DECLARE

Available in: ESQL, PSQL

Description

Declares a PSQL local variable.

Syntax

```
DECLARE [VARIABLE] varname <var_spec>;

<var_spec> ::= <type> [NOT NULL] [<coll>] [<default>]
               | CURSOR FOR (select-statement)
<type>     ::= sql_datatype | [TYPE OF] domain
<coll>     ::= COLLATE collation
<default>  ::= {= | DEFAULT} value
```

- If sql_datatype is a text type, it may include a character set.
- Obviously, a COLLATE clause is only allowed with text types.

DECLARE ... CURSOR

Added in: 2.0

Description

Declares a named cursor and binds it to its own SELECT statement. The cursor can later be opened, used to walk the result set, and closed again. Positioned updates and deletes (using WHERE CURRENT OF) are also supported. PSQL cursors are available in triggers, stored procedures and EXECUTE BLOCK statements.

Example

```
execute block
returns (relation char(31), sysflag int)
as
declare cur cursor for
  (select rdb$relation_name, rdb$system_flag from rdb$relations);
begin
  open cur;
  while (1=1) do
  begin
    fetch cur into relation, sysflag;
    if (row_count = 0) then leave;
    suspend;
  end
  close cur;
```

```
end
```

Notes:

- A FOR UPDATE clause is allowed in the SELECT statement, but not required for a positioned update or delete to succeed.
- Make sure that declared cursor names do not clash with any names defined later on in AS CURSOR clauses.
- If you need a cursor to loop through an output set, it is almost always easier – and less error-prone – to use a FOR SELECT statement with an AS CURSOR clause. Declared cursors must be explicitly opened, fetched from, and closed. Furthermore, you need to check row_count after every fetch and break out of the loop if it is zero. AS CURSOR takes care of all of that automatically. However, declared cursors give you more control over the sequence of events, and allow you to operate several cursors in parallel.
- The SELECT statement may contain named SQL parameters, like in "select name || :sfx from names where number = :num". Each parameter must be a PSQL variable that has been declared previously (this includes any in/out params of the PSQL module). When the cursor is OPENed, the parameter will be assigned the current value of the variable.
- Caution! If the value of a PSQL variable that is used in the SELECT statement changes during execution of the loop, the statement may (but will not always) be re-evaluated for the remaining rows. In general, this situation should be avoided. If you really need this behaviour, test your code thoroughly and make sure you know how variable changes affect the outcome. Also be advised that the behaviour may depend on the query plan, in particular the use of indices. As it is currently not strictly defined, it may change in some future version of Firebird.

See also:

- OPEN cursor
- FETCH cursor
- CLOSE cursor

back to top of page

DECLARE [VARIABLE] with initialization

Changed in: 1.5

Description

In Firebird 1.5 and above, a PSQL local variable can be initialized upon declaration. The VARIABLE keyword has become optional.

Example

```
create procedure proccie (a int)
returns (b int)
as
  declare p int;
  declare q int = 8;
```

```
  declare r int default 9;
  declare variable s int;
  declare variable t int = 10;
  declare variable u int default 11;
begin
  <intelligent code here>
end
```

back to top of page

DECLARE with DOMAIN instead of datatype

Added in: 2.1

Description

In Firebird 2.1 and above, PSQL local variables and input/output parameters can be declared with a domain instead of a data type. The TYPE OF modifier allows using only the domain's datatype and not its NOT NULL setting, CHECK constraint and/or default value.

Example

create procedure MyProc (a int, f ternbool)

```
 returns (b int, x type of bigfloat)
```

as

```
 declare p int;
 declare q int = 8;
 declare y stocknum default -1;
```

begin

```
 <very intelligent code here>
```

end (This example presupposes that TERNBOOL, BIGFLOAT and STOCKNUM are domains already defined in the database.)

Warning: If you change a domain's definition, existing PSQL code using that domain may become invalid. For information on how to detect this, please read the note The RDB$VALID_BLR field, near the end of this document.

back to top of page TYPE OF COLUMN in variable declaration Added in: 2.5

Description

Analogous to the TYPE OF domain syntax supported since version 2.1, it is now also possible to declare variables and parameters as having the type of an existing table or view column. Only the

type itself is used; in the case of string types, this includes the character set and the collation. Constraints and default values are never copied from the source column.

Example

create table cars (

```
make varchar(20),
model varchar(20),
weight numeric(4),
topspeed numeric(3),
constraint uk_make_model unique (make, model)
```

)

create procedure max_kinetic_energy

```
                                (make type of column cars.make,
                                 model type of column cars.model)
                  returns (max_e_kin double precision)
```

as

```
declare mass type of column cars.weight;
declare velocity type of column cars.topspeed;
```

begin

```
select weight, topspeed from cars
  where make = :make and model = :model
  into mass, velocity;
max_e_kin = 0.5 * mass * velocity * velocity;
```

end Warnings:

The collation of the source column is not always taken into consideration when comparisons (e.g. equality tests) are made, even though it should. This is due to a bug that has been fixed for Firebird 3. PSQL code using TYPE OF COLUMN may become invalid if the column's type is changed at a later time. For information on how to detect this, please read the note The RDB$VALID_BLR field?, near the end of this document. back to top of page COLLATE in variable declaration Added in: 2.1

Description

In Firebird 2.1 and above, a COLLATE clause is allowed in the declaration of text-type PSQL local variables and input/output parameters.

Example

create procedure GimmeText

```
returns (txt char(32) character set utf8 collate unicode)
```

as

```
declare simounao mytextdomain collate pt_br default 'não';
```

begin

```
<extremely intelligent code here>
```

end back to top of page NOT NULL in variable declaration Added in: 2.1

Description

In Firebird 2.1 and above, a NOT NULL constraint is allowed in the declaration of PSQL local variables and input/output parameters.

Example

create procedure Compute(a int not null, b int not null)

```
returns (outcome bigint not null)
```

as

```
declare temp bigint not null;
```

begin

```
<slightly disappointing code here>
```

end

From:
http://ibexpert.com/docu/ - **IBExpert**

Permanent link:
http://ibexpert.com/docu/doku.php?id=01-documentation:01-09-sql-language-references:firebird2.5-language-reference-update:psql-statements:declare

Last update: **2023/08/02 17:54**