# Firebird SQL Reference Guide

The complete reference of all SQL keywords and commands supported by Firebird

Members of the Firebird Documentation project

December 2007

# Table of Contents

# Introduction

The Firebird SQL Reference Guide contains an alphabetical index of all keywords and built-in-functions available in a Firebird database.

Note that not all terms are available everywhere. At the start of every entry there is an item "Availability" that tells in what context(s) a keyword or function can be used. The terms used there are described in the following.

## *DSQL*

Dynamic SQL is the context of a SQL client ( application ) sending SQL commands to the server.

## *ESQL*

Embedded SQL is the context of a SQL command embedded in an application. This is in essence the same as DSQL, except that every ESQL statement must be preceeded with the EXEC SQL keyword.

## *ISQL*

ISQL (or Interactive SQL) is a command line tool that is included in the Firebird distribution. It allows access to (almost) the full feature set available in Firebird, and is the recommended tool to narrow down the source of a potential problem with a SQL command should you find one. Unlike most other connectivity components and tools, ISQL shows also warning messages that may not be shown

## *PSQL*

PSQL (or Procedural SQL) is the SQL context used in Stored Procedures and Triggers. There are some special commands and keywords only available in PSQL, like the NEW and OLD context variables in triggers. But there are also some limitations against D/E/ISQL: as a rule of thumb, PSQL is limited to DML (Data Manipulation Language), while the other flavours also allow DDL (Data Definition Language) statements.

# Alphabetical keyword and function index

## *ABS() [2.1]*

Returns the absolute value of a number.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
ABS(<numeric expression>)
```

> **Important**
> <Notes>

| Argument | Description |
|---|---|
| <number expression> | The numeric expression whose absolute value is returned |

## Description

Returns the absolute value of a number. The result is always >= 0.

## Examples

```
select abs(amount) from transactions

select abs(4-7) from rdb$database
(returns 3)

select abs(NULL) from rdb$database
(returns NULL)
```

**See also:** SIGN()

# *ACOS() [2.1]*

Returns the arc cosine of a number.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
ACOS(<numeric expression>)
```

**Important**
The argument to ACOS must be in the range -1 to 1.

| Argument | Description |
|---|---|
| <number expression> | The numeric expression whose arc cosine is returned |

## Description

Returns the arc cosine of a number. Argument to

ACOS must be in the range -1 to 1. Returns a value

in the range 0 to PI.

## Examples

```
select acos(x) from y
```

**See also:** COS(), SIN()

# ALTER DATABASE

Adds secondary files to the current database.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
ALTER {DATABASE | SCHEMA}
   ADD <add_clause>;
<add_clause> = FILE 'filespec' [<fileinfo>] [<add_clause>
]
<fileinfo> = LENGTH [=] int [PAGE[S]]
 | STARTING [AT [PAGE]] int [<fileinfo>]
```

(This text is currently not included because of possible copyright issues.)

**See also:** CREATE DATABASE, DROP DATABASE

**See also:** the Data Definition Guide for more information about multifile databases and the Operations Guide for more information about exclusive database access.

# ALTER DATABASE BEGIN/END BACKUP [2.0]

(no contents yet)

# ALTER DOMAIN

Changes a domain definition.

**Availability:** DSQL ESQL ISQL PSQL ALTER DOMAIN { name | old_name TO new_name } SET DEFAULT {literal | NULL | USER} | DROP DEFAULT | ADD [CONSTRAINT] CHECK (<dom_search_condition>) | DROP CONSTRAINT | new_col_name | TYPE datatype; <dom_search_condition> = VALUE <operator> <val> | VALUE [NOT] BETWEEN <val> AND <val> | VALUE [NOT] LIKE <val> [ESCAPE <val>] | VALUE [NOT] IN (<val> [, <val> ...]) | VALUE IS [NOT] NULL | VALUE [NOT] CONTAINING <val> | VALUE [NOT] STARTING [WITH] <val> | (<dom_search_condition>) | NOT <dom_search_condition> | <dom_search_condition> OR <dom_search_condition> | <dom_search_condition> AND <dom_search_condition> <operator> = {= | < | > | <= | >= | !< | !> | < > | !=}

(This text is currently not included because of possible copyright issues.)

**See also:** CREATE DOMAIN, CREATE TABLE, DROP DOMAIN, For a complete discussion of creating domains, and using them to create column definitions, see Firebird domains in Using Firebird- Domains and Generators (ch. 15 p. 285). ALTER EXCEPTION Changes the message associated with an existing exception.

**Availability1:** DSQL ESQL ISQL PSQL

## Syntax

```
ALTER EXCEPTION name 'message'
```

**Argument1:** Description name Name of an existing exception message 'message' Quoted string containing ASCII values

**See also:** ALTER PROCEDURE, ALTER TRIGGER, CREATE EXCEPTION, CREATE PROCEDURE, CREATE TRIGGER, DROP EXCEPTION, For more information on creating, raising, and handling exceptions, refer to Using Firebird- Error trapping and handling (ch. 25 p. 549).

# ALTER EXTERNAL FUNCTION [2.0]

(no contents yet)

# ALTER INDEX

Activates or deactivates an index.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

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

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER TABLE, CREATE INDEX, DROP INDEX, SET STATISTICS

## *ALTER PROCEDURE*

Changes the definition of an existing stored procedure.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
ALTER PROCEDURE name
   [(param <datatype> [, param <datatype> ...])]
   [RETURNS (param <datatype> [, param <datatype> ...])]
   AS <procedure_body> [terminator]
```

(This text is currently not included because of possible copyright issues.)

**See also:** CREATE PROCEDURE for a complete description terminator Terminator defined by the ISQL SET TERM command to signify the end of the procedure body; required by ISQL

### Syntax

```
SET TERM <new terminator> <old terminator>
The <old terminator> is not part of the command, but the command
terminator. Because SET TERM is exclusively an ISQL command, the command
terminator is always required.
A procedure can be altered by its creator, the SYSDBA user and, on
Linux/UNIX, the root user and any user with root privileges..
Procedures in use are not altered until they are no longer in use.
ALTER PROCEDURE changes take effect when they are committed. Changes are
then reflected in all applications that use the procedure without
recompiling or relinking.
```

**See also:** CREATE PROCEDURE, DROP PROCEDURE, EXECUTE PROCEDURE, For more information on creating and using procedures, see Using Firebird- Programming on Firebird Server (ch. 25 p. 494)., For a complete description of the statements in procedure and trigger language, refer to

PSQL-Firebird Procedural Language. ALTER PROCEDURE Changes the definition of an existing stored procedure.

**Availability1:** DSQL ESQL ISQL PSQL

## Syntax

```
ALTER PROCEDURE name
   [(param <datatype> [, param <datatype> ...])]
   [RETURNS (param <datatype> [, param <datatype> ...])]
   AS <procedure_body> [terminator]
```

**Argument1:** Description name Name of an existing procedure param datatype Input parameters used by the procedure; valid datatypes are listed under CREATE PROCEDURE RETURNS param datatype Output parameters used by the procedure; valid datatypes are listed under CREATE PROCEDURE procedure_body The procedure body includes: o Local variable declarations o A block of statements in procedure and trigger language

**See also:** CREATE PROCEDURE for a complete description terminator Terminator defined by the ISQL SET TERM command to signify the end of the procedure body; required by ISQL

## Syntax

```
SET TERM <new terminator> <old terminator>
The <old terminator> is not part of the command, but the command
terminator. Because SET TERM is exclusively an ISQL command, the command
terminator is always required.
A procedure can be altered by its creator, the SYSDBA user and, on
Linux/UNIX, the root user and any user with root privileges..
Procedures in use are not altered until they are no longer in use.
ALTER PROCEDURE changes take effect when they are committed. Changes are
then reflected in all applications that use the procedure without
recompiling or relinking.
```

**See also:** CREATE PROCEDURE, DROP PROCEDURE, EXECUTE PROCEDURE, For more information on creating and using procedures, see Using Firebird- Programming on Firebird Server (ch. 25 p. 494)., For a complete description of the statements in procedure and trigger language, refer to PSQL-Firebird Procedural Language.

# *ALTER SEQUENCE .. RESTART WITH [2.0]*

Sets the current value of a sequence / generator

**Availability:** +DSQL +ESQL +ISQL -PSQL

## Syntax

```
ALTER SEQUENCE <name> RESTART WITH <start_value>
```

**Important**
ALTER SEQUENCE, like SET GENERATOR, is a good way to screw up the generation of key values! It is important to know that sequences and generators are outside of any transaction control.

| Argument | Description |
|---|---|
| <name> | name of the sequence / generator to be set |
| <start_value> | new starting value for the sequence / generator |

## Description

This is the SQL-99-compliant (and therefor recommended) syntax for the SET GENERATOR command.

It directly sets a sequence / generator to the given value.

The command is not available in PSQL since it is a DDL and not a DML statement

(this can, however, be surpassed by the use of EXECUTE STATEMENT).

This command is useful to reset e.g. an ID-generating sequence after a DELETE FROM <table>,

but in almost all other circumstances it is a dangerous thing to do.

Read the "Generator Guide" which is available as part of the Firebird documentation set

for an in-depth discussion of the use of sequences / generators, and esp. why it

is dangerous and not recommended to use this statement in live databases.

## Examples

```
ALTER SEQUENCE SEQ_ID_EMPLOYEE RESTART WITH 1;
(equivalent to SET GENERATOR SEQ_ID_EMPLOYEE TO 1)
```

**See also:** SET GENERATOR, CREATE SEQUENCE, DROP SEQUENCE, NEXT VALUE FOR

# *ALTER TABLE*

Changes a table by adding, dropping, or modifying columns or integrity constraints.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
ALTER TABLE table <operation> [, <operation> ...];
<operation> = ADD <col_def>
 | ADD <tconstraint>
 | ALTER [COLUMN] column_name <alt_col_clause>
 | DROP col
 | DROP CONSTRAINT constraint
<alt_col_clause> = TO new_col_name
 | TYPE new_col_datatype
 | POSITION new_col_position
<col_def> = col {<datatype> | COMPUTED [BY] (<expr>) |
domain}
 [DEFAULT {literal | NULL | USER}]
 [NOT NULL]
 [<col_constraint>]
 [COLLATE collation]
<datatype> =
 {SMALLINT | INTEGER | FLOAT | DOUBLE PRECISION}[<array_dim>]
 | (DATE | TIME | TIMESTAMP}  [<array_dim>]
 | {DECIMAL | NUMERIC} [(precision [, scale])] [<array_dim>]
 | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(int)]
  [<array_dim>] [CHARACTER SET charname]
 | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR}
  [VARYING] [(int)] [<array_dim>]
 | BLOB [SUB_TYPE {int | subtype_name}] [SEGMENT SIZE int]
  [CHARACTER SET charname]
 | BLOB [(seglen [, subtype])]
<array_dim> = [[x:]y [, [x:]y ...]]
<expr> = a valid SQL expression that results in a single value
<col_constraint> = [CONSTRAINT constraint]
 { UNIQUE
 | PRIMARY KEY
 | REFERENCES other_table [(other_col [, other_col ...])]
  [ON DELETE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
  [ON UPDATE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
 | CHECK (<search_condition>)}
<tconstraint> = [CONSTRAINT constraint]
 {{PRIMARY KEY | UNIQUE} (col [, col ...])
 | FOREIGN KEY (col [, col ...])
  REFERENCES other_table [(other_col [, other_col ...])]
   [ON DELETE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
   [ON UPDATE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
 | CHECK (<search_condition>)}
<search_condition> = <val> <operator> {<val> | (<
    select_one>)}
 | <val> [NOT] BETWEEN <val> AND <val>
 | <val> [NOT] LIKE <val> [ESCAPE <val>]
 | <val> [NOT] IN (<val> [, <val> ...] | <
select_list>)
 | <val> IS [NOT] NULL
 | <val> {>= | <=} <val>
 | <val> [NOT] {= | < | >} <val>
 | {ALL | SOME | ANY} (<select_list>)
 | EXISTS (<select_expr>)
 | SINGULAR (<select_expr>)
 | <val> [NOT] CONTAINING <val>
 | <val> [NOT] STARTING [WITH] <val>
 | (<search_condition>)
 | NOT <search_condition>
```

```
 | <search_condition> OR <search_condition>
 | <search_condition> AND <search_condition>
<val> = { col [<array_dim>] | :variable
 | <constant> | <expr> | <function>
 | udf ([<val> [, <val> ...]])
 | NULL | USER | RDB$DB_KEY | ? }
 [COLLATE collation]
<constant> = num | 'string' | _charsetname 'string'
<function> = COUNT (* | [ALL] <val> | DISTINCT <val>)
 | SUM ([ALL] <val> | DISTINCT <val>)
 | AVG ([ALL] <val> | DISTINCT <val>)
 | MAX ([ALL] <val> | DISTINCT <val>)
 | MIN ([ALL] <val> | DISTINCT <val>)
 | CAST (<val> AS <datatype>)
 | UPPER (<val>)
 | GEN_ID (generator, <val>)
<operator> = {= | < | > | <= | >= | !< | !> | <
> | !=}
<select_one> = SELECT on a single column; returns exactly one value.
<select_list> = SELECT on a single column; returns zero or more
values.
<select_expr> = SELECT on a list of values; returns zero or more
values.
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER DOMAIN, CREATE DOMAIN, CREATE TABLE, For more information about altering tables, see Using Firebird- Altering tables (ch. 17 p. 340).

## ALTER TRIGGER

Changes an existing trigger.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
ALTER TRIGGER name
   [ACTIVE | INACTIVE]
   [{BEFORE | AFTER} {DELETE | INSERT | UPDATE}]
   [POSITION number]
   [AS <trigger_body>] [terminator]
```

(This text is currently not included because of possible copyright issues.)

**See also:** CREATE TRIGGER, DROP TRIGGER, For a complete description of the statements in procedure and trigger language, PSQL-Firebird Procedural Language., For more information, see Using Firebird- Triggers (ch. 25 p. 532).

## ASCII_CHAR() [2.1]

Returns the ASCII character with the specified code

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
ASCII_CHAR(<numeric expression>)
```

> **Important**
> The argument to ASCII_CHAR must be in the range 0 to 255.

| Argument | Description |
|---|---|
| <numeric expression> | The code for the ASCII character to be returned |

## Description

Returns the ASCII character with the specified

code. The argument to ASCII_CHAR must be in the

range 0 to 255. The result is returned in character set

NONE.

## Examples

```
1. DSQL
select ascii_char(65) from rdb$database
(returns 'A')

2. PSQL
mystr = mystr || ascii_char(13) || ascii_char(10);
(adds a Carriage Return + Line Feed to mystr)

3. PSQL

The following selectable procedure returns the alphabet in upper and lower
case:

CREATE PROCEDURE ALPHABET
returns (ALPHA_UPPER char(26), ALPHA_LOWER char(26))
AS
declare variable i integer;
begin
  ALPHA_UPPER = '';
  ALPHA_LOWER = '';
```

```
  i = 0;

  while (i < 26) do
  begin
    ALPHA_UPPER = TRIM(ALPHA_UPPER) || ASCII_CHAR(i + 65);
    ALPHA_LOWER = TRIM(ALPHA_LOWER) || ASCII_CHAR(i + 65 + (ASCII_VAL('a')-
ASCII_VAL('A')));

    i = i + 1;
  end

  suspend;
end
```

**See also:** ASCII_VAL()

# ASCII_VAL() [2.1]

Returns the ASCII code of the first character of the specified string.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
ASCII_VAL(<val>)
```

> **Important**
> if <val> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <val> | A column, constant, host-language variable, expression, function, or UDF that evaluates to a character datatype |

## Description

Returns the ASCII code of the first character of the

specified string.

**Rules:** 1. Returns 0 if the string is empty 2. Throws an error if the first character is multi-byte 3. Returns NULL if <val> is (or evaluates to) NULL

## Examples

```
select ascii_val(x) from y

select ascii_val('A') from rdb$database
(returns 65)
```

**See also:** ASCII_CHAR()

## ASIN() [2.1]

Returns the arc sine of a number.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
ASIN(<number>)
```

> **Important**
> The argument to ASIN must be in the range -1 to 1.

| Argument | Description |
|---|---|
| <number> | The number or numeric expression whose arc sine is returned |

### Description

Returns the arc sine of a number. Argument to

ASIN must be in the range -1 to 1. Returns a value

in the range -PI/2 to PI/2.

### Examples

```
select asin(-1) from rdb$database
(returns 1,5707963267949 = -PI/2)

select asin(0) from rdb$database
(returns 0)

select asin(1) from rdb$database
(returns 1,5707963267949 = PI/2)
```

**See also:** COS(), SIN()

## ATAN() [2.1]

Returns the arc tangent of a number.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
ATAN(<number>)
```

> **Important**
> The argument to ATAN must be in the range -1 to 1.

| Argument | Description |
|---|---|
| <number> | The number or numeric expression whose arc tangent is returned |

### Description

Returns the arc sine of a number. Argument to

ATAN must be in the range -1 to 1. Returns a value

in the range -PI/2 to PI/2.

### Examples

```
select atan(-1) from rdb$database
(returns -0,7853981633974 = -PI/4)

select atan(0) from rdb$database
(returns 0)

select atan(1) from rdb$database
(returns 0,7853981633974 = PI/4)
```

**See also:** COS(), SIN()

# ATAN2() [2.1]

Returns the arc tangent of the first number / the second number.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
ATAN2(<number1>,<number2>)
```

> **Important**
> The arguments to ATAN2 must be in the range -1 to 1.

| Argument | Description |
|---|---|
| <number1> | The first numeric expression whose arc tangent is returned |
| <number2> | The second numeric expression whose arc tangent is returned |

## Description

Returns the arc tangent of the first number / the second

number. Returns a value in the range -PI to PI.

## Examples

```
select atan2(1,1) from rdb$database
(returns 0,7853981633974 = PI/4)

select atan2(0,0) from rdb$database
(returns 0)
```

**See also:** COS(), SIN()

# AVG( )

Calculates the average of numeric values in a specified column or expression.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
AVG ([ALL] value | DISTINCT value)
```

(This text is currently not included because of possible copyright issues.)

**See also:** COUNT( ), MAX( ), MIN( ), SUM( )

## BASED ON

Declares a host-language variable based on a column.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
BASED [ON] [dbhandle.]table.col[.SEGMENT] variable;
```

(This text is currently not included because of possible copyright issues.)

**See also:** BEGIN DECLARE SECTION, CREATE TABLE, END DECLARE SECTION

## BEGIN DECLARE SECTION

Identifies the start of a host-language variable declaration section.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
BEGIN DECLARE SECTION;
```

(This text is currently not included because of possible copyright issues.)

**See also:** BASED ON, END DECLARE SECTION

## BIN_AND() [2.1]

Returns the result of a binary AND operation performed on all arguments.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
BIN_AND( <number>[, <number> ...] )
```

**Important**
<Notes>

| Argument | Description |
|---|---|
| <number> | The numbers that the binary AND operation is executed on |

## Description

## Examples

```
SELECT bin_and(1,3,7) from rdb$database
(returns 1)

SELECT bin_and(2,6,10) from rdb$database
(returns 2)
```

**See also:** BIN_OR(), BIN_XOR()

# *BIN_OR() [2.1]*

Returns the result of a binary OR operation performed on all arguments.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
BIN_OR( <number>[, <number> ...] )
```

**Important**
<Notes>

| Argument | Description |
|---|---|
| <number> | The numbers that the binary OR operation is executed on |

## Description

## Examples

```
SELECT bin_and(1,3,7) from rdb$database
(returns 7)

SELECT bin_or(2,6,10) from rdb$database
(returns 14)
```

**See also:** BIN_AND(), BIN_XOR()

# *BIN_SHL() [2.1]*

Returns the result of a binary shift left operation performed on the arguments (first << second).

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
BIN_SHL( <number1>,<number2> )
```

**Important**
<number2> must be >= 0.

| Argument | Description |
|---|---|
| <number1> | The number that gets binary shifted left |
| <number2> | How many bits to shift <number1> left |

## Description

## Examples

```
SELECT bin_shl(16,1) from rdb$database
(returns 32)

SELECT bin_shl(16,4) from rdb$database
(returns 256)
```

**See also:** BIN_SHR()

## *BIN_SHR() [2.1]*

Returns the result of a binary shift right operation performed on the arguments (first >> second).

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
BIN_SHL( <number1>,<number2> )
```

> **Important**
> <number2> must be >= 0.

| Argument | Description |
|---|---|
| <number1> | The number that gets binary shifted right |
| <number2> | How many bits to shift <number1> right |

### Description

### Examples

```
SELECT bin_shr(16,1) from rdb$database
(returns 8)

SELECT bin_shr(16,4) from rdb$database
(returns 1)

SELECT bin_shr(16,8) from rdb$database
(returns 0)
```

**See also:** BIN_SHL()

## BIN_XOR() [2.1]

Returns the result of a binary XOR operation performed on all arguments.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
BIN_OR( <number>[, <number> ...] )
```

> **Important**
> <Notes>

| Argument | Description |
|---|---|
| <number> | The numbers that the binary XOR operation is executed on |

### Description

### Examples

```
SELECT bin_xor(1,3,7) from rdb$database
(returns 5)

SELECT bin_xor(2,6,10) from rdb$database
(returns 14)
```

**See also:** BIN_AND(), BIN_OR()

## BIT_LENGTH / CHAR_LENGTH / CHARACTER_LENGTH / OCTET_LENGTH [2.0]

These functions will return information about the size of strings.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
BIT_LENGTH(<val>)
CHAR_LENGTH(<val>)
CHARACTER_LENGTH(<val>)
OCTET_LENGTH(<val>)
```

**Important**
If no TRIM() is applied to <val>, trailing blanks in <val> will add to the result (see example).

| Argument | Description |
|---|---|
| <val> | A column, constant, host-language variable, expression, function, or UDF that evaluates to a character datatype |

## Description

These three new functions will return information about the size of strings:

1. BIT_LENGTH returns the length of a string in bits

2. CHAR_LENGTH/CHARACTER_LENGTH returns the length of a string in characters

3. OCTET_LENGTH returns the length of a string in bytes

## Examples

```
select
  rdb$relation_name,
  char_length(rdb$relation_name),
  bit_length(trim(rdb$relation_name)),
  char_length(trim(rdb$relation_name))
  octet_length(trim(rdb$relation_name))
from rdb$relations;
```

**See also:**

# *CASE [1.5]*

Allow the result of a column to be determined by the outcome of a group of exclusive conditions.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
simple CASE:
CASE <search expression>
    WHEN <value expression> THEN <result expression>
  { WHEN <value expression> THEN <result expression> }
[ ELSE <result expression> ]

searched CASE:
CASE
    WHEN <search condition> THEN <result expression>
  { WHEN <search condition> THEN <result expression> }
[ ELSE <result expression> ]
```

**Important**
<Notes>

| Argument | Description |
|---|---|
| <search expression> | The expression to be examined by the CASE construct |
| <value expression> | a constant for this CASE branch |
| <search condition> | an expression that, if it evaluates to TRUE, gives the result in this WHEN branch |
| <result expression> | the result returned when this WHEN or ELSE branch matches |

## Description

Allow the result of a column to be determined by the outcome of a group of exclusive conditions.

There are two variations of the CASE construct: simple and searched.

In the simple CASE, an expression following the keyword CASE is evaluated and compared against the various values in the simple when clauses. The result given after THEN in the first matching WHEN argument is returned.

In the searched CASE, every WHEN clause holds an expression that gets evaluated. The result will be the argument following the WHEN clause for the first WHEN clause that evaluates to true.

There are three more variations to CASE:

- NULLIF is equivalent to CASE WHEN V1 = V2 THEN NULL ELSE V1 END
- COALESCE is equivalent to CASE WHEN V1 IS NOT NULL THEN V1 ELSE V2 END
- DECODE is an inline version of CASE implemented as a function call

### Examples

```
Simple example:
SELECT
  o.ID,
  o.Description,
  CASE o.Status
    WHEN 1 THEN 'confirmed'
    WHEN 2 THEN 'in production'
    WHEN 3 THEN 'ready'
    WHEN 4 THEN 'shipped'
  ELSE 'unknown status ''' || o.Status || ''''
  END
FROM Orders o;

Searched example:
SELECT
  o.ID,
  o.Description,
  CASE
    WHEN (o.Status IS NULL) THEN 'new'
    WHEN (o.Status = 1) THEN 'confirmed'
    WHEN (o.Status = 3) THEN 'in production'
    WHEN (o.Status = 4) THEN 'ready'
    WHEN (o.Status = 5) THEN 'shipped'
  ELSE 'unknown status ''' || o.Status || ''''
  END
FROM Orders o;
```

**See also:** COALESCE(), NULLIF(), DECODE(), IF()

## CAST( )

Converts a column from one datatype to another.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
CAST (value AS datatype)
```

(This text is currently not included because of possible copyright issues.)

**See also:** UPPER( )

## CEIL() / CEILING() [2.1]

Returns a value representing the smallest integer that is greater than or equal to the input argument.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
{ CEIL | CEILING }( <number> )
```

> **Important**
> <Notes>

| Argument | Description |
|---|---|
| <number> | the number whose next-greater integer value is returned |

## Description

Returns a value representing the smallest integer that

is greater than or equal to the input argument.

## Examples

```
select ceil(1.0) from rdb$database
(returns 1)

select ceil(1.1) from rdb$database
(returns 2)

select ceil(-1.1) from rdb$database
(returns -1)
```

**See also:** FLOOR(), ROUND()

# *CLOSE*

Closes an open cursor.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
CLOSE cursor;
```

(This text is currently not included because of possible copyright issues.)

**See also:** CLOSE (BLOB), COMMIT, DECLARE CURSOR, FETCH, OPEN, ROLLBACK

# CLOSE (BLOB)

Terminates a specified blob cursor and releases associated system resources.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
CLOSE blob_cursor;
```

(This text is currently not included because of possible copyright issues.)

**See also:** DECLARE CURSOR (BLOB), FETCH (BLOB), INSERT CURSOR (BLOB), OPEN (BLOB)

# COALESCE [1.5]

a shortcut for a CASE construct returning the first non-NULL value

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
COALESCE ( <value expression> { , <value expression> } )
```

> **Important**
> <Notes>

| Argument | Description |
|---|---|
| <value expression> | an expression to be evaluated |

## Description

Allows a column value to be calculated by a number of expressions, from which the first expression to return a non-NULL value is returned as the output value.

- COALESCE (V1, V2) is equivalent to the following case specification: CASE WHEN V1 IS NOT NULL THEN V1 ELSE V2 END
- COALESCE (V1, V2,..., Vn), for n >= 3, is equivalent to the following case specification: CASE WHEN V1 IS NOT NULL THEN V1 ELSE COALESCE (V2,...,Vn) END

### Examples

```
SELECT
  PROJ_NAME AS Projectname,
  COALESCE(e.FULL_NAME,'[< not assigned >]') AS Employeename
FROM
  PROJECT p
    LEFT JOIN EMPLOYEE e
ON (e.EMP_NO = p.TEAM_LEADER);

SELECT
  COALESCE(Phone,MobilePhone,'Unknown') AS "Phonenumber"
FROM
  Relations;
```

**See also:** CASE, NULLIF(), DECODE(), IIF()

## COLLATE (BLOB) [2.0]

(no contents yet)

## COLLATE [PSQL] [2.1]

(no contents yet)

## COMMENT [2.0]

Allows to specify comments on database metadata

**Availability:** +DSQL +ESQL +ISQL -PSQL

### Syntax

```
COMMENT ON DATABASE IS ( <comment> | NULL )
COMMENT ON COLUMN <tblviewname>.<fieldname> IS  ( <
```

```
comment> | NULL )
COMMENT ON PARAMETER <procname>.<paramname> IS  ( <
comment> | NULL )
COMMENT ON <basic_type> <name> IS  ( <comment> | NULL )
```

**Important**
An empty literal string '' will act as NULL.

| Argument | Description |
|---|---|
| <comment> | the comment: a literal string constant (not an expression!) |
| <tblviewname> | name of a table or view |
| <fieldname> | name of a column in a table or view |
| <procname> | name of a stored procedure |
| <paramname> | name of a parameter of a stored procedure |
| <basic_type> | can be DOMAIN, TABLE, VIEW, PROCEDURE, TRIGGER, |
| | EXTERNAL FUNCTION, FILTER, EXCEPTION, GENERATOR, SEQUENCE, INDEX, ROLE, |
| | CHARACTER SET or COLLATION |
| <name> | name of a metadata object of type <basic_type> |

## Description

This command provides a way to set the RDB$DESCRIPTION field in all of the

RDB$ system tables using a SQL command - that is, without the need to directly update the RDB$ tables

(which is not recommended).

It allows to comment or document any metadata object in a database.

## Examples

```
COMMENT ON DATABASE IS 'This is a Firebird database';
SELECT RDB$DESCRIPTION FROM RDB$DATABASE;

COMMENT ON SEQUENCE SEQ_ID_LOG IS 'generates new IDs for the LOG table';
SELECT RDB$DESCRIPTION FROM RDB$GENERATORS WHERE RDB$GENERATOR_NAME='
SEQ_ID_LOG';

COMMENT ON COLUMN LOG.ID IS 'primary key of the LOG table';
SELECT RDB$DESCRIPTION FROM RDB$RELATION_FIELDS
WHERE RDB$RELATION_NAME='LOG' AND RDB$FIELD_NAME='ID';
```

**See also:** RDB$ system tables

# COMMIT

Makes a transaction's changes to the database permanent, and ends the transaction.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
COMMIT [WORK] [TRANSACTION name] [RELEASE] [RETAIN [SNAPSHOT]];
```

(This text is currently not included because of possible copyright issues.)

**See also:** DISCONNECT, ROLLBACK, For more information about handling transactions, see Using Firebird- Transactions in Firebird (ch. 8 p. 90).

# CONNECT

Attaches to one or more databases.

**Availability:** DSQL ESQL ISQL* PSQL *A subset of CONNECT options is available in ISQL.

## Syntax

```
ISQL form:
CONNECT 'filespec' [USER 'username'][PASSWORD 'password']
 [CACHE int] [ROLE 'rolename']
ESQL form:
CONNECT [TO] {ALL | DEFAULT} <config_opts>
 | <db_specs> <config_opts> [, <db_specs> <
config_opts>...];
<db_specs> = dbhandle
 | {'filespec' | :variable} AS dbhandle
<config_opts> =  [USER {'username' | :variable}]
 [PASSWORD {'password' | :variable}]
 [ROLE {'rolename' | :variable}]
 [CACHE int [BUFFERS]]
```

(This text is currently not included because of possible copyright issues.)

**See also:** DISCONNECT, SET DATABASE, SET NAMES

**See also:** Using Firebird- Configuring the database cache (ch. 5 p. 67) for more information about cache buffers and Managing Security in ch. 22 of the same volume for more information about database security.

# COS() [2.1]

Returns the cosine of a number.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
COS(<number>)
```

> **Important**
> If <number> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | The number or numeric expression whose cosine is returned |

## Description

Returns the cosine of a number. The angle is specified

in radians and returns a value in the range -1 to 1.

## Examples

```
select cos(0) from rdb$database
(returns 1)

select cos(-1) from rdb$database
(returns 0,5403023058681)

select cos(1) from rdb$database
(returns 0,5403023058681)
```

**See also:** SIN()

# COSH() [2.1]

Returns the hyperbolic cosine of a number.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
COSH(<number>)
```

> **Important**
> If <number> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | The number or numeric expression whose hyperbolic cosine is returned |

## Description

Returns the hyperbolic cosine of a number. The angle is specified

in radians and returns a value in the range -1 to 1.

## Examples

```
select cosh(0) from rdb$database
(returns 1)

select cosh(-1) from rdb$database
(returns 1,5430806348152)

select cosh(1) from rdb$database
(returns 1,5430806348152)
```

**See also:** SIN(), COS()

# *COT() [2.1]*

Returns Returns 1 / tan(argument).

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
COT(<number>)
```

**Important**
If <number> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | The number or numeric expression whose cotangent is returned |

### Description

Returns the cotangent of a number. The angle is specified

in radians and returns a value in the range -1 to 1.

### Examples

```
select cot (0) from rdb$database
(returns INF)

select cot(-1) from rdb$database
(returns -0,6420926159343)

select cot(1) from rdb$database
(returns 0,6420926159343)
```

**See also:** SIN()

## COUNT( )

Calculates the number of rows that satisfy a query's search condition.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
COUNT ( * | [ALL] value | DISTINCT value)
```

(This text is currently not included because of possible copyright issues.)

**See also:** AVG( ), MAX( ), MIN( ) SUM( )

## CREATE COLLATION [2.1]

(no contents yet)

# CREATE DATABASE

Creates a new database

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
CREATE {DATABASE | SCHEMA} 'filespec'
  [USER 'username' [PASSWORD 'password']]
  [PAGE_SIZE [=] int]
  [LENGTH [=] int [PAGE[S]]]
  [DEFAULT CHARACTER SET charset]
  [<secondary_file>];
<secondary_file> = FILE 'filespec' [<fileinfo>] [<
secondary_file>]
<fileinfo> = {[LENGTH [=] int [PAGE[S]] | STARTING [AT [PAGE]] int }
    [<fileinfo>]
```

**Important**
In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in ISQL, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument | Description |
|---|---|
| 'filespec' | A new database file specification; file naming conventions are platform-specific. See "Creating a database" for details |
| | about database file specification. |
| USER | 'username' Checks the username against valid user name and password combinations in the security database on the server where the |
| | database will reside |
| | o Windows client applications must provide a user name on attachment to a server |
| | o Any client application attaching to a database on NT or NetWare must provide a user name on attachment |
| PASSWORD | 'password' Checks the password against valid user name and password combinations in the security database on the server |
| | where the database will reside; can be up to 8 characters |
| | o Windows client applications must provide a user name and password on attachment to a server |
| | o Any client application attaching to a database on NT or NetWare must |

| Argument | Description |
|---|---|
| | provide a password on attachment |
| PAGE_SIZE | [=] int Size, in bytes, for database pages |
| | int can be 1024, 2048, 4096 (default), 8192 or 16384. |
| | From Firebird 2.1 onward, 1024 and 2048 can not be used any more. |
| DEFAULT | CHARACTER SET charset Sets default character set for a database |
| | charset is the name of a character set; if omitted, character set defaults to NONE |
| FILE | 'filespec' Names one or more secondary files to hold database pages after the primary file is filled. |
| | For databases created on remote servers, secondary file specifications cannot include a node name. |
| STARTING | [AT [PAGE]] int Specifies the starting page number for a secondary file. |
| LENGTH | [=] int [PAGE[S]] Specifies the length of a primary or secondary database file. |
| | Use for primary file only if defining a secondary file in the same statement. |

## Description

CREATE DATABASE creates a new, empty database and establishes the following characteristics for it:

- The name of the primary file that identifies the database for users.

By default, databases are contained in single files.

- The name of any secondary files in which the database is stored.

A database can reside in more than one disk file if additional file names are specified as secondary files.

If a database is created on a remote server, secondary file specifications cannot include a node name.

- The size of database pages.

Increasing page size can improve performance for the following reasons:

o Indexes work faster because the depth of the index is kept to a minimum.

o Keeping large rows on a single page is more efficient.

o Blob data is stored and retrieved more efficiently when it fits on a single page.

If most transactions involve only a few rows of data, a smaller page size might be appropriate,

since less data needs to be passed back and forth and less memory is used by the disk cache.

- The number of pages in each database file.
- The dialect of the database.

The initial dialect of the database is the dialect of the client that creates it.

For example, if you are using ISQL, either start it with the -sql_dialect n switch or issue the SET SQL DIALECT n command

before issuing the CREATE DATABASE command. Typically, you would create all databases in dialect 3.

Dialect 1 exists to ease the migration of legacy databases.

NOTE: To change the dialect of a database, use the gfix tool.

- The character set used by the database.

For a list of the character sets recognized by Firebird, see Character sets and collations available in Firebird.

Choice of DEFAULT CHARACTER SET limits possible collation orders to a subset of all available collation orders.

Given a specific character set, a specific collation order can be specified when data is selected, inserted, or updated in a column.

If you do not specify a default character set, the character set defaults to NONE.

Using character set NONE means that there is no character set assumption for columns; data is stored and retrieved just as you originally entered it.

You can load any character set into a column defined with NONE, but you cannot load that same data into another column that has been defined with a

different character set. In that case, no transliteration is performed between the source and destination character sets, and transliteration

errors may occur during assignment.

- System tables that describe the structure of the database.

After creating the database, you define its tables, views, indexes, and system views as well as any triggers, generators, stored procedures,

and UDFs that you need.

**Important**
In DSQL, you must execute CREATE DATABASE EXECUTE IMMEDIATE. The database handle and transaction name, if present, must be initialized to zero prior to use. Read-only databases Databases are always created in read-write mode. You can change a database to read-only mode in either of two ways: You can specify mode -read_only when you restore a backup or you can use gfix -mode read_only to change the mode of a read-write database to read- only. About file sizes Firebird dynamically expands the last file in a database as needed until it reaches the filesystem limit for shared access files. This applies to single-file database as well as to the last file of multifile databases. It is important to be aware of the maximum size allowed for shared access files in the filesystem environment where your databases live. Firebird database files are limited to 2GB in many environments. The total file size is the product of the number of database pages times the page size. The default page size is 4KB and the maximum page size is 16KB. However, Firebird files are small at creation time and increase in size as needed. The product of number of pages times page size represents a potential maximum size, not the size at creation.

## Examples

```
The following ISQL statement creates a database in the default directory
using ISQL:
CREATE DATABASE 'employee.gdb';

The next ESQL statement creates a database with a page size of 2048 bytes
rather than the default of 4096:
EXEC SQL
    CREATE DATABASE 'employee.gdb' PAGE_SIZE 2048;

The following ESQL statement creates a database stored in two files and
specifies its default character set:
EXEC SQL
    CREATE DATABASE 'employee.gdb'
        DEFAULT CHARACTER SET ISO8859_1
        FILE 'employee2.gdb' STARTING AT PAGE 10001;
```

**See also:** ALTER DATABASE, DROP DATABASE

**See also:** the following topics: o Multi-file databases o Character Sets and Collation Orders o Specifying database page size

# *CREATE DOMAIN*

Creates a column definition that is global to the database.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
CREATE DOMAIN domain [AS] <datatype>
   [DEFAULT {literal | NULL | USER}]
   [NOT NULL] [CHECK (<dom_search_condition>)]
   [COLLATE collation];
<datatype> =
```

```
 {SMALLINT|INTEGER|FLOAT|DOUBLE PRECISION}  [<array_dim>]
 | {DATE | TIME | TIMESTAMP}  [<array_dim>]
 | {DECIMAL | NUMERIC} [(precision [, scale])] [<array_dim>]
 | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(int)]
  [<array_dim>] [CHARACTER SET charname]
 | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR}
  [VARYING] [(int)] [<array_dim>]
 | BLOB [SUB_TYPE {int | subtype_name}] [SEGMENT SIZE int]
  [CHARACTER SET charname]
 | BLOB [(seglen [, subtype])]
<array_dim> = [[x:]y [, [x:]y ...]]
<dom_search_condition> =
 VALUE <operator> value
 | VALUE [NOT] BETWEEN value AND value
 | VALUE [NOT] LIKE value [ESCAPE value]
 | VALUE [NOT] IN (value [, value ...])
 | VALUE IS [NOT] NULL
 | VALUE [NOT] CONTAINING value
 | VALUE [NOT] STARTING [WITH] value
 | (<dom_search_condition>)
 | NOT <dom_search_condition>
 | <dom_search_condition> OR <dom_search_condition>
 | <dom_search_condition> AND <dom_search_condition>
<operator> = {= | < | > | <= | >= | !< | !> | <
> | !=}
```

(This text is currently not included because of possible copyright issues.)

**Note1:** Be careful not to create a domain with contradictory constraints, such as declaring a domain NOT NULL and assigning it a DEFAULT value of NULL. The datatype specification for a CHAR or VARCHAR text domain definition can include a CHARACTER SET clause to specify a character set for the domain. Otherwise, the domain uses the default database character set. For a complete list of character sets recognized by Firebird, see chapter 4, Character Sets and Collation Orders (p. 249). If you do not specify a default character set, the character set defaults to NONE. Using character set NONE means that there is no character set assumption for columns; data is stored and retrieved just as you originally entered it. You can load any character set into a column defined with NONE, but you cannot load that same data into another column that has been defined with a different character set. In these cases, no transliteration is performed between the source and destination character sets, so errors can occur during assignment. The COLLATE clause enables specification of a particular collation order for CHAR, VARCHAR, and NCHAR text datatypes. Choice of collation order is restricted to those supported for the domain's given character set, which is either the default character set for the entire database, or a different set defined in the CHARACTER SET clause as part of the datatype definition. For a complete list of collation orders recognized by Firebird, see chapter 4, Character Sets and Collation Orders (p. 249). Columns based on a domain definition inherit all characteristics of the domain. The domain default, collation clause, and NOT NULL setting can be overridden when defining a column based on a domain. A column based on a domain can add additional CHECK constraints to the domain CHECK constraint.

**See also:** ALTER DOMAIN, ALTER TABLE, CREATE TABLE, DROP DOMAIN, For more information about character set specification and collation orders, see xxxxx.

## CREATE EXCEPTION

Creates a used-defined error and associated message for use in stored procedures and triggers.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
CREATE EXCEPTION name 'message';
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER EXCEPTION, ALTER PROCEDURE, ALTER TRIGGER, CREATE PROCEDURE, CREATE TRIGGER, DROP EXCEPTION, For more information on creating, raising, and handling exceptions, see the Using Firebird- Error trapping and handling (ch. 25 p. 549).

## CREATE GENERATOR

Declares a generator to the database.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
CREATE GENERATOR name;
```

(This text is currently not included because of possible copyright issues.)

**See also:** GEN_ID( ), SET GENERATOR, DROP GENERATOR

## CREATE GLOBAL TEMPORARY TABLE [2.1]

(no contents yet)

## CREATE INDEX

Creates an index on one or more columns in a table.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]] INDEX index
   ON table (col [, col ...]);
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER INDEX, DROP INDEX, SELECT, SET STATISTICS

## CREATE INDEX COMPUTED BY [2.0]

(no contents yet)

## CREATE OR ALTER EXCEPTION [2.0]

(no contents yet)

## CREATE OR ALTER {TRIGGER | PROCEDURE } [1.5]

(no contents yet)

## CREATE PROCEDURE

Creates a stored procedure, its input and output parameters, and its actions.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
CREATE PROCEDURE name
   [(param <pdatatype> [, param <pdatatype> ...])]
   [RETURNS param <pdatatype> [, param <pdatatype> ...])]
   AS <procedure_body> [terminator]
<pdatatype> = BLOB | <datatype>

<procedure_body> =
 [<variable_declaration_list>]
 <block>
<variable_declaration_list> =
 DECLARE VARIABLE var <datatype>;
 [DECLARE VARIABLE var <datatype>; ...]
<block> =
BEGIN
 <compound_statement>
 [<compound_statement> ...]
END
<compound_statement> = <block> | statement;
<datatype> =  SMALLINT
 | INTEGER
 | FLOAT
 | DOUBLE PRECISION
 | {DECIMAL | NUMERIC} [(precision [, scale])]
 | {DATE | TIME | TIMESTAMP)
```

```
 | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR}
  [(int)]  [CHARACTER SET charname]
 | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(int)]
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER EXCEPTION, ALTER PROCEDURE, CREATE EXCEPTION, DROP EXCEPTION, DROP PROCEDURE, EXECUTE PROCEDURE, SELECT, For more information on creating and using procedures, see Using Firebird- Programming on Firebird Server (ch. 25 p. 494)., For a complete description of the statements in procedure and trigger language, see chapter 3, PSQL-Firebird Procedural Language (p. 222).

# CREATE ROLE

Creates a role.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
CREATE ROLE rolename;
```

(This text is currently not included because of possible copyright issues.)

**See also:** GRANT, REVOKE, DROP ROLE

# CREATE SEQUENCE [2.0]

Creates an integer number generator using SQL-99-compliant syntax

**Availability:** +DSQL -ESQL +ISQL -PSQL

## Syntax

```
  CREATE ( SEQUENCE | GENERATOR ) <name>
```

**Important**
<Notes>

| Argument | Description |
|---|---|
| <name> | The name for the new generator / sequence |

## Description

SEQUENCE is the SQL-99-compliant synonym for GENERATOR.

SEQUENCE is a syntax term described in the SQL specification, whereas GENERATOR is a legacy InterBase syntax term.

It is recommended to use the standard SEQUENCE syntax.

A sequence generator is a mechanism for generating successive exact numeric values, one at a time. A sequence

generator is a named schema object. In dialect 3 it is a BIGINT, in dialect 1 it is an INTEGER.

It is often used to implement guaranteed unique IDs for records, to construct

columns that behave like AUTOINC fields found in other RDBMSs.

## Examples

```
CREATE SEQUENCE SEQ_ID_EMPLOYEE;
```

**See also:** CREATE GENERATOR, NEXT VALUE FOR, DROP SEQUENCE, ALTER SEQUENCE, CREATE TRIGGER

> **Tip**
> For a complete discussion on the concept and useage of sequences / generators, see the "Generator Guide" that is available as part of the Firebird documentation set.

# *CREATE SHADOW*

Creates one or more duplicate, in-sync copies of a database.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
CREATE SHADOW set_num [AUTO | MANUAL] [CONDITIONAL]
   'filespec' [LENGTH [=] int [PAGE[S]]]
   [<secondary_file>];
<secondary_file> = FILE 'filespec' [<fileinfo>] [<
secondary_file>]
<fileinfo> = LENGTH [=] int [PAGE[S]] | STARTING [AT [PAGE]] int
 [<fileinfo>]
```

(This text is currently not included because of possible copyright issues.)

**See also:** DROP SHADOW

**See also:** Using Firebird- Database shadows (ch. 20 p. 379).

# CREATE TABLE

Creates a new table in an existing database.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
CREATE TABLE table [EXTERNAL [FILE] 'filespec']
   (<col_def> [, <col_def> | <tconstraint> ...]);
<col_def> = col {<datatype> | COMPUTED [BY] (<expr>) |
domain}
 [DEFAULT {literal | NULL | USER}]
 [NOT NULL]
 [<col_constraint>]
 [COLLATE collation]
<datatype> =
 {SMALLINT | INTEGER | FLOAT | DOUBLE PRECISION}[<array_dim>]
 | (DATE | TIME | TIMESTAMP}  [<array_dim>]
 | {DECIMAL | NUMERIC} [(precision [, scale])] [<array_dim>]
 | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(int)]
  [<array_dim>] [CHARACTER SET charname]
 | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR}
  [VARYING] [(int)] [<array_dim>]
 | BLOB [SUB_TYPE {int | subtype_name}] [SEGMENT SIZE int]
  [CHARACTER SET charname]
 | BLOB [(seglen [, subtype])]
<array_dim> = [[x:]y [, [x:]y ...]]
<expr> = a valid SQL expression that results in a single value
<col_constraint> = [CONSTRAINT constraint]
 { UNIQUE
 | PRIMARY KEY
 | REFERENCES other_table [(other_col [, other_col ...])]
  [ON DELETE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
  [ON UPDATE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
 | CHECK (<search_condition>)}
<tconstraint> = [CONSTRAINT constraint]
 {{PRIMARY KEY | UNIQUE} (col [, col ...])
 | FOREIGN KEY (col [, col ...])
  REFERENCES other_table [(other_col [, other_col ...])]
   [ON DELETE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
   [ON UPDATE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
 | CHECK (<search_condition>)}
<search_condition> = <val> <operator> {<val> | (<
    select_one>)}
 | <val> [NOT] BETWEEN <val> AND <val>
 | <val> [NOT] LIKE <val> [ESCAPE <val>]
 | <val> [NOT] IN (<val> [, <val> ...] | <
select_list>)
 | <val> IS [NOT] NULL
 | <val> {>= | <=} <val>
 | <val> [NOT] {= | < | >} <val>
```

```
  | {ALL | SOME | ANY} (<select_list>)
  | EXISTS (<select_expr>)
  | SINGULAR (<select_expr>)
  | <val> [NOT] CONTAINING <val>
  | <val> [NOT] STARTING [WITH] <val>
  | (<search_condition>)
  | NOT <search_condition>
  | <search_condition> OR <search_condition>
  | <search_condition> AND <search_condition>
<val> = { col [<array_dim>] | :variable
  | <constant> | <expr> | <function>
  | udf ([<val> [, <val> ...]])
  | NULL | USER | RDB$DB_KEY | ? }
  [COLLATE collation]
<constant> = num | 'string' | _charsetname 'string'
<function> = COUNT (* | [ALL] <val> | DISTINCT <val>)
  | SUM ([ALL] <val> | DISTINCT <val>)
  | AVG ([ALL] <val> | DISTINCT <val>)
  | MAX ([ALL] <val> | DISTINCT <val>)
  | MIN ([ALL] <val> | DISTINCT <val>)
  | CAST (<val> AS <datatype>)
  | UPPER (<val>)
  | GEN_ID (generator, <val>)
<operator> = {= | < | > | <= | >= | !< | !> | <
> | !=}
<select_one> = SELECT on a single column; returns exactly one value.
<select_list> = SELECT on a single column; returns zero or more
values.
<select_expr> = SELECT on a list of values; returns zero or more
values.
```

(This text is currently not included because of possible copyright issues.)

**Note1:** Constraints are not enforced on expressions.

**See also:** CREATE DOMAIN, DECLARE TABLE, GRANT, REVOKE, For more information, refer to Using Firebird- Tables (ch. 17 p. 313) and Managing Security in ch. 22 of the same volume.

# CREATE TRIGGER

Creates a trigger, including when it fires, and what actions it performs.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
CREATE TRIGGER name FOR table
   [ACTIVE | INACTIVE]
   {BEFORE | AFTER}
   {DELETE | INSERT | UPDATE}
   [POSITION number]
   AS <trigger_body> terminator
<trigger_body> = [<variable_declaration_list>] <block>
<variable_declaration_list> =
 DECLARE VARIABLE variable <datatype>;
 [DECLARE VARIABLE variable <datatype>; ...]
```

```
<block> =
BEGIN
 <compound_statement>
 [<compound_statement> ...]
END
<datatype> =  SMALLINT
 | INTEGER
 | FLOAT
 | DOUBLE PRECISION
 | {DECIMAL | NUMERIC} [(precision [, scale])]
 | {DATE | TIME | TIMESTAMP)
 | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR}
  [(int)]  [CHARACTER SET charname]
 | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(int)]
<compound_statement> = <block> | statement;
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER EXCEPTION, ALTER TRIGGER, CREATE EXCEPTION, CREATE PROCEDURE, DROP EXCEPTION, DROP TRIGGER, EXECUTE PROCEDURE, For a complete description of each statement, see chapter 3, PSQL-Firebird Procedural Language (p. 222)., For discussion of programming triggers, see Triggers, Coding the body of the code module and Implementing stored procedures and triggers in Using Firebird- Programming on Firebird Server (ch. 25 p. 494).

## CREATE TRIGGER ON CONNECT [2.1]

(no contents yet)

## CREATE TRIGGER ON DISCONNECT [2.1]

(no contents yet)

## CREATE TRIGGER ON TRANSACTION COMMIT [2.1]

(no contents yet)

## CREATE TRIGGER ON TRANSACTION ROLLBACK [2.1]

(no contents yet)

## CREATE TRIGGER ON TRANSACTION START [2.1]

(no contents yet)

## CREATE VIEW

Creates a new view of data from one or more tables.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
CREATE VIEW name [(view_col [, view_col ...])]
   AS <select> [WITH CHECK OPTION];
```

(This text is currently not included because of possible copyright issues.)

**Note1:** Although it is possible to create a view based on the output of a selectable stored procedure, it adds an unnecessary layer of dependency to do so. Using the output set of a stored procedure joined to a table, another view or another stored procedure is also theoretically possible but , in practice, it causes more trouble than it saves. With such complex requirements, it is almost invariably best to define the entire output within a selectable stored procedure. A view is updatable if: o It is a subset of a single table or another updatable view. o All base table columns excluded from the view definition allow NULL values. o The view's SELECT statement does not contain subqueries, a DISTINCT predicate, a HAVING clause, aggregate functions, joined tables, user- defined functions, or stored procedures. If the view definition does not meet these conditions, it is considered read-only.

**Note2:** Read-only views can be updated by using a combination of user-defined referential constraints, triggers, and unique indexes.

**See also:** CREATE TABLE, DROP VIEW, GRANT, INSERT, REVOKE, SELECT, UPDATE, For a complete discussion, see Using Firebird- Views (ch. 19 p. 363).

## *CREATE VIEW [with column alias] [2.1]*

(no contents yet)

## *CROSS JOIN [2.0]*

(no contents yet)

## *CURRENT_CONNECTION [1.5]*

context variable that holds the system ID of the current connection

**Availability:** +DSQL +ESQL +ISQL +PSQL

### Syntax

```
CURRENT_TRANSACTION
```

**Important**
Because the counter for this value is stored on the database header page , it will be reset after a database restore.

| Argument | Description |
|---|---|
| CURRENT_CONNECTION | returns the system identifier of the current connection |

## Description

This context variable holds the current connection's system ID (data type INTEGER). It can be used for e.g. logging purposes.

Every new connection that is made will receive a new, unique connection ID.

In the monitoring tables (V2.1 and up), the value of CURRENT_CONNECTION corresponds

to the field MON$ATTACHMENT_ID in MON$ATTACHMENTS, MON$TRANSACTIONS and MON$STATEMENTS.

**Note**
An active connection with a specific CURRENT_CONNECTION number will always correspond with one record in the MON$ATTACHMENTS table (but can have several associated transaction records in MON$TRANSACTIONS)

## Examples

```
Obtain the current connection ID in a trigger:
NEW.CON_ID = CURRENT_CONNECTION;

List all transactions that are bound to the current connection:
(V2.1 and up):
SELECT * FROM MON$TRANSACTIONS WHERE MON$ATTACHMENT_ID=CURRENT_CONNECTION

List all statements that are executed within the current connection context
,
even if they use different transactions (V2.1 and up):
SELECT * FROM MON$STATEMENTS WHERE MON$ATTACHMENT_ID=CURRENT_CONNECTION
```

**See also:** CURRENT_TRANSACTION, CURRENT_USER, CURRENT_ROLE

# *CURRENT_ROLE [1.5]*

Context variable returning the current SQL user's role

**Availability:** +DSQL +ESQL +ISQL +PSQL

## Syntax

```
CURRENT_ROLE
```

> **Important**
> <Notes>

| Argument | Description |
|---|---|
| CURRENT_ROLE | returns the name of the role of the current SQL user (if any) |

## Description

Returns the name of the role the current user logged in with (see also CURRENT_USER). If no role was

specified, it returns "NONE".

1. If you insist on using an InterBase v.4.x or 5.1 database with Firebird, ROLE is not supported, so current_role will be NONE (as mandated by the SQL standard in absence of an explicit role) even if the user passed a role name.

2. If you use IB 5.5, IB 6 or Firebird, the ROLE passed is verified. If the role does not exist, it is reset to NONE without returning an error.

This means that in FB you can never get an invalid ROLE returned by CURRENT_ROLE, because

it will be reset to NONE. This is in contrast with IB, where the bogus value is carried internally,

although it is not visible to SQL.

## Examples

```
SELECT CURRENT_ROLE FROM RDB$DATABASE

INSERT INTO RoleLog (ID, USERNAME)
VALUES (NEXT VALUE FOR SEQ_ID_ROLELOG, CURRENT_ROLE)
```

**See also:** CURRENT_USER, CURRENT_TRANSACTION, CURRENT_CONNECTION

# CURRENT_TRANSACTION [1.5]

context variable that holds the system ID of the current transaction

**Availability:** +DSQL +ESQL +ISQL +PSQL

## Syntax

```
CURRENT_TRANSACTION
```

> **Important**
> Because the counter for this value is stored on the database header page , it will be reset after a database restore.

| Argument | Description |
|---|---|
| CURRENT_TRANSACTION | returns the system identifier of the current transaction |

## Description

This context variable holds the current transaction's system ID (data type INTEGER). It can be used for e.g. logging purposes.

Every new transaction that is started will receive a new, unique transaction ID.

In the monitoring tables (V2.1 and up), the value of CURRENT_TRANSACTION corresponds to the fields MON$TRANSACTIONS.MON$TRANSACTION_ID and MON$STATEMENTS.MON$TRANSACTION_ID.

## Examples

```
Obtain the current transaction ID in a trigger:
NEW.TXN_ID = CURRENT_TRANSACTION;

List all statements that are executed within the current transaction
(V2.1 and up):
SELECT * FROM MON$STATEMENTS WHERE MON$TRANSACTION_ID=CURRENT_TRANSACTION
```

**See also:** CURRENT_CONNECTION, CURRENT_USER, CURRENT_ROLE

# CURRENT_USER [1.5]

Context variable returning the SQL user name

**Availability:** +DSQL +ESQL +ISQL +PSQL

### Syntax

```
CURRENT_USER
```

> **Important**
> <Notes>

| Argument | Description |
|---|---|
| CURRENT_USER | returns the name of the current SQL user |

### Description

CURRENT_USER is a DSQL synonym for USER that appears in the SQL standard. They are identical. There is no advantage of CURRENT_USER over USER.

### Examples

```
SELECT CURRENT_USER FROM RDB$DATABASE

INSERT INTO UserLog (ID, USERNAME)
VALUES (NEXT VALUE FOR SEQ_ID_USERLOG, CURRENT_USER)
```

**See also:** CURRENT_ROLE, CURRENT_TRANSACTION, CURRENT_CONNECTION

## CURSOR FOR [2.0]

(no contents yet)

## DATEADD() [2.1]

Returns a date/time/timestamp value increased (or decreased, when negative) by the specified amount of time.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
DATEADD( <number> <timestamp_part> FOR <date_time> )
DATEADD( <timestamp_part>, <number>, <date_time> )
timestamp_part ::= { YEAR | MONTH | DAY | WEEKDAY | HOUR | MINUTE | SECOND
}
```

**Important**
If any of the arguments is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <date_time> | The starting date, time or timestamp for the calculation |
| <number> | The offset to be added to <date_time> |
| <timestamp_part> | The unit for <number> |

## Description

Returns a date/time/timestamp value increased (or

decreased, when negative) by the specified amount

of time.

## Examples

```
select dateadd(1 day for current_date) from rdb$database
(returns tomorrow's date)

select dateadd(-1 day for current_date) from rdb$database
(returns yesterday's date)

select dateadd(weekday,1,current_date) from rdb$database
(returns the date of today's weekday in the next week)

select dateadd(weekday,1,current_timestamp) from rdb$database
(returns the timestamp of today's weekday in the next week with the current
time)
```

**See also:** DATEDIFF()

# *DATEDIFF() [2.1]*

Returns the interval between two dates/times/timestamps

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
DATEDIFF( <timestamp_part> FROM <date_time1> FOR <
date_time2> )
DATEDIFF( <timestamp_part>, <date_time1>, <date_time2> )
timestamp_part ::= { YEAR | MONTH | DAY | WEEKDAY | HOUR | MINUTE | SECOND
}
```

**Important**
If any of the arguments is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <date_time1> | The first date, time or timestamp for the calculation |
| <date_time2> | The second date, time or timestamp for the calculation |
| <timestamp_part> | The unit for <number> |

## Description

Returns an exact numeric value representing the interval

of time from the first date/time/timestamp value

to the second one.

**Rules:** 1. Returns a positive value if the second value is greater than the first one, negative when the first one is greater, or zero when they are equal. 2. Comparison of date with time values is invalid. 3. YEAR, MONTH, DAY and WEEKDAY cannot be used with time values. 4. HOUR, MINUTE and SECOND cannot be used with date values. 5. All timestamp_part values can be used with timestamp values.

## Examples

```
select datediff(SECOND,cast(current_date as timestamp),current_timestamp)
from rdb$database
(returns the number of seconds elapsed since midnight. The CAST is
necessayr because of Rule 2)

select datediff(DAY,dateadd(1 weekday for current_date),current_date) from
rdb$database
(returns -7)
```

```
select datediff(SECOND,current_time,current_time) from rdb$database
(returns 0)

select datediff(SECOND,current_date,current_date) from rdb$database
(throws an error because of Rule 5, returns NULL)
```

**See also:** DATEADD()

# DECLARE CURSOR

Defines a cursor for a table by associating a name with the set of rows specified in a SELECT statement.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
(ESQL only):
DECLARE cursor CURSOR FOR <select> [FOR UPDATE OF <col> [, <
col>...]];
Blob form: See DECLARE CURSOR (BLOB)
```

(This text is currently not included because of possible copyright issues.)

**See also:** CLOSE, DECLARE CURSOR (BLOB), FETCH, OPEN, PREPARE, SELECT

# DECLARE CURSOR (BLOB)

Declares a blob cursor for read or insert.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
DECLARE cursor CURSOR FOR
   {READ BLOB column FROM table
   | INSERT BLOB column INTO table}
   [FILTER [FROM subtype] TO subtype]
   [MAXIMUM_SEGMENT length];
```

(This text is currently not included because of possible copyright issues.)

**See also:** CLOSE (BLOB), FETCH (BLOB), INSERT CURSOR (BLOB), OPEN (BLOB)

# DECLARE EXTERNAL FUNCTION

Declares an existing user-defined function (UDF) to a database.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
DECLARE EXTERNAL FUNCTION name [datatype | CSTRING (int)
    [, datatype | CSTRING (int) ...]]
   RETURNS {datatype [BY VALUE] | [BY DESCRIPTOR] | CSTRING (int)}
    | PARAMETER int_pos} [FREE_IT]
   ENTRY_POINT 'entryname'
   MODULE_NAME 'modulename';
```

(This text is currently not included because of possible copyright issues.)

**Note1:** that beginning with Firebird 1, you must list the path in the Firebird configuration file if it is other than ib_install_dir/UDF. A path name is no longer useful in the DECLARE EXTERNAL FUNCTION statement. The Firebird configuration file is called ibconfig on Windows machines, isc_config on Linux/UNIX machines.

**See also:** DROP EXTERNAL FUNCTION, For more information about writing and using UDFs, see Using Firebird- Working with UDFs and Blob Filters (ch. 26 p. 572)., For declarations of the UDFs in the ib_udf and FBUDF libraries, see User-defined Functions on page 257 in chapter 6.

# DECLARE FILTER

Declares an existing blob filter to a database.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
DECLARE FILTER filter
   INPUT_TYPE subtype OUTPUT_TYPE subtype
   ENTRY_POINT 'entryname' MODULE_NAME 'modulename';
```

(This text is currently not included because of possible copyright issues.)

**See also:** DROP FILTER, For more information about BLOB subtypes and instructions on writing blob filters, see Using Firebird- BLOB filters (ch. 26 p. 596) and associated topics in that section.

# DECLARE STATEMENT

Identifies dynamic SQL statements before they are prepared and executed in an embedded program.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
DECLARE <statement> STATEMENT;
```

(This text is currently not included because of possible copyright issues.)

**See also:** EXECUTE, EXECUTE IMMEDIATE, PREPARE

## DECLARE TABLE

Describes the structure of a table to the preprocessor, gpre, before it is created with CREATE TABLE.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
DECLARE table TABLE (<table_def>);
```

(This text is currently not included because of possible copyright issues.)

**See also:** CREATE DOMAIN, CREATE TABLE

## DECODE() [2.1]

a shortcut for a CASE ... WHEN ... ELSE expression.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
   DECODE( <expression>, <search>, <result>[ , <
search>, <result> ... ] [, <default> ] )
```

> **Important**
> <Notes>

| Argument | Description |
|---|---|
| <expression> | The expression to decode |
| <search> | a possible match for <expression> |
| <result> | the value returned when <expression> matches the preceeding <search> value |
| <default> | the value returned when none of the <search> values matched <expression> |

## Description

DECODE is an inline version of a CASE ... WHEN ... ELSE construct.

## Examples

```
select decode(state, 0, 'deleted', 1, 'active', 'unknown') from x
(returns 'deleted' when state equals 0, 'active' when state equals 1 and
otherwise returns 'unknown')

select decode(rdb$system_flag,1,'SYSTEM',0,'USER','unknown') from rdb$
triggers
(returns 'SYSTEM' for system triggers and 'USER' for user-defined ones.
```

> **Note**
> the output column's name is 'CASE'.)

**See also:** CASE

# DELETE

Removes rows in a table or in the active set of a cursor.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
ESQL and DSQL form:
```

(This text is currently not included because of possible copyright issues.)

**See also:** DECLARE CURSOR, FETCH, GRANT, OPEN, REVOKE, SELECT, For more informa-

tion about using cursors, see the Embedded SQL Guide (EmbedSQL.pdf) of the InterBase(R) 6 documentation set, obtainable from Borland.

# DESCRIBE

Provides information about columns that are retrieved by a dynamic SQL (DSQL) statement, or information about dynamic parameters that statement passes.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
DESCRIBE [OUTPUT | INPUT] statement
   {INTO | USING} SQL DESCRIPTOR xsqlda;
```

(This text is currently not included because of possible copyright issues.)

**See also:** EXECUTE, EXECUTE IMMEDIATE, PREPARE, For more information about ESQL programming and the XSQLDA descriptor, see the Embedded SQL Guide of the InterBase(R) 6 documentation set, available from Borland.

# DISCONNECT

Detaches an application from a database.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
DISCONNECT {{ALL | DEFAULT} | dbhandle [, dbhandle] ...]};
```

(This text is currently not included because of possible copyright issues.)

**See also:** COMMIT, CONNECT, ROLLBACK, SET DATABASE

# DROP DATABASE

Deletes the currently attached database.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
DROP DATABASE;
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER DATABASE, CREATE DATABASE

## DROP DEFAULT [2.0]

(no contents yet)

## DROP DOMAIN

Deletes a domain from a database.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
DROP DOMAIN name;
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER DOMAIN, ALTER TABLE, CREATE DOMAIN

## DROP EXCEPTION

Deletes an exception from a database.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
DROP EXCEPTION name
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER EXCEPTION, ALTER PROCEDURE, ALTER TRIGGER, CREATE EXCEPTION, CREATE PROCEDURE, CREATE TRIGGER

# DROP EXTERNAL FUNCTION

Removes a user-defined function (UDF) declaration from a database.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
DROP EXTERNAL FUNCTION name;
```

(This text is currently not included because of possible copyright issues.)

**See also:** DECLARE EXTERNAL FUNCTION

# DROP FILTER

Removes a blob filter declaration from a database.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
DROP FILTER name;
```

(This text is currently not included because of possible copyright issues.)

**See also:** DECLARE FILTER

# DROP GENERATOR

Removes a generator from a database.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
DROP EXTERNAL FUNCTION name;
```

(This text is currently not included because of possible copyright issues.)

**See also:** CREATE GENERATOR

## DROP GENERATOR revisited [1.5]

(no contents yet)

## DROP INDEX

Removes an index from a database.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
DROP INDEX name;
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER INDEX, CREATE INDEX, For more information about integrity constraints and system-defined indexes, see Using Firebird- Tables (ch. 17 p. 313)., For a discussion of indexing and related issues, see Indexes in ch. 18 of the same volume.

## DROP PROCEDURE

Deletes an existing stored procedure from a database.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
DROP PROCEDURE name
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER PROCEDURE, CREATE PROCEDURE, EXECUTE PROCEDURE

## DROP ROLE

Deletes a role from a database.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
DROP ROLE rolename;
```

(This text is currently not included because of possible copyright issues.)

**See also:** CREATE ROLE, GRANT, REVOKE

## *DROP SEQUENCE [2.0]*

Removes a sequence or generator from a database

**Availability:** +DSQL +ESQL +ISQL -PSQL

### Syntax

```
DROP SEQUENCE <name>
```

> **Important**
> It is not possible to drop a sequence when it is used by e.g. a trigger. You can query the RDB$DEPENCIES table, column RDB$DEPENDED_ON_NAME, to find out what triggers and / or stored procedures use a sequence.

| Argument | Description |
|---|---|
| <name> | name of the sequence / generator to be dropped |

### Description

To remove a sequence from a database, use DROP SEQUENCE.

This command is equivalent to DROP GENERATOR, but uses the SQL-99-compliant

SEQUENCE syntax. It is therefor recommended to use this syntax instead of DROP GENERATOR.

### Examples

```
DROP SEQUENCE SEQ_ID_EMPLOYEE;
```

**See also:** DROP GENERATOR, CREATE SEQUENCE, ALTER SEQUENCE, NEXT VALUE FOR

## DROP SHADOW

Deletes a shadow from a database.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
DROP SHADOW set_num;
```

(This text is currently not included because of possible copyright issues.)

**See also:** CREATE SHADOW

## DROP TABLE

Removes a table from a database.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
DROP TABLE name;
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER TABLE, CREATE TABLE

## DROP TRIGGER

Deletes an existing user-defined trigger from a database.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```

```

```
DROP TRIGGER name
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER TRIGGER, CREATE TRIGGER

## DROP VIEW

Removes a view definition from the database.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
DROP VIEW name;
```

(This text is currently not included because of possible copyright issues.)

**See also:** CREATE VIEW

## END DECLARE SECTION

Identifies the end of a host-language variable declaration section.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
END DECLARE SECTION;
```

(This text is currently not included because of possible copyright issues.)

**See also:** BASED ON, BEGIN DECLARE SECTION

## EVENT INIT

Registers interest in one or more events with the Firebird event manager.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
EVENT INIT request_name [dbhandle]
   ('string' | :variable [, 'string' | :variable ...]);
```

(This text is currently not included because of possible copyright issues.)

**See also:** CREATE PROCEDURE, CREATE TRIGGER, EVENT WAIT, SET DATABASE, For more information about events, see How events work, Handling events on a client and related topics in Using Firebird- Programming on Firebird Server (ch. 25 p. 494).

## EVENT WAIT

Causes an application to wait until notified of an event's occurrence.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
EVENT WAIT request_name;
```

(This text is currently not included because of possible copyright issues.)

**See also:** EVENT INIT, For more information about events, see How events work, Handling events on a client and related topics in Using Firebird- Programming on Firebird Server (ch. 25 p. 494).

## EXECUTE

Executes a previously prepared dynamic SQL (DSQL) statement.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
EXECUTE [TRANSACTION transaction] statement
   [USING SQL DESCRIPTOR xsqlda] [INTO SQL DESCRIPTOR xsqlda];
```

(This text is currently not included because of possible copyright issues.)

**See also:** DESCRIBE, EXECUTE IMMEDIATE, PREPARE, For more information about ESQL programming and the XSQLDA, see the Embedded SQL Guide (EmbedSQL.pdf) available from Bor-

land.

## EXECUTE BLOCK [2.0]

(no contents yet)

## EXECUTE IMMEDIATE

Prepares a dynamic SQL (DSQL) statement, executes it once, and discards it.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
EXECUTE IMMEDIATE [TRANSACTION transaction]
   {:variable | 'string'} [USING SQL DESCRIPTOR xsqlda];
```

(This text is currently not included because of possible copyright issues.)

**See also:** DESCRIBE, EXECUTE IMMEDIATE, PREPARE, For more information about ESQL programming and the XSQLDA, see the Embedded SQL Guide.

## EXECUTE PROCEDURE

Calls a stored procedure.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
ESQL form:
EXECUTE PROCEDURE [TRANSACTION transaction]
 name [:param [[INDICATOR]:indicator]]
  [, :param [[INDICATOR]:indicator] ...]
 [RETURNING_VALUES :param [[INDICATOR]:indicator]
  [, :param [[INDICATOR]:indicator] ...]];
DSQL form:
EXECUTE PROCEDURE TRANSACTION transaction
 name [param [, param ...]]
  [RETURNING_VALUES param [, param ...]]

ISQL form:
EXECUTE PROCEDURE name [param [, param ...]]
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER PROCEDURE, CREATE PROCEDURE, DROP PROCEDURE, For more information about indicator variables, see the Embedded SQL Guide (EmbedSQL.pdf) from the InterBase(R) 6 documentation set, available from Borland.

## EXECUTE STATEMENT [1.5]

(no contents yet)

## EXP() [2.1]

Returns the exponential e to the argument.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
EXP(<number>)
```

> **Important**
> If <number> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | The number |

### Description

Returns the exponential e to the argument.

### Examples

```
select EXP(0) from rdb$database
(returns 1)

select EXP(1) from rdb$database
(returns 2,718281828459 or e)

select EXP(2) from rdb$database
(returns 7,3890560989307 or e^2)
```

**See also:** POWER()

# EXTRACT( )

Extracts date and time information from DATE, TIME, and TIMESTAMP values.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
EXTRACT (part FROM value)
```

(This text is currently not included because of possible copyright issues.)

# FETCH

Retrieves the next available row from the active set of an opened cursor.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
ESQL form:
FETCH cursor
 [INTO :hostvar [[INDICATOR] :indvar]
 [, :hostvar [[INDICATOR] :indvar] ...]];
DSQL form:
FETCH cursor {INTO | USING} SQL DESCRIPTOR xsqlda
Blob form: See FETCH (BLOB).
```

(This text is currently not included because of possible copyright issues.)

**See also:** CLOSE, DECLARE CURSOR, DELETE, FETCH (BLOB), OPEN, For more information about cursors and XSQLDA, see the Embedded SQL Guide (EmbedSQL.pdf) from the InterBase(R) 6 documentation set, available from Borland.

# FETCH (BLOB)

Retrieves the next available segment of a blob column and places it in the specified local buffer.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
FETCH cursor INTO
   :<buffer> [[INDICATOR] :segment_length];
```

(This text is currently not included because of possible copyright issues.)

**See also:** BASED ON, CLOSE (BLOB), DECLARE CURSOR (BLOB), INSERT CURSOR (BLOB), OPEN (BLOB)

## FIRST(m) SKIP(n)

Optional sub-clauses to a SELECT statement-FIRST(m) produces the first m rows of an ordered output set, discarding the remainder, while SKIP(n) causes the first n rows of an ordered output set to be discarded and begins the output at row n+1. Both FIRST and SKIP are optional. If both are present, they interact in the output.

(This text is currently not included because of possible copyright issues.)

## FLOOR() [2.1]

Returns a value representing the greatest integer that is lesser than or equal to the input argument.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
   FLOOR( <number> )
```

> **Important**
> <Notes>

| Argument | Description |
|---|---|
| <number> | the number whose next-greater integer value is returned |

### Description

Returns a value representing the greatest integer that

is lesser than or equal to the input argument.

### Examples

```
select floor(1.0) from rdb$database
(returns 1)

select floor(1.9) from rdb$database
(returns 1)

select floor(-1.1) from rdb$database
(returns -2)
```

**See also:** CEIL(), ROUND()

## FOR UPDATE [WITH LOCK] [1.5]

(no contents yet)

## GDSCODE [1.5]

(no contents yet)

## GEN_ID( )

Produces a system-generated integer value.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
GEN_ID (generator, step)
```

(This text is currently not included because of possible copyright issues.)

**See also:** CREATE GENERATOR, SET GENERATOR

## GEN_UUID() [2.1]

Returns a universal unique number.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
GEN_UUID()
```

## Description

Returns a universal unique number.

## Examples

```
insert into records (id) value (gen_uuid());
```

**See also:** GEN_ID()

# GRANT

Assigns privileges to users for specified database objects.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
GRANT  {<privileges>  ON [TABLE] {tablename | viewname}
  TO {<object> | <userlist> | GROUP UNIX_group}
 | EXECUTE ON PROCEDURE procname   TO {<object> | <userlist>}
 | <role_granted> TO {PUBLIC | <role_grantee_list>}};
<privileges> =  ALL [PRIVILEGES] | <privilege_list>
<privilege_list> = {
 SELECT
 | DELETE
 | INSERT
 | UPDATE [(col [, col ...])]
 | REFERENCES [(col [, col ...])]
 }
 [, <privilege_list> ...]
<object> = {
 PROCEDURE procname
 | TRIGGER trigname
 | VIEW viewname
 | PUBLIC
 }
 [, <object> ...]
<userlist> = {
 [USER] username
 | rolename
 | UNIX_user
```

```
 }
 [, <userlist> ...]
 [WITH GRANT OPTION]
<role_granted> = rolename [, rolename ...]
<role_grantee_list> = [USER] username [, [USER] username ...]
 [WITH ADMIN OPTION]
```

(This text is currently not included because of possible copyright issues.)

**See also:** REVOKE, For more information about privileges, see Using Firebird- Database-level security (ch. 22 p. 429).

## *HASH() [2.1]*

Returns a HASH of a value.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
HASH(<string>)
```

> **Important**
> If <string> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <string> | The string the hash is calculated from |

### Description

Returns a HASH of a value.

### Examples

```
select HASH('') from rdb$database
(returns 0)

select HASH('Firebird') from rdb$database
(returns 20678676612)

select HASH('Firebird'||NULL) from rdb$database
(returns NULL)
```

**See also:**

## *IIF [2.0]*

Shortcut function for a two-branch CASE construct

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
IIF (<search_condition>, <value1>, <value2>)
```

> **Important**
> <Notes>

| Argument | Description |
|---|---|
| <search_condition> | The condition to be evaluated |
| <value1> | The result returned if the <search_condition> evaluates to TRUE |
| <value2> | The result returned if the <search_condition> evaluates to FALSE |

### Description

IIF() returns the value of the first sub-expression if the given search condition evaluates to TRUE, otherwise it

returns a value of the second sub-expression. It is implemented as a shortcut function for the following CASE construct:

CASE

WHEN <search_condition> THEN <value1>

ELSE <value2>

END

### Examples

```
SELECT IIF(VAL > 0, VAL, -VAL) FROM OPERATION
```

**See also:** CASE, COALESCE(), NULLIF(), DECODE()

# INSERT

Adds one or more new rows to a specified table.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
INSERT [TRANSACTION transaction] INTO <object> [(col [, col ...])]
   {VALUES (<val> [, <val> ...]) | <select_expr>};
<object> = tablename | viewname
<val> = {:variable | <constant> | <expr> | (<
single_select_expr>)
 | <function> | udf ([<val> [, <val> ...]])
 | NULL | USER | RDB$DB_KEY | ?
 } [COLLATE collation]
<constant> = num | 'string' | _charsetname 'string'
<function> = CAST (<val> AS <datatype>)
 | UPPER (<val>)
 | GEN_ID (generator, <val>)
```

(This text is currently not included because of possible copyright issues.)

**Argument1:** Description TRANSACTION transaction Name of the transaction that controls the execution of the INSERT INTO object Name of an existing table or view into which to insert data col Name of an existing column in a table or view into which to insert values VALUES (val [, val ...]) Lists values to insert into the table or view; values must be listed in the same order as the target columns select_expr Query that returns row values to insert into target columns

**See also:** GRANT, REVOKE, SET TRANSACTION, UPDATE

# INSERT CURSOR (BLOB)

Inserts data into a blob cursor in units of a blob segment-length or less in size.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
INSERT CURSOR cursor
   VALUES (:buffer [INDICATOR] :bufferlen);
```

(This text is currently not included because of possible copyright issues.)

**See also:** CLOSE (BLOB), DECLARE CURSOR (BLOB), FETCH (BLOB), OPEN (BLOB)

# INSERT INTO ... DEFAULT VALUES [2.1]

Inserts a record without supplying field values

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
INSERT INTO <table> DEFAULT VALUES [RETURNING <values>]
```

**Important**
<Notes>

| Argument | Description |
|---|---|
| <table> | the table to insert a record into |
| <values> | optional return parameters (see RETURNING) |

## Description

Allows to INSERT without supplying values, if Before Insert triggers and/or declared defaults are available for every column and none is dependent on the presence of any supplied 'NEW' value.

## Examples

```
INSERT INTO TableWithDefaults DEFAULT VALUES;
```

**See also:** INSERT, RETURNING, UPDATE OR INSERT

# INSERTING, UPDATING, DELETEING [1.5]

(no contents yet)

# LEAVE / BREAK [1.5]

(no contents yet)

## LEAVE [<label_name>] [2.0]

(no contents yet)

## LEFT() [2.1]

Returns the substring of a specified length

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
LEFT( <string expression>, <numeric expression> )
```

> **Important**
> if either of the arguments is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <string> | the string expression (e.g. a field) where the output gets copied from |
| <numeric expression> | denotes how many chars the output will contain |

### Description

Returns the substring of a specified length that appears

at the start of a left-to-right string.

### Examples

```
select left('Firebird', 4) from rdb$database
returns 'Fire'

select left('', 10) from rdb$database
returns ''
```

**See also:** RIGHT()

## LIKE ... ESCAPE?? [1.5]

(no contents yet)

## LIST() [2.1]

Returns a string with concatenated matches

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
LIST '(' [ {ALL | DISTINCT} ] <value expression> [',' <delimiter
value>] ')'
<delimiter value> ::= { <string literal> | <parameter> |
<variable> }
```

> **Important**
> <Notes>

| Argument | Description |
|---|---|
| <value expression> | The expression to be concatenated |
| <delimiter value> | The separator inserted between any matches |

### Description

This function returns a string result with the concatenated non-NULL values from a group. It returns NULL if

there are no non-NULL values.

**Rules:** 1. If neither ALL nor DISTINCT is specified, ALL is implied. 2. If <delimiter value> is omitted, a comma is used to separate the concatenated values. Other Notes 1. Numeric and date/time values are implicitly converted to strings during evaluation. 2. The result value is of type BLOB with SUB_TYPE TEXT for all cases except list of BLOB with different subtype. 3. Ordering of values within a group is implementation-defined.

### Examples

```
/* A */
```

```
SELECT LIST(ID, ':')
FROM MY_TABLE

/* B */
SELECT TAG_TYPE, LIST(TAG_VALUE)
FROM TAGS
GROUP BY TAG_TYPE
```

**See also:**

## *LN() [2.1]*

Returns the natural logarithm of a number.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
LN(<number>)
```

> **Important**
> If <number> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | The number wgose natural logarithm is returned |

### Description

Returns the natural logarithm of a number.

### Examples

```
select ln(0) from rdb$database
(throws the error 'expression evaluation not supported' and returns NULL)

select ln(1) from rdb$database
(returns 0)

select ln(10) from rdb$database
(returns 2,302585092994)

select ln(exp(1)) from rdb$database
(returns 1)
```

**See also:** EXP()

## LOG() [2.1]

returns the logarithm base x of y.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
LOG( <number1>, <number2> )
```

> **Important**
> If either of the arguments is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number1> | The logarithm base |
| <number2> | The number whose loagithm base <number1> is calculated |

### Description

returns the logarithm base x of y.

### Examples

```
select log(1,10) from rdb$database
(returns INF)

select log(0,0) from rdb$database
(returns NAN)

select log(exp(1),10) from rdb$database
(returns 2,302585092994)

select log(10,10000) from rdb$database
(returns 4)
```

**See also:** LOG10()

# LOG10() [2.1]

Returns the logarithm base ten of a number.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
LOG10(<number>)
```

> **Important**
> If <number> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | The number whose logarith base 10 is calculated |

## Description

Returns the logarithm base ten of a number. The function is equivalent

to LOG(10,<number>).

## Examples

```
select log10(0) from rdb$database
(returns -INF)

select log10(1) from rdb$database
(returns 0)

select log10(10) from rdb$database
(returns 1)

select log10(10000) from rdb$database
(returns 4)
```

**See also:** LOG()

# LOWER() [2.0]

Converts a string to all lowercase.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
LOWER (<val>)
```

| Argument | Description |
|---|---|
| val | A column, constant, host-language variable, expression, function, or UDF that evaluates to a character datatype |

### Description

LOWER() converts a specified string to all lowercase characters. If applied to character sets that have no case differentiation, LOWER() has no effect.

### Examples

```
The following ISQL statement changes the name, BMatthews, to bmatthews:
UPDATE EMPLOYEE
 SET EMP_NAME = LOWER ('BMatthews')
 WHERE EMP_NAME = 'BMatthews';
The next ISQL statement creates a domain called PROJNO with a CHECK
constraint that requires the value of the column to be all lowercase:
CREATE DOMAIN PROJNO
 AS CHAR(5)
 CHECK (VALUE = LOWER (VALUE));
```

**See also:** CAST( ), UPPER()

## *LPAD() [2.1]*

prepends string2 to the beginning of string1

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
  LPAD( <string1>, <number> [, <string2> ] )
```

**Important**
If either of the arguments is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <string1> | the string expression to be padded. |
| <number> | the length of the output string |
| <string2> | the string to be prepended (default is a blank or space) |

## Description

LPAD(string1, length, string2) prepends string2 to

the beginning of string1 until the length of the result

string becomes equal to length.

**Rules:** 1. If the second string is omitted the default value is one space. 2. If the result string would exceed the length, the second string is truncated.

## Examples

```
select LPAD('TEST',10) from rdb$database
(returns '      TEST', see Rule 1)

select LPAD('TEST',10,'x') from rdb$database
(returns 'xxxxxxTEST')

select LPAD('TEST',10,'1234') from rdb$database
(returns '123412TEST', see Rule 2)

select LPAD('1234567890',5,'x') from rdb$database
(returns '12345', that is: the output string is limited in length to <
number>)
```

**See also:**

# *MAX( )*

Retrieves the maximum value in a column.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
MAX ([ALL] <val> | DISTINCT <val>)
```

(This text is currently not included because of possible copyright issues.)

**See also:** AVG( ), COUNT( ), CREATE DATABASE, CREATE TABLE, MIN( ), SUM( )

## *MAXVALUE() [2.1]*

Returns the maximum value of a list of values

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
MAXVALUE( <number> [,<number>] )
```

> **Important**
> If any of the arguments is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | a number or numeric expression |

### Description

Returns the maximum value of a list of values

### Examples

```
select MAXVALUE(1,5,3) from rdb$database
(returns 5)

select MAXVALUE(1,5,NULL) from rdb$database
(returns NULL)
```

**See also:** MAX(), MIN(), MINVALUE()

# MIN( )

Retrieves the minimum value in a column.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
MIN ([ALL] <val> | DISTINCT <val>)
```

(This text is currently not included because of possible copyright issues.)

**See also:** AVG( ), COUNT( ), CREATE DATABASE, CREATE TABLE, MAX( ), SUM( )

# MINVALUE() [2.1]

Returns the minimum value of a list of values

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
  MINVALUE( <number> [,<number>] )
```

> **Important**
> If any of the arguments is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | a number or numeric expression |

## Description

Returns the minimum value of a list of values

## Examples

```
select MINVALUE(1,5,3) from rdb$database
(returns 1)
```

```
select MINVALUE(1,5,NULL) from rdb$database
(returns NULL)
```

**See also:** MAX(), MIN(), MAXVALUE()

## MOD() [2.1]

returns the remainder part of the division of X by Y.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
MOD( <number1>, <number2> )
```

> **Important**
> <Notes>

| Argument | Description |
|---|---|
| <number1> | The number or numeric expression the modulo is calculated from |
| <number2> | The number or numeric expression that <number1> is divided by to calculate the modulo |

### Description

Modulo: MOD(X, Y) returns the remainder part of

the division of X by Y.

### Examples

```
select MOD(10,3) from rdb$database
(returns 1)

select MOD(10,5) from rdb$database
(returns 0)

select MOD(-10,3) from rdb$database
(returns -1)
```

**See also:** TRUNC()

## MON$ Tables [2.1]

(no contents yet)

## NATURAL JOIN [2.1]

(no contents yet)

## NEXT VALUE FOR [2.0]

Generates the next value for a sequence / generator

**Availability:** +DSQL +ESQL +ISQL +PSQL

### Syntax

```
NEXT VALUE FOR <name>
```

> **Important**
> <Notes>

| Argument | Description |
|---|---|
| <name> | name of the sequence / generator whose next value is returned |

### Description

Generates and returns the next value for a sequence.

The NEXT VALUE FOR <name> expression is a synonym for GEN_ID(<name>, 1), using the SQL-99-compliant SEQUENCE syntax.

While the GEN_ID() function allows an optional step or increment value to be supplied in the function call, the increment is implicitly set to 1 when using NEXT VALUE FOR.

### Examples

```
This example generates a new value for the ID column using a sequence,
and returns that new value to the caller:
INSERT INTO EMPLOYEE (ID, NAME)
VALUES (NEXT VALUE FOR SEQ_ID_EMPLOYEE, 'John Smith')
RETURNING ID;
```

**See also:** GEN_ID(), CREATE SEQUENCE, ALTER SEQUENCE, DROP SEQUENCE

> **Tip**
> For more information about the use of sequences, refer to the "Generator Guide" that is available as part of the Firebird documentation set.

## *NULLIF [1.5]*

Returns NULL for a sub-expression if it has a specific value, otherwise returns the value of the subexpression.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
NULLIF ( <value expression1> , <value expression2> )
```

> **Important**
> <Notes>

| Argument | Description |
|---|---|
| <value expression1> | The value returned when it is not NULL |
| <value expression2> | Tha value returned if <value expression1> evaluates to NULL |

### Description

Returns NULL for a sub-expression if it has a specific value, otherwise returns the value of the subexpression.

NULLIF (V1, V2) is equivalent to the following case specification: CASE WHEN V1 = V2 THEN NULL ELSE V1 END

### Examples

```
UPDATE PRODUCTS SET STOCK = NULLIF(STOCK,0)
```

**See also:** CASE, COALESCE(), DECODE(), IIF()

## OPEN

Retrieve specified rows from a cursor declaration.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
ESQL form:
OPEN [TRANSACTION transaction] cursor;
DSQL form:
OPEN [TRANSACTION transaction] cursor [USING SQL DESCRIPTOR xsqlda]
Blob form: See OPEN (BLOB).
```

(This text is currently not included because of possible copyright issues.)

**See also:** CLOSE, DECLARE CURSOR, FETCH OPEN (BLOB) Opens a previously declared blob cursor and prepares it for read or insert.

**Availability1:** DSQL ESQL ISQL PSQL

### Syntax

```
OPEN [TRANSACTION name] cursor
   {INTO | USING} :blob_id;
```

**Argument1:** Description TRANSACTION name Specifies the transaction under which the cursor is opened Default: The default transaction cursor Name of the blob cursor INTO | USING Depending on blob cursor type, use one of these: INTO: For INSERT BLOB USING: For READ BLOB blob_id Identifier for the blob column

**See also:** CLOSE (BLOB), DECLARE CURSOR (BLOB), FETCH (BLOB), INSERT CURSOR (BLOB)

## OVELAY() [2.1]

Returns string1 replacing the substring FROM start FOR length by string2.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
   OVERLAY( <string1> PLACING <string2> FROM <start> [
FOR <length> ] )
```

**Important**
If either of the arguments is (or evaluates to) NULL, the result is NULL . Use the FOR <length> clause with care - see the examples below!

| Argument | Description |
|---|---|
| arg1 | arg1 desc |

## Description

Returns string1 replacing the substring FROM start

FOR length by string2.

The OVERLAY function is equivalent to:

SUBSTRING(<string1>, 1 FOR <start> - 1) || <string2> || SUBSTRING(<string1>, <start> + <length>)

If <length> is not specified, CHAR_LENGTH( <string2> ) is implied.

If <length> is specified, then

the <length> characters of <string1> starting with character #<start>

will be replaced with the entire <string2>, that is <string2> will not be

clipped or padded to adjust it to <length>.

## Examples

```
select OVERLAY('1234567890' PLACING 'ABCD' FROM 3) from rdb$database
(retunrs '12ABCD7890')

select OVERLAY('1234567890' PLACING 'ABCD' FROM 9) from rdb$database
(returns '12345678ABCD' - note the output is longer than string1!)

select OVERLAY('1234567890' PLACING 'ABCD' FROM 3 FOR 2 ) from rdb$database
(returns '12ABCD567890')

select OVERLAY('1234567890' PLACING 'ABCD' FROM 3 FOR 4 ) from rdb$database
(returns '12ABCD7890')
```

```
select OVERLAY('1234567890' PLACING 'ABCD' FROM 3 FOR 6 ) from rdb$database
(returns '12ABCD90')
```

**See also:** SUBSTRING()

## PI() [2.1]

Returns the number PI

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
   PI()
```

### Description

Returns the number PI with a precision of 13 decimals

### Examples

```
select PI() from rdb$database
(returns 3,1415926535898)
```

**See also:** SIN(), COS()

## POSITION() [2.1]

returns the position of the substring X in the string Y.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
   POSITION(<string1> IN <String2>)
```

**Important**
If either of the arguments is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <string1> | the string whose position is to be found in <string2> |
| <string2> | the string where <string1> is searched in |

## Description

Returns the position of the substring

X in the string Y. Returns 0 if X is not found

within Y. The character matching is NOT case sensitive.

## Examples

```
select POSITION('bird' IN 'Firebird') from rdb$database
(returns 5)

select POSITION('Bird' IN 'Firebird') from rdb$database
(returns 0 - search is not case sensitive!)
```

**See also:** SUBSTRING()

# *POWER() [2.1]*

Returns X to the power of Y

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
POWER( <number1>, <number2> )
```

**Important**
If either of the arguments is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number1> | The number that is put to the power of <number2> |

### Description

Returns X to the power of Y. The function is equivalent to <number1>^<number2>

### Examples

```
select power(2,16) from rdb$database
(returns 65536)

select power(10,6) from rdb$database
(returns 1000000)

select power(10,1.5) from rdb$database
(returns 31,6227766016838)

select power(10,-1) from rdb$database
(returns 0.1)
```

**See also:** EXP()

## PREPARE

Prepares a statement for execution in embedded SQL.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
PREPARE [TRANSACTION transaction] statement
   [INTO SQL DESCRIPTOR xsqlda] FROM {:variable | 'string'};
```

(This text is currently not included because of possible copyright issues.)

**Note1:** The previous statement could also be prepared and described in the following manner: EXEC SQL PREPARE Q FROM :buf; EXEC SQL DESCRIBE Q INTO SQL DESCRIPTOR xsqlda;

**See also:** DESCRIBE, EXECUTE, EXECUTE IMMEDIATE

## RAND() [2.1]

Returns a random value in the range between 0 and 1

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
RAND()
```

### Description

Returns a random value in the range between 0 and 1

### Examples

```
select rand() from rdb$database
(returns a random double precision value with up to 13 decimals)
```

**See also:**

## *RDB$GET_CONTEXT [2.0]*

(no contents yet)

## *RDB$SET_CONTEXT [2.0]*

(no contents yet)

## *RECREATE EXCEPTION [2.0]*

(no contents yet)

## *RECREATE PROCEDURE*

RECREATE PROCEDURE redefines an existing stored procedure to a database.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
RECREATE PROCEDURE name
   [(param <pdatatype> [, param <pdatatype> ...])]
   [RETURNS param <pdatatype> [, param <pdatatype> ...])]
   AS <procedure_body> [terminator]
```

(This text is currently not included because of possible copyright issues.)

**See also:** DROP PROCEDURE, CREATE PROCEDURE, ALTER PROCEDURE

## RECREATE TABLE

RECREATE TABLE redefines an existing table to a database.

**Availability:** DSQL ESQL ISQL PSQL

(This text is currently not included because of possible copyright issues.)

### Syntax

```
RECREATE TABLE table [EXTERNAL [FILE] 'filespec']
   (<col_def> [, <col_def> | <tconstraint> ...]);
```

**See also:** DROP TABLE, CREATE TABLE, ALTER TABLE

## RECREATE TRIGGER [2.0]

(no contents yet)

## RECREATE VIEW

(This text is currently not included because of possible copyright issues.)

## RELEASE SAVEPOINT [1.5]

(no contents yet)

# REPLACE() [2.1]

Replaces all occurrences of <findstring> in <stringtosearch> with <replstring>.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
   REPLACE( <stringtosearch>, <findstring>, <replstring>
)
```

> **Important**
> If either of the arguments is (or evaluates to) NULL, the result is NULL .

| Argument | Description |
|---|---|
| <stringtosearch> | the string to be searched and replaced in <findstring> |
| <findstring> | the string where <stringtosearch> is searched in |
| <replstring> | the string to replace <findstring> |

## Description

Replaces all occurrences of <findstring> stringtosearch>

with <replstring>. Search is NOT case sensitive.

## Examples

```
select REPLACE('Firebird','i','1') from rdb$database
(returns 'F1reb1rd')

select REPLACE('Firefox','f','b') from rdb$database
(returns 'Firebox' - search is not case sensitive)

select REPLACE('123123','2','two') from rdb$database
(returns '1two31two3')

select REPLACE('ABCDE','B','BCB') from rdb$database
(returns 'ABCBCDE' - replace is not recursive)
```

**See also:** POSITION(), SUBSTRING()

# RETURNING [2.1]

Returns columns from an INSERT, UPDATE or DELETE operation to the caller.

**Availability:** +DSQL +ESQL +ISQL +PSQL

## Syntax

```
INSERT INTO ... VALUES (...)
  [RETURNING <column_list> [INTO <variable_list>]]

INSERT INTO ... SELECT ...
  [RETURNING <column_list> [INTO <variable_list>]]

UPDATE OR INSERT INTO ... VALUES (...) ...
  [RETURNING <column_list> [INTO <variable_list>]]

UPDATE ...
  [RETURNING <column_list> [INTO <variable_list>]]

DELETE FROM ...
  [RETURNING <column_list> [INTO <variable_list>]]
```

**Important**
In DSQL, the statement always returns the set, even if the operation had no effect on any record. Hence, at this stage of implementation, the potential exists to return an "empty" set. (This may be changed in a future version.)

| Argument | Description |
|---|---|
| <column_list> | The list of columns to be returned as a result of the respective operation |
| <variable_list> | optional list of result variables to take the returned values (PSQL only) |

## Description

The purpose of the RETURNING clause is to enable the column values stored into a table as a result of the INSERT,

UPDATE OR INSERT, UPDATE and DELETE statements to be returned to the client.

The most likely usage is for retrieving the value generated for a primary key inside a BEFORE-trigger. The

RETURNING clause is optional and is available in both DSQL and PSQL, although the rules differ slightly.

In DSQL, the execution of the operation itself and the return of the set occur in a single protocol round trip.

Because the RETURNING clause is designed to return a singleton set in response to completing an

operation on

a single record, it is not valid to specify the clause in a statement that inserts, updates or deletes multiple records.

**Rules:** for Using a RETURNING Clause: 1. The INTO part (i.e. the variable list) is allowed in PSQL only, for assigning the output set to local variables. It is rejected in DSQL. 2. The presence of the RETURNING clause causes an INSERT statement to be described by the API as isc_info_sql_stmt_exec_procedure rather than isc_info_sql_stmt_insert. Existing connectivity drivers should already be capable of supporting this feature without special alterations. 3. The RETURNING clause ignores any explicit record change (update or delete) that occurs as a result of the execution of an AFTER trigger. 4. OLD and NEW context variables can be used in the RETURNING clause of UPDATE and UPDATE OR INSERT statements. 5. In UPDATE and UPDATE OR INSERT statements, field references that are unqualified or qualified by table name or relation alias are resolved to the value of the corresponding NEW context variable.

### Examples

```
1.
INSERT INTO T1 (F1, F2)
VALUES (:F1, :F2)
RETURNING F1, F2 INTO :V1, :V2;

2.
INSERT INTO T2 (F1, F2)
VALUES (1, 2)
RETURNING ID INTO :PK;

3.
DELETE FROM T1
WHERE F1 = 1
RETURNING F2;

4. UPDATE T1
SET F2 = F2 * 10
RETURNING OLD.F2, NEW.F2;
```

**See also:** INSERT, UPDATE, DELETE, UPDATE OR INSERT

## *REVERSE() [2.1]*

Returns a string in reverse order.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
REVERSE(<string expression>)
```

**Important**
if <string expression> is (or evaluates to) NULL, the result is NULL.

| Argument | Description |
|---|---|
| <string expression> | the string to be returned in reverse order |

## Description

Returns a string in reverse order. Useful function

for creating an expression index that indexes strings

from right to left.

## Examples

```
create index people_email on people
computed by (reverse(email));

select * from people
where reverse(email) starting with reverse('.br');

select reverse('Firebird') from rdb$database;
(returns 'driberiF')

select reverse('reliefpfeiler') from rdb$database;
(returns 'reliefpfeiler', which is an existing German word!)
```

**See also:**

# *REVOKE*

Withdraws privileges from users for specified database objects.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
REVOKE [GRANT OPTION FOR] {<privileges> ON [TABLE]
   {tablename | viewname}
  FROM {<object> | <userlist> | <rolelist> | GROUP
UNIX_group}
 | EXECUTE ON PROCEDURE procname
  FROM {<object> | <userlist>}
```

```
 | <role_granted> FROM {PUBLIC | <role_grantee_list>}};
<privileges> = ALL [PRIVILEGES] | <privilege_list>
<privilege_list> = {
 SELECT
 | DELETE
 | INSERT
 | UPDATE [(col [, col ...])]
 | REFERENCES [(col [, col ...])]
 }
 [, <privilege_list> ...]
<object> = {
 PROCEDURE procname
 | TRIGGER trigname
 | VIEW viewname
 | PUBLIC
 }

 [, <object> ...]
<userlist> = [USER] username [, [USER] username ...]
<rolelist> = rolename [, rolename]
<role_granted> = rolename [, rolename ...]
<role_grantee_list> = [USER] username [, [USER] username ...]
```

(This text is currently not included because of possible copyright issues.)

**See also:** GRANT

## REVOKE ADMIN OPTION FROM [2.0]

(no contents yet)

## RIGHT() [2.1]

Returns the rightmost part of a string.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
  RIGHT( <string>, <numeric expression> )
```

> **Important**
> If either of the arguments is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <string> | the string expression (e.g. a field) where the output gets copied from |
| <numeric expression> | denotes how many chars the output will contain |

### Description

Returns a substring, of the specified length, from

the right-hand end of a string.

### Examples

```
select right('Firebird',4) from rdb$database
(returns 'bird')

select right('Firebird',10) from rdb$database
(returns 'Firebird', that is the output is not padded if <string> is
shorter than 10)
```

**See also:** LEFT(), SUBSTRING()

## ROLLBACK

Restores the database to its state prior to the start of the current transaction.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
ROLLBACK [TRANSACTION name] [WORK] [RELEASE];
```

(This text is currently not included because of possible copyright issues.)

**See also:** COMMIT, DISCONNECT, For more information about controlling transactions, see Using Firebird- Transactions in Firebird (ch. 8 p. 90). .

## ROLLBACK RETAIN [2.0]

(no contents yet)

## ROLLBACK [WORK] TO [SAVEPOINT] [1.5]

(no contents yet)

# ROUND() [2.1]

Returns a number rounded to the specified scale.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
ROUND(<number1>,<number2>)
```

> **Important**
> If any of the arguments is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number1> | The number or numeric expression to be rounded |
| <number2> | The scale (number of decimal places) <number1> is rounded to |

## Description

Returns a number rounded to the specified scale. If the scale

(<number2>) is negative, the integer part of the value is rounded.

## Examples

```
select round(0.123456789,6) from rdb$database
(returns 0.123457)

select round(0.123456789,3) from rdb$database
(returns 0.123)

select round(12345.6789,0) from rdb$database
(returns 12346.0)

select round(12345.6789,-3) from rdb$database
(returns 12000.0)
```

**See also:** TRUNC()

# ROWS [2.0]

(no contents yet)

## ROW_COUNT [1.5]

(no contents yet)

## RPAD() [2.1]

appends string2 to the end of string1

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
RPAD( <string1>, <number> [, <string2> ] )
```

> **Important**
> If either of the arguments is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <string1> | the string expression to be padded. |
| <number> | the length of the output string |
| <string2> | the string to be appended (default is a blank or space) |

### Description

RPAD(string1, length, string2) appends string2 to

the end of string1 until the length of the result

string becomes equal to length.

**Rules:** 1. If the second string is omitted the default value is one space. 2. If the result string would exceed the length, the second string is truncated.

### Examples

```
select RPAD('TEST',10) from rdb$database
(returns 'TEST      ', see Rule 1)
```

```
select RPAD('TEST',10,'x') from rdb$database
(returns 'TESTxxxxxx')

select RPAD('TEST',10,'1234') from rdb$database
(returns 'TEST123412', see Rule 2)

select RPAD('1234567890',5,'x') from rdb$database
(returns '12345', that is: the output string is limited in length to <
number>)
```

**See also:** LPAD()

## SAVEPOINT [1.5]

(no contents yet)

## SELECT

Retrieves data from one or more tables.

**Availability:** DSQL ESQL ISQL PSQL* * In PSQL, a variant syntax for SELECT is available. For details, refer to notes on SELECT and FOR SELECT...INTO...DO in the chapter PSQL-Firebird Procedural Language.

### Syntax

```
SELECT [TRANSACTION transaction]
   [ {[FIRST int] [SKIP int]} ]
   [DISTINCT | ALL]
   {* | <val> [, <val> ...]}
   [INTO :var [, :var ...]]
   FROM <tableref> [, <tableref> ...]
   [WHERE <search_condition>]
   [GROUP BY col [COLLATE collation] [, col [COLLATE collation] ...]
   [HAVING <search_condition>]
   [UNION [ALL] <select_expr>]
   [PLAN <plan_expr>]
   [ORDER BY <order_list>]
   [FOR UPDATE [OF col [, col ...]]];
<val> = {
 col [<array_dim>] | :variable
 | <constant> | <expr> | <function>
 | udf ([<val> [, <val> ...]])
 | NULL | USER | RDB$DB_KEY | ?
 } [COLLATE collation] [AS alias]
<array_dim> = [[x:]y [, [x:]y ...]]
<constant> = num | 'string' | _charsetname 'string'
<function> =  COUNT (* | [ALL] <val> | DISTINCT <val>)
 | SUM ([ALL] <val> | DISTINCT <val>)
 | AVG ([ALL] <val> | DISTINCT <val>)
 | MAX ([ALL] <val> | DISTINCT <val>)
 | MIN ([ALL] <val> | DISTINCT <val>)
```

```
 | CAST (<val> AS <datatype>)
 | UPPER (<val>)
 | GEN_ID (generator, <val>)
<tableref> = <joined_table> | table | view | procedure
 [(<val> [, <val> ...])] [alias]
<joined_table> = <tableref> <join_type> JOIN <
tableref>
 ON <search_condition> | (<joined_table>)
<join_type> = [INNER] JOIN
 | {LEFT | RIGHT | FULL } [OUTER]} JOIN
<search_condition> = <val> <operator> {<val> | (<
    select_one>)}
 | <val> [NOT] BETWEEN <val> AND <val>
 | <val> [NOT] LIKE <val> [ESCAPE <val>]
 | <val> [NOT] IN (<val> [, <val> ...] | <
select_list>)
 | <val> IS [NOT] NULL
 | <val> {>= | <=} <val>
 | <val> [NOT] {= | < | >} <val>
 | {ALL | SOME | ANY} (<select_list>)
 | EXISTS (<select_expr>)
 | SINGULAR (<select_expr>)
 | <val> [NOT] CONTAINING <val>
 | <val> [NOT] STARTING [WITH] <val>
 | (<search_condition>)
 | NOT <search_condition>
 | <search_condition> OR <search_condition>
 | <search_condition> AND <search_condition>
<operator> = {= | < | > | <= | >= | !< | !> | <
> | !=}
<plan_expr> =
 [JOIN | [SORT] [MERGE]] ({<plan_item> | <plan_expr>}
 [, {<plan_item> | <plan_expr>} ...])
<plan_item> = {table | alias}
 {NATURAL | INDEX (<index> [, <index> ...]) | ORDER <
index>}
<order_list> =
 {col | int} [COLLATE collation]
  [ASC[ENDING] | DESC[ENDING]]
  [, <order_list> ...]
```

(This text is currently not included because of possible copyright issues.)

**Argument1:** Description TRANSACTION transaction Name of the transaction under control of which the statement is executed; ESQL only SELECT [DISTINCT | ALL] Specifies data to retrieve. DISTINCT prevents duplicate values from being returned. ALL, the default, retrieves every value SELECT {[FIRST m] | [SKIP n]} ... ORDER BY ... FIRST m returns an output set consisting of m rows, optionally SKIPping n rows and returning a set beginning (n+1) rows from the "top" of the set specified by the rest of the SELECT specification. If SKIP n is used and the [FIRST m] parameter is omitted, the output set returns all rows in the SELECT specification except the "top" n rows. These parameters generally make sense only if applied to a sorted set. {*|val [, val ...]} The asterisk (*) retrieves all columns for the specified tables val [, val ...] retrieves a list of specified columns, values, and expressions INTO :var [, var ...] Singleton select in ESQL only; specifies a list of host-language variables into which to retrieve values FROM tableref [, tableref ...] List of tables, views, and stored procedures from which to retrieve data; list can include joins and joins can be nested table Name of an existing table in a database view Name of an existing view in a database procedure Name of an existing stored procedure that functions like a SELECT statement alias Brief, alternate name for a table, view, or column; after declaration in tableref, alias can stand in for subsequent references to a table or view joined_table A table reference consisting of a JOIN join_type Type of join to perform. Default: INNER WHERE search_condition Specifies a condition that limits rows retrieved to a subset of all available rows GROUP BY col [, col ...] Partitions the results of a query, assembling the output into

groups formed on the basis of common values in all of the output columns named in the grouping list. Precedence of grouping columns is left=high. Aggregations apply to the grouping column having the lowest precedence. COLLATE collation Specifies the collation order for the data retrieved by the query HAVING search_condition Used with GROUP BY; specifies a condition that limits grouped rows returned UNION [ALL] Combines two or more tables that are fully or partially identical in structure; the ALL option keeps identical rows separate instead of folding them together into one PLAN plan_expr Specifies the access plan for the Firebird optimizer to use during retrieval plan_item Specifies a table and index method for a plan ORDER BY order_list Specifies columns to order, either by column name or ordinal number in the query, and the order (ASC or DESC) in which rows to return the rows

**See also:** DECLARE CURSOR, DELETE, INSERT, UPDATE, For discussions of topics related to query specifications and SQL, see Using Firebird- Firebird SQL & Queries (ch. 9 p. 110)., For a full discussion of data retrieval in embedded programming using DECLARE CURSOR and SELECT, see the Embedded SQL Guide (EmbedSQL) of the InterBase(R) 6 documentation set, available from Borland.

# SET DATABASE

Declares a database handle for database access.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
SET {DATABASE | SCHEMA} dbhandle =
    [GLOBAL | STATIC | EXTERN][COMPILETIME][FILENAME] 'dbname'
   [USER 'name' PASSWORD 'string']
   [RUNTIME [FILENAME]
   {'dbname' | :var}
   [USER {'name' | :var} PASSWORD {'string' |:var}]];
```

(This text is currently not included because of possible copyright issues.)

**See also:** COMMIT, CONNECT, ROLLBACK, SELECT, For more information on the security database, seeUsing Firebird- Managing Security (ch. 22 p. 414)..

# SET DEFAULT [2.0]

(no contents yet)

# SET GENERATOR

Sets a new value for an existing generator.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
SET GENERATOR name TO int;
```

(This text is currently not included because of possible copyright issues.)

**See also:** CREATE GENERATOR, CREATE PROCEDURE, CREATE TRIGGER, GEN_ID( )

## SET HEAD[ing] toggle [2.0]

(no contents yet)

## SET NAMES

Specifies an active character set to use for subsequent database attachments.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
SET NAMES [charset | :var];
```

(This text is currently not included because of possible copyright issues.)

**See also:** CONNECT, SET DATABASE, For more information about character sets and collation orders, see Using Firebird- Character Sets and Collation Orders (ch. 16 p. 301).

## SET SQL DIALECT

Declares the SQL Dialect for database access.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
SET SQL DIALECT n;
```

(This text is currently not included because of possible copyright issues.)

**See also:** SHOW SQL DIALECT

## SET SQLDA_DISPLAY ON/OFF [2.0]

(no contents yet)

## SET STATISTICS

Recomputes the selectivity of a specified index.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
SET STATISTICS INDEX name;
```

(This text is currently not included because of possible copyright issues.)

**See also:** ALTER INDEX, CREATE INDEX, DROP INDEX

## SET TRANSACTION

Starts a transaction and optionally specifies its behavior.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
SET TRANSACTION [NAME transaction]
   [READ WRITE | READ ONLY]
   [WAIT | NO WAIT]
   [[ISOLATION LEVEL] {SNAPSHOT [TABLE STABILITY]
    | READ COMMITTED [[NO] RECORD_VERSION]}]
   [RESERVING <reserving_clause>
    | USING dbhandle [, dbhandle ...]];
<reserving_clause> = table [, table ...]
 [FOR [SHARED | PROTECTED] {READ | WRITE}] [, <reserving_clause>]
```

(This text is currently not included because of possible copyright issues.)

**See also:** COMMIT, ROLLBACK, SET NAMES, For more information about transactions, see Using Firebird- Transactions in Firebird (ch. 8 p. 90).

# SHOW SQL DIALECT

Returns the current client SQL dialect setting and the database SQL dialect value.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
SHOW SQL DIALECT;
```

(This text is currently not included because of possible copyright issues.)

**See also:** SET SQL DIALECT

# SIGN() [2.1]

Returns the sign of a number

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
SIGN( <number> )
```

**Important**
If <number> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | The number or numeric expression whose wign is returned |

## Description

Returns 1, 0, or -1 depending on whether the input

value is positive, zero or negative, respectively.

## Examples

```
select SIGN(-99) from rdb$database
(returns -1)

select SIGN(0) from rdb$database
(returns 0)

select SIGN(99) from rdb$database
(returns 1)
```

**See also:** ABS()

## SIN() [2.1]

Returns the sine of a number.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
SIN(<number>)
```

> **Important**
> If <number> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | The number or numeric expression whose sine is returned |

### Description

Returns the sine of a number. The angle is specified

in radians and returns a value in the range -1 to 1.

### Examples

```
select sin(0) from rdb$database
(returns 0)

select sin(-1) from rdb$database
(returns -0,8414709848079)
```

```
select sin(1) from rdb$database
(returns 0,8414709848079)

select sin(PI()) from rdb$database
(returns 0)

select sin(PI()/2) from rdb$database
(returns 1)
```

**See also:** COS(), SINH()

## SINH() [2.1]

Returns the hyperbolic sine of a number.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
SINH(<number>)
```

> **Important**
> If <number> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | The number or numeric expression whose sine is returned |

### Description

Returns the hyperbolic sine of a number. The angle is specified

in radians and returns a value in the range -1 to 1.

### Examples

```
select sinh(0) from rdb$database
(returns 0)

select sinh(-1) from rdb$database
(returns -1,1752011936438)

select sinh(1) from rdb$database
(returns 1,1752011936438)
```

**See also:** COS(), SIN()

# SQL Commands

(no contents yet)

# SQLCODE [1.5]

(no contents yet)

# SQRT() [2.1]

Returns the square root of a number

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
SQRT(<number>)
```

> **Important**
> If <number> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | The number od numeric expression whose square root is returned |

## Description

Returns the square root of a number

## Examples

```
select sqrt(0) from rdb$database
(returns 0)

select sqrt(9) from rdb$database
```

```
(returns 3)

select sqrt(-1) from rdb$database
(throws the error 'expression evaluation not supported', returns NULL)
```

**See also:** POWER()

## SUBSTRING( )

Returns a string of specified length from within an input string, starting from a specified position in the input string.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
SUBSTRING(<input-string> <pos> [FOR <length>])
```

(This text is currently not included because of possible copyright issues.)

**See also:** The user-defined (external) functions substr and substrlen

## SUM( )

Totals the numeric values in a specified column.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
SUM ([ALL] <val> | DISTINCT <val>)
```

(This text is currently not included because of possible copyright issues.)

**See also:** AVG( ), COUNT( ), MAX( ), MIN( )

## TAN() [2.1]

Returns the tangent of a number

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
TAN(<number>)
```

> **Important**
> If <number> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | The number whose tangent is returned |

## Description

Returns the tangent of an input number that is expressed

in radians.

## Examples

```
select tan(0) from rdb$database
(returns 0)

select tan(-1) from rdb$database
(returns -1,5574077246549)

select tan(1) from rdb$database
(returns 1,5574077246549)
```

**See also:** COT(), TANH()

# *TANH() [2.1]*

Returns the hyperbolic tangent of a number

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
TANH(<number>)
```

**Important**
If <number> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | The number whose hyperbolic tangent is returned |

## Description

Returns the hyperbolic tangent of an input number that is expressed

in radians.

## Examples

```
select tanh(0) from rdb$database
(returns 0)

select tan(-1) from rdb$database
(returns -0,7615941559558)

select tanh(1) from rdb$database
(returns 0,7615941559558)
```

**See also:** COT(), TAN()

# *TEMPLATE for new entries [VER]*

Short description

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
<Syntax>
```

**Important**
<Notes>

| Argument | Description |
|---|---|
| arg1 | arg1 desc |

## Description

## Examples

```
```

**See also:**

# *TRIM() [2.0]*

trims characters (default: blanks) from the left and/or right of a string.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
simple: TRIM (<val>)

complete:
TRIM <left paren> [ [ <trim specification> ] [ <trim
character> ]
FROM ] <value expression> <right paren>
<trim specification> ::= LEADING | TRAILING | BOTH
<trim character> ::= <value expression>
```

| Argument | Description |
|---|---|
| val | A column, constant, host-language variable, expression, function, or UDF that evaluates to a character datatype |

## Description

TRIM() trims characters (default: blanks) from the left and/or right of a string.

**Rules:** 1. If <trim specification> is not specified, BOTH is assumed. 2. If <trim character> is not specified, ' ' is assumed. 3. If <trim specification> and/or <trim character> is specified , FROM should be specified. 4. If <trim specification> and <trim character> is not specified, FROM should not be specified.

## Examples

```
```

**See also:** RPAD(), LPAD()

# *TRUNC() [2.1]*

Returns the integral part of a number.

**Availability:** DSQL ESQL ISQL PSQL

## Syntax

```
TRUNC( <number> )
```

> **Important**
> If <number> is (or evaluates to) NULL, the result is NULL

| Argument | Description |
|---|---|
| <number> | The number or numeric expression whose integral part is returned |

## Description

Returns the integral part of a number. The function is equal to FLOOR() for

positive numbers.

## Examples

```
select trunc(1.1) from rdb$database
(returns 1)

select trunc(-1.1) from rdb$database
(returns -1, note FLOOR() would return -2 here.)
```

**See also:** FLOOR(), CEIL()

## TYPE OF [domains in PSQL] [2.1]

(no contents yet)

## UNION DISTINCT [2.0]

(no contents yet)

## UPDATE

Changes the data in all or part of an existing row in a table, view, or active set of a cursor.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
ESQL form:
UPDATE [TRANSACTION transaction] {table | view}
 SET col = <val> [, col = <val> ...]
 [WHERE <search_condition> | WHERE CURRENT OF cursor];
DSQL and ISQL form:
UPDATE {table | view}
 SET col = <val> [, col = <val> ...]
 [WHERE <search_condition>
<val> = {
 col [<array_dim>]
 | :variable
 | <constant>
 | <expr>
 | <function>
 | udf ([<val> [, <val> ...]])
 | NULL
 | USER
 | ?
}
 [COLLATE collation]
<array_dim> = [[x:]y [, [x:]y ...]]
<constant> = num | 'string' | _charsetname 'string'
<function> = CAST (<val> AS <datatype>)
 | UPPER (<val>)
 | GEN_ID (generator, <val>)
<expr> = A valid SQL expression that results in a single value.
<search_condition> = See CREATE TABLE for a full description.
Notes on the UPDATE statement
o  In SQL and ISQL, you cannot use val as a parameter placeholder (like "?"
).
o  In DSQL and ISQL, val cannot be a variable.
o  When you need to qualify a constant string with a specific character set
, prefix the character set name with an underscore, to indicate the name is
not a regular SQL identifier. You must use, for example:
 _WIN1252 'This is my string';
o  You cannot specify a COLLATE clause for blob columns
.
```

(This text is currently not included because of possible copyright issues.)

**See also:** DELETE, GRANT, INSERT, REVOKE, SELECT

# UPDATE OR INSERT [2.1]

Updates or inserts a record depending on whether it is already present

**Availability:** +DSQL +ESQL +ISQL +PSQL

## Syntax

```
UPDATE OR INSERT INTO <table or view> [(<column_list1>)]
VALUES (<value_list>)
[MATCHING <column_list2>]
[RETURNING <column_list3> [INTO <variable_list>]]
```

**Important**
INSERT and UPDATE permissions are needed on <table or view>. A "multiple rows in singleton select" error will be raised if the RETURNING clause is present and more than one record matches the search condition.

| Argument | Description |
|---|---|
| <table or view> | the table or view where the update or insert takes place |
| <column_list1> | optional list of fields to update or insert |
| <value_list> | list of field values to update or insert |
| <column_list2> | list of fields that determine whether or not the record already exists |
| <column_list3> | optional list of returned values (see RETURNING) |
| <variable_list> | optional list of variables where the RETURNING values are returned into |

## Description

This syntax has been introduced to enable a record to be either updated or inserted, according to whether or not

it already exists (checked with IS NOT DISTINCT).

When MATCHING is omitted, the existence of a primary key is required.

If the RETURNING clause is present, then the statement is described as

isc_info_sql_stmt_exec_procedure by the API; otherwise, it is described as isc_info_sql_stmt_insert.

### Examples

```
In the first example it is assumed that T1 has a primary key (e.g. on F1):
1. UPDATE OR INSERT INTO T1 (F1, F2)
   VALUES (:F1, :F2);

The second example returns the updated or inserted ID:
2. UPDATE OR INSERT INTO EMPLOYEE (ID, NAME)
   VALUES (:ID, :NAME)
   RETURNING ID;

Here the decision to INSERT or to UPDATE is based on F1, be it the primary
key or not:
3. UPDATE OR INSERT INTO T1 (F1, F2)
   VALUES (:F1, :F2)
   MATCHING (F1);

In this example, in case ID already existed, the OLD contents of field NAME
is returned:
4. UPDATE OR INSERT INTO EMPLOYEE (ID, NAME)
   VALUES (:ID, :NAME)
   RETURNING OLD.NAME;
```

**See also:** INSERT, UPDATE, RETURNING

## UPPER( )

Converts a string to all uppercase.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
UPPER (<val>)
```

(This text is currently not included because of possible copyright issues.)

**See also:** CAST( ) and the user-defined (external) function lower( )

## WHENEVER

Traps SQLCODE errors and warnings.

**Availability:** DSQL ESQL ISQL PSQL

### Syntax

```
WHENEVER {NOT FOUND | SQLERROR | SQLWARNING}
   {GOTO label | CONTINUE};
```

(This text is currently not included because of possible copyright issues.)

## *WITH [RECURSIVE] (CTE) [2.1]*

(no contents yet)

# Document history

**Revision History**

| | | |
|---|---|---|
| 0.1 | FI | First Beta |

# License note