EMBEDDED SQL

Part 2: Dynamic Statements

University of Waterloo
List of Slides

1 Dynamic SQL
2 Dynamic SQL: a Roadmap
3 EXECUTE IMMEDIATE
4 PREPARE
5 Parametric Statements
6 Simple statement: EXECUTE
7 Query with many answers: CURSOR
8 Unknown number/types of variables??
9 SQLDA: a description of tuples
10 SQLDA (cont.)
11 DESCRIBE
12 SQLDA (cont.)
13 SQLDA and parameter passing
14 Putting it together: adhoc.sqc
15 adhoc.sqc (cont.)
16 adhoc.sqc (cont.)
17 adhoc.sqc (cont.)
18 Example
19 Summary
Dynamic SQL

**GOAL:** to *execute* a string as a SQL statement.

Problems:

- How do we know a string is a valid statement?  
  ⇒ parsing and compilation?

- How do we execute
  ⇒ queries? (where does the answer go?)
  ⇒ updates? (how many rows affected?)

- What is we don’t know anything about the string?

. . . we develop an “adhoc” application that accepts an SQL statement as an argument and executes it (and prints out the answer, if any).
Dynamic SQL: a Roadmap

- STRING
- PREPARE
- STATEMENT
- EXECUTE
- DECLARE CURSOR
- EXECUTE IMMEDIATE
- # ROWS
- DESCRIBE
- SQLDA
- OPEN/FETCH/CLOSE
- TUPLES
- # ROWS
Execution of non-parametric statements **without answer(s)**:

```
EXEC SQL EXECUTE IMMEDIATE :string;
```

where `:string` is a host variable containing the ASCII representation of the query.

- `:string` may not return an answer nor contain parameters (see later).
- used for constant statements executed only once
  
  \[ \Rightarrow :\text{string} \text{ is } compiled \text{ every time we pass through.} \]
Often we better compile a :string into a stmt:

EXEC SQL PREPARE stmt FROM :string;

- used for repeatedly executed statements (avoids recompilation each time we want to execute them)
- :string may be a query (and return answers).
- :string may contain parameters.
- stmt is not a host variable but an identifier of the statement used by the preprocessor (careful: can’t be used in recursion!)
Parametric Statements

- Static embedded SQL
  - host variables as parameters
  - mostly used in **WHERE** clause(s)

- Dynamic SQL (strings) and **parameters**?
  - we can change the string (recompilation)
  - use **parameter marker**: "?" in the string

  Values for "?"s are substituted when the statement is to be executed
Simple statement: EXECUTE

EXEC SQL EXECUTE stmt;
    USING :var1 [,\ldots,:vark];

- for statements that don’t return tuples
  ⇒ database modification (\textbf{INSERT}, \ldots)
  ⇒ transactions (\textbf{COMMIT})
  ⇒ data definition (\textbf{CREATE} \ldots)

- values of \texttt{:var1}, \ldots, \texttt{:vark} are substituted for the parameters (in order of appearance)
  ⇒ mismatch causes SQL runtime error!

- \texttt{sqlca.sqlerrd[2]} contains the number of affected tuples (for updates)
Embedded SQL: Dynamic Statements: 8

Query with many answers: CURSOR

EXEC SQL DECLARE cname CURSOR FOR stmt;
EXEC SQL OPEN  cname
    USING :var1 [,....,:vark];
EXEC SQL FETCH cname
    INTO  :out1 [,....,:outn];
EXEC SQL CLOSE cname;

- for queries we use cursor (like in the static case).
- :var1,...,:vark – supply query parameters.
- :out1,...,:outn – store the resulting tuple.
Unknown number/types of variables??

We need a dynamic descriptor area.

The standard says:

- `ALLOCATE DESCRIPTOR descr`
- `GET DESCRIPTOR descr what`
- `SET DESCRIPTOR descr what`
  
  where `what` is
  
  ⇒ get/set the value for `COUNT`
  
  ⇒ get/set value for `i`-th attribute: `VALUE :i assgn`

  you can use use `DATA`, `TYPE`, `INDICATOR`, ...

- `DESCRIBE [INPUT|OUTPUT] stmt INTO descr`

In practice we have to use a `sqlda` descriptor explicitly...
SQLDA: a description of tuples

\texttt{sql\texttt{d}a} is a SQL description area that defines a single tuple (this is how the DBMS communicates with the application in the end).

- It contains (among other things):
- The string ‘SQLDA’ (for identification)
- Number of allocated entries for attributes
- Number of actual attributes; 0 if none
- For every attribute
  1. (numeric code of) type
  2. length of storage for the attribute
  3. pointer to a data variable
  4. pointer to a indicator variable
  5. name (string and its length)
SQLDA (cont.)

A modern version (DB2):

```c
struct sqlname /* AttributeName */
{
    short length; /* Name length [1..30] */
    char data[30]; /* Variable or Column name */
};

struct sqlvar /* Attribute Descriptor */
{
    short sqltype; /* Variable data type */
    short sqllen; /* Variable data length */
    char *SQL_POINTER sqldata; /* data buffer */
    short *SQL_POINTER sqlind; /* null indicator */
    struct sqlname sqlname; /* Variable name */
};

struct sqlda /* Main SQLDA */
{
    char sqldaid[8]; /* Eye catcher = 'SQLDA ' */
    long sqldabc; /* SQLDA size in bytes=16+44*SQLN */
    short sqln; /* Number of SQLVAR elements */
    short sqld; /* Number of used SQLVAR elements */
    struct sqlvar sqlvar[1]; /* first SQLVAR element */
};
```
SQLDA (cont.)

An archaic version (ORACLE):

```c
struct SQLDA {
    long   N;  /* Descriptor size in number of entries */
    char  *V[];  /* Arr of addresses of main variables (data) */
    long   L[];  /* Arr of lengths of data buffers */
    short  T[];  /* Arr of types of buffers */
    short  *I[];  /* Arr of addresses of indicator vars */
    long   F;  /* Number of variables found by DESCRIBE */
    char  *S[];  /* Arr of variable name pointers */
    short  M[];  /* Arr of max lengths of attribute names */
    short  C[];  /* Arr of current lengths of attribute names */
    char  *X[];  /* Arr of indicator name pointers */
    short  Y[];  /* Arr of max lengths of ind. names */
    short  Z[];  /* Arr of cur lengths of ind. names */
};
```
A prepared statement can be described; the description is stored in the SQLDA structure.

EXEC SQL DESCRIBE stmt INTO sqlda

The result is:

- the number of result attributes
  - 0: not a query
  - negative: SQLDA to small [ORACLE]
- for every attribute in the answer
  - its name and length
  - its type
SQLDA and parameter passing

We can use a **SQLDA** descriptor to supply parameters and/or to get the result.

You **fill in the values and types** and then use the description area in the following statements:

```
EXEC SQL EXECUTE stmt
   USING DESCRIPTOR :sqlda;

EXEC SQL OPEN cname
   USING DESCRIPTOR :sqlda;

EXEC SQL FETCH cname
   USING DESCRIPTOR :sqlda;
```

...essentially in places we used `:var1.,...,:vark.`
**Putting it together: adhoc.sqc**

**adhoc** is an application that executes an SQL statement provided as its argument on the command line.

Declarations:

```c
#include <stdio.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;

EXEC SQL BEGIN DECLARE SECTION;
    char db[6] = "cs448";
    char sqlstmt[1000];
EXEC SQL END DECLARE SECTION;

struct sqlda *select;
```
Start up and **prepare** the statement:

```c
int main(int argc, char *argv[]) {
    int i, isnull; short type;

    printf("Sample C program : ADHOC interactive SQL\n");

    /* bail out on error */
    EXEC SQL WHENEVER SQLERROR  GO TO error;

    /* connect to the database */
    EXEC SQL CONNECT TO :db;
    printf("Connected to DB2\n");

    strncpy(sqlstmt,argv[1],1000);  
    printf("Processing <%s>\n",sqlstmt);

    /* compile the sql statement */
    EXEC SQL PREPARE stmt FROM :sqlstmt;

    init_da(&select,1);

    /* now we find out what it is */
    EXEC SQL DESCRIBE stmt INTO :*select;

    i= select->sqld;
```
... its a query:

```c
if (i>0) {
    printf("... looks like a query\n");

    /* new SQLDA to hold enough descriptors for answer */
    init_da(&select,i);

    /* get the names, types, etc... */
    EXEC SQL DESCRIBE stmt INTO :select;

    printf("Number of select variables <%d>\n",select->sqld);
    for (i=0; i<select->sqld; i++) {
        printf(" variable %d <%.*s (%d%s [%d])>
", 
                 i,
                select->sqlvar[i].sqlname.length,
                select->sqlvar[i].sqlname.data,
                select->sqlvar[i].sqltype,
                ( (select->sqlvar[i].sqltype&1)==1 ?
                    "": " not null"),
                select->sqlvar[i].sqllen);
    }
    printf("\n");
```
...more processing for queries:

```c
for (i=0; i<select->sqld; i++) {
    select->sqlvar[i].sqldata=malloc(select->sqlvar[i].sqllen);
    select->sqlvar[i].sqlind=malloc(sizeof(short));
    *select->sqlvar[i].sqlind = 0;
}
EXEC SQL DECLARE cstmt CURSOR FOR stmt;
EXEC SQL OPEN cstmt;
EXEC SQL WHENEVER NOT FOUND GO TO end;
for (i=0; i<select->sqld; i++)
    printf("%-*.*s ",select->sqlvar[i].sqllen,
           select->sqlvar[i].sqlname.length,
           select->sqlvar[i].sqlname.data);

printf("\n");
for (;;) {
    EXEC SQL FETCH cstmt USING DESCRIPTOR :*select;
    for (i=0; i<select->sqld; i++)
        if ( *(select->sqlvar[i].sqlind) < 0 )
            print_var("NULL", select->sqlvar[i].sqltype,
                    select->sqlvar[i].sqlname.length,
                    select->sqlvar[i].sqllen);
        else
            print_var(select->sqlvar[i].sqldata,
                     select->sqlvar[i].sqltype,
                     select->sqlvar[i].sqlname.length,
                     select->sqlvar[i].sqllen);
    printf("\n");
}
end: printf("\n");
```
...otherwise its a simple statement:

} else {
    printf(" ... looks like an update\n");

    EXEC SQL EXECUTE stmt;
};
printf("Rows processed: %d\n",sqlca.sqlerrd[2]);

/* and get out of here */
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);
}
Example

bash$ ./adhoc "select * from author"

Sample C program: ADHOC interactive SQL

Password:

Connected to ORACLE

Processing <select * from author>

... looks like a query

Number of select variables <3>

variable 0 <AID (2 not null [14])>
variable 1 <NAME (96 [22])>
variable 2 <URL (96 [42])>

<table>
<thead>
<tr>
<th>AID</th>
<th>NAME</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Toman, David</td>
<td><a href="http://brics.dk/~david">http://brics.dk/~david</a></td>
</tr>
<tr>
<td>2</td>
<td>Chomicki, Jan</td>
<td><a href="http://cs.monmouth.edu/~c">http://cs.monmouth.edu/~c</a></td>
</tr>
<tr>
<td>3</td>
<td>Saake, Gunter</td>
<td><strong>null</strong></td>
</tr>
<tr>
<td>4</td>
<td>Niwinski, Damian</td>
<td><a href="http://zls.mimuw.edu.pl/">http://zls.mimuw.edu.pl/</a>~</td>
</tr>
<tr>
<td>5</td>
<td>Snodgrass, Richard T.</td>
<td><a href="http://www.cs.arizona.edu">http://www.cs.arizona.edu</a></td>
</tr>
</tbody>
</table>

Rows processed: 5
Summary

- given a string:
  - simple statement used once: \texttt{EXECUTE IMMEDIATE}
  - otherwise: \texttt{PREPARE}

- given a statement handle (using \texttt{PREPARE}):  
  - simple statement: \texttt{EXECUTE}
  - query: \texttt{DECLARE CURSOR}
    and process as a ordinary cursor
  - unknown: \texttt{DESCRIBE}

Remember to supply correct host variables/sqlda for all parameter and answer tuples!