Programming Techniques

- Objective:
  - To access a database from an application program (as opposed to interactive interfaces)
- Why?
  - An interactive interface is convenient but not sufficient
    - A majority of database operations are made thru application programs

Database Programming Approaches

- Embedded commands:
  - Database commands are embedded in a general-purpose programming language
- Library of database functions:
  - Available to the host language for database calls; known as an *API (*Application Programming Interface)
- A special-purpose language
  - Minimizes impedance mismatch

Impedance Mismatch

- Incompatibilities between a host programming language and the database model, e.g.,
  - type mismatch and incompatibilities; requires a new binding for each language
  - set vs. record-at-a-time processing
    - need special iterators to loop over query results and manipulate individual values

Steps in Database Programming

1. Client program *opens a connection* to the database server
2. Client program *submits queries to and/or updates* the database
3. When database access is no longer needed, client program *closes (terminates) the connection*

Embedded SQL

- Most SQL statements can be embedded in a general-purpose host programming language such as COBOL, C, Java
- An embedded SQL statement is distinguished from the host language statements by enclosing it between **`EXEC SQL`** or **`EXEC SQL BEGIN`** and a matching **`END-EXEC`** or **`EXEC SQL END`** (or semicolon)
  - Syntax may vary with language
  - *Shared variables* (used in both languages) usually prefixed with a colon (:) in SQL

Example: Variable Declaration in Language C

- Variables inside **DECLARE** are shared and can appear (while prefixed by a colon) in SQL statements

```
int loop;
EXEC SQL BEGIN DECLARE SECTION;
    varchar dname[16], fname[16], …;
    char ssn[10], bdate[11], …;
    int dno, dnumber, SQLCODE, …;
EXEC SQL END DECLARE SECTION;
```

- **SQLCODE** is used to communicate errors/exceptions between the database and the program

Embedded SQL in C

```
loop = 1;
while (loop) {
  prompt ("Enter SSN: ", ssn);
  EXEC SQL
      select FNAME, LNAME, ADDRESS, SALARY
      into :fname, :lname, :address, :salary
      from EMPLOYEE where SSN == :ssn;
  if (SQLCODE == 0) printf(fname, …);
  else printf("SSN does not exist: ", ssn);
  prompt("More SSN? (1=yes, 0=no): ", loop);
  }
```

Embedded SQL in C

- A **cursor** (iterator) is needed to process multiple tuples
- **`FETCH`** commands move the cursor to the *next* tuple
- **`CLOSE CURSOR`** indicates that the processing of query results has been completed

```
EXEC SQL BEGIN DECLARE SECTION;
int v1; VARCHAR v2;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL DECLARE foo CURSOR FOR
        SELECT a, b FROM test;
...
do
{       ...
        EXEC SQL
        FETCH NEXT FROM foo INTO :v1, :v2; ... }
while (...);
```

Dynamic SQL

- Objective:
  - Composing and executing new (not previously compiled) SQL statements at run-time
    - a program accepts SQL statements from the keyboard at run-time
- Dynamic query can be complex
  - because the type and number of retrieved attributes are unknown at compile time

Dynamic SQL: An Example

```
EXEC SQL BEGIN DECLARE SECTION;
varchar sqlupdatestring[256];
EXEC SQL END DECLARE SECTION;
…
prompt ("Enter update command:", sqlupdatestring);
EXEC SQL PREPARE sqlcommand FROM :sqlupdatestring;
EXEC SQL EXECUTE sqlcommand;
```

Embedded SQL in Java

- SQLJ: a standard for embedding SQL in Java
- An SQLJ translator converts SQL statements into Java
  - These are executed thru the *JDBC* interface
- Certain classes have to be imported
  - e.g., **`java.sql`**

Example: Embedded SQL in Java

```
ssn = readEntry("Enter a SSN: ");
try {
   #sql{select FNAME< LNAME, ADDRESS, SALARY
   into :fname, :lname, :address, :salary
   from EMPLOYEE where SSN = :ssn};
}
catch (SQLException se) {
   System.out.println("SSN does not exist: ",+ssn);
   return;
}
System.out.println(fname + " " + lname + … );
```

Database Programming using an API

- Embedded SQL provides static database programming
- API: Dynamic database programming with a library of functions
  - Advantage:
    - No preprocessor needed
  - Disadvantage:
    - SQL syntax checks to be done at run-time

Java Database Connectivity

- JDBC:
  - function calls for Java/database interaction
- A Java program with JDBC functions can access any relational DBMS that has a JDBC driver

Steps in JDBC Database Access

1. Import JDBC library **`(java.sql.*)`**
2. Load JDBC driver:
   `Class.forname("oracle.jdbc.driver.OracleDriver")`
3. Define appropriate variables
4. Create a connection object (via **`getConnection`**)
5. Create a statement object from the **`Statement`** class:
   - 1. PreparedStatment 2. CallableStatement
6. Identify statement parameters (designated by question marks)
7. Bound parameters to program variables
8. Execute SQL statement (referenced by an object) via JDBC's **`executeQuery`**
9. Process query results (returned in an object of type ResultSet)
   - **`ResultSet`** is a 2-dimentional table

Database Stored Procedures

- Persistent procedures/functions (modules) are stored locally and executed by the database server
  - As opposed to execution by clients
- Advantages:
  - If the procedure is needed by many applications, it can be invoked by any of them (thus reduce duplications)
  - Execution by the server reduces communication costs
- Disadvantages:
  - Every DBMS has its own syntax and this can make the system less portable
  - (examine help on Stored Procedures for SQL Server)

Stored Procedure Constructs

- A stored procedure

```
CREATE PROCEDURE procedure-name (params)
local-declarations
procedure-body;
```

- A stored function

```
CREATE FUNCTION fun-name (params) RETRUNS return-type
local-declarations
function-body;
```

- Calling a procedure or function

```
CALL procedure-name/fun-name (arguments);
```

Example

```
CREATE FUNCTION DEPT_SIZE (IN deptno INTEGER)
RETURNS VARCHAR[7]
DECLARE TOT_EMPS INTEGER;

SELECT COUNT (*) INTO TOT_EMPS
   FROM SELECT EMPLOYEE WHERE DNO = deptno;
IF TOT_EMPS > 100 THEN RETURN "HUGE"
   ELSEIF TOT_EMPS > 50 THEN RETURN "LARGE"
   ELSEIF TOT_EMPS > 30 THEN RETURN "MEDIUM"
   ELSE RETURN "SMALL"
ENDIF;
```

Example from SQL Server Books Online

```
CREATE PROCEDURE OrderSummary @MaxQuantity INT OUTPUT AS
– SELECT to return a result set summarizing -- employee sales.
SELECT Ord.EmployeeID, SummSales = SUM(OrDet.UnitPrice *
    OrDet.Quantity)
FROM Orders AS Ord JOIN [Order Details] AS OrDet ON (Ord.OrderID =
    OrDet.OrderID)
GROUP BY Ord.EmployeeID
ORDER BY Ord.EmployeeID
-- SELECT to fill the output parameter with the
-- maximum quantity from Order Details.
SELECT @MaxQuantity = MAX(Quantity) FROM [Order Details]
-- Return the number of all items ordered.
RETURN (SELECT SUM(Quantity) FROM [Order Details])
```

Summary of Programming Techniques

- A database may be accessed in an interactive mode
- Most often, however, data in a database is manipulate via application programs
- Several methods of database programming:
  - Embedded SQL
  - Dynamic SQL
  - Stored Procedure and Function

SQL Triggers

- Database Programming can also be accomplished by incorporating Triggers
- Objective: to monitor a database and take initiate action when a condition occurs
- Triggers are expressed in a syntax that includes the following:
  - Event
    - Such as an insert, deleted, or update operation
  - Condition
  - Action
    - To be taken when the condition is satisfied

Example: SQL Triggers

- A trigger to compare an employee's salary to his/her supervisor during insert or update operations:

```
CREATE TRIGGER INFORM_SUPERVISOR
BEFORE INSERT OR UPDATE OF
   SALARY, SUPERVISOR_SSN ON EMPLOYEE
   FOR EACH ROW
        WHEN
        (NEW.SALARY> (SELECT SALARY FROM EMPLOYEE
                      WHERE SSN=NEW.SUPERVISOR_SSN))
        INFORM_SUPERVISOR (NEW.SUPERVISOR_SSN,NEW.SSN);
```