Outline

- Embedded SQL.
- Call-Level Interface (CLI).
- Java Database Connectivity (JDBC).

Embedded SQL

- Standard for combining SQL with a host language.
- SQL statements are converted to procedure calls in the host language by a preprocessor.
- Begin SQL statements with `EXEC SQL`.

Shared Variables

- The interface between SQL and the host language is through shared variables.
  - EXEC SQL BEGIN DECLARE SECTION;
  - declarations of shared variables in host language syntax
  - EXEC SQL END DECLARE SECTION;

Use of Shared Variables

- In SQL, shared variables are preceded by a colon.
  - Can be used as constants in SQL statements.
  - Can get values from SQL statements and pass values to host language.
- In the host language, shared variables are used as any other variables.

Example

- Look up the price that a given bar charges for a given beer.
  - EXEC SQL BEGIN DECLARE SECTION;
  - char aBeer[21], aBar[21];
  - float aPrice;
  - EXEC SQL END DECLARE SECTION;
  - /* read in the beer and the bar */
  - EXEC SQL SELECT price INTO :aPrice
  - FROM Sells
  - WHERE beer = :aBeer AND bar = :aBar;
  - /* print the price */
Embedded Queries

- **Modification queries.**
  - Return no results; can be used anywhere.
- **Single-row select queries.**
  - Return a single tuple; can be read into shared variables.
- **Multiple-row select queries.**
  - Return many tuples; can be used with cursors.

Cursors

- **Declare a cursor.**
  
  \[
  \text{EXEC SQL DECLARE c CURSOR FOR <query>;}\]
- **Open a cursor.**
  
  \[
  \text{EXEC SQL OPEN c;}\]
- **Fetch a tuple.**
  
  \[
  \text{EXEC SQL FETCH c INTO <vars>;}\]

Example (1/2)

- Find the prices of all beers sold in Spoon.
  
  \[
  \text{EXEC SQL BEGIN DECLARE SECTION;}
  
  \text{char aBeer[21];}
  
  \text{float aPrice;}
  
  \text{EXEC SQL END DECLARE SECTION;}
  
  \text{EXEC SQL DECLARE spoonBeers CURSOR FOR}
  
  \text{SELECT beer, price}
  
  \text{FROM Sells}
  
  \text{WHERE bar = 'Spoon';}\]

Example (2/2)

  
  \[
  \text{EXEC SQL OPEN CURSOR spoonBeers;}
  
  \text{while(1) {}
  
  \text{EXEC SQL FETCH spoonBeers}
  
  \text{INTO :aBeer, :aPrice;}
  
  \text{if (NO_MORE_TUPLES) break;}
  
  \text{/* print out the beer and the price */}
  
  }\]

  
  \[
  \text{EXEC SQL CLOSE CURSOR spoonBeers;}\]

Modifying Base Relations

- A cursor can range over a base relation.
  
  \[
  \text{EXEC SQL DECLARE c CURSOR FOR Sells;}\]
- Modifications can be made only to the current tuple.
  
  \[
  \text{EXEC SQL DELETE FROM Sells}
  
  \text{WHERE CURRENT OF c;}\]
- Any condition can be applied in the host language.

Dynamic SQL

- So far, fixed queries with possibly some parameters.
- What if we want run ad-hoc queries?
- **Dynamic SQL**
  - Prepare statement (not known at compile time.)
  - Execute statement.
Dynamic SQL Syntax

- Prepare a query.
  ```sql
  EXEC SQL PREPARE <query-name>
  FROM <query>;
  ```
- Execute a query.
  ```sql
  EXEC SQL EXECUTE <query-name>;
  ```

Example

- Read a query and run it.
  ```sql
  EXEC SQL BEGIN DECLARE SECTION;
  char query[255];
  EXEC SQL END DECLARE SECTION;
  while (1)
      /* read query */
      EXEC SQL PREPARE q FROM :query;
      EXEC SQL EXECUTE q;
  ```

Execute-Immediate

- If the query is to be executed only once the prepare and execute statements can be combined.
  ```sql
  EXEC SQL EXECUTE IMMEDIATE <query>;
  ```

SQL/CLI

- Call-Level Interface: call library functions and procedures within a host language.
- Data types:
  - Environments: DBMS installation.
  - Connections: logins to DBMS.
  - Statements: SQL statements.
  - Descriptions: query results or parameters.

Data Type Instances

- Create environment, connection, and statement handles with SQLAllocHandle(T,I,O)
  - T is the type, e.g. SQL_HANDLE_ENV.
  - I is the input handle (higher-level handle):
    - statement < connection < environment
  - O is the output handle.

Example

```c
SQLHENV myEnv;
SQLHDBC myCon;
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &myEnv);
SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon);
```
Processing Statements

- Prepare and execute.
  SQLPrepare(<statement-handle>,
              <statement>,
              <length of statement>)
  SQLExecute(<statement-handle>)

Example

```
SQLPrepare(myStmt, "SELECT bar, beer
           FROM Sells WHERE price < 3.00",
           SQL_NTS)
SQLExecute(myStmt)
or
SQLExecDirect(myStmt, "SELECT bar, beer
           FROM Sells WHERE price < 3.00",
           SQL_NTS)
```

Fetching Tuples

- Every statement has an implied cursor associated with it.
- SQLFetch(<stmt-handle>) returns the next tuple from the result of the executed statement.

Binding Variables

- Before fetching we need to indicate where the tuple attributes should be stored.
  SQLBindCol(<stmt-handle>,
              <attribute-pos>,
              <attribute-type>,
              <var-ptr>,
              <var-size>,
              <var-info-ptr>);

Parameterized Queries

- Bind variables to query parameters, so you can execute the same statement several times with different parameters.
  SQLPrepare(myStmt, "INSERT(bar, beer)
           VALUES(?,?),
           SQL_NTS);
  SQLBindParameter(myStmt, 1, aBar, SQL_CHAR,
                   &aBar);
  SQLBindParameter(myStmt, 2, aBeer, SQL_CHAR,
                   &aBeer);
  while (SQLFetch(myStmt) != SQL_NO_DATA)
  {
    /* Cheers! */
  }
```
JDBC

- Java Database Connectivity (JDBC)
  - Similar to SQL/CLI and ODBC but adapted to object-oriented Java.
  - JDBC drivers are similar to environments in CLI.
  - Platform, implementation, and installation dependent.
  - DriverManager object.

JDBC Connection

- Connect with DriverManager by specifying the DBMS URL, username, and password.
  ```java
  Connection myCon = DriverManager.getConnection(<DB URL>, <username>, <password>);
  ```

Statements

- Two types of statements:
  - Statement can accept any string that is an SQL statement and execute it.
  - PreparedStatement has a fix SQL statement.
  ```java
  Statement s1 = myCon.createStatement();
  PreparedStatement s2 = myCon.prepareStatement(<SQL-stmt>);
  ```

Executing Statements

- JDBC distinguishes between queries and modifications.
- Both Statement and PreparedStatement have two methods:
  - executeQuery
  - executeUpdate
- For Statement the methods take a parameter.

Example

```java
PreparedStatement s2 = myCon.prepareStatement("SELECT bar,beer FROM Sells WHERE price < 3.0");
ResultSet cheapBeers = s2.executeQuery();
Statement s1 = myCon.createStatement();
S1.executeUpdate("INSERT INTO Sells Values('Spoon', 'Bud', 3.0)");
```
Accessing Attributes

- Call an appropriate method, depending on the type of attribute, on the ResultSet object.
  - Position of the attribute is a parameter
  - `getInt(i)`, `getString(i)`, `getFloat(i)`.

Example

```java
while (cheepBeers.next()) {
    aBar = cheepBeers.getString(1);
    aBeers = cheepBeers.getString(2);
    /* print out a map to the bar */
}
```

Parameterized Queries

- `PreparedStatement` can be parameterized
  - Use `?` to denote a parameter.
  - Use methods `setString`, `setInt`, `setFloat`.
  - Then run `executeQuery` or `Update`.