Native SQL & its use with DATABASE CONNECTION in SAP

Version 1

created by Bhaskar Tripathi on Dec 28, 2012 6:59 PM, last modified by Bhaskar Tripathi on Dec 29, 2012 3:51 PM

NATIVE SQL in SAP

By Bhaskar Tripathi
29-Dec-2012

Introduction

Open SQL is used to access database table defined in ABAP dictionary irrespective of the database platform the R/3 system is based. In contrast, native SQL statements are used to access database tables not defined in ABAP dictionary and hence allow us to integrate data not part of R/3 system.

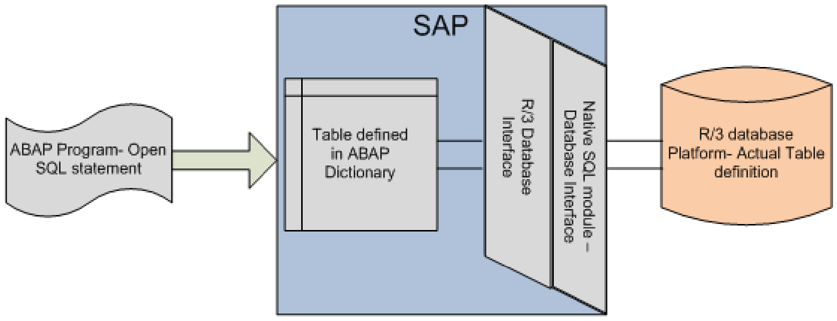
SAP
ABAP Program- Open SQL statement
Table defined in ABAP Dictionary
R/3 Database Interface
Native SQL module – Database Interface
R/3 database Platform- Actual Table definition

Figure 1.1 Database access through Open SQL statement

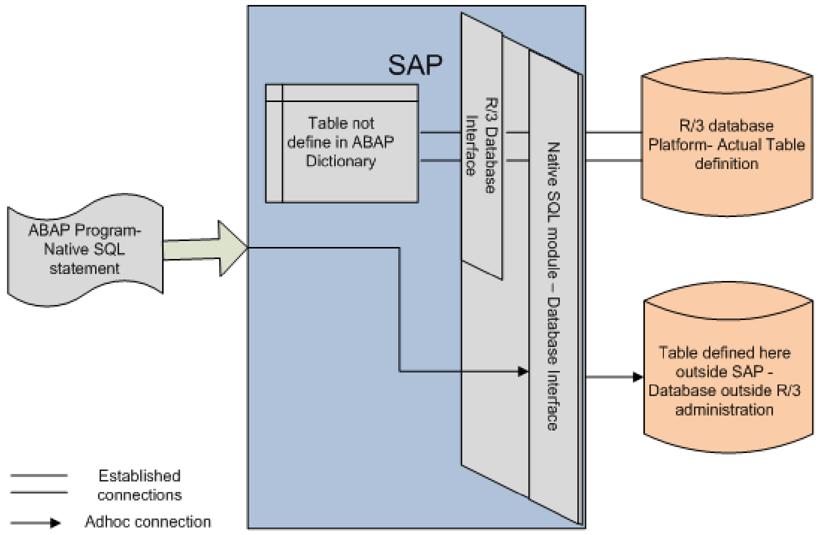
SAP
Table not define in ABAP Dictionary
R/3 Database Interface
Native SQL module – Database Interface
R/3 database Platform- Actual Table definition
ABAP Program- Native SQL statement
Table defined here outside SAP - Database outside R/3 administration
Established connections
Adhoc connection

Figure 1.2 Database access (database outside SAP administration) through Native SQL statement bypassing R/3 Database Interface

Since native SQL are database specific statements, program containing these statements will not run in different database systems.

This document is intended to describe the statements which are used with Oracle database.

The document divides the topic into following sections:

1. Executing Native SQL statements.
2. Establishing/disconnecting connection to external database.
3. Selecting data from a table
4. Inserting/Updating table entry
5. Statements for common operations
6. Handling exception from native SQL statements

Executing Native SQL statements

To indicate the ABAP processor that the following statement is native SQL statement and hence to bypass R/3 database interface, we need to precede such statement by EXEC SQL and followed by ENDEXEC as shown below:

```
EXEC SQL [PERFORMING ABAP_ROUTINE].
      SELECT   pernr
                  INTO :lv_pernr
      FROM PERNRTAB
ENDEXEC.
```

The addition PERFORMING is applicable only with the SELECT native SQL statement (also, not applicable in OO ABAP context) and calls the routine ABAP_ROUTINE after each successful call of SELECT statement. This addition is further explained in section “Selecting data from a table”.

Establishing/disconnecting connection to external database

Before communicating with external database, connection must be established with it. Connection with external database is established using the statement:

```
EXEC SQL.
  CONNECT TO dbs [AS con]
ENDEXEC.
```

The addition **AS** can be used to name the connection. **dbs** is the name of the external database system. We store this name in SAP database table DBCON (field ‘CON_NAME’). **dbs** is specified either as literal or using a variable. When a variable is used with dbs, it should be preceded by a colon ‘:’. This representation of variable in EXEC SQL statement is called as **Host Variable**. For example,

```
EXEC SQL.
  CONNECT TO ‘EXT_DB’
ENDEXEC.
```

Above statement can be written as:

```
DATA: lv_dbs.

lv_dbs = ‘EXT_DB’.
EXEC SQL.
  CONNECT TO :lv_dbs
ENDEXEC.
```

Disconnection to external database is done using the statement:

```
EXEC SQL.
  DISCONNECT dbs.
ENDEXEC.
```

Selecting data from a table

Once the connection is established, a SELECT statement can be used to fetch data from external database. A single record extraction can be done using the statement:

```
EXEC SQL [PERFORMING <form_routine>].
  SELECT   <column_name>
      INTO <host_variable>
      FROM <table_name>
      WHERE <sql_condition>
ENDEXEC.
```

For example,

```
EXEC SQL.
  SELECT   position
      INTO :lv_position
      FROM ZRESPERNR
      WHERE pernr = '10874138'
ENDEXEC.
```

The above statement tries to fetch position of the employee whose pernr is 10874138 in table ZRESPERNR.

But if we want to extract multiple records, we have to modify the statement as:

```
EXEC SQL PERFORMING add_rec.
  SELECT   position
      INTO :lv_position
      FROM ZRESPERNR
      WHERE pernr = '10874138'
ENDEXEC.

FORM add_rec.
  "Display or store LV_POSITION locally.
ENDFORM.
```

PERFORMING addition implicitly opens a cursor for the SELECT statement following it and also calls the specified form routine. This way we can fetch multiple records. However, **this is not allowed in ABAP objects** since global classes do not have access to global data and subroutines of the calling ABAP program.

For ABAP objects, we have to explicitly open cursor to extract multiple records. Following example explain this.

```
DATA: lv_pernr       TYPE pernr_d,
           gt_zrespernr TYPE STANDARD TABLE OF pernr_d.
EXEC SQL.
  OPEN cursor FOR
  SELECT pernr FROM ZRESPERNR
ENDEXEC
DO.
  CLEAR lv_pernr.
EXEC SQL.
    FETCH NEXT c1 INTO :lv_pernr
  ENDEXEC.
  IF sy-subrc <> 0.
    EXIT.
  ELSE.
    APPEND lv_pernr TO gt_zrespernr.
  ENDIF.
ENDDO.
EXEC SQL.
  CLOSE cursor
```

```
ENDEXEC.
```

Inserting/Updating table entry

A new record is inserted with an **INSERT statement** which has the following syntax:

```
INSERT INTO<table_name>
  ( <field 1>, <field 2>,...<Field n> )
  VALUES
  ( <val 1>, <val 2>,...<val n> )
```

Where <field n> represents name of the field and <valn> the value for the field. Value <val n> can be specified as literal or as host variable. Also, any function (producing output value which is compatible to target field format) valid in the given external database can be used in place of <val n>.

Following example inserts a new record in table ZRESPERNR:

```
DATA:  lv_pernr TYPE pernr_d.
lv_pernr = '10007588'.

EXEC SQL.
INSERT INTO ZRESPERNR
         ( PERNR, NAME, POSITION, DOJ )
      VALUES
         ( :lv_pernr, 'STEVE JOB', 'CEO', TO_DATE(:sy-datum,'MM/DD/YYYY') )
ENDEXEC.
```

Records in a table laying in the external database are updated using **UPDATE statement**:

```
UPDATE <table_name>
  SET <field 1> = <val 1>, <field 2> = <val 2>...<fieldn> = <val n>
  WHERE <sql_condition>
```

Following example updates the value in field POSITION for records where current position name is 'CEO'.

```
DATA: lv_ceo TYPE char20.
lv_ceo = 'CEO'.
EXEC SQL.
UPDATE ZRESPERNR
  SET POSITION = 'CFO'
  WHERE POSTION = :lv_ceo.
ENDEXEC.
```

Statements for common operations

Aggregate Function: With native SQL, we can use various aggregate functions which are very useful while selecting rows from external database. There are five such aggregate functions:

- Count function - COUNT function returns the number of rows in a table satisfying the criteria specified in the WHERE clause. Eg. the following statement returns total number or employees whose employee number is < '10007483'.

```
      DATA:lv_rows TYPE i.
      EXEC SQL.
        SELECT   COUNT(pernr)
            INTO :lv_rows
            FROM ZRESPERNR
            WHERE pernr < '10007483'
```

```
ENDEXEC.
```

- Sum function - SUM function returns the sum of all selected columns. Eg. the following statement returns total of annual salaries of employees whose employee number is < ‘10007483’.

```
DATA: lv_amount TYPE p length 15 DECIMALS 2.
EXEC SQL.
SELECT   SUM(ansal)
    INTO :lv_amount
    FROM ZRESPERNR
   WHERE pernr < '10007483'
ENDEXEC.
```

- Avg function - AVG function calculates the average value of a column of numeric type. Eg. the following statement returns average annual salaries of employees whose employee number is < ‘10007483’.

```
DATA: lv_amount TYPE p length 15 DECIMALS 2.
EXEC SQL.
  SELECT   AVG(ansal)
      INTO :lv_amount
      FROM ZRESPERNR
     WHERE pernr < '10007483'
ENDEXEC.
```

- Max function - MAX is used to find the maximum value or highest value of a certain column. Eg. the following statement returns largest annual salary among the employees whose employee number is < ‘10007483’.

```
DATA: lv_amount TYPE p length 15 DECIMALS 2.
EXEC SQL.
  SELECT   MAX(ansal)
     INTO :lv_amount
     FROM ZRESPERNR
    WHERE pernr < '10007483'
ENDEXEC.
```

- Min function - MIN is used to find the minimum value or lowest value of a certain column. Eg. the following statement returns smallest annual salary among the employees whose employee number is < ‘10007483’.

```
DATA: lv_amount TYPE p length 15 DECIMALS 2.
EXEC SQL.
  SELECT   MIN(ansal)
     INTO :lv_amount
     FROM ZRESPERNR
    WHERE pernr < '10007483'
ENDEXEC.
```

Note: Above functions also have many variations which are not discussed here.

System Date & Time: Sometimes we need system (external database) current date and time in our program. We can fetch system date & time using table DUAL. DUAL is a temporary table maintained in Oracle data dictionary having exactly one row and used to select pseudo columns. Following example following code snippet fetches system date and time from external database.

```
DATA: lv_oracle_date TYPE c LENGTH 40,
      lv_oracle_timestamp TYPE c LENGTH 40.

EXEC SQL.
  select CURRENT_DATE
  from dual
  into :lv_oracle_date
ENDEXEC.

EXEC SQL.
  select CURRENT_TIMESTAMP
    from dual
    into :lv_oracle_timestamp
ENDEXEC.

WRITE: ‘DATE: ‘, lv_oracle_date.
WRITE: / ‘TIMESTAMP: ‘, lv_oracle_timestamp.
```

Handling exception from native SQL statements

Handling errors/exception while working with native SQL statements is very important as various checks, such as source and target data type compatibility check are not performed during compilation.
When a native SQL statement fails, it sets the value of SY-SUBRC to non-zero. We use this value to raise an exception of type CX_SY_NATIVE_SQL_ERROR. We can then catch the same exception and import the error message to display in our program. Following example explains this.

```
DATA: lv_exc_ref TYPE REF TO cx_sy_native_sql_error,
      lv_error_text TYPE string.
TRY.

  EXEC SQL.
      CONNECT TO :lv_store_dbs
  ENDEXEC.
  IF sy-subrc <> 0.
    RAISE EXCEPTION TYPE cx_sy_native_sql_error.
  ENDIF.

  CATCH cx_sy_native_sql_error INTO lv_exc_ref.
    CLEAR: lv_error_text.
    lv_error_text  = lv_exc_ref->get_text( ).
    MESSAGE lv_error_text  TYPE ‘I’.
ENDTRY.
```

Summary

Native SQL is used to access data from external database. In this document we saw the architecture of SAP in the context of data access through open SQL and native SQL. We then saw how to execute native SQL statements and how to establish/disconnect connections to external database. We took example of Oracle SQL and discussed statements for selecting table rows, inserting/updating data into tables, useful functions while selecting data and how
to handle errors/exceptions.

References:

1. http://help.sap.com
2. Personal Experience

8928 Views

Average User Rating

(1 rating)

G+1 0 Twe Like 1

2 Comments

Suhas Saha Dec 31, 2012 6:37 AM

SAP recommends the usage of ADBC (ABAP Database Connectivity) to access the external DBs.

The advantages of ADBC being:

1. Class-based access to external DB
2. Dynamic access which was not possible via Native SQL.

Cheers,
Suhas

Like (0)

Jack Chow Chee Yong May 31, 2013 3:48 AM (in response to Suhas Saha)

Dear all,

Is there any other solution for Dynamic access like dynamic table?

Like (0)