

IF/Prolog V5.2

Generic SQL Interface

Is there

anything you would like to tell us about this manual?
Please send us your comments.

Siemens AG Austria
PSE KB B2
Gudrunstrasse 11
A-1100 Vienna
Austria

Fax.: +43-1-1707 56992

email: prolog@siemens.at

The information in this document is subject to change and does not represent a commitment on the part of Siemens AG Austria. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X/Open and the X device are trademarks of X/Open Company Ltd.

Copyright ©Siemens AG Austria, 1999. All rights reserved.

The reproduction, transmission, translation or exploitation of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. Delivery subject to availability; right of technical modifications reserved.

Contents

Contents	iii
1 IF/SQL Interface	1
1.1 Introduction	2
1.1.1 Cursor Concept	2
1.1.2 Dynamic SQL Commands	2
1.1.3 Interpretation of Dynamic SQL Commands	2
1.1.4 Data Conversion	3
1.1.5 Data Types	3
1.1.6 Data Retrieval	3
1.1.7 Bind Variables	4
1.1.8 Return Codes	4
1.1.9 Dual SQL Commands	4
1.1.10 No Prolog Specialities	4
1.1.11 Prolog Exception Handling	4
1.2 How to use dynamic SQL commands in IF/Prolog	5
1.3 IF/SQL Predicates Reference	6
sql_begin/1 – Begin INFORMIX transaction	7
sql_close/2 – Close cursor	8
sql_close_db/1 – Close INFORMIX database	9
sql_commit/1 – Commit transaction	10
sql_connect_db/2/3/4 – Connect to INGRES database	11
sql_declare/5 – Declare cursor	12

sql_descr_in/2 – Describe input variables	14
sql_descr_out/2 – Describe output variables	16
sql_disconnect_db/1 – Disconnect from INGRES database	17
sql_dual_cmd/2 – Dual cursor	18
sql_errmsg/1 – Retrieve SQL error message	19
sql_execute/2 – Execute command	20
sql_fetch/4 – Read database record	21
sql_fetch_buf/2 – Read database record	22
sql_fetch_n/5 – Read database records	23
sql_get_val/5 – Extract value	24
sql_inout/3 – Query bind and select variables	25
sql_inquire_ingres/4 – Retrieve INGRES error information	26
sql_logoff/0 – Logout from ORACLE	27
sql_logon/3 – Logon to ORACLE	28
sql_open/2 – Open cursor	29
sql_open_db/2/3 – Open INFORMIX database	30
sql_rollback/1 – Rollback transaction	31
1.4 Examples	32
1.4.1 A Dynamic SQL DELETE Command	32
1.4.2 A Dynamic SQL INSERT Command	33
1.4.3 A Dynamic SQL SELECT Command	34

Index

Chapter 1

IF/SQL Interface

”Embedded SQL” Interface for IF/Prolog

The IF/SQL predicates added to IF/Prolog provide a set of functions to manage SQL commands to a Relational Database Management System (RDBMS). The IF/SQL predicates are available for all RDBMS which support the ANSI standard for the **Embedded SQL** Pre-compiler Interface. At present they are available for ORACLE/PRO*C, INFORMIX/ESQL and INGRES/ESQL. Other relational databases will follow in the future.

The generic database interface is distributed as source in the installation directory *\$PRO-ROOT/DEMOS/sql*. The interface can be configured by editing the **Makefile** for the corresponding database system (e.g. *oracle7.mk* for Oracle Database System Version 7.x).

The IF/SQL predicates were released with IF/Prolog 4.0. Its functionality replaces the ORACLE CALL Interface released with IF/Prolog 3.X.

1.1 Introduction

The IF/SQL predicates reflect the functionality of the **Embedded SQL** Precompiler Programming Interface for 3GL programming languages such as FORTRAN, COBOL, Pascal and C.

1.1.1 Cursor Concept

SQL commands can be executed at once or they are associated with a cursor. A **cursor** logically identifies an SQL command as long as it is activated.

The **Cursor concept** of **Embedded SQL** is a concept to navigate within sets of data records to retrieve or manipulate data or, more generally, to manipulate a set of database operations. **Manipulation** of an SQL command here means to declare and pre-translate, instantiate, execute, reinstantiate, reexecute, or to forget the command.

In this way, complex database operations can be performed from an application with the maximum of efficiency.

Several SQL commands can be executed **in parallel** independent from standard Prolog semantics. This multiple cursor concept is known from **Embedded SQL** for C, COBOL, Fortran and Pascal. Now it may be combined with the ability of symbolic expressions, pattern matching and backtracking within Prolog.

1.1.2 Dynamic SQL Commands

An SQL command is called **dynamic** if the SQL command string is yet unknown at compile time. A dynamic SQL command may be regarded as function with formal parameters for input and output, pretranslated by the RDBMS. Input parameters are place-holders in an SQL command string, called **bind variables**. Output parameters are elements of the select list in an SQL SELECT command.

After declaration the function can be repeatedly called with different instantiations for the bind variables.

1.1.3 Interpretation of Dynamic SQL Commands

Dynamic SQL commands called via **Embedded SQL** from C, Pascal or COBOL etc. need explicit program code to manage **SQL descriptor areas**, memory allocation, alignment, indices for data elements, specifications of data types, etc. and need precompilation, compilation and linkage after each modification of the program code.

Using the IF/SQL predicates allows runtime declaration of SQL commands within the IF/Prolog interpreter. The management of the **SQL descriptor areas**, memory allocation,

database type specification, etc. are completely hidden from the user and SQL commands became part of symbolic programming.

After runtime declaration dynamic SQL commands can be interpretatively instantiated and executed as often as you like.

1.1.4 Data Conversion

The conversion of data from the database communication buffers (SQL descriptor areas) into Prolog terms and vice-versa is supported.

Values provided within Prolog to be instantiated for bind variables are automatically converted to the data format expected by the RDBMS.

For data retrieval (with the SELECT command) it is possible to define which Prolog data type the retrieved values should be converted to. If no definition is given data are passed as atoms with default length as defined to the RDBMS.

1.1.5 Data Types

Standard RDBMS data types that are provided are **CHAR(N)**, **NUMBER(N)**, **NUMBER(N, M)** and **DATE**.

Depending on the RDBMS, further data types are supported:

- INGRES
INTEGER, SMALLINT, FLOAT, CHAR(N), VARCHAR(N), DATE, MONEY
- ORACLE
LONG
- INFORMIX
DECIMAL, SMALLFLOAT, SMALLINT, SERIAL, MONEY

This means that, e.g. in ORACLE, up to 64 KB text data can be stored in one database data field.

1.1.6 Data Retrieval

Data retrieval is performed by first fetching a database record into a logical (output) buffer and then transferring values from it into the application. Complete data records may be gotten as Prolog lists as well as single data elements referred to by name or its relative number within the output buffer.

For all data elements, indicators are generated to control the case of NULL values and truncation of strings.

A special IF/SQL predicate (**sql_fetch_n/5**) also allows to fetch many tuples at a time within Prolog.

1.1.7 Bind Variables

Within a dynamic SQL command string some bind variables might occur that parametrize the SQL command. That makes it possible to specify or modify at runtime, e.g. the WHERE clause for a command or actual values for an INSERT or UPDATE command, without re-translation.

In a SQL statement, bind variable names for ORACLE are prefixed by a colon ':'. Bind variables for INFORMIX and INGRES have to be a question mark '?'.

Bind variables must be instantiated before execution of the SQL command. For each bind variable an instantiation value must be provided in the logical input buffer associated with the SQL command. Examples are given below.

1.1.8 Return Codes

IF/SQL predicates that call SQL commands provide an argument for the return code from RDBMS. Some error codes that are related to the IF/SQL Interface have been added.

A return code value < 0 indicates that a database error has occurred. For ORACLE a particular error message can be retrieved by the predicate `sql_errmsg/1`. For error codes from INFORMIX it must be referred to the Informix Manual. INGRES error messages can be retrieved by `sql_inquire_ingres/4`.

1.1.9 Dual SQL Commands

A dynamic SQL command with bind variables may be defined as **dual** to a SELECT command. That is, the output buffer of the SELECT command is automatically used as initial input buffer of the dual command. No instantiation or only partial modification of the dual input buffer must be performed before execution of the dual command is possible. See predicate `sql_dual_cmd/2`.

1.1.10 No Prolog Specialities

IF/SQL predicates for IF/Prolog are independent from Prolog's special semantics. All IF/SQL predicates are **primitive** Prolog builtin predicates. They do not backtrack.

Backtrackable SQL predicates like those from IF/Prolog's previous ORACLE CALL interface, can easily be implemented within Prolog itself.

1.1.11 Prolog Exception Handling

Violation of input parameter argument types in IF/SQL predicates result in exceptions as usual within IF/Prolog.

1.2 How to use dynamic SQL commands in IF/Prolog

A complete transaction of dynamic database operations performed with IF/SQL predicates consists of the following steps:

1. Declaration of an SQL command, pretranslation by the RDBMS, definition of a cursor for it (**sql_declare/5**).
2. Instantiation of the bind variables of the SQL command if some exist (**sql_descr_in/2**).
3. Execution of the command (**sql_open/2**).
4. Access to data, transfer into the application (for SQL SELECT commands only):
 - (a) **sql_fetch_buf/2** and **sql_get_value/5**
 - (b) **sql_descr_out/2** and **sql_fetch/4** or **sql_fetch_n/5**
5. Commit of transaction (**sql_commit/1**) or rollback of transaction (**sql_rollback/1**) after some SQL INSERT, SQL UPDATE or SQL DELETE commands.
6. Releasing the cursor for the command (**sql_close/2**).

Any SQL command (user privileges, data definition, data retrieval and data manipulation, etc.) can be executed in this way.

Steps 1, 2, 3, 4, 5, 6 must occur in this order.

Only steps 1 and 3 must always be executed.

Step 2 is only necessary for commands with bind variables.

Steps 4a or 4b only make sense for SQL SELECT commands.

Steps 2, 3, 4 can be executed as often as you like.

Different database operations, i.e. commands with different cursors can be executed in **parallel** as you like, that is they can be interleaved or nested within one another.

Cursor = 'immediate' has a special meaning for a call to **sql_declare/5**. It executes the given SQL command at once after translation. (Bind or select variables cannot exist!) It is equivalent to **sql_execute(+SqlCmd, -RC)**, which performs interpretative execution of the command by the RDBMS. You can use the predicate **sql_execute/2** for SQL commands that must be called only once. You can also compare the performance with that of dynamic SQL commands.

1.3 IF/SQL Predicates Reference

The arguments of the following predicates should be or become instantiated as follows:

<i>Role:</i>	<i>Prolog Type:</i>
Cursor	atom
SqlCmd	atom
Tuple	list of fields: [Value1, Value2, ...]
Tuple_list	list of tuples: [Tuple1, Tuple2, ...]
Ind	indicator value: integer $N \geq -1$
IndTuple	list of indicators: Ind1, Ind2, ...
Type	one of {integer, double, atom, atom(N), date}
RC	SQL return code: integer

Begin INFORMIX transaction

`sql_begin(-RC)`

This predicate starts a transaction for an INFORMIX database with a transaction log file.

Example

```
?- sql_begin(RC).
```

```
RC      = 0
```

Close cursor

`sql_close(+Cursor, -RC)`

This predicate releases the internal data structures associated with *Cursor*. The corresponding SQL command is no more activated.

$RC = 0$ indicates success.

$RC < 0$ indicates an error.

Example

```
?- sql_close(c1, RC).
```

```
RC      = 0
```

Close INFORMIX database

`sql_close_db(-RC)`

This predicate closes an INFORMIX database for use.

Example

```
?- sql_close_db(RC) .
```

```
RC      = 0
```

Commit transaction

`sql_commit(-RC)`

This predicate performs end of transaction. The database in its current state is written to disk.

For an INFORMIX database a transaction log file must have been defined.

Example

```
?- sql_commit(RC).
```

```
RC      = 0
```

Connect to INGRES database

```
sql_connect_db(+DbName, -RC)
```

```
sql_connect_db(+DbName, +User, -RC)
```

```
sql_connect_db(+DbName, +User, +Flags, -RC)
```

These predicates establish a connection with an INGRES database. Normally your login userid is also used as INGRES user. You may overwrite this with the *User* argument.

RC = 0 indicates success.

RC < 0 indicates an INGRES error.

Example

```
?- sql_connect_db(test, RC).
```

```
RC      = 0
```

Declare cursor

sql_declare(+Cursor, +SqlCmd, -NoIn, -NoOut, -RC)

The SQL command string given as atom *SqlCmd* is parsed and *Cursor* is associated to it.

NoIn specifies the number of bind variables within *SqlCmd*. *NoOut* gives the number of the SELECT list variables.

RC = 0 indicates success.

RC < 0 indicates an SQL error (see **sql_errmsg/1**).

The SQL keywords within *SqlCmd* may be given in lowercase or uppercase letters.

Bind variables names have to be preceded by a colon ':' (ORACLE) or have to be a '?' (INFORMIX and INGRES).

The select list variables have to be columns of the retrieved table or expressions built from such columns, such as min(<column>), <column>+1, etc.

For the names of bind and select list variables see **sql_inout/2**.

sql_declare(immediate, SqlCmd, _, _, RC) is equivalent to **sql_execute(SqlCmd, RC)**.

This executes the SQL command at once after translation. Bind or select variables cannot exist.

Example

```
%      ORACLE
?- _SqlCmd = 'SELECT ename, dep FROM EMP where dep=20',
   sql_declare(c2, _SqlCmd, NoIn, NoOut, RC).

NoIn    = 0
NoOut   = 2
RC      = 0

?- _SqlCmd = 'select * from emp where ename = :name ',
   sql_declare(c3, _SqlCmd, NoIn, NoOut, RC).

NoIn    = 1
NoOut   = 6
RC      = 0

?- _SqlCmd = 'insert into emp2 (ename, dep) values (:x, :y)',
   sql_declare(c4, _SqlCmd, NoIn, NoOut, RC).

NoIn    = 2
NoOut   = 0
RC      = 0
```

```
%      INFORMIX, INGRES
?- _SqlCmd = 'SELECT ename, dep FROM EMP where dep=20',
   sql_declare(c2, _SqlCmd, NoIn, NoOut, RC).

NoIn    = 0
NoOut   = 2
RC      = 0

?- _SqlCmd = 'select * from emp where ename = ?',
   sql_declare(c3, _SqlCmd, NoIn, NoOut, RC).

NoIn    = 1
NoOut   = 6
RC      = 0

?- _SqlCmd = 'insert into emp2 (ename, dep) values (?, ?)',
   sql_declare(c4, _SqlCmd, NoIn, NoOut, RC).

NoIn    = 2
NoOut   = 0
RC      = 0
```

Describe input variables

`sql_descr_in(+Cursor, +ValueSpec)`

This predicate performs instantiation of bind variables to the command associated with *Cursor*.

ValueSpec may be one of:

N = *Value*

Name = *Value*

or a list of specifications, e.g.

[*N1* = *Value1*, *Name2* = *Value2*, ...].

ORACLE bind variables:

Bind variables can be referenced to by a relative number *N* or by *Name* (without the colon ':'). Lowercase or uppercase letters within *Name* are not significant.

INFORMIX and INGRES bind variables:

Bind variables have to be referenced to by a relative number *N*.

NULL values can be specified by the Prolog Term '[]'. Note that ORACLE does not allow NULL values for an SQL INSERT command (only for SQL UPDATE).

Example

The following examples refer to bind variables of an SQL command like:

```
'select * from emp where dep = :dep and job = :job'
```

```
?- sql_descr_in(c1, job = 'Manager').
```

```
yes
```

```
?- sql_descr_in(c1, 'JOB' = 'Musician').
```

```
yes
```

```
?- sql_descr_in(c1, 2 = 20).
```

```
yes
```

```
?- sql_descr_in(c1, [ dep = 20, 1 = 'Musician' ] ).
```

```
yes
```

```
?- sql_descr_in(c1, 2 = [] ).
```

```
yes
```

Describe output variables

`sql_descr_out(+Cursor, +TypeSpec)`

This predicate defines the data type of data elements to be fetched later from the logical output buffer associated to *Cursor* via `sql_fetch/4` or `sql_fetch_n/5`.

TypeSpec is one of

$N_i = Type_i$,

$Column_i = Type_i$,

a list of *TypeSpec*, e.g. [$N_1 = Type_1, Column_2 = Type_2, \dots$] ,

or a list of *Types*, [$Type_1, Type_2, \dots$].

$Type_i$ is one of `integer`, `double`, `atom`, `atom(N)` or `date`. The order of arguments is significant only in the latter form of *TypeSpec*. Default for $Type_i$ is `atom`.

`atom(N)` restricts the length of an atom string to *N*.

Example

The following examples refer to bind variables of an SQL command like:

```
'select ename, depno, hiredate from emp
where depno = :dep and hiredate = :date'
```

```
?- sql_descr_out(c2, 1=atom).
```

```
yes
```

```
?- sql_descr_out(c2, dep=double).
```

```
yes
```

```
?- sql_descr_out(c2, [ 1=atom(10), dep=double, 3=date ] ).
```

```
yes
```

```
?- sql_descr_out(c2, [ atom(10), double, date ] ).
```

```
yes
```

Disconnect from INGRES database

`sql_disconnect_db(-RC)`

This predicate disconnects from an INGRES database which was previously used.

Example

```
?- sql_disconnect_db(RC).
```

```
RC      = 0
```

Dual cursor

`sql_dual_cmd(+Cursor1, +Cursor2)`

The logical output buffer associated to *Cursor1* is defined as logical input buffer for *Cursor2*.

Cursor1 must have been declared for a SELECT command. *Cursor2* must have been declared for an SQL command with bind variables that correspond to the select variables list to *Cursor1*.

Having fetched a record into the output buffer for *Cursor1*, the input buffer for *Cursor2* now is automatically initialized. No instantiation or only partial modification of the "dual" input buffer must be performed before execution of *Cursor2* is possible.

Example

```
?- sql_dual_cmd(c2, c4).
```

```
yes
```

For the declaration of the cursors *c2* and *c4* see the examples given for `sql_declare/5`.

Retrieve SQL error message

`sql_errmsg(-Atom)`

This predicate retrieves the SQL error message of the last SQL predicate. If the last predicate returned 0, *Atom* will be unified with ”.

This predicate is provided for ORACLE and INFORMIX only.

Example

```
?- sql_errmsg(Msg).
```

```
Msg      = 'ORA-0942:table or view does not exists'
```

Execute command

`sql_execute(+SqlCmd, -RC)`

This predicate executes interpretation of the SQL command *SqlCmd*. You may compare its performance with the execution of dynamic SQL commands.

SqlCmd cannot contain bind or select variables.

If another predicate for a specific purpose exists, it should be used instead of `sql_execute/2`. E.g. you should use `sql_open_db/2` to open an INFORMIX database.

Example

```
?- sql_execute('create index on emp (depno)', RC).
```

```
RC      = 0
```

Read database record

sql_fetch(+Cursor, -Tuple, -IndTuple, -RC)

This predicate reads the first or next data record from the query result into the logical output buffer associated with *Cursor*. The corresponding SELECT command must have been executed before with **sql_open/2**.

Tuple will be unified with a list of the data elements. *IndTuple* will be unified with a list of corresponding indicator values.

RC = 1 indicates a successful fetch.

RC = 0 if no (more) record was found.

RC < 0 indicates an SQL error (see **sql_errmsg/1**).

Example

```
?- sql_fetch(c2, Tuple, IndTuple, RC).  
  
Tuple    = ['Smith',20,date(89, 12, 31)]  
IndTuple = [0,0,0]  
RC       = 1
```

Read database record

sql_fetch_buf(+*Cursor*, -*RC*)

This predicate reads the first or next data record from the query result into the logical output buffer associated with *Cursor*. The corresponding SELECT command must have been executed before with **sql_open/2**. Together with **sql_get_value/5** it offers an alternative method for data retrieval.

RC = 1 indicates successful fetch.

RC = 0 if no (more) data found.

Example

```
?- sql_fetch_buf(c2, RC).
```

```
RC      = 1
```

Read database records

sql_fetch_n(+Cursor, ?N, -TupleList, -IndTupleList, -RC)

This predicate reads the next N data records from the query result into the logical output buffer associated with *Cursor*. The corresponding SELECT command must have been executed before with **sql_open/2**.

TupleList will be unified with a list of tuples, one value tuple list for each record.

IndTupleList will be unified with a list of corresponding indicator value lists.

$RC > 0$ gives the number of tuples found.

$RC = 0$ if no (more) record was found.

$RC < 0$ indicates an SQL error (see **sql_errmsg/1**).

Example

```
?- sql_fetch_n(c2, N, Tuples, IndTuples, RC).
```

```
Tuples = [ ['Smith',20,date(89,12,31)],
            ['Meyer',20,date(88,12,6)],
            ['Pipenbrinc',0,'']
          ]
IndTuples = [ [0,0,0], [0,0,0], [10,-1,-1] ]
N          = 3
RC         = 3
```

Extract value

`sql_get_val(+Cursor, +Column, +Type, -Value, -Ind)`

This predicate extracts the current value for *Column* out of the logical output buffer associated with *Cursor*. One (or more) corresponding call(s) to `sql_fetch_buf/2` must have been successfully performed before.

Value will be converted to *Type*. *Type* must be one of `integer`, `double`, `atom`, `atom(N)` or `date`.

Ind returns an indicator value for *Value*:

Ind = -1 indicates a NULL value.

Ind = 0 indicates success.

Ind > 0 indicates truncation of a string.

Example

```
?- sql_get_val(c2, ename, atom, Value, Ind).
```

```
Value = 'Musician'  
Ind   = 0
```

```
?- sql_get_val(c2, dep, double, Value, Ind).
```

```
Value = 20.0  
Ind   = 0
```

```
?- sql_get_val(c2, hiredate, date, Value, Ind).
```

```
Value = date(89,12,31)  
Ind   = 0
```

```
?- sql_get_val(c2, hiredate, atom, Value, Ind).
```

```
Value = '31-DEC-89'  
Ind   = 0
```

Query bind and select variables

`sql_inout(+Cursor, -BindVarList, -ColumnList)`

The names of bind and select variables of the SQL command associated with *Cursor* will be unified with *BindVarList* and *ColumnList*.

Example

```
?- sql_inout(c1, L1, L2).
```

```
L1      = [job,dep]
```

```
L2      = []
```

```
?- sql_inout(c2, L1, L2).
```

```
L1      = []
```

```
L2      = [ename,depno,hiredate]
```

Retrieve INGRES error information

`sql_inquire_ingres(-ErrorText, -ErrorNo, -RowCount, -EndQuery)`

This predicate can be used to ask for full error message after the previous INGRES SQL command returned a negative error code. See your INGRES SQL Reference Manual for more information.

Example

```
?- sql_execute('select * from emp', RC).
```

```
RC      = -2350
```

```
yes
```

```
?- sql_inquire_ingres(Text, No, Row, End).
```

```
Text    = 'E_US092E EXECUTE IMMEDIATE:  
          a SELECT statement cannot be a target of this command.'
```

```
No      = 2350
```

```
Row     = -1
```

```
End     = 0
```

Logout from ORACLE

sql_logoff

This predicate performs logout from ORACLE.

Example

```
?- sql_logoff.
```

Logon to ORACLE

`sql_logon(+Userid, +Passwd, -RC)`

This predicate performs a logon to ORACLE.

RC = 0 indicates success.

RC < 0 indicates an ORACLE error (see `sql_errmsg/1`).

Example

```
?- sql_logon(scott, tiger, RC).
```

```
RC      = 0
```

Open cursor

`sql_open(+Cursor, -RC)`

This predicate executes the SQL command associated with *Cursor*.

RC = 0 indicates success.

RC < 0 indicates an SQL error (see `sql_errmsg/1`).

If a declared SQL command string contains bind variables, they must be instantiated with `sql_descr_in/2` before execution.

Example

```
?- sql_open(c2, RC).
```

```
RC      = 0
```

Open INFORMIX database

`sql_open_db(+DbName, -RC)`

`sql_open_db(+DbName, -RC, +Mode)`

These predicates open an INFORMIX database for use in exclusive or nonexclusive mode.

The argument *Mode* is an atom `exclusive` or `nonexclusive` (default).

DbName is expected to be a directory name of an INFORMIX database.

RC = 0 indicates success.

RC < 0 indicates an INFORMIX error (see INFORMIX manual).

Example

```
?- sql_open_db(stores, RC).
```

```
RC      = 0
```

Rollback transaction

`sql_rollback(-RC)`

This predicate performs a rollback of transaction. It results in an undo of all data manipulation done since the last commit (ORACLE and INGRES) or begin of transaction (INFORMIX).

For an INFORMIX database a transaction log file must have been defined.

Example

```
?- sql_rollback(RC).
```

```
RC      = 0
```

1.4 Examples

The examples below use ORACLE syntax. Other database systems might have similar but different syntax.

1.4.1 A Dynamic SQL DELETE Command

```
?- SqlCmd = 'DELETE FROM EMP WHERE ENAME = :name',  
   sql_declare(c1, SqlCmd, NoIn, NoOut, RC).
```

```
NoIn    = 1  
NoOut   = 0  
RC      = 0
```

The SQL command is parsed and can later be identified by the cursor **c1**.

When the **sql_declare/5** predicate has successfully completed its parsing ($RC = 0$) it returns: the number *NoIn* of bind variables and the number *NoOut* of select variables, that is how many columns will be output.

```
?- sql_descr_in(c1, 'name' = 'SCOTT'),  
   sql_open(c1, RC).
```

```
RC      = 0
```

```
?- sql_descr_in(c1, 'NAME' = 'SMITH'),  
   sql_open(c1, RC).
```

```
RC      = 0
```

```
?- sql_descr_in(c1, 1 = 'WALLACE'),  
   sql_open(c1, RC).
```

```
RC      = 0
```

Data tuples in the 'EMP' table that exist for employees 'SCOTT', 'SMITH' and 'WALLACE', are deleted.

Bind variables can be referred to by name or relative number.

1.4.2 A Dynamic SQL INSERT Command

```
?- SqlCmd = 'INSERT INTO EMP (ENAME, JOB, DEPNO, SALARY)
           VALUES (:name, :dep, :sal)',
   sql_declare(c2, SqlCmd, NoIn, NoOut, RC).
```

```
NoIn    = 3
NoOut   = 0
RC      = 0
```

```
?- sql_descr_in(c2, [ 1='SMITH', sal=29500.50, dep=20 ] ).
```

The bind variables are provided with values.

```
?- sql_open(c2, RC).
```

```
RC      = 0
```

A tuple for 'SMITH' is added to the table 'EMP'. The columns which haven't been named are assigned to NULL values.

```
?- sql_descr_in(c2, [ 1='SCOTT', sal='61098.0', dep='20' ] ).
```

Bind variables can also be instantiated by any atomic terms.

```
?- sql_open(c2, RC).
```

```
RC      = 0
```

A second tuple for an employee 'SCOTT' is added.

1.4.3 A Dynamic SQL SELECT Command

```
?- _SqlCmd = 'SELECT ENAME, DEPNO, HIREDATE, SALARY FROM EMP
           WHERE ENAME=:name OR (DEPNO = :dep AND SALARY >= :sal)',
   sql declare(c3, _SqlCmd, N1, N2, RC).
```

```
N1      = 3
N2      = 4
RC      = 0
```

This SQL SELECT command contains 3 bind variables for input and 4 select variables for output.

```
?- sql_inout(c3, In, Out).
```

```
In      = [name, dep, sal]
Out     = [ename, depno, hiredate, salary]
```

The names of bind and select variables can be obtained with the predicate **sql_inout/3**.

```
?- BindSpec = [ sal=29999.99, dep=20, name='SCOTT' ],
   sql_descr_in(c3, BindSpec).
```

The WHERE clause of the SELECT command is specified.

```
?- sql_open(c3, RC).
```

```
RC      = 0
```

The SELECT command is executed by the RDBMS.

```
?- TypeSpec = [ depno=integer, hiredate=date, salary=double ],
   sql_descr_out(c3, TypeSpec).
```

The types for the application's retrieval of columns is specified. Type **atom** is default.

```
?- sql_fetch(c3, Tuple, Warnings, RC).
```

```
Tuple   = ['SMITH', 30, date(87, 12, 7), 29500.50]
Warnings = [0, 0, 0, 0]
RC      = 1
```

The first record of data is retrieved.

```
?- sql_fetch_n(c3, N, TupleList, IndTupleList, RC).
```

```
N          = 5
TupleList   = [['SCOTT', 20, '', 61098.0], ...]
IndTupleList = [[0, 0, -1, 0], ...]
RC         = 5
```

`sql_fetch_n/5` will fetch a maximum of N data tuples. If N is uninstantiated when calling `sql_fetch_n/5`, all data tuples found will be fetched and N (and RC) is set to their number.

Otherwise, if N is instantiated it specifies a maximum value for the number of data tuples to be fetched. $0 \leq RC \leq N$ then gives the actual number of tuples found.

For 'SCOTT' there is no HIREDATE available, so a NULL value is returned and the corresponding indicator is set to -1.

Without having to re-parse the SELECT command the selection criteria can be modified using `sql_descr_in/2` and further database queries can be executed as follows:

```
?- sql_descr_in(c3, [ ... ] ),
   sql_open(c3, 0),
   sql_descr_out(c3, [ ... ] ),
   Max = 4,
   repeat,
   sql_fetch_n(c3, Max, Tuples, _, RC),
   perform_data(RC, Tuples),
   RC < Max,
   !,
   ...
```

```
?- sql_close(c3, 0).
```

If the application likes to retrieve only single data elements at a time, it can be done as follows:

```
?- sql_descr_in(c3, [ ... ] ),
   sql_open(c3, 0),
   sql_descr_out(c3, [ ... ] ),
   repeat,
   sql_fetch_buf(c3, RC),
   (
     RC = 1,
     ...
     sql_get_val(c3, ename, atom, Name, Ind1),
     sql_get_val(c3, salary, double, Salary, Ind2),
     ...
```

```
    fail
;
    RC = 0,
    !
).
```

```
?- sql_close(c3, 0).
```

Index

- cursor
 - close, 8
 - declare, 12
 - dual, 18
 - input variables, 14
 - open, 29
 - output variables, 16
- database
 - close, 9
 - connect, 11
 - disconnect, 17
 - open, 30
 - read, 21–23
- error message, 19, 26
- execute command, 20
- extract value, 24
- INFORMIX
 - begin transaction, 7
 - close database, 9
 - open database, 30
- INGRES
 - connect to database, 11
 - disconnect from database, 17
 - error message, 26
- ORACLE
 - logon, 28
 - logout, 27
- SQL
 - error message, 19
- transaction
 - begin, 7
 - commit, 10
 - rollback, 31
- variables
 - query, 25