1) can refer itself only in FROM clause

Руководство по языку SQL СУБД Firebird

			Рекурсивная часть CTE (<строка>) может ссылаться сама на себя только в кляузе FROM
-104	336397226	dsql_cte_cycle	CTE '@1' has cyclic dependencies CTE '<строка>' имеет циклические зависимости
-104	336397227	dsql_cte_outer_join	Recursive member of CTE can't be member of an outer join Рекурсивная часть CTE не может являться членом outer join
-104	336397228	dsql_cte_mult_references	Recursive member of CTE can't reference itself more than once Рекурсивный член CTE не может ссылаться на себя более одного раза
-104	336397229	dsql_cte_not_a_union	Recursive CTE (@1) must be an UNION Рекурсивный CTE (<строка>) должен содержать UNION
-104	336397230	dsql_cte_nonrecurs_after_recurs	CTE '@1' defined non-recursive member after recursive В CTE '<строка>' определена нерекурсивная часть после рекурсии
-104	336397231	dsql_cte_wrong_clause	Recursive member of CTE '@1' has @2 clause Рекурсивная часть CTE '<строка>' содержит кляузу '<строка>'
-104	336397232	dsql_cte_union_all	Recursive members of CTE (@1) must be linked with another members via UNION ALL Рекурсивная часть CTE (<строка>) должна быть связана с остальными частями через UNION ALL
-104	336397233	dsql_cte_miss_nonrecursive	Non-recursive member is missing in CTE '@1' Нерекурсивная часть отсутствует в CTE '<строка>'
-104	336397234	dsql_cte_nested_with	WITH clause can't be nested Кляуза WITH не может быть вложенной
-104	336397235	dsql_col_more_than_once_using	Column @1 appears more than once in USING clause Столбец <строка> используется более одного раза в кляузе USING

Руководство по языку SQL СУБД Firebird

-104	336397237	dsql_cte_not_used	CTE "@1" is not used in query CTE "<строка>" не используется в запросе
-105	335544702	like_escape_invalid	Invalid ESCAPE sequence Неверная ESCAPE последовательность
-105	335544789	extract_input_mismatch	Specified EXTRACT part does not exist in input datatype Указанная часть-параметр для EXTRACT не существует для входного типа данных
-150	335544360	read_only_rel	Attempted update of read-only table Попытка обновления таблицы только для чтения
-150	335544362	read_only_view	Cannot update read-only view @1 Не могу обновить view <строка> в режиме read-only
-150	335544446	non_updatable	Not updatable Не обновляемое
-150	335544546	constraint_on_view	Cannot define constraints on views Не могу определять констрейны на view
-151	335544359	read_only_field	Attempted update of read-only column Попытка обновить доступный только для чтения столбец
-155	335544658	dsql_base_table	@1 is not a valid base table of the specified view <строка> не является верной базовой таблицей для указанного view
-157	335544598	specify_field_err	Must specify column name for view select expression Вы обязаны указать имя столбца для выражения выборки view
-158	335544599	num_field_err	Number of columns does not match select list Число столбцов не соответствует списку выборки
-162	335544685	no_dbkey	Dbkey not available for multi-table views Значение Dbkey не доступно для мультитабличных view

Руководство по языку SQL СУБД Firebird

-170	335544512	prcmismat	Input parameter mismatch for procedure @1 Входные параметры не соответствуют для процедуры <строка>
-170	335544619	extern_func_err	External functions cannot have more than 10 parametrs UDF не может иметь более чем 10 параметров

-170	335544850	prc_out_param_mismatch	Output parameter mismatch for procedure @1 Несоответствующие входные параметры для процедуры <строка>
-171	335544439	funmismat	Function @1 could not be matched Функция <строка> не может быть согласована
-171	335544458	invalid_dimension	Column not array or invalid dimensions (expected @1, encountered @2) Столбец не является массивом или неверная размерность (ожидается <строка>, встретилась <строка>)
-171	335544618	return_mode_err	Return mode by value not allowed for this data type Режим возврата по значению для допускается для этого типа данных
-171	335544873	array_max_dimensions	Array data type can use up to @1 dimensions Тип данных массив не может использовать свыше <число> размерностей
-172	335544438	funnotdef	Function @1 is not defined Функция <строка> не определена
-203	335544708	dyn fld_ambiguous	Ambiguous column reference. Неоднозначная ссылка на столбец.
-204	336003085	dsq_ambiguous_field_name	Ambiguous field name between @1 and @2 Неоднозначное имя поля между <строка> и <строка>
-204	335544463	gennotdef	Generator @1 is not defined Генератор <строка> не определен

Руководство по языку SQL СУБД Firebird

-204	335544502	stream_not_defined	Reference to invalid stream number Ссылка на неверный номер потока
-204	335544509	charset_not_found	CHARACTER SET @1 is not defined Набор символов <строка> не определен
-204	335544511	prcnotdef	Procedure @1 is not defined Процедура <строка> не определена
-204	335544515	codnotdef	Status code @1 unknown Код статуса <строка> неизвестен
-204	335544516	xcpnotdef	Exception @1 not defined Пользовательское исключение <строка> не определено
-204	335544532	ref_cnstrnt_notfound	Name of Referential Constraint not defined in constraints table. Имя реляционного констрейна не определено в таблице констрейнов
-204	335544551	grant_obj_notfound	Could not find table/procedure for GRANT Для операции GRANT не могу найти таблицу/процедуру
-204	335544568	text_subtype	Implementation of text subtype @1 not located. Реализация текстового подтипа <строка> не обнаружена.
-204	335544573	dsql_datatype_err	Data type unknown Неизвестный тип данных.
-204	335544580	dsql_relation_err	Table unknown Неизвестная таблица
-204	335544581	dsql_procedure_err	Procedure unknown Неизвестная процедура
-204	335544588	collation_not_found	COLLATION @1 for CHARACTER SET @2 is not defined Тип сортировки <строка> для набора символов <строка> не определен
-204	335544589	collation_not_for_charset	COLLATION @1 is not valid for specified CHARACTER SET Тип сортировки <строка> не верная для указанного набора символов

Руководство по языку SQL СУБД Firebird

-204	335544595	dsql_trigger_err	Trigger unknown Триггер неизвестен
-204	335544620	alias_conflict_err	Alias @1 conflicts with an alias in the same statement Алиас <строка> конфликтует с алиасом в том же выражении
-204	335544621	procedure_conflict_error	Alias @1 conflicts with a procedure in the same statement Алиас <строка> конфликтует в процедурой в том же выражении
-204	335544622	relation_conflict_err	Alias @1 conflicts with a table in the same statement Алиас <строка> конфликтует с таблицей в том же выражении
-204	335544635	dsql_no_relation_alias	There is no alias or table named @1 at this scope level Там нет алиаса или так именуется таблица на этом уровне области
-204	335544636	indexname	There is no index @1 for table @2 Там нет индекса <строка> для таблицы <строка>
-204	335544640	collation_requires_text	Invalid use of CHARACTER SET or COLLATE Неверное использование набора символов или типа сортировки
-204	335544662	dsql_blob_type_unknown	BLOB SUB_TYPE @1 is not defined Подтип BLOB <строка> не определен
-204	335544759	bad_default_value	Can not define a not null column with NULL as default value Не могу описать not null столбец при значении по умолчанию NULL
-204	335544760	invalid_clause	Invalid clause--- '@1' Неверная кляуза --- '<строка>'
-204	335544800	too_many_contexts	Too many Contexts of Relation/Procedure/Views. Maximum allowed is 255. В контексте слишком большое количество таблиц/процедур/view. Максимально допустимое количество 255.

Руководство по языку SQL СУБД Firebird

-204	335544817	bad_limit_param	Invalid parameter to FIRST. Only integers ≥ 0 are allowed. Неверный параметр для FIRST. Возможны только целые числа ≥ 0 .
-204	335544818	bad_skip_param	Invalid parameter to SKIP. Only integers ≥ 0 are allowed. Неверный параметр для SKIP. Возможны только целые числа ≥ 0 .
-204	335544837	bad_substring_offset	Invalid offset parameter @1 to SUBSTRING. Only positive integers are allowed. Неверный параметр смещения <строка> для SUBSTRING. Возможны только положительные целые числа.
-204	335544853	bad_substring_length	Invalid length parameter @1 to SUBSTRING. Negative integers are not allowed. Неверный параметр длины <строка> для SUBSTRING. Отрицательные целые числа не доступны.
-204	335544854	charset_not_installed	CHARACTER SET @1 is not installed Набор символов <строка> не установлен.
-204	335544855	collation_not_installed	COLLATION @1 for CHARACTER SET @2 is not installed Сортировка <строка> для набора символов <строка> не установлена.
-204	335544867	subtype_for_internal_use	Blob sub_types bigger than 1 (text) are for internal use only Подтип BLOB со значением более чем 1 (TEXT) предназначены только для внутреннего использования
-205	335544396	fldnotdef	Column @1 is not defined in table @2 Столбец <строка> не определен в таблице <строка>
-205	335544552	grant_fld_notfound	Could not find column for GRANT Не могу найти строку для операции GRANT
-205	335544883	fldnotdef2	Column @1 is not defined in procedure @2 Столбец <строка> не определен в процедуре <строка>

Руководство по языку SQL СУБД Firebird

-206	335544578	dsql_field_err	Column unknown Столбец неизвестен
-206	335544587	dsql_blob_err	Column is not a BLOB Столбец не является BLOB
-206	335544596	dsql_subselect_err	Subselect illegal in this context В данном контексте подселект запрещен
-206	336397208	dsql_line_col_error	At line @1, column @2 В строке <строка>, колонка <число>
-206	336397209	dsql_unknown_pos	At unknown line and column В неизвестных строке и столбце
-206	336397210	dsql_no_dup_name	Column @1 cannot be repeated in @2 statement Столбец <строка> не может быть повторен в выражении <строка>
-208	335544617	order_by_err	Invalid ORDER BY clause Неверная кляуза ORDER BY
-219	335544395	relnotdef	Table @1 is not defined Таблица <строка> не определена
-219	335544872	domnotdef	Domain @1 is not defined Домен <строка> не определен
-230	335544487	walw_err	WAL Writer error Ошибка записи WAL
-231	335544488	logh_small	Log file header of @1 too small Заголовок файла лога <строка> слишком мал
-232	335544489	logh_inv_version	Invalid version of log file @1 Неверная версия файла лога <строка>
-233	335544490	logh_open_flag	Log file @1 not latest in the chain but open flag still set Лог файл <строка> не последний в цепочке, но флаг «открыт» еще установлен
-234	335544491	logh_open_flag2	Log file @1 not closed properly; database recovery may be required Лог файл <строка> не верно закрыт; может понадобится операция

Руководство по языку SQL СУБД Firebird

			восстановления базы данных
-235	335544492	logh_diff_dbname	Database name in the log file @1 is different Имя базы данных в файле лога отличается
-236	335544493	logf_unexpected_eof	Unexpected end of log file @1 at offset @2 Неожиданное окончание файла лога <строка> по смещению <число>
-237	335544494	logr_incomplete	Incomplete log record at offset @1 in log file @2 Неполная запись лога по смещению <число> в файл лога <строка>
-238	335544495	logr_header_small	Log record header too small at offset @1 in log file @2 Заголовок записи лога слишком мал по смещению <число> в файле лога <строка>
-239	335544496	logb_small	Log block too small at offset @1 in log file @2 Блок лога слишком мал по смещению <число> в файле лога <строка>
-239	335544691	cache_too_small	Insufficient memory to allocate page buffer cache Недостаточно памяти для размещения страниц буфера кэш
-239	335544693	log_too_small	Log size too small Размер лога слишком мал
-239	335544694	partition_too_small	Log partition size too small Размер раздела лога слишком мал
-243	335544500	no_wal	Database does not use Write-ahead Log База данных не использует запись с упреждением лога
-257	335544566	start_cm_for_wal	WAL defined; Cache Manager must be started first Обнаружено WAL; Менеджер КЭШа должен стартовать первым
-260	335544690	cache_redef	Cache redefined Кэш переопределен

Руководство по языку SQL СУБД Firebird

-260	335544692	log_redef	Log redefined Лог переопределен
-261	335544695	partition_not_supp	Partitions not supported in series of log file specification Разделы не поддерживаются в серии спецификации файла журнала
-261	335544696	log_length_spec	Total length of a partitioned log must be specified Общая длина раздельного лога должна быть определена
-281	335544637	no_stream_plan	Table @1 is not referenced in plan Таблица <строка> не упоминается в плане
-282	335544638	stream_twice	Table @1 is referenced more than once in plan; use aliases to distinguish Таблица <строка> упомянута более раза в плане; используйте алиасы чтобы различить
-282	335544643	dsql_self_join	The table @1 is referenced twice; use aliases to differentiate Таблица <строка> упомянута дважды; используйте алиасы чтобы различить
-282	335544659	duplicate_base_table	Table @1 is referenced twice in view; use an alias to distinguish to differentiate Таблица <строка> упомянута дважды во view; используйте алиасы чтобы различить
-282	335544660	view_alias	View @1 has more than one base table; use aliases to distinguish View <строка> использует более одного разабазовую таблицу; используйте алиасы чтобы различить
-282	335544710	complex_view	Navigational stream @1 references a view with more than one base table Навигационный поток <строка> ссылается на view с более чем одной базовой таблицей
-283	335544639	stream_not_found	Table @1 is referenced in the plan but not the from list В плане ссылаются на таблицу <строка>, но она не в списке

Руководство по языку SQL СУБД Firebird

-284	335544642	index_unused	Index @1 cannot be used in the specified plan Индекс <строка> не может быть использован в указанном плане
-291	335544531	primary_key_notnull	Column used in a PRIMARY constraint must be NOT NULL. Столбец, использованный в первичном ключе должен быть NOT NULL.
-292	335544534	ref_cnstrnt_update	Cannot update constraints (RDB\$REF_CONSTRAINTS). Не могу обновить констренты (RDB\$REF_CONSTRAINTS).
-293	335544535	check_cnstrnt_update	Cannot update constraints (RDB\$CHECK_CONSTRAINTS). Не могу обновить констренты (RDB\$CHECK_CONSTRAINTS).
-294	335544536	check_cnstrnt_del	Cannot delete CHECK constraint entry (RDB\$CHECK_CONSTRAINTS) Не удастся удалить констрент CHECK (RDB\$CHECK_CONSTRAINTS)
-295	335544545	rel_cnstrnt_update	Cannot update constraints (RDB\$RELATION_CONSTRAINTS). Не могу обновить констренты (RDB\$RELATION_CONSTRAINTS).
-296	335544547	invld_cnstrnt_type	Internal gds software consistency check (invalid RDB\$CONSTRAINT_TYPE) Внутренняя ошибка программного обеспечения на согласованность (неверная таблица RDB\$CONSTRAINT_TYPE)
-297	335544558	check_constraint	Operation violates check constraint @1 on view or table @2 Операция нарушает check констрент <строка> на view или таблицу <строка>
-313	336003099	upd_ins_doesnt_match_pk	UPDATE OR INSERT field list does not match primary key of table @1 В UPDATE OR INSERT список полей не соответствует первичному ключу таблицы <строка>
-313	336003100	upd_ins_doesnt_match_matching	UPDATE OR INSERT field list does not match MATCHING clause В UPDATE OR INSERT список полей не соответствует кляузе MATCHING

Руководство по языку SQL СУБД Firebird

-313	335544669	dsql_count_mismatch	Count of column list and variable list do not match Количество столбцов и переменных в списке не соответствует
-314	335544565	transliteration_failed	Cannot transliterate character between character sets Не могу подвергнуть транслитерации символ между наборами символов
-315	336068815	dyn_dtype_invalid	Cannot change datatype for column @1. Changing datatype is not supported for BLOB or ARRAY columns. Не могу изменить тип данных для столбца <строка>. Изменение типа данных не поддерживается для BLOB полей и полей с массивами.
-383	336068814	dyn_dependency_exists	Column @1 from table @2 is referenced in @3 Столбец <строка> из таблицы <строка> упоминается в <строка>
-401	335544647	invalid_operator	Invalid comparison operator for find operation Неверный оператор сравнения для операции поиска
-402	335544368	segstr_no_op	Attempted invalid operation on a BLOB Попытка неверной операции с BLOB
-402	335544414	blobnotsup	BLOB and array data types are not supported for @1 operation Типы данных BLOB и массив не поддерживаются для операции <строка>
-402	335544427	datnotsup	Data operation not supported Операция данных не поддерживается
-406	335544457	out_of_bounds	Subscript out of bounds Индекс вне границ
-407	335544435	nullsegkey	Null segment of UNIQUE KEY Null сегмент для уникального ключа
-413	335544334	convert_error	Conversion error from string "@1" Ошибка конвертации для строки <строка>
-413	335544454	nofilter	Filter not found to convert type @1 to type @2

Руководство по языку SQL СУБД Firebird

			Фильтр не обнаружен для конвертации типа <строка> в тип <строка>
-413	335544860	blob_convert_error	Unsupported conversion to target type BLOB (subtype @1) Не поддерживается преобразование в целевой тип BLOB (подтип <строка>)
-413	335544861	array_convert_error	Unsupported conversion to target type ARRAY Не поддерживается преобразование в целевой тип массив
-501	335544577	dsql_cursor_close_err	Attempt to reclose a closed cursor Попытка перезакрыть закрытый курсор
-502	336003090	dsql_cursor_redefined	Statement already has a cursor @1 assigned Для предложения уже имеется назначенный <строка> курсор
-502	336003091	dsql_cursor_not_found	Cursor @1 is not found in the current context Курсор <строка> не обнаружен для текущего контекста
-502	336003092	dsql_cursor_exists	Cursor @1 already exists in the current context Курсор <строка> уже существует для указанного контекста
-502	336003093	dsql_cursor_rel_ambiguous	Relation @1 is ambiguous in cursor @2 Соотношение <строка> неоднозначно в курсоре <строка>
-502	336003094	dsql_cursor_rel_not_found	Relation @1 is not found in cursor @2 Соотношение <строка> не найдено в курсоре <строка>
-502	336003095	dsql_cursor_not_open	Cursor is not open Курсор не открыт
-502	335544574	dsql_decl_err	Invalid cursor declaration Неверная декларация курсора
-502	335544576	dsql_cursor_open_err	Attempt to reopen an open cursor Попытка переоткрыть открытый курсор
-504	336003089	dsql_cursor_invalid	Empty cursor name is not allowed

Руководство по языку SQL СУБД Firebird

			Для курсора не доступно пустое имя
-504	335544572	dsql_cursor_err	Invalid cursor reference Недопустимая ссылка на курсор
-508	335544348	no_cur_rec	No current record for fetch operation Нет текущей записи для операции FETCH
-510	335544575	dsql_cursor_update_err	Cursor @1 is not updatable Курсор <строка> является необновляемым
-518	335544582	dsql_request_err	Request unknown Запрос неизвестен
-519	335544688	dsql_open_cursor_request	The prepare statement identifies a prepare statement with an open cursor Подготовка выражения определила подготовку выражения с открытием курсора
-530	335544466	foreign_key	Violation of FOREIGN KEY constraint "@1" on table "@2" Нарушение ограничения внешнего ключа <строка> для таблицы <строка>
-530	335544838	foreign_key_target_doesnt_exist	Foreign key reference target does not exist Ссылка на целевое значение внешнего ключа не существует.
-530	335544839	foreign_key_references_present	Foreign key references are present for the record Ссылки внешнего ключа присутствуют для записи
-531	335544597	dsql_crdb_prepare_err	Cannot prepare a CREATE DATABASE/SCHEMA statement Не могу подготовить к выполнению оператор CREATE DATABASE/SCHEMA
-532	335544469	trans_invalid	Transaction marked invalid by I/O error Транзакция помечена как недействительная из за ошибки ввода - вывода
-551	335544352	no_priv	No permission for @1 access to @2 @3 Нет разрешения <строка> для доступа

Руководство по языку SQL СУБД Firebird

			к <строка> <строка>
-551	335544790	insufficient_svc_privileges	Service @1 requires SYSDBA permissions.Reattach to the Service Manager using the SYSDBA account. Сервис <строка> требует привелегии SYSDBA. Переприсоединитесь к менеджеру сервисов используя учетную запись SYSDBA.
-552	335544550	not_rel_owner	Only the owner of a table may reassign ownership Только владелец таблицы может переназначить права владения.
-552	335544553	grant_nopriv	User does not have GRANT privileges for operation Пользователь не имеет привелегий для операции GRANT
-552	335544707	grant_nopriv_on_base	User does not have GRANT privileges on base table/view for operation Пользователь не имел в базе прав GRANT для таблицы / view для операции
-553	335544529	Existing_priv_mod	Cannot modify an existing user privilege Не могу изменить существующие пользовательские привелегии
-595	335544645	stream_crack	The current position is on a crack Текущая позиция «в трещине»
-596	335544644	stream_bof	Illegal operation when at beginning of stream Неверная операция при начале потока
-597	335544632	dsql_file_length_err	Preceding file did not specify length, so @1 must include starting page number Предыдущий файл не указал длины, так что <строка> должен включать число стартовых страниц
-598	335544633	dsql_shadow_number_err	Shadow number must be a positive integer Номер тени должен быть целым положительным числом
-599	335544607	node_err	Gen.c: node not supported Gen.c.: ноды (узлы) не

Руководство по языку SQL СУБД Firebird

			поддерживаются
-599	335544625	node_name_err	A node name is not permitted in a secondary, shadow, cache or log file name Имя узла не допускается в именах вторичного, теневого, КЭШа или в имени файла лога.
-600	335544680	crrp_data_err	Sort error: corruption in data structure Ошибка сортировки: разрушения в структуре данных
-601	335544646	db_or_file_exists	Database or file exists База данных или файл не существует
-604	335544593	dsql_max_arr_dim_exceeded	Array declared with too many dimensions Массив задекларирован со слишком многими размерностями
-604	335544594	dsql_arr_range_error	Illegal array dimension range Неверный диапазон размерности массива
-605	335544682	dsql_field_ref	Inappropriate self-reference of column Жалоба на ссылку «сам на себя» столбца
-607	336003074	dsql_dbkey_from_non_table	Cannot SELECT RDB\$DB_KEY from a stored procedure. Невозможен SELECT RDB\$DB_KEY из хранимой процедуры
-607	336003086	dsql_udf_return_pos_err	External function should have return position between 1 and @1 Внешняя функция должна иметь позицию возврата между 1 и <число>
-607	336003096	dsql_type_not_supp_ext_tab	Data type @1 is not supported for EXTERNAL TABLES. Relation '@2', field '@3' Тип данных <строка> не поддерживается для внешних таблиц. Таблица <строка>, поле <строка>
-607	335544351	no_meta_update	Unsuccessful metadata update Неудачное обновление метаданных
-607	335544549	systrig_update	Cannot modify or erase a system trigger Не возможно изменить или стереть системный триггер

Руководство по языку SQL СУБД Firebird

-607	335544657	dsql_no_blob_array	Array/BLOB/DATE data types not allowed in arithmetic Типы данных Массив/BLOB/даты не возможны для арифметических операций
-607	335544746	Reftable_requires_pk	"REFERENCES table" without "(column)" requires PRIMARY KEY on referenced table "Таблицы ссылок" без «(столбца)» требуют первичного ключа связанной таблице
-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 Невозможна таблица состоящая только из одних вычислительных полей и констрейнов.
-607	336397206	dsql_table_not_found	Table @1 does not exist Таблица <строка> не существует
-607	336397207	dsql_view_not_found	View @1 does not exist Представление (view) <строка> не существует
-607	336397212	dsql_no_array_computed	Array and BLOB data types not allowed in computed field Типы данных массив и BLOB не подходят для вычисляемых полей
-607	336397214	dsql_only_can_subscript_array	Scalar operator used on field @1 which is not an array Скалярный оператор используется по полю <строка>, которое не является массивом
-612	336068812	dyn_domain_name_exists	Cannot rename domain @1 to @2. A domain with that name already exists. Не могу переименовать домен из <строка> в <строка>. Домен с таким именем уже существует.
-612	336068813	dyn_field_name_exists	Cannot rename column @1 to @2. A column with that name already exists in table @3. Не могу переименовать столбец <строка> в <строка>. Столбец с таким именем уже существует в таблице

Руководство по языку SQL СУБД Firebird

			<строка>.
-615	335544475	relation_lock	Lock on table @1 conflicts with existing lock Блокировка в таблице <строка> конфликтует с существующей блокировкой
-615	335544476	record_lock	Requested record lock conflicts with existing lock Требуемая блокировка записи конфликтует с существующей блокировкой
-615	335544507	range_in_use	Refresh range number @1 already in use Обновленный диапазон имен <строка> уже используется
-616	335544530	primary_key_ref	Cannot delete PRIMARY KEY being used in FOREIGN KEY definition. Не могу удалить первичный ключ используемый в определении внешнего ключа.
-616	335544539	integ_index_del	Cannot delete index used by an Integrity Constraint Не могу удалить индекс используемый в констрейне.
-616	335544540	integ_index_mod	Cannot modify index used by an Integrity Constraint Не могу изменить индекс используемый в констрейне.
-616	335544541	check_trig_del	Cannot delete trigger used by a CHECK Constraint Не могу удалить триггер используемый в констрейне типа CHECK
-616	335544543	cnstrnt fld_del	Cannot delete column being used in an Integrity Constraint. Не могу удалить столбец используемый в констрейне целостности
-616	335544630	dependency	There are @1 dependencies Есть <число> зависимостей
-616	335544674	del_last_field	Last column in a table cannot be deleted Последний столбец таблицы не может быть удален

Руководство по языку SQL СУБД Firebird

-616	335544728	integ_index_deactivate	Cannot deactivate index used by an integrity constraint Не могу деактивировать индекс используемый в констрейне целостности
-616	335544729	integ_deactivate_primary	Cannot deactivate index used by a PRIMARY/UNIQUE constraint Не могу деактивировать индекс используемый в констрейне первичного ключа/ уникальности
-617	335544542	check_trig_update	Cannot update trigger used by a CHECK Constraint Не могу обновить триггер используемый в констрейне типа CHECK
-617	335544544	cnstrnt fld_rename	Cannot rename column being used in an Integrity Constraint. Не могу переименовать столбец, используемый в констрейне целостности
-618	335544537	integ_index_seg_del	Cannot delete index segment used by an Integrity Constraint Не могу удалить сегмент индекса, используемый в констрейне целостности данных
-618	335544538	integ_index_seg_mod	Cannot update index segment used by an Integrity Constraint Не могу обновить сегмент индекса, используемый в констрейне целостности данных
-625	335544347	not_valid	Validation error for column @1, value "@2" Ошибка проверки данных для столбца <строка>, значение «<строка>»
-625	335544879	not_valid_for_var	Validation error for variable @1, value "@2" Ошибка проверки данных для переменной <строка>, значение «<строка>»
-625	335544880	not_valid_for	Validation error for @1, value "@2" Ошибка проверки данных для <строка>, значение «<строка>»
-637	335544664	dsql_duplicate_spec	Duplicate specification of @1- not supported

Руководство по языку SQL СУБД Firebird

			Дубликат спецификации для <строка> - не поддерживается.
-637	336397213	dsql_implicit_domain_name	Implicit domain name @1 not allowed in user created domain Неявное доменное имя <строка> не допускается для доменов, создаваемых пользователями
-660	336003098	primary_key_required	Primary key required on table @1 Для таблицы <строка> требуется первичный ключ
-660	335544533	foreign_key_notfound	Non-existent PRIMARY or UNIQUE KEY specified for FOREIGN KEY. Не существующий первичный или ключ уникальности указан для внешнего ключа
-660	335544628	idx_create_err	Cannot create index @1 Не могу создать индекс <строка>
-663	335544624	idx_seg_err	Segment count of 0 defined for index @1 Количество сегментов равное 0 определено для индекса <строка>
-663	335544631	idx_key_err	Too many keys defined for index @1 Слишком много ключей определено для индекса <строка>
-663	335544672	key_field_err	Too few key columns found for index @1 (incorrect column name?) Слишком много столбцов ключей обнаружено для индекса <строка> (неверные имена столбцов?)
-664	335544434	keytoobig	Key size exceeds implementation restriction for index "@1" Размер ключа превысил ограничения реализации для индекса <строка>
-677	335544445	ext_err	@1 extension error <строка> ошибка расширения
-685	335544465	bad_segstr_type	Invalid BLOB type for operation Неверный тип BLOB –а для операции
-685	335544670	blob_idx_err	Attempt to index BLOB column in index @1 Попытка индексации BLOB столбца в

Руководство по языку SQL СУБД Firebird

			индексе <строка>
-685	335544671	array_idx_err	Attempt to index array column in index @1 Попытка индексации столбца с типом массив в индексе <строка>
-689	335544403	badpagtyp	Page @1 is of wrong type (expected @2, found @3) Страница <число> имеет неверный тип (ожидается <число>, обнаружена <число>)
-689	335544650	page_type_err	Wrong page type Неверный тип страницы
-690	335544679	no_segments_err	Segments not allowed in expression index @1 Сегменты не допускаются в выражении индекса <строка>
-691	335544681	rec_size_err	New record size of @1 bytes is too big Новый размер записи в <число> байт является слишком большим
-692	335544477	max_idx	Maximum indexes per table (@1) exceeded Максимум количества индексов на одну таблицу (<строка>) превышен
-693	335544663	req_max_clones_exceeded	Too many concurrent executions of the same request Слишком много конкурентных выполнений одного и того же запроса
-694	335544684	no_field_access	Cannot access column @1 in view @2 Не могу получить доступ к столбцу <строка> представления/view <строка>
-802	335544321	arith_except	Arithmetic exception, numeric overflow, or string truncation Арифметическое исключение, числовое переполнение или строковое обрезание
-802	335544836	concat_overflow	Concatenation overflow. Resulting string cannot exceed 32K in length. Переполнение при конкатенации. Результирующая строка не может превышать 32 Кб.
-803	335544349	no_dup	Attempt to store duplicate value (visible to active transactions) in unique index

Руководство по языку SQL СУБД Firebird

			"@1" Попытка сохранить дубликат значения (видимые при активных транзакциях) в уникальном индексе "<строка>"
-803	335544665	unique_key_violation	Violation of PRIMARY or UNIQUE KEY constraint "@1" on table "@2" Нарушение ограничения первичного или уникального ключа "<строка>" в таблице "<строка>"
-804	336003097	dsql_feature_not_supported_ods	Feature not supported on ODS version older than @1.@2 Опция не поддерживается в ODS старше чем <число>.<число>
-804	335544380	wronumarg	Wrong number of arguments on call Неверное число аргументов в вызове
-804	335544583	dsql_sqlda_err	SQLDA missing or incorrect version, or incorrect number/type of variables SQLDA отсутствует или неверной версии, либо некорректный номер/ тип переменной
-804	335544584	dsql_var_count_err	Count of read-write columns does not equal count of values Количество столбцов для чтения/записи не равно количеству значений
-804	335544586	dsql_function_err	Function unknown Функция неизвестна
-804	335544713	dsql_sqlda_value_err	Incorrect values within SQLDA structure Некорректные значения в SQLDA структуре
-804	336397205	dsql_too_old_ods	ODS versions before ODS@1 are not supported ODS версий до версии ODS<строка> не поддерживаются
-806	335544600	col_name_err	Only simple column names permitted for VIEW WITH CHECK OPTION Только простые имена столбцов допустимы для опции VIEW WITH CHECK
-807	335544601	where_err	No WHERE clause for VIEW WITH

Руководство по языку SQL СУБД Firebird

			CHECK OPTION Нет кляузы WHERE для опции VIEW WITH CHECK
-808	335544602	table_view_err	Only one table allowed for VIEW WITH CHECK OPTION Только одна таблица позволена для опции VIEW WITH CHECK
-809	335544603	distinct_err	DISTINCT, GROUP or HAVING not permitted for VIEW WITH CHECK OPTION Для опции VIEW WITH CHECK OPTION не допустимы DISTINCT, GROUP или HAVING
-810	335544605	subquery_err	No subqueries permitted for VIEW WITH CHECK OPTION Нет подзапросов разрешенных для for VIEW WITH CHECK OPTION
-811	335544652	sing_select_err	Multiple rows in singleton select Несколько строк для единичной выпорки
-816	335544651	ext_readonly_err	Cannot insert because the file is readonly or is on a read only medium. Не могу вставить по причине файла только для чтения или среды носителя только для чтения
-816	335544715	extfile_uns_op	Operation not supported for EXTERNAL FILE table @1 Операция не поддерживается для внешней таблицы <строка>
-817	336003079	isc_sql_dialect_conflict_num	DB dialect @1 and client dialect @2 conflict with respect to numeric precision @3. Диалект БД <число> и диалект клиентской программы <число> конфликтует с соблюдением числовой точности <строка>
-817	336003101	upd_ins_with_complex_view	UPDATE OR INSERT without MATCHING could not be used with views based on more than one table UPDATE OR INSERT без MATCHING не могут быть использованы с View базирующимися на более чем одной таблице

Руководство по языку SQL СУБД Firebird

-817	336003102	dsql_incompatible_trigger_type	Incompatible trigger type Несовместимый тип триггера
-817	336003103	dsql_db_trigger_type_cant_change	Database trigger type can't be changed Триггер типа «база данных» не может быть изменен
-817	335544361	read_only_trans	Attempted update during read-only transaction Попытка выполнить изменения во время выполнения транзакции только для чтения
-817	335544371	segstr_no_write	Attempted write to read-only BLOB Попытка записи в BLOB только для чтения
-817	335544444	read_only	Operation not supported Операция не поддерживается
-817	335544765	read_only_database	Attempted update on read-only database Попытка записи в базу данных находящуюся в режиме только для чтения
-817	335544766	must_be_dialect_2_and_up	SQL dialect @1 is not supported in this database SQL диалект <число> не поддерживается в этой базе данных
-817	335544793	ddl_not_allowed_by_db_sql_dial	Metadata update statement is not allowed by the current database SQL dialect @1 Обновление метаданных этой строкой не поддерживается из-за текущего диалекта базы данных <число>
-820	335544356	obsolete_metadata	Metadata is obsolete Метаданных являются устаревшими
-820	335544379	wrong_ods	Unsupported on-disk structure for file @1; found @2.@3, support @4.@5 Неподдерживаемая ODS для файла <строка>, обнаружена <число>.<число >, поддерживается <число>.<число >
-820	335544437	wrodysver	Wrong DYN version Неверная версия DYN
-820	335544467	high_minor	Minor version too high found @1 expected @2 Минорная версия слишком высокая,

Руководство по языку SQL СУБД Firebird

			обнаружена <строка>, ожидалась <строка>
-820	335544881	need_difference	Difference file name should be set explicitly for database on raw device Файл разницы должен быть явно задан для базы данных на «сыром устройстве»
-823	335544473	invalid_bookmark	Invalid bookmark handle Неверный дескриптор закладки
-824	335544474	bad_lock_level	Invalid lock level @1 Неверный уровень блокировки <строка>
-825	335544519	bad_lock_handle	Invalid lock handle Неверный дескриптор блокировки
-826	335544585	dsql_stmt_handle	Invalid statement handle Неверный дескриптор выражения
-827	335544655	invalid_direction	Invalid direction for find operation Неверное выражение для операции «поиск»
-827	335544718	invalid_key	Invalid key for find operation Неверный ключ для операции поиска
-828	335544678	inval_key_posn	Invalid key position Неверный положение ключа
-829	336068816	dyn_char fld_too_small	New size specified for column @1 must be at least @2 characters. Новый размер указанный для столбца должен быть по крайней мере <число> символов
-829	336068817	dyn_invalid_dtype_conversion	Cannot change datatype for @1. Conversion from base type @2 to @3 is not supported. Не могу изменить тип данных для <строка>. Преобразование из базового типа <строка> в <строка> не поддерживается.
-829	336068818	dyn_dtype_conv_invalid	Cannot change datatype for column @1 from a character type to a non-character type. Не могу поменять тип данных для столбца <строка> из символьного типа

Руководство по языку SQL СУБД Firebird

			в несимвольный.
-829	336068829	max_coll_per_charset	Maximum number of collations per character set exceeded В наборе символов превышено максимальное число наборов сортировок
-829	336068830	invalid_coll_attr	Invalid collation attributes Неверный атрибуты параметров сортировки
-829	336068852	dyn_scale_too_big	New scale specified for column @1 must be at most @2. Новый масштаб, указанный для столбца <строка> должен быть не более <число>
-829	336068853	dyn_precision_too_small	New precision specified for column @1 must be at least @2. Новая точность указанная для столбца <строка> должна быть по крайней мере <число>
-829	335544616	field_ref_err	Invalid column reference Неверная ссылка столбца
-830	335544615	field_aggregate_err	Column used with aggregate Столбец используется в агрегатах
-831	335544548	primary_key_exists	Attempt to define a second PRIMARY KEY for the same table Попытка определить второй первичный ключ для таблицы
-832	335544604	key_field_count_err	FOREIGN KEY column count does not match PRIMARY KEY Количество столбцов внешнего ключа не совпадает с первичным ключем
-833	335544606	expression_eval_err	Expression evaluation not supported Вычисляемые выражения не поддерживаются
-833	335544810	date_range_exceeded	Value exceeds the range for valid dates Значение превышает пределы установленные для действительных дат
-834	335544508	range_not_found	Refresh range number @1 not found Обновленный диапазон номеров

Руководство по языку SQL СУБД Firebird

			<строка> не найден
-835	335544649	bad_checksum	Bad checksum Неверная контрольная сумма
-836	335544517	except	Exception @1 Исключение <строка>
-836	335544848	except2	Exception @1 Исключение <строка>
-837	335544518	cache_restart	Restart shared cache manager Рестарт общего менеджера КЭШа
-838	335544560	shutwarn	Database @1 shutdown in @2 seconds База данных <строка> уйдет в состояние шатдаун через <число> секунд
-841	335544677	version_err	Too many versions Слишком много версий
-842	335544697	precision_err	Precision must be from 1 to 18 Точность должна быть в пределах от 1 до 18
-842	335544698	scale_nogt	Scale must be between zero and precision Масштаб должен быть между нулем и значением точности
-842	335544699	expec_short	Short integer expected Ожидается короткое целое
-842	335544700	expec_long	Long integer expected Ожидается длинное целое
-842	335544701	expec_ushort	Unsigned short integer expected Ожидается беззнаковое короткое целое
-842	335544712	expec_positive	Positive value expected Ожидается положительное значение
-901	335740929	gfix_db_name	Data base file name (@1) already given Имя файла базы данных (<строка>) уже дали
-901	336330753	gbak_unknown_switch	Found unknown switch Обнаружена неизвестная опция

Руководство по языку SQL СУБД Firebird

			командной строки
-901	336920577	gstat_unknown_switch	Found unknown switch Обнаружена неизвестная опция командной строки
-901	336986113	fbsvcmgr_bad_am	Wrong value for access mode Неверное значение для режима доступа
-901	335740930	gfix_invalid_sw	Invalid switch @1 Неверный параметр командной строки <строка>
-901	335544322	bad_dbkey	Invalid database key Неверный ключ базы данных
-901	336986114	fbsvcmgr_bad_wm	Wrong value for write mode Неверное значение для режима записи
-901	336330754	gbak_page_size_missing	Page size parameter missing Параметра размера страницы отсутствует
-901	336920578	gstat_retry	Please retry, giving a database name Пожалуйста повторите, давая имя базы данных
-901	336986115	fbsvcmgr_bad_rs	Wrong value for reserve space Неверное значение для зарезервированного пространства
-901	336920579	gstat_wrong_ods	Wrong ODS version, expected @1, encountered @2 Неверная версия ODS, ожидается <строка>, встретилась <строка>
-901	336330755	gbak_page_size_toobig	Page size specified (@1) greater than limit (16384 bytes) Указанный размер страницы (<число>) больше ограничения (16384 bytes)
-901	335740932	gfix_incmp_sw	Incompatible switch combination Несовместимая комбинация ключей командной строки
-901	336920580	gstat_unexpected_eof	Unexpected end of database file. Неожиданный конец файла базы данных.

Руководство по языку SQL СУБД Firebird

-901	336330756	gbak_redir_output_missing	Redirect location for output is not specified Перенаправление места для вывода не указано
-901	336986116	fbsvcmgr_info_err	Unknown tag (@1) in info_svr_db_info block after isc_svc_query() Неизвестный тэг (<строка>) в блоке info_svr_db_info после вызова isc_svc_query()
-901	335740933	gfix_replay_req	Replay log pathname required Воспроизведение лога требует пути до него
-901	336330757	gbak_switches_conflict	Conflicting switches for backup/restore Конфликтующие опции командной строки для бекапа/рестора
-901	336986117	fbsvcmgr_query_err	Unknown tag (@1) in isc_svc_query() results Неизвестный тег в результатах isc_svc_query()
-901	335544326	bad_dpb_form	Unrecognized database parameter block Блок параметров базы данных не распознан
-901	335740934	gfix_pgbuf_req	Number of page buffers for cache required Требуется указать количество страниц для буфера кэша
-901	336986118	fbsvcmgr_switch_unknown	Unknown switch "@1" Неизвестный параметр командной строки «<строка>»
-901	336330758	gbak_unknown_device	Device type @1 not known Тип устройства <строка> не известен
-901	335544327	bad_req_handle	Invalid request handle Неверный указатель запроса
-901	335740935	gfix_val_req	Numeric value required Требуется числовое значение
-901	336330759	gbak_no_protection	Protection is not there yet Опция «защита» еще не есть
-901	335544328	bad_segstr_handle	Invalid BLOB handle Неверный указатель на BLOB

Руководство по языку SQL СУБД Firebird

-901	335740936	gfix_pval_req	Positive numeric value required Требуется положительное числовое значение
-901	336330760	gbak_page_size_not_allowed	Page size is allowed only on restore or create Параметр «размер страницы» доступен только при восстановлении или создании
-901	335544329	bad_segstr_id	Invalid BLOB ID Неверный ID BLOB
-901	335740937	gfix_trn_req	Number of transactions per sweep required Требуется количество транзакция для операции sweep
-901	336330761	gbak_multi_source_dest	Multiple sources or destinations specified Несколько источников или направлений указаны
-901	335544330	bad_tpb_content	Invalid parameter in transaction parameter block Неверный параметр в блоке параметров транзакции
-901	336330762	gbak_filename_missing	Requires both input and output filenames Требуется оба имени файла лоя ввода и для вывода
-901	335544331	bad_tpb_form	Invalid format for transaction parameter block Неверный формат для блока параметров для транзакции
-901	336330763	gbak_dup_inout_names	Input and output have the same name. Disallowed. Ввод и вывод имеют одинаковое имя. Неразрешается.
-901	335740940	gfix_full_req	"full" or "reserve" required Параметры "full" или "reserve" требуются
-901	335544332	bad_trans_handle	Invalid transaction handle (expecting explicit transaction start) Неверная ссылка на транзакцию (ожидаю явного старта транзакций)

Руководство по языку SQL СУБД Firebird

-901	336330764	gbak_inv_page_size	Expected page size, encountered "@1" Ожидалось размер страницы, встретилост «<строка>»
-901	335740941	gfix_username_req	User name required Требуется имя пользователя
-901	336330765	gbak_db_specified	REPLACE specified, but the first file @1 is a database Указан параметр ЗАМЕНА, но первый файл <строка> является базой данных
-901	335740942	gfix_pass_req	Password required Требуется пароль
-901	336330766	gbak_db_exists	Database @1 already exists.To replace it, use the-REP switch База данных <строка> уже существует. Чтобы заменить ее, используйте переключатель командной строки -REP
-901	335740943	gfix_subs_name	Subsystem name Имя подсистемы
-901	336723983	gsec_cant_open_db	Unable to open database Не могу открыть базу данных
-901	336330767	gbak_unk_device	Device type not specified Тип устройства не указан
-901	336723984	gsec_switches_error	Error in switch specifications Ошибка в указании опций командной строки
-901	335740945	gfix_sec_req	Number of seconds required Число секунд требуется
-901	335544337	excess_trans	Attempt to start more than @1 transactions Попытка начать более чем <строка> транзакций
-901	336723985	gsec_no_op_spec	No operation specified Операция не определена
-901	335740946	gfix_nval_req	Numeric value between 0 and 32767 inclusive required Цифровое значение между 0 и 32767 включительно требуется

Руководство по языку SQL СУБД Firebird

-901	336723986	gsec_no_usr_name	No user name specified Не определено имя пользователя
-901	335740947	gfix_type_shut	Must specify type of shutdown Требуется определить тип шатдауна
-901	335544339	infinap	Information type inappropriate for object specified Тип данных подходит для указанного объекта
-901	336723987	gsec_err_add	Add record error Ошибка добавления записи
-901	335544340	infona	No information of this type available for object specified Нет информации этого типа доступной для указанного объекта
-901	336723988	gsec_err_modify	Modify record error Ошибка изменения записи
-901	336330772	gbak_blob_info_failed	Gds_\$blob_info failed Операция Gds_\$blob_info не удалась
-901	335740948	gfix_retry	Please retry, specifying an option Пожалуйста повторите, уточните опцию
-901	335544341	infunk	Unknown information item Незнакомый элемент информации
-901	336723989	gsec_err_find_mod	Find/modify record error Ошибка поиска/изменения записи
-901	336330773	gbak_unk_blob_item	Do not understand BLOB INFO item @1 Не понятный элемент BLOB INFO <строка>
-901	335544342	integ_fail	Action cancelled by trigger (@1) to preserve data integrity Действие отменено триггером (<строка>) чтобы сохранить целостность данных
-901	336330774	gbak_get_seg_failed	Gds_\$get_segment failed Операция Gds_\$get_segment не удалась
-901	336723990	gsec_err_rec_not_found	Record not found for user: @1 Запись не найдена для пользователя:

Руководство по языку SQL СУБД Firebird

			<строка>
-901	336723991	gsec_err_delete	Delete record error Ошибка удаления записи
-901	336330775	gbak_close_blob_failed	Gds_\$close_blob failed Операция Gds_\$close_blob не удалась
-901	335740951	gfix_retry_db	Please retry, giving a database name Пожалуйста повторите, давая имя базы данных
-901	336330776	gbak_open_blob_failed	Gds_\$open_blob failed Операция Gds_\$open_blob не удалась
-901	336723992	gsec_err_find_del	Find/delete record error Ошибка поиска/удаления записи
-901	335544345	lock_conflict	Lock conflict on no wait transaction Ошибка блокировки при не ждущей транзакции
-901	336330777	gbak_put_blr_gen_id_failed	Failed in put_blr_gen_id Операция put_blr_gen_id не удалась
-901	336330778	gbak_unk_type	Data type @1 not understood Тип данных <строка> не понят
-901	336330779	gbak_comp_req_failed	Gds_\$compile_request failed Операция Gds_\$compile_request не удалась
-901	336330780	gbak_start_req_failed	Gds_\$start_request failed Операция Gds_\$start_request не удалась
-901	336723996	gsec_err_find_disp	Find/display record error Ошибка в Найти/показать запись
-901	336330781	gbak_rec_failed	gds_\$receive failed Операция gds_\$receive не удалась
-901	336920605	gstat_open_err	Can't open database file @1 Не могу открыть файл базы данных <строка>
-901	336723997	gsec_inv_param	Invalid parameter, no switch defined Неверный параметр, нет ключа командной строки

Руководство по языку SQL СУБД Firebird

-901	335544350	no_finish	Program attempted to exit without finishing database Программа попыталась завершиться без отсоединения от базы данных
-901	336920606	gstat_read_err	Can't read a database page Не могу проситать страницу базы данных
-901	336330782	gbak_rel_req_failed	Gds_\$release_request failed Операция Gds_\$release_request не удалась
-901	336723998	gsec_op_specified	Operation already specified Операция уже определена
-901	336920607	gstat_sysmemex	System memory exhausted Системная память исчерпана
-901	336330783	gbak_db_info_failed	gds_\$database_info failed Операция gds_\$database_info не удалась
-901	336723999	gsec_pw_specified	Password already specified Пароль уже определен
-901	336724000	gsec_uid_specified	Uid already specified UID уже определен
-901	336330784	gbak_no_db_desc	Expected database description record Ожидалась описание записи базы данных
-901	335544353	no_recon	Transaction is not in limbo Транзакция не является limbo (не находится в подвешенном состоянии)
-901	336724001	gsec_gid_specified	Gid already specified GID уже указан
-901	336330785	gbak_db_create_failed	Failed to create database @1 Ошибка создания базы данных <строка>
-901	336724002	gsec_proj_specified	Project already specified Парметр «проект» уже указан
-901	336330786	gbak_decomp_len_error	RESTORE: decompression length error Восстановление: ошибка длины декомпрессии

Руководство по языку SQL СУБД Firebird

-901	335544355	no_segstr_close	BLOB was not closed BLOB был не закрытым
-901	336330787	gbak_tbl_missing	Cannot find table @1 Не могу найти таблицу <строка>
-901	336724003	gsec_org_specified	Organization already specified Организация уже указана
-901	336330788	gbak_blob_col_missing	Cannot find column for BLOB Не могу найти столбец для BLOB
-901	336724004	gsec_fname_specified	First name already specified Параметр «имя» уже указано
-901	335544357	open_trans	Cannot disconnect database with open transactions (@1 active) Не могу отключить базу данных от открытых коннектов (<число> активных)
-901	336330789	gbak_create_blob_failed	Gds_\$create_blob failed Операция Gds_\$create_blob failed не удалась
-901	336724005	gsec_mname_specified	Middle name already specified Второе имя (отчество) уже указано
-901	335544358	port_len	Message length error (encountered @1, expected @2) Ошибка в длине сообщения (встретилась <строка>, ожидалось <строка>)
-901	336330790	gbak_put_seg_failed	Gds_\$put_segment failed Операция Gds_\$put_segment не удалась
-901	336724006	gsec_lname_specified	Last name already specified Фамилия уже указана
-901	336330791	gbak_rec_len_exp	Expected record length Ошибка в ожидаемой длине записи
-901	336724008	gsec_inv_switch	Invalid switch specified Неверный параметр командной строки указан
-901	336330792	gbak_inv_rec_len	Wrong length record, expected @1 encountered @2 Неверная длина записи, ожидалось

Руководство по языку SQL СУБД Firebird

			<строка>, встретилась <строка>
-901	336330793	gbak_exp_data_type	Expected data attribute Ожидаемый атрибут данных
-901	336724009	gsec_amb_switch	Ambiguous switch specified Неоднозначный переключатель командной строки указан
-901	336330794	gbak_gen_id_failed	Failed in store_blr_gen_id Ошибка в store_blr_gen_id
-901	336724010	gsec_no_op_specified	No operation specified for parameters Нет операции указанной для параметров
-901	335544363	req_no_trans	No transaction for request Нет транзакции для запроса
-901	336330795	gbak_unk_rec_type	Do not recognize record type @1 Не распознан тип записи <строка>
-901	336724011	gsec_params_not_allowed	No parameters allowed for this operation Нет параметров доступных для этой операции
-901	335544364	req_sync	Request synchronization error Ошибка синхронизации запроса
-901	336724012	gsec_incompat_switch	Incompatible switches specified Указанные несовместимые переключатели командной строки
-901	336330796	gbak_inv_bkup_ver	Expected backup version 1..8. Found @1 Ожидалась версия бекапа в пределах 1 .. 8. Обнаружена <число>
-901	335544365	req_wrong_db	Request referenced an unavailable database Запрос ссылается на недоступный базу данных
-901	336330797	gbak_missing_bkup_desc	Expected backup description record Ожидается описание бекапа
-901	336330798	gbak_string_trunc	String truncated Усечение строки
-901	336330799	gbak_cant_rest_record	warning-- record could not be restored Предупреждение – запись не может

Руководство по языку SQL СУБД Firebird

			быть восстановлена
-901	336330800	gbak_send_failed	Gds_\$send failed Ошибка Gds_\$send
-901	335544369	segstr_no_read	Attempted read of a new, open BLOB Попытка чтения нового, а открыт BLOB
-901	336330801	gbak_no_tbl_name	No table name for data Не указано имя таблицы для данных
-901	335544370	segstr_no_trans	Attempted action on blob outside transaction Попытка действия на BLOB за пределами транзакции
-901	336330802	gbak_unexp_eof	Unexpected end of file on backup file Неожиданный признак конца файла в файле бекапа
-901	336330803	gbak_db_format_too_old	Database format @1 is too old to restore to Формат базы данных <строка> слишком старый чтобы восстановить базу
-901	335544372	segstr_wrong_db	Attempted reference to BLOB in unavailable database Попытка сослаться на BLOB в недоступной базе данных
-901	336330804	gbak_inv_array_dim	Array dimension for column @1 is invalid Размерность массива для столбца <строка> неверная
-901	336330807	gbak_xdr_len_expected	Expected XDR record length Ожидаемая длина записи XDR
-901	335544376	unres_rel	Table @1 was omitted from the transaction reserving list Таблица <строка> была исключена из списка резервирования транзакций
-901	335544377	uns_ext	Request includes a DSRI extension not supported in this implementation Запрос включает в себя расширение DSRI не поддерживается в этой реализации
-901	335544378	wish_list	Feature is not supported Опция не поддерживается

Руководство по языку SQL СУБД Firebird

-901	335544382	random	@1
-901	335544383	fatal_conflict	Unrecoverable conflict with limbo transaction @1 Неустранимый конфликт с лимбо транзакцией <строка>
-901	335740991	gfix_exceed_max	Internal block exceeds maximum size Внутренний блок превышает максимальный размер
-901	335740992	gfix_corrupt_pool	Corrupt pool Разрушение пула
-901	335740993	gfix_mem_exhausted	Virtual memory exhausted Виртуальная память исчерпана
-901	336330817	gbak_open_bkup_error	Cannot open backup file @1 Не могу открыть файл бекапа <строка>
-901	335740994	gfix_bad_pool	Bad pool id Неверный ID пула
-901	336330818	gbak_open_error	Cannot open status and error output file @1 Не могу получить статус и произошла ошибка выходного файла <строка>
-901	335740995	gfix_trn_not_valid	Transaction state @1 not in valid range. Состояние транзакции <число> вне пределов верного диапазона.
-901	335544392	bdbincon	Internal error Внутренняя ошибка
-901	336724044	gsec_inv_username	Invalid user name (maximum 31 bytes allowed) Неверное имя пользователя (максимум 31 байт доступен)
-901	336724045	gsec_inv_pw_length	Warning- maximum 8 significant bytes of password used Внимание – максимум 8 значащих байт для пароля используется
-901	336724046	gsec_db_specified	Database already specified База данных уже указана
-901	336724047	gsec_db_admin_specified	Database administrator name already specified Имя администратора базы данных

Руководство по языку SQL СУБД Firebird

			уже указано.
-901	336724048	gsec_db_admin_pw_specified	Database administrator password already specified Пароль администратора базы данных уже указан.
-901	336724049	gsec_sql_role_specified	SQL role name already specified Имя SQL роли уже указаны
-901	335741012	gfix_unexp_eoi	Unexpected end of input Неожиданный конец ввода
-901	335544407	dbbnotzer	Database handle not zero Указатель базы данных не ноль
-901	335544408	tranotzer	Transaction handle not zero Указатель транзакции не ноль
-901	335741018	gfix_recon_fail	Failed to reconnect to a transaction in database @1 Сбой реконнекта транзакции в базе данных <строка>
-901	335544418	trainlim	Transaction in limbo Транзакция в лимбо (зависла)
-901	335544419	notinlim	Transaction not in limbo Транзакция не в лимбо (зависла)
-901	335544420	traoutsta	Transaction outstanding Транзакция выдалась за пределы
-901	335544428	badmsgnum	Undefined message number Неопределенный номер сообщения
-901	335741036	gfix_trn_unknown	Transaction description item unknown Элемент описания транзакции неизвестен
-901	335741038	gfix_mode_req	"read_only" or "read_write" required Требуются параметры "read_only" или "read_write"
-901	335544431	blocking_signal	Blocking signal has been received Был получен блокирующий сигнал
-901	335741042	gfix_pzval_req	Positive or zero numeric value required Требуются положительное или нулевое

Руководство по языку SQL СУБД Firebird

			значение
-901	335544442	noargacc_read	Database system cannot read argument @1 СУБД не может прочитать аргумент <строка>
-901	335544443	noargacc_write	Database system cannot write argument @1 СУБД не может записать аргумент <строка>
-901	335544450	misc_interpreted	@1
-901	335544468	tra_state	Transaction @1 is @2 Транзакция <строка> является <строка>
-901	335544485	bad_stmt_handle	Invalid statement handle Неверный указатель выражения
-901	336330934	gbak_missing_block_fac	Blocking factor parameter missing Параметр blocking factor отсутствует
-901	336330935	gbak_inv_block_fac	Expected blocking factor, encountered "@1" Ожидался Blocking factor, встретилось <строка>
-901	336330936	gbak_block_fac_specified	A blocking factor may not be used in conjunction with device CT Blocking factor не могут быть использованы в сочетании с устройством CT
-901	336068796	dyn_role_does_not_exist	SQL role @1 does not exist SQL роль <строка> не существует
-901	336330940	gbak_missing_username	User name parameter missing Параметр имя пользователя отсутствует
-901	336330941	gbak_missing_password	Password parameter missing Параметр пароль отсутствует
-901	336068797	dyn_no_grant_admin_opt	User @1 has no grant admin option on SQL role @2 Пользователь <строка> не имеет администраторской опции GRANT в SQL роли <строка>

Руководство по языку SQL СУБД Firebird

-901	335544510	lock_timeout	Lock time-out on wait transaction В ждущей транзакции произошел тайм-аут блокировки
-901	336068798	dyn_user_not_role_member	User @1 is not a member of SQL role @2 Пользователь <строка> не является членом SQL роли
-901	336068799	dyn_delete_role_failed	@1 is not the owner of SQL role @2 <строка> не принадлежит SQL роли <строка>
-901	336068800	dyn_grant_role_to_user	@1 is a SQL role and not a user <строка> является SQL ролью, а не пользователем
-901	336068801	dyn_inv_sql_role_name	User name @1 could not be used for SQL role Имя пользователя не может использоваться для названия SQL роли
-901	336068802	dyn_dup_sql_role	SQL role @1 already exists SQL роль <строка> уже существует
-901	336068803	dyn_kywd_spec_for_role	Keyword @1 can not be used as a SQL role name Ключевое слово <строка> не может быть использовано в качестве имени роли
-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. SQL роль не поддерживается в более старых версиях СУБД. Требуется выполнить операцию бекап – респоторе для базы данных.
-901	336330952	gbak_missing_skipped_bytes	missing parameter for the number of bytes to be skipped Отсутствующий параметр для количества байт был обойден
-901	336330953	gbak_inv_skipped_bytes	Expected number of bytes to be skipped, encountered "@1" Ожидалось число байтов для пропуска, встретилось "<строка >"
-901	336068820	dyn_zero_len_id	Zero length identifiers are not allowed Идентификаторы с нулевой длиной не

Руководство по языку SQL СУБД Firebird

			допускаюся
-901	336330965	gbak_err_restore_charset	Character set Ошибка при восстановлении: Character set
-901	336330967	gbak_err_restore_collation	Collation Ошибка при восстановлении: Collation
-901	336330972	gbak_read_error	Unexpected I/O error while reading from backup file Неожиданная ошибка операции ввода – вывода при чтении файла бекапа
-901	336330973	gbak_write_error	Unexpected I/O error while writing to backup file Неожиданная ошибка операции ввода – вывода при записи файла бекапа
-901	336068840	dyn_wrong_gtt_scope	@1 cannot reference @2 <строка> не может ссылаться на <строка>
-901	336330985	gbak_db_in_use	Could not drop database @1 (database might be in use) Не могу удалить базу данных <строка> (операция DROP) (база данных может использоваться)
-901	336330990	gbak_sysmemex	System memory exhausted Системная память исчерпана
-901	335544559	bad_svc_handle	Invalid service handle Неверный дескриптор службы
-901	335544561	wrospbver	Wrong version of service parameter block Неверная версия блока параметров сервиса
-901	335544562	bad_spb_form	Unrecognized service parameter block Нераспознан блок параметров сервиса
-901	335544563	svcnotdef	Service @1 is not defined Сервис <строка> не определен
-901	336068856	dyn_ods_not_supp_feature	Feature '@1' is not supported in ODS @2.@3 Фича '<строка>' не поддерживается в ODS <число>.<число>

Руководство по языку SQL СУБД Firebird

-901	336331002	gbak_restore_role_failed	SQL role Ошибка восстановления SQL роль
-901	336331005	gbak_role_op_missing	SQL role parameter missing Параметр SQL роль отсутствует
-901	336331010	gbak_page_buffers_missing	Page buffers parameter missing Параметр Page buffers отсутствует
-901	336331011	gbak_page_buffers_wrong_param	Expected page buffers, encountered "@1" Ожидался параметр page buffers, встретилось <строка>
-901	336331012	gbak_page_buffers_restore	Page buffers is allowed only on restore or create Параметр page buffers позволяет только восстановить или создать
-901	336331014	gbak_inv_size	Size specification either missing or incorrect for file @1 Размер спецификации или отсутствует или некорректный для файла <строка>
-901	336331015	gbak_file_outof_sequence	File @1 out of sequence Файл <строка> выпал из последовательности
-901	336331016	gbak_join_file_missing	Can't join-- one of the files missing Не могу соединить – один из файлов отсутствует
-901	336331017	gbak_stdin_not_supptd	standard input is not supported when using join operation Поток stdin не поддерживается при использовании операции «соединить» (join)
-901	336331018	gbak_stdout_not_supptd	Standard output is not supported when using split operation Поток stdout не поддерживается при использовании операции “разделить”(split)
-901	336331019	gbak_bkup_corrupt	Backup file @1 might be corrupt Файл бэкапа <строка> может быть поврежден
-901	336331020	gbak_unk_db_file_spec	Database file specification missing Файл спецификации базы данных

Руководство по языку SQL СУБД Firebird

			отсутствует
-901	336331021	gbak_hdr_write_failed	Can't write a header record to file @1 Не могут записать заголовок в файл <строка>
-901	336331022	gbak_disk_space_ex	Free disk space exhausted Свободное место на диске исчерпано
-901	336331023	gbak_size_lt_min	File size given (@1) is less than minimum allowed (@2) Размер файла дан (<число>) – является меньшим чем минимально допущенный (<число>)
-901	336331025	gbak_svc_name_missing	Service name parameter missing Параметр имя службы отсутствует
-901	336331026	gbak_not_ownr	Cannot restore over current database, must be SYSDBA or owner of the existing database. Не могу восстановить поверх текущей базы данных, должен быть SYSDBA или владелец существующей базы данных.
-901	336331031	gbak_mode_req	"read_only" or "read_write" required Требуются режимы "read_only" или "read_write"
-901	336331033	gbak_just_data	Just data ignore all constraints etc. Только данные, игнорируя все условия контроля целостности данных
-901	336331034	gbak_data_only	Restoring data only ignoring foreign key, unique, not null & other constraints Режим восстановления «только данные» игнорирующий все ограничения по внешним ключам, условиям уникальности данных, NOT NULL и прочим условиям проверки целостности данных
-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 Операция UNION не поддерживается

Руководство по языку SQL СУБД Firebird

-901	335544614	dsql_construct_err	Unsupported DSQL construct Неподдерживаемая конструкция DSQL
-901	335544623	dsql_domain_err	Illegal use of keyword VALUE Неверное использование слова 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 Указанный домен или столбец – источник <строка> не существует
-901	335544656	dsql_var_conflict	Variable @1 conflicts with parameter in same procedure Переменная <строка> конфликтует с одноименным параметром в процедуре
-901	335544666	svr_version_too_old	Server version too old to support all CREATE DATABASE options Слишком старая версия сервера чтобы поддерживать все опции команды CREATE DATABASE
-901	335544673	no_delete	Cannot delete Не могу удалить
-901	335544675	sort_err	Sort error Ошибка сортировки
-901	335544703	svcnoexe	Service @1 does not have an associated executable Сервис <строка> не имеет связанного исполнителя
-901	335544704	net_lookup_err	Failed to locate host machine. Не удалось найти хост машины
-901	335544705	service_unknown	Undefined service @1/@2. Неопределенная служба <строка> / <строка>.
-901	335544706	host_unknown	The specified name was not found in the hosts file or Domain Name Services. Указанное имя не найдено в файле hosts или в DNS.

Руководство по языку SQL СУБД Firebird

-901	335544711	unprepared_stmt	Attempt to execute an unprepared dynamic SQL statement. Попытка выполнить неподготовленный DSQL запрос.
-901	335544716	svc_in_use	Service is currently busy: @1 Сервис в настоящий момент занят: <строка>
-901	335544731	tra_must_sweep	
-901	335544740	udf_exception	A fatal exception occurred during the execution of a user defined function. Фатальное исключение произошло во время выполнения UDF.
-901	335544741	lost_db_connection	Connection lost to database Соединение с базой данных потеряно
-901	335544742	no_write_user_priv	User cannot write to RDB\$USER_PRIVILEGES Пользователь не может писать в таблицу RDB\$USER_PRIVILEGES
-901	335544767	blob_filter_exception	A fatal exception occurred during the execution of a blob filter. Фатальное исключение произошло во время исполнения BLOB фильтра.
-901	335544768	exception_access_violation	Access violation.The code attempted to access a virtual address without privilege to do so. Ошибка доступа. Код попытался получить доступ к виртуальному адресу без соответствующих привелегий на это.
-901	335544769	exception_datatype_missalignment	Datatype misalignment.The attempted to read or write a value that was not stored on a memory boundary. Тип данных не выровнен. Попытка прочитать или записать значение, которое не хранится в границах области памяти.
-901	335544770	exception_array_bounds_exceeded	Array bounds exceeded. The code attempted to access an array element that is out of bounds. Превышение границ массива. Код пытался получить доступ к элементам массива за его пределами.

Руководство по языку SQL СУБД Firebird

-901	335544771	exception_float_denormal_operand	<p>Float denormal operand. One of the floating-point operands is too small to represent a standard float value.</p> <p>Аномальное число с плавающей точкой. Один из операндов с плавающей точкой слишком мал, чтобы представить стандартным значением с плавающей точкой.</p>
-901	335544772	exception_float_divide_by_zero	<p>Floating-point divide by zero. The code attempted to divide a floating-point value by zero.</p> <p>Числа с плавающей точкой; деление на ноль. Код попытался выполнить деление числа с плавающей точкой на ноль.</p>
-901	335544773	exception_float_inexact_result	<p>Floating-point inexact result. The result of a floating-point operation cannot be represented as a decimal fraction.</p> <p>Числа с плавающей точкой; неточный результат. Результат операции с плавающей точкой не может быть представлен в виде десятичной дроби.</p>
-901	335544774	exception_float_invalid_operand	<p>Floating-point invalid operand. An indeterminate error occurred during a floating-point operation.</p> <p>Числа с плавающей точкой; неверная операция. Неопределяемая ошибка произошла во время операций с плавающей точкой.</p>
-901	335544775	exception_float_overflow	<p>Floating-point overflow. The exponent of a floating-point operation is greater than the magnitude allowed.</p> <p>Числа с плавающей точкой; переполнение. Показатель операции с плавающей точкой больше, чем допустимая величина.</p>
-901	335544776	exception_float_stack_check	<p>Floating-point stack check. The stack overflowed or underflowed as the result of a floating-point operation.</p> <p>Числа с плавающей точкой; проверка стека. Стек переполнен или показатель операции с плавающей точкой меньше величины допустимого, в результате операции с плавающей точкой.</p>
-901	335544777	exception_float_underflow	<p>Floating-point underflow. The exponent of a floating-point operation is less than the magnitude allowed.</p> <p>Числа с плавающей точкой. Показатель операции с плавающей</p>

Руководство по языку SQL СУБД Firebird

			точкой меньше величины допустимого.
-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. Целые числа; деление на ноль. Код попытался выполнить операцию целочисленного деления целого числа на ноль.
-901	335544779	exception_integer_overflow	Integer overflow. The result of an integer operation caused the most significant bit of the result to carry. Переполнение целого числа. В результате операций с целыми числами был выставлен самый старший бит, отвечающий за перенос.
-901	335544780	exception_unknown	An exception occurred that does not have a description. Exception number @1. Произошло исключение, но оно не имеет описания. Номер исключения <число>
-901	335544781	exception_stack_overflow	Stack overflow. The resource requirements of the runtime stack have exceeded the memory available to it. Переполнение стека. Требования ресурсов к операциям к стеку превысили память отведенную под него.
-901	335544782	exception_sigsegv	Segmentation Fault. The code attempted to access memory without priviledges. Ошибка сегментации. Код попытался получить доступ к области памяти без соответствующих привелегий.
-901	335544783	exception_sigill	Illegal Instruction. The Code attempted to perfrom an illegal operation. Неверная инструкция. Код попытался выполнить нелегальную операцию.
-901	335544784	exception_sigbus	Bus Error. The Code caused a system bus error. Ошибка шины. Код вызвал системную ошибку шины.
-901	335544785	exception_sigfpe	Floating Point Error. The Code caused an Arithmetic Exception or a floating point exception. Ошибка операции с плавающей точкой. Код вызвал арифметическое исключение или исключение операций

Руководство по языку SQL СУБД Firebird

			с плавающей точкой.
-901	335544786	ext_file_delete	Cannot delete rows from external files. Невозможно удалить строки из внешних таблиц/файлов
-901	335544787	ext_file_modify	Cannot update rows in external files. Не могут обновлять строки во внешних файлах.
-901	335544788	adm_task_denied	Unable to perform operation.You must be either SYSDBA or owner of the database Не могу выполнить операцию. Вы должны быть SYSDBA или владельцем базы данных.
-901	335544794	cancelled	Operation was cancelled Операция была отменена
-901	335544797	svcnouser	User name and password are required while attaching to the services manager Для доступа к менеджеру сервисов требуется имя пользователя и пароль
-901	335544801	datatype_notsup	Data type not supported for arithmetic Тип данных не поддерживается для арифметических операций
-901	335544803	dialect_not_changed	Database dialect not changed. Диалект базы данных не изменен.
-901	335544804	database_create_failed	Unable to create database @1 Не могу создать базу данных <строка>
-901	335544805	inv_dialect_specified	Database dialect @1 is not a valid dialect. Диалект базы данных <строка> не является верным диалектом
-901	335544806	valid_db_dialects	Valid database dialects are @1. Верным диалектом базы данных является <строка>.
-901	335544811	inv_client_dialect_specified	Passed client dialect @1 is not a valid dialect. Переданный клиентской программой диалект <строка> не является верным диалектом
-901	335544812	valid_client_dialects	Valid client dialects are @1. Верным клиентским диалектом является <строка>.

Руководство по языку SQL СУБД Firebird

-901	335544814	service_not_supported	Services functionality will be supported in a later version of the product Функциональность сервисов будет поддерживаться в дальнейших версиях продукта
-901	335544820	invalid_savepoint	Unable to find savepoint with name @1 in transaction context Не могу найти точку сохранения транзакции с именем <строка> в контексте транзакции
-901	335544835	bad_shutdown_mode	Target shutdown mode is invalid for database "@1" Указание режима «шатдаун» неверное для базы данных "<строка>"
-901	335544840	no_update	Cannot update Не могу обновить
-901	335544842	stack_trace	@1
-901	335544843	ctx_var_not_found	Context variable @1 is not found in namespace @2 Контекстная переменная <строка> не найдена в пространстве имен <строка>
-901	335544844	ctx_namespace_invalid	Invalid namespace name @1 passed to @2 Неверное имя пространства имен <строка> передается в <строка>
-901	335544845	ctx_too_big	Too many context variables Слишком много контекстных переменных
-901	335544846	ctx_bad_argument	Invalid argument passed to @1 Неверный аргумент передан в <строка>
-901	335544847	identifier_too_long	BLR syntax error. Identifier @1... is too long Ошибка синтаксиса BLR. Идентификатор <строка> является слишком большим
-901	335544859	invalid_time_precision	Time precision exceeds allowed range (0-@1) Уровень точности времени (тип данных TIME) превышает допустимый диапазон (0 - <число>)

Руководство по языку SQL СУБД Firebird

-901	335544866	met_wrong_gtt_scope	@1 cannot depend on @2 <строка> не может зависеть от <строка>
-901	335544868	illegal_prc_type	Procedure @1 is not selectable (it does not contain a SUSPEND statement) Процедура <строка> не является процедурой выборки (она не содержит оператор SUSPEND)
-901	335544869	invalid_sort_datatype	Datatype @1 is not supported for sorting operation Тип данных <строка> не поддерживается для операции сортировка
-901	335544870	collation_name	COLLATION @1 Порядок сортировки <строка>
-901	335544871	domain_name	DOMAIN @1 ДОМЕН <строка>
-901	335544874	max_db_per_trans_allowed	A multi database transaction cannot span more than @1 databases Мультибазовая транзакция не может занимать более чем <число> баз данных
-901	335544876	bad_proc_BLR	Error while parsing procedure @1's BLR Ошибка во время разбора BLR процедуры <строка>
-901	335544877	key_too_big	Index key too big Ключ индекса слишком велик
-901	336397211	dsql_too_many_values	Too many values (more than @1) in member list to match against Слишком много значений (более чем <число>) в списке членов,, чтобы соответствовать против
-901	336397236	dsql_unsupp_feature_dialect	Feature is not supported in dialect @1 Особенность не поддерживается в диалексте <число>
-902	335544333	bug_check	Internal gds software consistency check (@1) Внутренняя проверка целостности программного обеспечения (<строка>)
-902	335544335	db_corrupt	Database file appears corrupt (@1) Файл базы данных является

Руководство по языку SQL СУБД Firebird

			поврежденным (<строка>)
-902	335544344	io_error	I/O error for file "@2" Ошибка ввода – вывода для файла <строка>
-902	335544346	metadata_corrupt	Corrupt system table Повреждение системной таблицы
-902	335544373	sys_request	Operating system directive @1 failed Директива операционной системы <строка> не удалась
-902	335544384	badblk	Internal error Внутренняя ошибка
-902	335544385	invpoolcl	Internal error Внутренняя ошибка
-902	335544387	relbadblk	Internal error Внутренняя ошибка
-902	335544388	blktoobig	Block size exceeds implementation restriction Размер блока превышает ограничение реализации
-902	335544394	badodsver	Incompatible version of on-disk structure Несовместимая версия ODS
-902	335544397	dirtypage	Internal error Внутренняя ошибка
-902	335544398	waitfortra	Internal error Внутренняя ошибка
-902	335544399	doubleloc	Internal error Внутренняя ошибка
-902	335544400	nodnotfnd	Internal error Внутренняя ошибка
-902	335544401	dupnodfnd	Internal error Внутренняя ошибка
-902	335544402	locnotmar	Internal error Внутренняя ошибка
-902	335544404	corrupt	Database corrupted База данных повреждена

Руководство по языку SQL СУБД Firebird

-902	335544405	badpage	Checksum error on database page @1 Ошибка контрольной суммы на странице базы данных <число>
-902	335544406	badindex	Index is broken Индекс поврежден
-902	335544409	trareqmis	Transaction--request mismatch (synchronization error) Транзакция – несоответствие запроса (ошибка синхронизации)
-902	335544410	badhndcnt	Bad handle count Неверный счетчик указателей
-902	335544411	wrotpbver	Wrong version of transaction parameter block Неверная версия блока параметров транзакции
-902	335544412	wroblrver	Unsupported BLR version (expected @1, encountered @2) Неподдерживаемая версия BLR (ожидалась <строка> встретилась <строка>)
-902	335544413	wrodpbver	Wrong version of database parameter block Неверная версия блока параметров базы данных
-902	335544415	badrelation	Database corrupted База данных разрушена
-902	335544416	nodetach	Internal error Внутренняя ошибка
-902	335544417	notremote	Internal error Внутренняя ошибка
-902	335544422	dbfile	Internal error Внутренняя ошибка
-902	335544423	orphan	Internal error Внутренняя ошибка
-902	335544432	lockmanerr	Lock manager error Ошибка менеджера блокировок

Руководство по языку SQL СУБД Firebird

-902	335544436	sqlerr	SQL error code = @1 SQL ошибка код = <число>
-902	335544448	bad_sec_info	
-902	335544449	invalid_sec_info	
-902	335544470	buf_invalid	Cache buffer for page @1 invalid Буфер КЭШа для страницы <число> неверный
-902	335544471	indexnotdefined	There is no index in table @1 with id @2 Не существует индекса в таблице <строка> с идентификатором <число>
-902	335544472	login	Your user name and password are not defined. Ask your database administrator to set up a Firebird login. Ваши имя и пароль не определены. Чтобы установить соединение с Firebird обратитесь к администратору базы данных.
-902	335544506	shutinprog	Database @1 shutdown in progress Выполняется отсанов (shutdown) базы данных <строка>
-902	335544528	shutdown	Database @1 shutdown База данных <строка> в режиме «шатдаун»
-902	335544557	shutfail	Database shutdown unsuccessful Не успешный шатдаун базы данных
-902	335544569	dsq_err	Dynamic SQL Error Ошибка динамического SQL
-902	335544653	psw_attach	Cannot attach to password database Невозможно соединиться с базой данных пароля
-902	335544654	psw_start_trans	Cannot start transaction for password database Невозможно стартовать транзакцию для базы данных пароля
-902	335544717	err_stack_limit	Stack size insufficient to execute current request Размер стека недостаточен для

Руководство по языку SQL СУБД Firebird

			выполнения текущего запроса
-902	335544721	network_error	Unable to complete network request to host "@1". Невозможно завершить сетевой запрос на хост "<строка>"
-902	335544722	net_connect_err	Failed to establish a connection. Ошибка при установлении соединения
-902	335544723	net_connect_listen_err	Error while listening for an incoming connection. Ошибка при прослушивании входного соединения
-902	335544724	net_event_connect_err	Failed to establish a secondary connection for event processing. Ошибка при установлении вторичного соединения для обработки события
-902	335544725	net_event_listen_err	Error while listening for an incoming event connection request. Ошибка при прослушивании входного запроса для события соединения
-902	335544726	net_read_err	Error reading data from the connection. Ошибка чтения данных из соединения
-902	335544727	net_write_err	Error writing data to the connection. Ошибка записи данных в соединение
-902	335544732	unsupported_network_drive	Access to databases on file servers is not supported. Доступ к базам данных в файловых серверах не поддерживается
-902	335544733	io_create_err	Error while trying to create file Ошибка ввода - вывода при попытке создания файла
-902	335544734	io_open_err	Error while trying to open file Ошибка ввода - вывода при попытке открытия файла

Руководство по языку SQL СУБД Firebird

-902	335544735	io_close_err	Error while trying to close file Ошибка ввода - вывода при попытке закрытия файла
-902	335544736	io_read_err	Error while trying to read from file Ошибка ввода - вывода при попытке чтения из файла
-902	335544737	io_write_err	Error while trying to write to file Ошибка ввода - вывода при попытке записи в файл
-902	335544738	io_delete_err	Error while trying to delete file Ошибка ввода - вывода при попытке удаления файла
-902	335544739	io_access_err	Error while trying to access file Ошибка ввода - вывода при попытке доступа к файлу
-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. Ваше регистрационное имя <строка> то же, что и имя роли SQL. Уточните у вашего администратора базы данных допустимое регистрационное имя Firebird.
-902	335544791	file_in_use	The file @1 is currently in use by another process. Try again later. Файл <строка> в настоящее время используется другим процессом, Попробуйте позже.
-902	335544795	unexp_spb_form	Unexpected item in service parameter block, expected @1 Неопределенный элемент в блоке пара метров сервиса, ожидается <строка>
-902	335544809	extern_func_dir_error	Function @1 is in @2, which is not in a permitted directory for external functions. Функция <строка> находится в <стро-

Руководство по языку SQL СУБД Firebird

			ка>, что не является доступным каталогом для внешних функций.
-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. Файл превысил максимальный размер 2 Гбайт, Добавьте другой файл базы данных или используйте 64битовую версию Firebird.
-902	335544831	conf_access_denied	Access to @1 "@2" is denied by server administrator Доступ к <строка> "<строка>" отвергнут администратором сервера
-902	335544834	cursor_not_open	Cursor is not open Курсор не открыт
-902	335544841	cursor_already_open	Cursor is already open Курсор уже открыт
-902	335544856	att_shutdown	Connection shutdown Соединение остановлено
-902	335544882	long_login	Login name too long (@1 characters, maximum allowed @2) Наименование логина слишком велико (<строка> символов, максимально возможно <число>)
-904	335544324	bad_db_handle	Invalid database handle (no active connection) Указатель базы данных неверный (нет активного соединения)
-904	335544375	unavailable	Unavailable database Недоступная база данных
-904	335544381	imp_exc	Implementation limit exceeded Исчерпан лимит выполнения
-904	335544386	nopoolids	Too many requests Слишком много запросов

Руководство по языку SQL СУБД Firebird

-904	335544389	bufexh	Buffer exhausted Исчерпан буфер
-904	335544391	bufinuse	Buffer in use Буфер используется
-904	335544393	reqinuse	Request in use Запрос используется
-904	335544424	no_lock_mgr	No lock manager available Нет доступного менеджера блокировок
-904	335544430	virmemexh	Unable to allocate memory from operating system Невозможно выделить память в опе рационной системе
-904	335544451	update_conflict	Update conflicts with concurrent update Изменение конфликтует с конкурирующим обновлением
-904	335544453	obj_in_use	Object @1 is in use Объект <строка> используется
-904	335544455	shadow_accessed	Cannot attach active shadow file Невозможно соединиться с активным файлом теневой копии
-904	335544460	shadow_missing	A file in manual shadow @1 is unavailable Файл в ручной теневой копии <число> недоступен
-904	335544661	index_root_page_full	Cannot add index, index root page is full. Невозможно добавить индекс, корневая страница индексов заполнена
-904	335544676	sort_mem_err	Sort error: not enough memory Ошибка сортировки: недостаточно памяти
-904	335544683	req_depth_exceeded	Request depth exceeded. (Recursive definition?) Превышена глубина запроса (рекурсивное определение?)

Руководство по языку SQL СУБД Firebird

-904	335544758	sort_rec_size_err	Sort record size of @1 bytes is too big байт Размер записи сортировки в <число> байт слишком велик
-904	335544761	too_many_handles	Too many open handles to database Слишком много открыто дескрипторов базы данных
-904	335544792	service_att_err	Cannot attach to services manager Не могу подключиться к менеджеру сервисов
-904	335544799	svc_name_missing	The service name was not specified. Не указано имя сервиса
-904	335544813	optimizer_between_err	Unsupported field type specified in BETWEEN predicate. Указан неподдерживаемый тип поля в предикате BETWEEN
-904	335544827	exec_sql_invalid_arg	Invalid argument in EXECUTE STATEMENT-cannot convert to string Неверный аргумент в EXECUTE STATEMENT – невозможно конвертировать в строку
-904	335544828	exec_sql_invalid_req	Wrong request type in EXECUTE STATEMENT '@1' Неверный тип запроса в EXECUTE STATEMENT '<строка>'
-904	335544829	exec_sql_invalid_var	Variable type (position @1) in EXECUTE STATEMENT '@2' INTO does not match returned column type Тип переменной (позиция <число>) в EXECUTE STATEMENT '<строка>' INTO не соответствует возвращаемому типу столбца
-904	335544830	exec_sql_max_call_exceeded	Too many recursion levels of EXECUTE STATEMENT Слишком много уровней рекурсии в EXECUTE STATEMENT
-904	335544832	wrong_backup_state	Cannot change difference file name while database is in backup mode Не могу изменить файл – разницу пока база данных находится в режиме бекапа

Руководство по языку SQL СУБД Firebird

-904	335544852	partner_idx_incompat_type	Partner index segment no @1 has incompatible data type Сегмент <строка> индекса – партнера содержит не совместимый тип данных
-904	335544857	blobtoobig	Maximum BLOB size exceeded Достигнут максимальный размер BLOB
-904	335544862	record_lock_not_supp	Stream does not support record locking Поток не поддерживает блокировку записей.
-904	335544863	partner_idx_not_found	Cannot create foreign key constraint @1. Partner index does not exist or is inactive. Не могу создать конструкцию внешнего ключа. Индекс - партнер не существует или является неактивным.
-904	335544864	tra_num_exc	Transactions count exceeded. Perform backup and restore to make database operable again Превышено число допустимых транзакций. Выполнение бекапа и ресторе сделает вновь базу данных работоспособной.
-904	335544865	field_disappeared	Column has been unexpectedly deleted Столбец был неожиданно удален
-904	335544878	concurrent_transaction	Concurrent transaction number is @1 Число конкурирующих транзакций <число>
-906	335544744	max_att_exceeded	Maximum user count exceeded. Contact your database administrator. Превышен максимум счетчика пользователей. Свяжитесь с вашим администратором базы данных.
-909	335544667	drdb_completed_with_errs	Drop database completed with errors Удаление базы данных завершилось с ошибками
-911	335544459	rec_in_limbo	Record from transaction @1 is stuck in limbo Запись транзакции <число> становится зависшей
-913	335544336	Deadlock	Deadlock Взаимная блокировка

Руководство по языку SQL СУБД Firebird

-922	335544323	bad_db_format	File @1 is not a valid database Файл <строка> не является допустимой базой данных
-923	335544421	connect_reject	Connection rejected by remote interface Соединение отменено удаленным интерфейсом
-923	335544461	cant_validate	Secondary server attachments cannot validate databases Вторичные подключения к серверу не могут проверять базу данных
-923	335544464	cant_start_logging	Secondary server attachments cannot start logging Вторичные подключения к серверу не могут начинать логгирование
-924	335544325	bad_dpb_content	Bad parameters on attach or create database Неверные параметры при подключении или создании базы данных
-924	335544441	bad_detach	Database detach completed with errors Отключение от базы данных завершилось с ошибками
-924	335544648	conn_lost	Connection lost to pipe server Потеря соединения с каналом сервера
-926	335544447	no_rollback	No rollback performed Не выполнен откат транзакции
-999	335544689	ib_error	Firebird error Ошибка Firebird