Module 5: Applications Programming and SQL

Fall 2026

Cheriton School of Computer Science

CS 338: Intro to Database Management

Outline

Unit 1: **Embedded SQL (in C)**

Unit 2: Stored Procedures

Database Applications

- SQL isn't sufficient to write general applications.

  $\Rightarrow$ somehow extend a general-purpose PL with SQL

- Language considerations:

  $\Rightarrow$ via library calls (CLI/ODBC)

  $\Rightarrow$ via Embedded SQL

  $\Rightarrow$ via (usually OO) PLs with *persistent* data types

- At the least, a two-tier client-server architecture will apply.

  $\Rightarrow$ SQL runs on the server

  $\Rightarrow$ application runs on the client

Embedded SQL

SQL Statements are embedded in a host language such as C, C++, FORTRAN, and so on.

The application is *preprocessed* to produce a pure host language program with library calls.

Advantages:

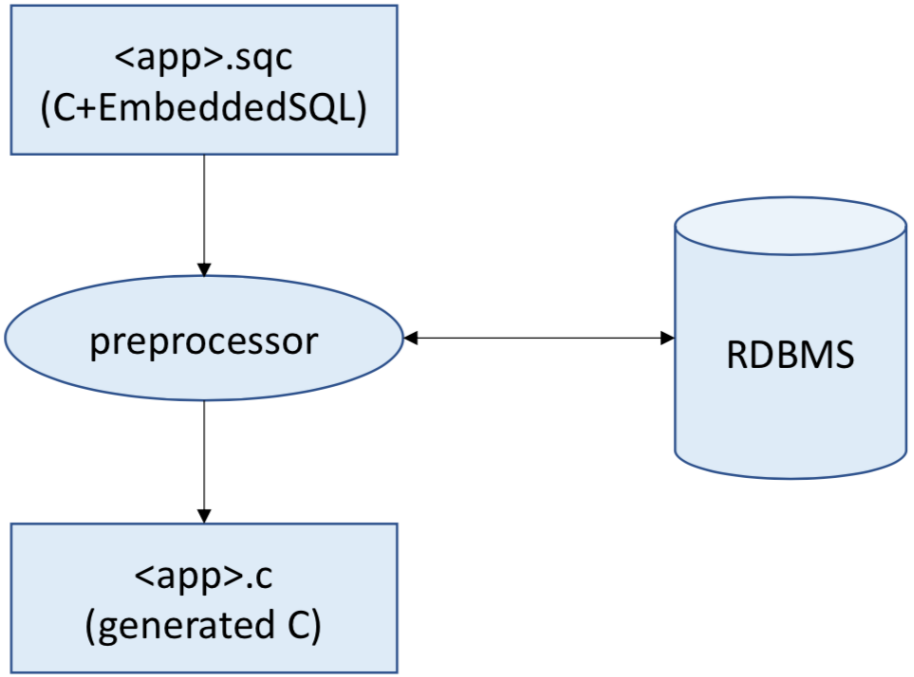
- Preprocessing of (static) parts of queries becomes possible.
- Much easier to use.
- Potentially more secure.

Disadvantages:

- Requires a pre-compiler.
- Generated host language program might be bound to a particular RDBMS engine.

  $\Rightarrow$ DB2 has this property

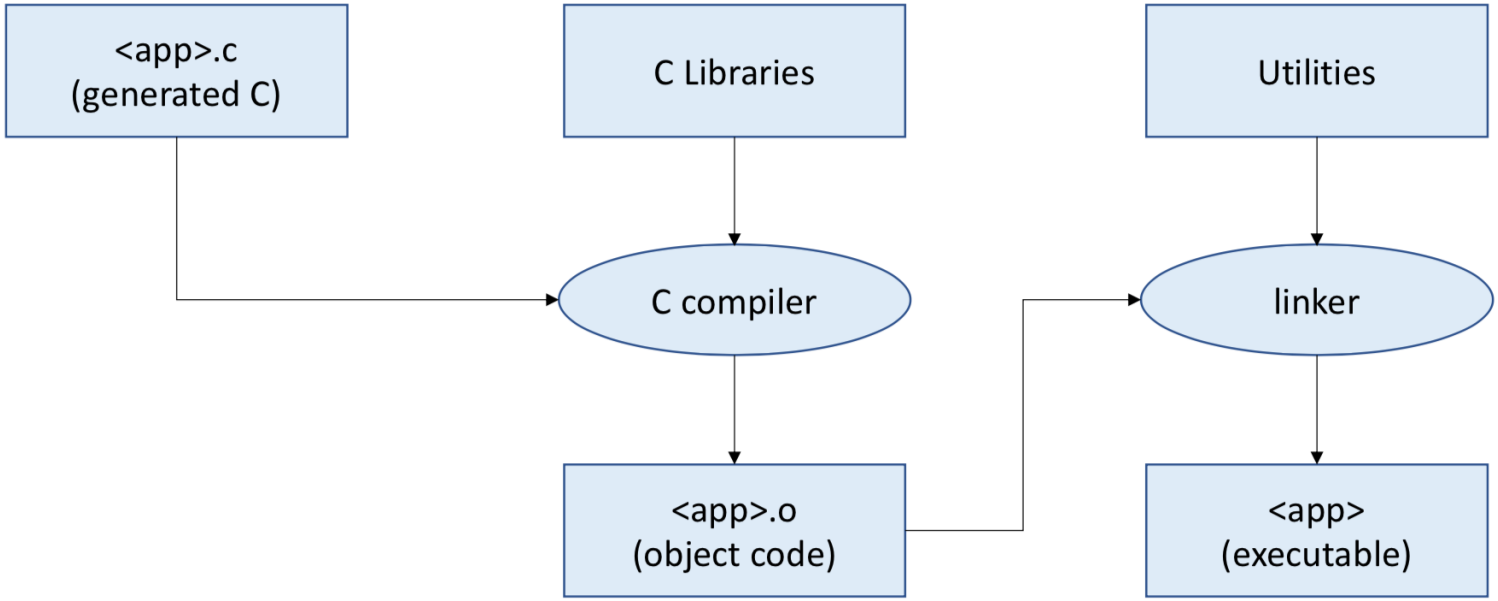
Development Process for Embedded SQL Applications



- Warning: Not all RDBMSs communicate with the server to *compile* at this time.

  $\Rightarrow$ some products simply generate a CLI interface, e.g., ODBC

Development Process (cont'd)



Reminder: For DB2, <app> will be bound to a particular server engine.

Development Process (cont'd)

Sample Makefile source, preamble:

```
# Source file for app: <app>.sqc
NAME= <app>

#############################################
# don't touch anything below this line
#############################################
DB2PATH = /home/db2inst2/sqllib
# The following compile and link options are for the gcc
CC=gcc

CFLAGS=-I$(DB2PATH)/include
#LIBS=-L$(DB2PATH)/lib -R$(DB2PATH)/lib -ldb2
LIBS=-L$(DB2PATH)/lib -ldb2
```

Development Process (cont'd)

Sample Makefile source, generating an executable:

```
all: $(NAME)

$(NAME): $(NAME).sqc util.o
db2 connect to cs348
db2 prep $(NAME).sqc bindfile
db2 bind $(NAME).bnd
db2 connect reset
$(CC) $(CFLAGS) -c $(NAME).c
$(CC) $(CFLAGS) -o $(NAME) $(NAME).o util.o $(LIBS)

clean:
rm -f $(NAME) $(NAME).c $(NAME).o $(NAME).bnd

util.o : util.c
$(CC) -c util.c $(CFLAGS)
```

Source downloadable from class web site.

Embedded SQL (cont.)

Considerations:

- How much can SQL be parameterized?

  $\Rightarrow$ how to pass parameters into SQL

  $\Rightarrow$ how to get results of queries

  $\Rightarrow$ exception handling

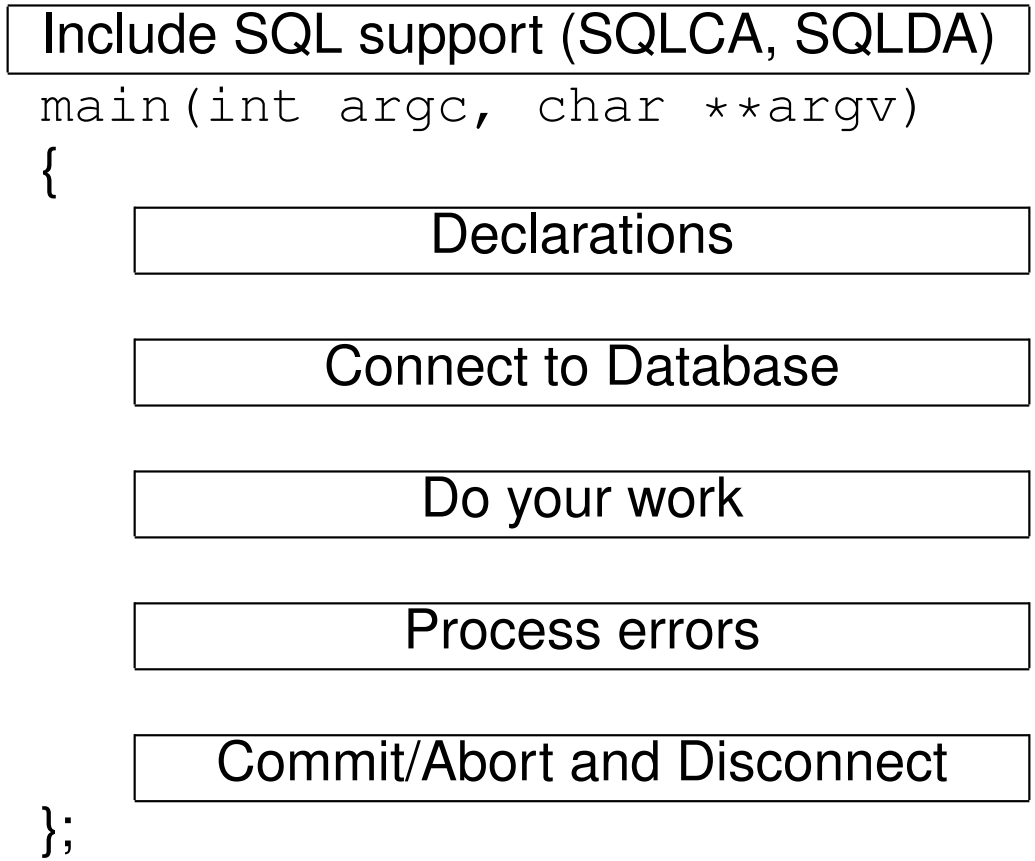
- Static versus dynamic SQL statements.

How much does the DBMS know about an application?

$\Rightarrow$ precompiling: `PREP`

$\Rightarrow$ binding: `BIND`

Application Structure



Declarations

- Include SQL communication area:

```
EXEC SQL INCLUDE SQLCA;
```

  Defines:

  $\Rightarrow$ return code of SQL statements (sqlcode)

  $\Rightarrow$ error messages (if any)

  $\Rightarrow$ required

- An SQL statement can be inserted by embedding the following where C statements can occur:

```
EXEC SQL <sql statement>;
```

Host Variables

Host Variables

SQL statements can have parameters that are local or global variables in the embedding language.

- Host variables communicate *single scaler values* between an SQL a statement and the embedding language.
- Must be declared within an SQL declare section:

```
EXEC SQL BEGIN DECLARE SECTION;
(only declarations of host variables in host language go here)
EXEC SQL END DECLARE SECTION;
```

- Can then be used in `EXEC SQL` statements.
- To distinguish them from SQL identifiers, they are prefixed by ":", the colon character.

Exception Handling

What if a SQL statement fails?

- Check `sqlcode != 0`, or
- Use exception handling SQL commands:

```
EXEC SQL WHENEVER SQLERROR GO TO <label>;
EXEC SQL WHENEVER SQLWARNING GO TO <label>;
EXEC SQL WHENEVER NOT FOUND GO TO <label>;
```

$\Rightarrow$ `<label>` must be in scope

Connecting to a Database (sample1)

```
#include <stdio.h>
#include <stdlib.h>
#include "util.h"

EXEC SQL INCLUDE SQLCA;

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

   EXEC SQL BEGIN DECLARE SECTION;
      char db[6] = "cs348";
   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);
```

Connecting to a Database (sample1, cont'd)

```
error:
    check_error("My error",&sqlca);
    EXEC SQL WHENEVER SQLERROR CONTINUE;

    EXEC SQL ROLLBACK;
    EXEC SQL CONNECT reset;
    exit(1);
}
```

Preparing your Application in 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]
   ```

Note: A Makefile to do this is available on class web site.

$\Rightarrow$ sets options

$\Rightarrow$ knows the path(s) and libraries

Connecting to a Database (sample1,cont'd)

```
make all NAME=sample1

db2 connect to cs348

   Database Connection Information

 Database server        = DB2/LINUXX8664 11.1.4.4
 SQL authorization ID   = GWEDDELL
 Local database alias   = CS348

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.
gcc -I/home/db2inst2/sqllib/include  -c sample1.c
gcc -I/home/db2inst2/sqllib/include  -o sample1 sample1.o util.o -L/home/db2inst2/sq
```

Connecting to a Database (sample1, cont'd)

With no errors:

```
./sample1

Sample C program: CONNECT
Connected to DB2
```

With wrong database name:

```
./sample1

Sample C program: CONNECT
--- error report ---
ERROR occurred : My error.
SQLCODE : 4294966283
SQL1013N  The database alias name or database name "CS248" could not be found.
SQLSTATE=42705
--- end error report ---
```

Adding SQL DML Statements

With a surrounding infrastructure, now consider adding executable DML statements:

- Simple statements:

  $\Rightarrow$ "constant" statements

  $\Rightarrow$ statements with parameters

  $\Rightarrow$ queries returning a single tuple

- General queries with many answers.

- Dynamic queries (covered in Module 6).

Executing Simple Statements (sample2)

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

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "util.h"

EXEC SQL INCLUDE SQLCA;

int main(int argc, char *argv[]) {
   int i;
   char pubidstr[10];

   EXEC SQL BEGIN DECLARE SECTION;
      char db[6] = "cs348";
      int pubid;
      char title[30];
   EXEC SQL END DECLARE SECTION;

   printf("Sample C program: SAMPLE2\n" );

   EXEC SQL CONNECT TO :db;

   printf("Connected to DB2\n");
```

Executing Simple Statements (sample2, cont'd)

```
    EXEC SQL WHENEVER SQLERROR  GO TO error;

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

      EXEC SQL WHENEVER NOT FOUND GO TO nope;
      EXEC SQL SELECT title INTO :title
               FROM   publication
               WHERE  pubid = :pubid;

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

    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);
}
```

Executing Simple Statements (sample2, cont'd)

```
./sample2 1 7 3 5

Sample C program: SAMPLE2
Connected to DB2
1: Mathematical Logic
7: *** not found ***
3: Principles of DB Systems
5: *** not found ***
```

It is important that at most *one* title is returned for each publication identifier.

NULLs and Indicator Variables

- What if a result value is NULL?

  $\Rightarrow$ not a valid value in the datatype

- Embedded SQL uses an extra indicator variable, e.g.:

```
smallint ind; // (within a BEGIN/END DECLARE SECTION)
...
EXEC SQL SELECT firstname INTO :firstname
                          INDICATOR :ind
         FROM ... ;
```

  If `ind` $<$ 0 then `firstname` is NULL.

- If the indicator variable is not provided and the result is a null we get an run-time error.

- The same rules apply for host variables in updates.

Impedance Mismatch

SQL CURSOR

For SQL queries possibly returning more than one answer, an SQL CURSOR is defined and used in an iterator protocol.

1. Declaring a cursor:

```
EXEC SQL DECLARE <name> CURSOR FOR
         <query>;
```

2. An iterator protocol using a cursor:

```
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>;
```

Executing Queries with Many Answers (sample3)

Write a program that lists all author names and publication titles with author name matching a pattern given as an argument.

```
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 -> %10s: %s\n",apat,name,title);
   };
   end:
   EXEC SQL CLOSE author;
   ...
}
```

Executing Queries with Many Answers (sample3, cont'd)

```
./sample3 "%"

Sample C program: SAMPLE3
Connected to DB2
         % ->         Sue: Mathematical Logic
         % ->         Sue: Trans. on Databases
         % ->         Sue: Query Languages
         % ->        John: Trans. on Databases

./sample3 "%u_"

Sample C program: SAMPLE3
Connected to DB2
        %u_ ->         Sue: Mathematical Logic
        %u_ ->         Sue: Trans. on Databases
        %u_ ->         Sue: Query Languages
```

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!

Outline

Unit 1: Embedded SQL (in C)

Unit 2: **Stored Procedures**

Stored Procedures

Idea

A *stored procedure* executes application logic directly inside the RBDMS server.

Possible implementations:

- An ability to invoke externally-compiled application; and/or
- Support for SQL/PSM (or a vendor-specific equivalent).

Reasons for stored procedures:

- Reduces volume of data transfer between client and server;
- Centralizes application code at the server; and
- Conceptual schema enhancement.

A Stored Procedure Example: Atomic-Valued Function

```
CREATE FUNCTION sumSalaries(dept CHAR(3))
    RETURNS DECIMAL(9,2)
LANGUAGE SQL
RETURN
    SELECT sum(salary)
    FROM employee
    WHERE workdept = dept
```

A Stored Procedure Example: Atomic-Valued Function

```
db2 => SELECT deptno, sumSalaries(deptno) AS sal \
    => FROM department

DEPTNO SAL
------ -----------
A00      128500.00
B01       41250.00
C01       90470.00
D01               -
D11      222100.00
D21      150920.00
E01       40175.00
E11      104990.00
E21       95310.00

  9 record(s) selected.
```

A Stored Procedure Example: Table-Valued Function

```
CREATE FUNCTION deptSalariesF(dept CHAR(3))
    RETURNS TABLE(salary DECIMAL(9,2))
    LANGUAGE SQL
RETURN
    SELECT salary
    FROM employee
    WHERE workdept = dept
```

A Stored Procedure Example: Table-Valued Function

```
db2 => SELECT * FROM TABLE \
    => (deptSalariesF(CAST('A00' AS CHAR(3)))) AS s

SALARY
------------
   52750.00
   46500.00
   29250.00

  3 record(s) selected.
```

A Stored Procedure Example: Branching

```
CREATE PROCEDURE UPDATE_SALARY_IF
       (IN employee_number CHAR(6), INOUT rating SMALLINT)
   LANGUAGE SQL
BEGIN
   DECLARE not_found CONDITION FOR SQLSTATE '02000';
   DECLARE EXIT HANDLER FOR not_found
      SET rating = -1;
   IF rating = 1 THEN
      UPDATE employee
      SET salary = salary * 1.10, bonus = 1000
      WHERE empno = employee_number;
   ELSEIF rating = 2 THEN
      UPDATE employee
      SET salary = salary * 1.05, bonus = 500
      WHERE empno = employee_number;
   ELSE
      UPDATE employee
      SET salary = salary * 1.03, bonus = 0
      WHERE empno = employee_number;
   END IF;
END
```