SQL in Application Code

- Most often DBMS access is performed from applications
  - Command-line interface only suitable for advanced users, admins

- SQL commands can be called from within a host language
  - C, Java, etc.
  - Application connects to the database

Two main approaches:
- Embed SQL in the host language (Embedded SQL, SQLJ)
- Create special API to call SQL commands (JDBC)

Important Aspects

- Mapping to language variables and data structures
  - SQL statements “linked” to host variables (+ status variables)
  - Alignment of SQL and host language “native” types

- The “impedance mismatch” problem
  - SQL relations are multisets of records (of unknown cardinality)
  - Some host languages may not support such data types
    - Modern ones do (STL in C++, Collections in Java)

- SQL supports a mechanism called a cursor
  - Bridges the gap between SQL sets and native variables

Outline

- Embedded SQL
- Dynamic SQL
- JDBC (API)
- SQLJ (Embedded)
- Stored procedures

Overview of Embedded SQL

```
Preprocessor
$gcc -L/disk/sd0d/tools... prog.c
```

Need to link to standard libraries as well as DBMS library (e.g. Oracle)
Overview of Constructs

- Connect to DBMS
  ```sql
  EXEC SQL CONNECT :username IDENTIFIED BY :password;
  ```

- Declaring variables (shared with SQL):
  ```sql
  EXEC SQL BEGIN DECLARE SECTION;
  ... 
  EXEC SQL END DECLARE SECTION;
  ```

- Executing Statements
  ```sql
  EXEC SQL _Statement;
  ```

Variables

```sql
EXEC SQL BEGIN DECLARE SECTION;
  char    c_sname[20]; /*convention is c_ prefix*/
  long    c_sid;
  int     c_rating;
  float   c_age;
EXEC SQL END DECLARE SECTION;
```

- In `EXEC SQL` lines, variables are prefixed by `:`.

- Two special error-handling variables:
  - `SQLCODE` (long, is negative if an error has occurred)
  - `SQLSTATE` (char[6], codes for common errors)
    - `'00000'` = no error
    - `'02000'` = no data
    (recall that in C, that is 5 chars + \0 terminator)

Type casting

- How are various SQL types mapped to native types?
  - Oracle uses the following mapping:
    | SQL       | C         |
    |-----------|-----------|
    | number    | integer   |
    | number(p,s) | short, long, float or double (depending on values p and s) |
    | char(n)   | char[n+1] |
    | date      | char[9]   |

- Type must match in SQL statement assignments!

Executing Queries

- Insertion (no need to return value in host program)
  ```sql
  EXEC SQL INSERT INTO Sailor(name) VALUES ('Lubber');
  ```

- If only one single tuple is retrieved
  ```sql
  EXEC SQL SELECT name INTO :c_name, FROM Sailors where sid = :c_sid;
  ```

- Will fail at runtime if more than one tuple retrieved!

Cursors

- Mechanism that helps traversing data
  - Can declare a cursor on a relation or query statement
  - Inspect tuple, or even modify/delete tuples (when allowed)

- Operations:
  - open or close a cursor
  - fetch a tuple and move cursor to next tuple in result
  - move the cursor

- Can use `ORDER BY` to control tuple order
  - `ORDER BY` fields must also appear in `SELECT` clause

Cursor Example

```sql
EXEC SQL DECLARE sinfo CURSOR FOR
SELECT s.name, s.age
FROM Sailors s, Boats b, Reserves r
WHERE s.sid = r.sid AND r.bid = b.bid AND s.rating > :c_minrating
ORDER BY s.name;
Variable :c_minrating evaluated at the time OPEN is executed!

EXEC SQL OPEN sinfo;
EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
printf("%s is %d years old\n", :c_sname, :c_age);
EXEC SQL CLOSE sinfo;
```
**Error Handling**

- Define action to execute when event/error occurs
  
  ```sql
  EXEC SQL WHENEVER condition action;
  ```

- Condition can be
  - `SQLWARNING`
  - `SQLERROR` (SQLSTATE will indicate error code)
  - `NOT FOUND` (SQLSTATE='02000', means no data retrieved)

- Action
  - `stop`, `goto` label, `continue`, `do` (function call)

Examples:

```sql
EXEC SQL WHENEVER sqlerror goto report_error;
EXEC SQL WHENEVER not found goto notfound;
```

**Extended Example**

```sql
EXEC SQL OPEN sinfo;
/* traverse result set with cursor */
do {
  EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
  printf("%s is %d years old\n", c_sname, c_age);
} while (strcmp(SQLSTATE,"02000")!=0);
EXEC SQL CLOSE sinfo;
```

**Dynamic SQL**

- Embedded SQL parses query strings at compile time
  - What if query is not known in advance?
  - Interactive apps: spreadsheet, graphical DBMS frontend

- Allow construction of SQL statements on-the-fly

```sql
char c_sqlstring[]="DELETE FROM Sailors WHERE rating>5";
EXEC SQL PREPARE readytogo FROM :c_sqlstring;
EXEC SQL EXECUTE readytogo;
```