EMBEDDED SQL

Part 1: Static Statements

University of Waterloo
List of Slides

1
2 Database Applications
3 How does client/server work?
4 Embedded SQL
5 Embedded SQL (cont.)
6 Application Structure
7 Declarations
8 Host Variables
9 Errors
10 Dummy Application (DB2)
11 Preparing your Application (DB2)
12 Example of a build (DB2)
13 Summary on Preparing an Application
14 “Real” SQL Statements
15 Simple Application
16 Simple Application (cont.)
17 NULLs and Indicator Variables
18 Impedance Mismatch
19 Application with a Cursor
20 Application with a Cursor (cont.)
21 Cursors and Updates
22 Example
23 Summary
Database Applications

- SQL isn't sufficient to write general applications.
  ⇒ connect it with a general-purpose PL!

- Language considerations:
  ⇒ Library calls (CLI/ODBC)
  ⇒ Embedded SQL
  ⇒ Advanced persistent PL (usually OO)

- Client-server:
  ⇒ SQL runs on the server
  ⇒ Application runs on the client
How does client/server work?

<table>
<thead>
<tr>
<th>Application</th>
<th>DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>main() {</td>
<td>dbms-start</td>
</tr>
<tr>
<td>CONNECT to DB</td>
<td>make connection</td>
</tr>
<tr>
<td>SQL STATEMENT</td>
<td>execute query</td>
</tr>
<tr>
<td>get answer(s)</td>
<td>commit changes</td>
</tr>
<tr>
<td>COMMIT</td>
<td>disconnect</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>
Embedded SQL

- SQL Statements are *embedded* into a *host language* (C, C++, FORTRAN, ...)

- The application is *preprocessed*
  pure host language program + library calls

  ⇒ Advantages:
  * Preprocessing of (static) parts of queries
  * MUCH easier to use

  ⇒ Disadvantages:
  * Needs precompiler
  * Needs to be *bound* to a database
Embedded SQL (cont.)

- Considerations:
  - How much can SQL be parameterized?
    - How to pass parameters into SQL?
    - How to get results?
    - Errors?
  - Static vs. dynamic SQL statements.
- How much does the DBMS know about an application?
  - precompiling: PREP
  - binding: BIND
    - in ORACLE both done using the proc tool
Application Structure

#include SQL support (SQLCA, SQLDA)

int main(int argc, char **argv)
{
    // Declarations
    // Connect to Database
    // Do your work
    // Process errors
    // Commit/Abort and Disconnect

    //
}
Declarations

- Include SQL communication area:

  ```sql
  EXEC SQL  INCLUDE SQLCA;
  ```

- it defines:
  - the return code of SQL statements (sqlcode)
  - the error messages (if any)
  - ... you can’t live without it.

- SQL statements inserted using magic words
  ```sql
  EXEC SQL <sql statement>
  ```
Host Variables

...are used to pass values between a SQL statement and the rest of the program:

- parameters in SQL statements:
  
  communicate **single values**
  
  between SQL and the host language variable

- must be declared within SQL declare section:
  
  ```
  EXEC SQL BEGIN DECLARE SECTION;
  
  declarations of variables to be used in SQL statements go here
  
  EXEC SQL END DECLARE SECTION;
  ```

- can be used in the **EXEC SQL** statements:
  
  ⇒ to distinguish them from SQL identifiers
  
  they are preceded by `:` (colon)
Errors

What if a SQL statement fails?

- check `sqlcode != 0`
- use “exception” handling:
  ```sql
  EXEC SQL WHENEVER SQLERROR GO TO lbl;
  EXEC SQL WHENEVER SQLWARNING GO TO lbl;
  EXEC SQL WHENEVER NOT FOUND GO TO lbl;
  ```
  → designed for COBOL (lbl has to be in scope).
#include <stdio.h>
#include "util.h"

EXEC SQL INCLUDE SQLCA;

int main(int argc, char *argv[]) {

    EXEC SQL BEGIN DECLARE SECTION;
    char db[6] = "cs448"
    EXEC SQL END DECLARE SECTION;

    printf("Sample C program: CONNECT\n");

    EXEC SQL WHENEVER SQLERROR GO TO error;

    EXEC SQL CONNECT TO :db;

    printf("Connected to DB2\n");

    EXEC SQL COMMIT;
    EXEC SQL CONNECT reset;
    exit(0);

error:
    check_error("My error", &sqlca);
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK;
    EXEC SQL CONNECT reset;
    exit(1);
}
Preparing your Application (DB2)

1. write the application in a file called `<name>.sqc`

2. preprocess the application:
   
   `db2 prep <name>.sqc`

3. compile the application:
   
   `cc -c -O <name>.c`

4. link with DB2 libraries:
   
   `cc -o <name> <name>.o -L... -l...`

5. run it:
   
   `./<name> [arguments]`

USE the provided Makefile

⇒ sets options
⇒ knows the path(s) and libraries
Example of a build (DB2)

bash$ make NAME=sample1

db2 connect to cs448

Database Connection Information

Database server = DB2/SUN 6.1.0
SQL authorization ID = DAVID
Local database alias = CS448

db2 prep sample1.sqc bindfile

LINE MESSAGES FOR sample1.sqc
----- -----------------------------------------------
SQL0060W The "C" precompiler is in progress.
SQL0091W Precompilation or binding was ended with
"0" errors and "0" warnings.

db2 bind sample1.bnd

LINE MESSAGES FOR sample1.bnd
----- -----------------------------------------------
SQL0061W The binder is in progress.
SQL0091N Binding was ended with "0" errors and
"0" warnings.

db2 connect reset

DB20000I The SQL command completed successfully.

c -I/usr/db2/include -c sample1.c

c -I/usr/db2/include -o sample1 sample1.o util.o
   -L/usr/db2/lib -R/usr/db2/lib -ldb2
Summary on Preparing an Application

1. Source Files With SQL Statements

2. Precompiler (db2 PREP)
   - PACKAGE Create a Package
   - BINDFILE Create a Bind File

3. Host Language Compiler

4. Host Language Linker

5. Binder (db2 BIND)

6. Executable Program

Database Manager Package (Package)
“Real” SQL Statements

So far we introduced only the surrounding infrastructure. Now for the real SQL statements:

- simple statements:
  - “constant” statements
  - statements with parameters
  - statements returning a single tuple

- general queries with many answers

- dynamic queries
Simple Application

Write a program that for each publication id supplied as an argument prints out the title of the publication:

```c
main(int argc, char *argv[]) {
  ...

  printf("Connected to Oracle\n");

  for (i=1; i<argc; i++) {
    strncpy(pubid,argv[i],8);

    EXEC SQL WHENEVER NOT FOUND GO TO nope;

    EXEC SQL SELECT title INTO :title
      FROM publication
      WHERE pubid = :pubid;

    printf("%10s: %s\n",pubid,title);
    continue;
    nope:
      printf("%10s: *** not found *** \n",pubid);
  }

  EXEC SQL COMMIT RELEASE;
  exit(0);

  ...
}
```
Simple Application (cont.)

bash$ ./sample2 ChTo98 nopubid
Sample C program: SAMPLE2
  ChTo98: Temporal Logic in Information Systems
  nopubid: *** not found ***

⇒ it is important that at most
  one title is returned for each pubid.
NULLs and Indicator Variables

- what if a host variable is assigned a NULL?
  ⇒ not a valid value in the datatype
  ⇒ ESQL uses an extra Indicator variable, e.g.:

  `smallint ind;`

  ```
  SELECT firstname INTO :firstname
  INDICATOR :ind
  FROM ...
  ```

  then if \texttt{ind} < 0 then \texttt{firstname} is NULL

- if the indicator variable is not provided and the result is a null we get an error

- the same applies for updates.
Impedance Mismatch

- What if we `EXEC SQL` a query and it returns more than one tuple?

- Solution: use a `cursor`:
  1. Declare a `cursor`:
     ```sql
     EXEC SQL DECLARE <name> CURSOR FOR <query>;
     ```
  2. Iterate over it:
     ```sql
     EXEC SQL OPEN <name>;
     EXEC SQL WHENEVER NOT FOUND GO TO end;
     for (;;) {
       <set up host parameters>
       EXEC SQL FETCH <name> INTO <host variables>;
       <process the fetched tuple>
     }
     end:
     EXEC SQL CLOSE <name>;
```
Application with a Cursor

Write a program that lists all author names and publication titles where the author name matches a pattern given to the applications as an argument:

```c
main(int argc, char *argv[]) {
    ...

    strncpy(apat,argv[1],8);

    EXEC SQL DECLARE author CURSOR
        FOR SELECT name, title
            FROM author, wrote, publication
            WHERE name LIKE :apat
                AND aid=author
                AND pubid=publication;

    EXEC SQL OPEN author;
    EXEC SQL WHENEVER NOT FOUND GO TO end;
    for (;;) {
        EXEC SQL FETCH author INTO :name, title;
        printf("%10s -> %20s: %s\n",apat,name,title);
    }
    end:
    EXEC SQL COMMIT RELEASE;
    exit(0);
    ...}
```
Application with a Cursor (cont.)

bash$ ./sample3 "%"
Sample C program: SAMPLE3
  % -> Toman, David : Temporal Logic in Information
  % -> Toman, David : Datalog with Integer Periodic
  % -> Toman, David : Point-Based Temporal Extensio
  % -> Chomicki, Jan : Logics for Databases and Info
  % -> Chomicki, Jan : Datalog with Integer Periodic
  % -> Chomicki, Jan : Temporal Logic in Information
  % -> Saake, Gunter : Logics for Databases and Info
bash$ ./sample3 "T%"
Sample C program: SAMPLE3
  T% -> Toman, David : Temporal Logic in Information
  T% -> Toman, David : Datalog with Integer Periodic
  T% -> Toman, David : Point-Based Temporal Extensio
Cursors and Updates

- cursors iterate over tuples in the answer
- you can change the tuple the cursor points to
  ⇒ remember updating views? (same rules here)
- the value to be changed has to be specified in the cursor declaration:

  \[
  \text{EXEC SQL DECLARE } <\text{name}> \text{ CURSOR}
  \]
  \[
  \text{FOR } <\text{query}>
  \]
  \[
  \text{FOR UPDATE } [ \text{ OF } <\text{attribs}> ];
  \]

- the actual change:

  \[
  \text{EXEC SQL FETCH } <\text{cursor}> \text{ INTO } <\text{vars}>;
  \]
  \[
  \text{if } <\text{cond on variables}>
  \]
  \[
  \quad \text{EXEC SQL UPDATE } <\text{cursor}> \text{ SET } ... \n  \]
  \[
  \quad \text{WHERE CURRENT OF } <\text{name}>;
  \]
  
  ⇒ the \text{UPDATE} must
  match the cursor declaration.
Example

main(int argc, char *argv[]) {

    EXEC SQL DECLARE author CURSOR
        FOR SELECT name
            FROM author
            WHERE url IS NULL
        FOR UPDATE OF url;

    EXEC SQL OPEN author;
    EXEC SQL WHENEVER NOT FOUND GO TO end;

    for (; ;) {
        EXEC SQL FETCH author INTO :name;
        printf("Author '%s' has no URL\n", name);
        printf("Enter new URL to fix or <cr> to delete: ");
        gets(url);
        if (strcmp(url,"")==0) {
            printf("Deleting '%s'\n",name);
            EXEC SQL DELETE FROM author
                WHERE CURRENT OF author;
        } else {
            printf("Setting URL for '%s' to '%s'\n",name,url);
            EXEC SQL UPDATE author
                SET url = :url
                WHERE CURRENT OF author;
        }
    }
end:

}
Summary

- Declarations:

  EXEC SQL INCLUDE SQLCA;
  EXEC SQL BEGIN DECLARE SECTION;
  <host variables here>
  EXEC SQL END DECLARE SECTION;

- Simple statements:

  EXEC SQL <SQL statement>;

- Queries (with multiple answers)

  EXEC SQL DECLARE <id> CURSOR FOR <qry>;
  EXEC SQL OPEN <id>;
  do {
    EXEC SQL FETCH <id> INTO <vars>;
  } while (SQLCODE == 0);
  EXEC SQL CLOSE <id>;

- Don’t forget to check errors!!