# Dynamic Embedded SQL

David Toman

School of Computer Science
University of Waterloo

Introduction to Databases CS348

## Dynamic SQL

**Goal**

*execute* a *string* as a SQL statement

Problems:

- How do we know a string is a valid statement?

  ⇒ parsing and compilation?

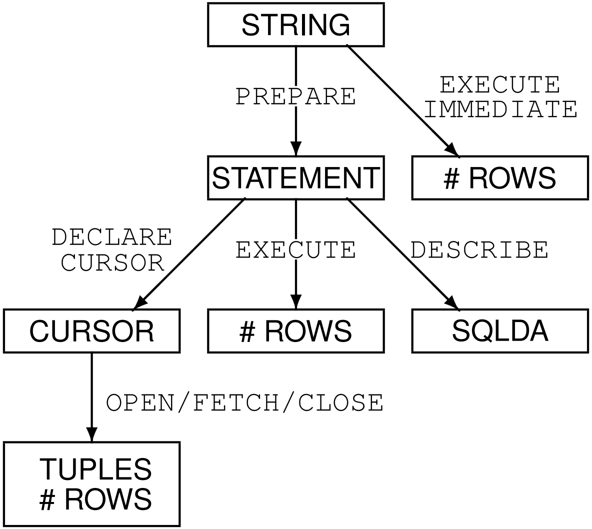
- How do we execute

  ⇒ queries? (where does the answer go?)
  ⇒ updates? (how many rows affected?)

- What if 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 answers, if any).

## Dynamic SQL: a Roadmap



## EXECUTE IMMEDIATE

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
- used for constant statements executed only once

  $\Rightarrow$ `:string` is *compiled* every time we pass through.

## PREPARE

We better **compile** a `:string` into a `stmt`...

```
EXEC SQL PREPARE stmt FROM :string;
```

`stmt` can now used for repeatedly executed statements

$\Rightarrow$ 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

How do we pass parameters into SQL statements?

- Static embedded SQL

  $\Rightarrow$ host variables as parameters

- Dynamic SQL (strings) and **parameters**?

  $\Rightarrow$ we can change the string (recompilation)
  $\Rightarrow$ use **parameter marker**: a `"?"` in the string

**Idea**

Values for `"?"`s are substituted when the statement is to be executed

## Simple statement: EXECUTE

How do we execute a prepared “non-query?”

```
EXEC SQL EXECUTE stmt
         USING   :var1 [,...,:vark];
```

- for statements that don’t return tuples
  ⇒ database modification (`INSERT`, ...)
  ⇒ transactions (`COMMIT`)
  ⇒ data definition (`CREATE ...`)

- values of `:var1 ,..., :vark` are substituted
  for the parameter markers (in order of appearance)
  ⇒ mismatch causes SQL runtime error!

## Query with many answers: CURSOR

How do we execute a prepared "query?"

```
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.
- `sqlca.sqlerrd[2]` the number of retrieved tuples.

## Unknown number/types of variables??

How do we know/learn what kind of statement a string represents?

We need/use 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 tuple structure

The `sqlda` data structure is a SQL **description area** that defines how a single tuple looks like, where are the data, etc. . .
this is how the DBMS communicates with the application.

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 ala DB2

```
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 indiciator */
  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 ala ORACLE6

```
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          */
};
```

## DESCRIBE

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

- 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: **fill in the values and types** and then use the description area as follows.

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

EXEC SQL OPEN cname
         USING DESCRIPTOR :sqlda;

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

...`:sqlda` essentially replaces `: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:

```
#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;
```

## adhoc.sqc (cont.)

Start up and **prepare** the statement:

```
int main(int argc, char *argv[]) {
   int i, isnull; short type;
   printf("Sample C program : ADHOC interactive SQL\n");

   EXEC SQL WHENEVER SQLERROR  GO TO error;

   EXEC SQL CONNECT TO :db;
   printf("Connected to DB2\n");

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

   EXEC SQL PREPARE stmt FROM :sqlstmt;

   init_da(&select,1);

   EXEC SQL DESCRIBE stmt INTO :*select;

   i= select->sqld;
```

## adhoc.sqc (cont.)

### . . . its a query:

```
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])>\n",
                 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");
```

## adhoc.sqc (cont.)

. . . more processing for queries: prepare buffers and print a header.

```
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;
};

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");
```

## adhoc.sqc (cont.)

…more processing for queries: fetch and print answers.

```
    EXEC SQL DECLARE cstmt CURSOR FOR stmt;
    EXEC SQL OPEN cstmt;
    EXEC SQL WHENEVER NOT FOUND GO TO end;
    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");
```

## adhoc.sqc (cont.)

…otherwise its a simple statement: just execute it.

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

      EXEC SQL EXECUTE stmt;
    };

    /* 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-2.05b$ ./adhoc "select * from author"
Sample C program : ADHOC interactive SQL
Connected to DB2
Processing <select * from author>
      ... looks like a query
Number of select variables <3>
  variable 0 <AID (496 not null [4])>
  variable 1 <NAME (453 [22])>
  variable 2 <URL (453 [42])>

AID NAME          URL
  1 Toman, David  http://db.uwaterloo.ca/~david
  2 Chomicki, Jan http://cs.buffalo.edu/~chomick
  3 Saake, Gunter NULL
```

## Summary

- given a string:
  - ⇒ unknown: `DESCRIBE`
  - ⇒ simple statement used once: `EXECUTE IMMEDIATE`
  - ⇒ otherwise: `PREPARE`
- given a statement handle (using `PREPARE`):
  - ⇒ simple statement: `EXECUTE`
  - ⇒ query: `DECLARE CURSOR`

    and then process as a ordinary cursor

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