SQL with other Programming Languages

Murali Mani

Why?

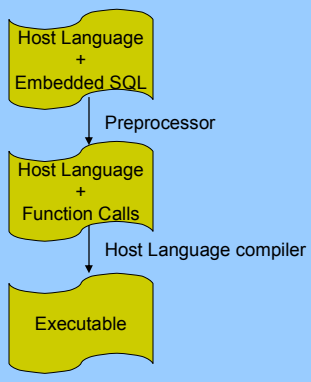
- SQL is not for general purpose programming.
- Eg: Suppose we have a table R (a, factorialA), we want to populate the table as follows:
  - The user inputs a, the program should compute the factorial of a and insert these 2 attributes as 1 row in R.
- We need to integrate SQL with general purpose Programming Language such as COBOL, Fortran, C, Java …

Murali Mani

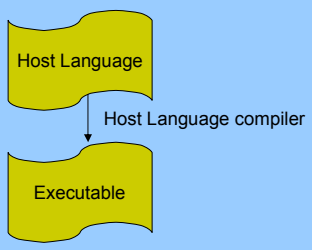
Solutions

- Embedded SQL (eg: Pro*C, SQLJ …)
  - SQL Embedded in **host language**
  - Preprocessor converts SQL statement to API calls
  - A host language compiler compiles the resulting code.
- Call Level Interface (CLI) (eg: JDBC, ODBC, PHP …)
  - Libraries of functions provided explicitly for SQL statements
  - No preprocessor, in stead host language compiler compiles the code.

Architecture



Embedded SQL architecture



CLI architecture

Embedded SQL: Main Constructs

- Connect to DB

  ```
  EXEC SQL CONNECT
  ```

- Declare variables that can be used by both SQL and host language

  ```
  EXEC SQL BEGIN DECLARE SECTION
  …
  EXEC SQL END DECLARE SECTION
  ```

- Executing SQL statements

  ```
  EXEC SQL …
  ```


Embedding SQL in C: Oracle

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlca.h>

EXEC SQL BEGIN DECLARE SECTION;
   VARCHAR userid[20];
   VARCHAR passwd[20];
   int value;
EXEC SQL END DECLARE SECTION;

void sql_error (char *msg) {
   printf ("%s", msg); exit (1);
}
```


Embedding SQL in C: Oracle

```
int main () {
   strcpy (userid.arr, "mmani");
   userid.len = strlen (userid.arr);
   strcpy (passwd.arr, "mmani");
   passwd.len = strlen (passwd.arr);

   EXEC SQL WHENEVER SQLERROR DO sql_error ("Oracle Error\n");
   EXEC SQL CONNECT :userid IDENTIFIED BY :passwd;

   EXEC SQL CREATE TABLE Test (a int);
   EXEC SQL INSERT INTO Test VALUES (1);

   EXEC SQL SELECT MAX (a) INTO :value from R;
   printf ("Max value=%d\n",value);
}
```


Cursors

```
EXEC SQL DECLARE myCursor CURSOR FOR SELECT pNumber,
   pName from Professor;
EXEC SQL OPEN myCursor;
EXEC SQL WHENEVER NOT FOUND DO break;
while (1) {
   EXEC SQL FETCH myCursor INTO :num, :name;
   …
}
EXEC SQL CLOSE myCursor;
```

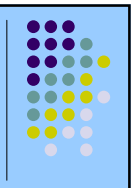

Updating with Cursors

```
EXEC SQL DECLARE myCursor CURSOR FOR SELECT pNumber,
  pName from Professor FOR UPDATE OF pName;
EXEC SQL OPEN myCursor;
EXEC SQL WHENEVER NOT FOUND DO break;
while (1) {
  EXEC SQL FETCH myCursor INTO :num, :name;
  …
  EXEC SQL UPDATE Professor SET pName='X' WHERE
  CURRENT OF myCursor;
}
EXEC SQL CLOSE myCursor;
EXEC SQL COMMIT;
```


Checking if a SQL statement returned null

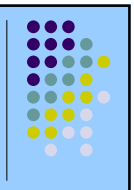
```
short isNullNumber;
…
EXEC SQL FETCH myCursor into :num:isNullNumber,
  :name;
if (isNullNumber == -1) {
  printf ("pNumber is null\n");
}
…
```


Compiling



- Create files with extension .pc, such as test.pc
- Preprocessor = proc
  - Available at `$ORACLE_HOME/bin`
- SET your library path as
  - `setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${ORACLE_HOME}/lib`

Compiling



- We will make them as

```
make -f
$ORACLE_HOME/precomp/demo/proc/demo_proc.mk
build EXE=test OBJS=test.o
```

(or)

```
$ORACLE_HOME/bin/proc iname=test.pc code=cpp
parse=none
g++ -I$ORACLE_HOME/precomp/public test.c
    -L$ORACLE_HOME/lib -lclntsh -lm
```


To find about SQL states

- Use the special variable called SQLSTATE, SQLCODE
- Also there are macros such as SQLERROR, SQLWARNING, NOT FOUND

Scrollable cursors

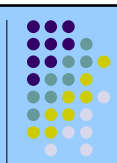
- We may need cursors that can go to any position in the result set, go back, go over the result set multiple times etc.
- Define a scrollable cursor as

```
EXEC SQL DECLARE myCursor SCROLL CURSOR FOR
  select sNumber, sName from student;
```

- We fetch from scrollable cursor as

```
EXEC SQL FETCH RELATIVE 2 myCursor
INTO :snumber:isNullNumber, :sname;
```


Scrollable cursors

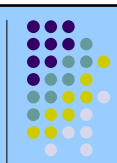


- We can use the following to move the cursor "pointer" around
  - NEXT to give next tuple
  - PRIOR to give previous tuple
  - FIRST to give first tuple in result set
  - LAST to give last tuple in result set
  - RELATIVE <num> where <num> is any positive or negative integer. RELATIVE 1 = NEXT, RELATIVE -1 = PRIOR
  - ABSOLUTE <num>, if num is positive, we count from first, otherwise we count from last. ABSOLUTE 1 = FIRST, ABSOLUTE -1 = LAST

Other Embedded SQL

- Embed SQL in Java for Oracle: SQLJ
  - SQLJ is part of SQL standard !!
- Similar to Embedding SQL in C
- Check Sample code

Call Level Interface (CLI)



- CLI
  - In Embedded SQL, the code depended on the DBMS vendor.
- CLI Examples
  - ODBC (Open Database Connectivity)
  - JDBC (Java Database Connectivity)

JDBC passing parameters to queries

```
PreparedStatement myStmt =
   myCon.prepareStatement ("INSERT INTO Student
   (sNumber, sName) VALUES (?, ?)");
myStmt.setInt (1, num);
myStmt.setString (2, name);

(or)
String query = "INSERT INTO Student (sNumber,
   sName) VALUES (" + num + ", '" + name + "')";
```


Tips while using JDBC

- DML modifications are usually committed (unlike ProC)
- Look up `interface java.sql.Connection`
  - `commit ()`
  - `rollback ()`
  - `setAutoCommit ()`