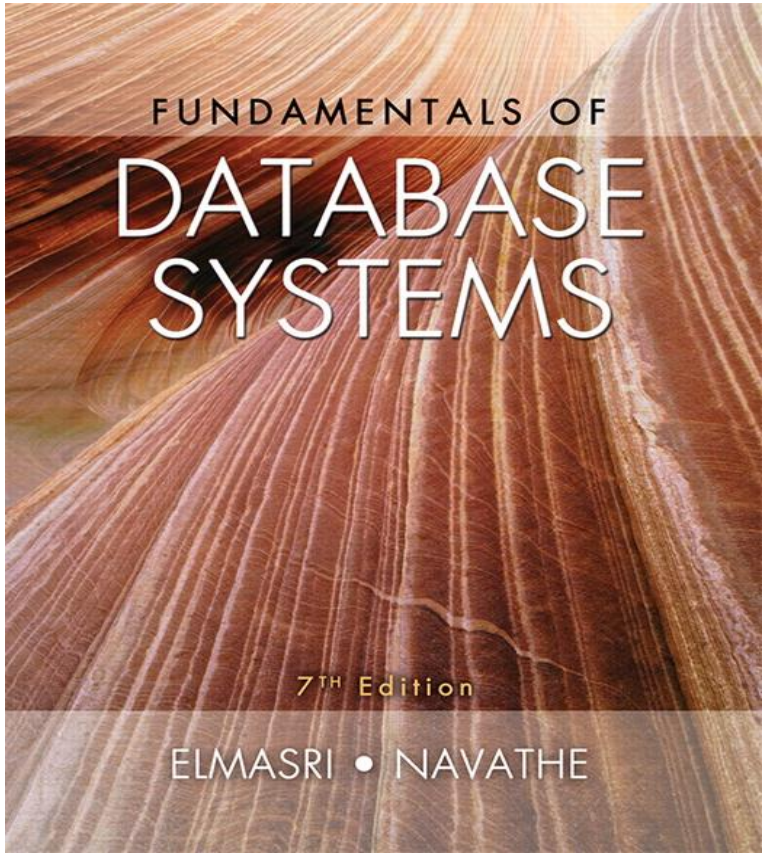
Comp-4150: Advanced and Practical Database Systems

- Ramez Elmasri , Shamkant B. Navathe(2016) Fundamentals of Database Systems (7th Edition), Pearson, isbn 10: 0-13-397077-9; isbn-13:978-0-13-397077-7.

Chapter 10:

Introduction to SQL Programming Techniques



Chapter 10: Introduction to SQL Programming Techniques: Outline

- 1. Database Programming: Techniques and Issues
- 2. Embedded SQL, Dynamic SQL, and SQLJ
- 3. Database Programming with Function Calls: SQL/CLI and JDBC
- 4. Database Stored Procedures and SQL/PSM (Oracle PL/SQL in Part B)
- 5. Connecting to database server through a DBMS client desktop software (E.g., SQL developer with Oracle)
- 6. Building a Web database application program for access through a web browser (e.g., with PHP scripting). (discussed in Chapter 11 and Part C)
- 5. Comparing the First Three Approaches (Embedded SQL, Function Calls, Stored Procedures)

Introduction to SQL Programming Techniques

- **1. Database applications: Techniques and Issues**
- **What are some of the techniques developed for accessing databases from application programs?**
- **Most database access in practical applications (e.g., student information system) is accomplished through software programs that implement database applications through the following three main approaches:**
- 2. Embedding Database Commands in a general-purpose programming language called Host language such as:
  - Java (embedded version is SQLJ), C/C++/C# (embedded SQL with C is Pro*C), COBOL, or some other programming language, (e.g., Dynamic SQL).

Introduction to SQL Programming Techniques

- 3. Database Programming with Function Calls and Class Libraries: SQL/CLI (eg., ODBC), JDBC connectivity
- 4. Database Stored Procedures and SQL/PSM
- SQL standards (e.g., Oracle PL/SQL, to be covered more in full in Part B of Course with the text book `Oracle PL/SQL by Example, 5th edition with authors Benjamin Rosenzweig and Elena Rakhimov).
- 5. Connecting to database server through a DBMS client desktop software (E.g., SQL developer with Oracle)
- 6. Building a Web database application program for access through a web browser (e.g., with PHP scripting). (discussed in Chapter 11 and Part C)
- 7. Comparing the Three Approaches of Embedded SQL, Function Calls and Stored Procedures (e.g., Oracle PL/SQL).

10.1. Database Programming: Techniques and Issues

- SQL standards are
  - Continually evolving
  - Each DBMS vendor may have some variations from standard
- **Interactive interface (e.g., through a query processor as Oracle SQLPlus or through a DBMS desktop front-end tool such as MS Access or SQL Developer, consists of**
  - SQL commands typed directly into a monitor
- To Execute a **filename.sql of SQL commands, use:**
  - *@filename.sql*
- **Application programs** or **database applications**
  - Used as canned transactions by the end users access a database and may have web interface.

10.1.1 Three Approaches to Database Programming (Discussed in Chapter 10)

- **1. Embedding** database commands in a general-purpose host programming language such as C or Java.
  - Here, Database statements are identified by a special prefix. For example, in an embedded SQL program with C host programming language, an embedded SQL statement is prefixed with the keywords: EXEC SQL
  - **Precompiler** or **preprocessor** scans the source program code
    - Identifies database statements and extracts them for processing by the DBMS
  - This first Approach is Called **embedded SQL**

10.1.1 Three Approaches to Database Programming (Discussed in Chapter 10)

- 2. Using a library of database functions (API) provided by the host language to connect to the DBMS.
  - **Library of functions** available to the host programming language (e.g. JDBC with Java)
  - **These functions in this second approach are called Application programming interface (API)**
- 3. Designing a brand-new database programming language
  - **Database programming language** designed from scratch by adding programming constructs of decision, repetition, functions and others to SQL.
  - These are called stored procedures (eg. Oracle PL/SQL)
- The First two approaches are more commonly used.

10.1.2 Impedance Mismatch

- Impedance mismatch discusses the problems that may occur when databases are queried using any of the three database programming approaches (e.g., Oracle PL/SQL and programming language model of methods 1 and 2)
- **1. Data model Binding** for each host programming language with the data model of the database can be challenging.
  - Thus, each such embedded C language program specifies for each attribute relational database (e.g., Varchar) type the compatible programming language types (e.g., char[ ])
- 2. Cursor or iterator variable has to be used to
  - Loop over the tuples in a query result that are typically presented as a relation in a relational database and these operations can appear complex.

10.1.3 Typical Sequence of Interaction in Database Programming

- When writing a database application program, a common sequence of interaction is:
- 1. The program must first Open a connection to database server usually by specifying the internet address (URL) of the machine where the server is located, plus providing a login account name and password for the database access.
- 2. Once the connection is established, the program can Interact with database by submitting queries, updates, and other database commands.
- 3. When the program no longer needs access to a particular database, it should terminate or close connection to the database

10.2. Embedded SQL, Dynamic SQL, and SQLJ

- **Examples are:**
- **1. Embedded SQL**
  - C language
- **2. SQLJ**
  - Java language
- Programming language called **host language**

10.2.1. Retrieving Single Tuples with Embedded SQL

- `EXEC SQL`
  - Prefix
  - **Preprocessor** separates embedded SQL statements from host language code
  - Terminated by a matching `END-EXEC`
    - Or by a semicolon (;)
- **Shared variables**
  - Used in both the C program and the embedded SQL statements
  - Prefixed by a colon (:) in SQL statement

Figure 10.1 C program variables used in the embedded SQL examples E1 and E2.

```
0) int loop ;
1) EXEC SQL BEGIN DECLARE SECTION ;
2) varchar dname [16], fname [16], lname [16], address [31] ;
3) char ssn [10], bdate [11], sex [2], minit [2] ;
4) float salary, raise ;
5) int dno, dnumber ;
6) int SQLCODE ; char SQLSTATE [6] ;
7) EXEC SQL END DECLARE SECTION ;
```

10.2.1. Retrieving Single Tuples with Embedded SQL (cont'd.)

- Connecting to the database

```
CONNECT TO <server name>AS <connection name>
AUTHORIZATION <user account name and password> ;
```

- Change connection

```
SET CONNECTION <connection name> ;
```

- Terminate connection

```
DISCONNECT <connection name> ;
```

10.2.1 Retrieving Single Tuples with Embedded SQL (cont'd.)

- **SQLCODE** and **SQLSTATE** communication variables
  - Used by DBMS to communicate exception or error conditions
- SQLCODE variable
  - 0 = statement executed successfully
  - 100 = no more data available in query result
  - < 0 = indicates some error has occurred

10.2.1. Retrieving Single Tuples with Embedded SQL (cont'd.)

- SQLSTATE
  - String of five characters
  - '00000' = no error or exception
  - Other values indicate various errors or exceptions
  - For example, '02000' indicates 'no more data' when using SQLSTATE
  - Example 10.1: Given the Company Database example table Employee already in your database, write an embedded SQL program in C/C++ that will accept as input a social security number of an employee and prints some information from the Employee record.

Figure 10.2 Program segment E1, a C program segment with embedded SQL.

```
   //Program Segment E1:
0) loop = 1 ;
1) while (loop) {
2)   prompt("Enter a Social Security Number: ", ssn) ;
3)   EXEC SQL
4)     SELECT Fname, Minit, Lname, Address, Salary
5)     INTO :fname, :minit, :lname, :address, :salary
6)     FROM EMPLOYEE WHERE Ssn = :ssn ;
7)   if (SQLCODE = = 0) printf(fname, minit, lname, address, salary)
8)     else printf("Social Security Number does not exist: ", ssn) ;
9)   prompt("More Social Security Numbers (enter 1 for Yes, 0 for No): ", loop) ;
10)  }
```

10.2.2. Retrieving Multiple Tuples with Embedded SQL Using Cursors

- Cursor
  - Points to a single tuple (row) from result of query
- **OPEN CURSOR** command
  - Fetches query result and sets cursor to a position before first row in result
  - Becomes current row for cursor
- **FETCH** commands
  - Moves cursor to next row in result of query

Figure 10.3 Program segment E2, a C program segment that uses cursors with embedded SQL for update purposes.

```
   //Program Segment E2:
 0) prompt("Enter the Department Name: ", dname) ;
 1) EXEC SQL
 2)   SELECT Dnumber INTO :dnumber
 3)   FROM DEPARTMENT WHERE Dname = :dname ;
 4) EXEC SQL DECLARE EMP CURSOR FOR
 5)   SELECT Ssn, Fname, Minit, Lname, Salary
 6)   FROM EMPLOYEE WHERE Dno = :dnumber
 7)   FOR UPDATE OF Salary ;
 8) EXEC SQL OPEN EMP ;
 9) EXEC SQL FETCH FROM EMP INTO :ssn, :fname, :minit, :lname, :salary ;
10) while (SQLCODE = = 0) {
11)   printf("Employee name is:", Fname, Minit, Lname) ;
12)   prompt("Enter the raise amount: ", raise) ;
13)   EXEC SQL
14)     UPDATE EMPLOYEE
15)     SET Salary = Salary + :raise
16)     WHERE CURRENT OF EMP ;
17)   EXEC SQL FETCH FROM EMP INTO :ssn, :fname, :minit, :lname, :salary ;
18)   }
19) EXEC SQL CLOSE EMP ;
```

10.2.2. Retrieving Multiple Tuples with Embedded SQL Using Cursors (cont'd.)

- **`FOR UPDATE OF`**
  - List the names of any attributes that will be updated by the program
- Fetch orientation
  - Added using value: `NEXT`, `PRIOR`, `FIRST`, `LAST`, `ABSOLUTE` *i*, and `RELATIVE` *i*

DECLARE <cursor name> [**INSENSITIVE**] [**SCROLL**] **CURSOR**
[**WITH HOLD**] **FOR** <query specification>
[**ORDER BY** <ordering specification>]
[**FOR READ ONLY** | **FOR UPDATE** [**OF** <attribute list>]] ;

10.2.3. Specifying Queries at Runtime Using Dynamic SQL

- **Dynamic SQL**
  - Execute different SQL queries or updates dynamically at runtime
- Dynamic update
- Dynamic query

Figure 10.4 Program segment E3, a C program segment that uses dynamic SQL for updating a table.

```
    //Program Segment E3:
0)  EXEC SQL BEGIN DECLARE SECTION ;
1)  varchar sqlupdatestring [256] ;
2)  EXEC SQL END DECLARE SECTION ;
    ...
3)  prompt("Enter the Update Command: ", sqlupdatestring) ;
4)  EXEC SQL PREPARE sqlcommand FROM :sqlupdatestring ;
5)  EXEC SQL EXECUTE sqlcommand ;
    ...
```

10.2.4. SQLJ: Embedding SQL Commands in Java

- Standard adopted by several vendors for embedding SQL in Java
- Import several class libraries
- **Default context**
- Uses **exceptions** for error handling
  - `SQLException` is used to return errors or exception conditions

Figure 10.5 Importing classes needed for including SQLJ in Java programs in Oracle, and establishing a connection and default context.

```
1)  import java.sql.* ;
2)  import java.io.* ;
3)  import sqlj.runtime.* ;
4)  import sqlj.runtime.ref.* ;
5)  import oracle.sqlj.runtime.* ;
    ...
6)  DefaultContext cntxt =
7)  oracle.getConnection("<url name>", "<user name>", "<password>", true) ;
8)  DefaultContext.setDefaultContext(cntxt) ;
    ...
```

Figure 10.6 Java program variables used in SQLJ examples J1 and J2.

```
1) string dname, ssn , fname, fn, lname, ln,
   bdate, address ;
2) char sex, minit, mi ;
3) double salary, sal ;
4) integer dno, dnumber ;
```

Figure 10.7 Program segment J1, a Java program segment with SQLJ.

```
   //Program Segment J1:
 1) ssn = readEntry("Enter a Social Security Number: ") ;
 2) try {
 3)   #sql { SELECT Fname, Minit, Lname, Address, Salary
 4)     INTO :fname, :minit, :lname, :address, :salary
 5)     FROM EMPLOYEE WHERE Ssn = :ssn} ;
 6) } catch (SQLException se) {
 7)     System.out.println("Social Security Number does not exist: " + ssn) ;
 8)     Return ;
 9)   }
10) System.out.println(fname + " " + minit + " " + lname + " " + address
      + " " + salary)
```

10.2.5. Retrieving Multiple Tuples in SQLJ Using Iterators

- **Iterator**
  - Object associated with a collection (set or multiset) of records in a query result
- **Named iterator**
  - Associated with a query result by listing attribute names and types in query result
- **Positional iterator**
  - Lists only attribute types in query result

Figure 10.8 Program segment J2A, a Java program segment that uses a **named iterator** to print employee information in a particular department.

```
    //Program Segment J2A:
 0) dname = readEntry("Enter the Department Name: ") ;
 1) try {
 2)   #sql { SELECT Dnumber INTO :dnumber
 3)     FROM DEPARTMENT WHERE Dname = :dname} ;
 4) } catch (SQLException se) {
 5)   System.out.println("Department does not exist: " + dname) ;
 6)   Return ;
 7)   }
 8) System.out.printline("Employee information for Department: " + dname) ;
 9) #sql iterator Emp(String ssn, String fname, String minit, String lname,
      double salary) ;
10) Emp e = null ;
11) #sql e = { SELECT ssn, fname, minit, lname, salary
12)   FROM EMPLOYEE WHERE Dno = :dnumber} ;
13) while (e.next()) {
14)   System.out.printline(e.ssn + " " + e.fname + " " + e.minit + " " +
        e.lname  + " " + e.salary) ;
15) } ;
16) e.close() ;
```

Figure 10.9 Program segment J2B, a Java program segment that uses a **positional iterator** to print employee information in a particular department.

```
    //Program Segment J2B:
 0) dname = readEntry("Enter the Department Name: ") ;
 1) try {
 2)   #sql { SELECT Dnumber INTO :dnumber
 3)     FROM DEPARTMENT WHERE Dname = :dname} ;
 4) } catch (SQLException se) {
 5)   System.out.println("Department does not exist: " + dname) ;
 6)   Return ;
 7)   }
 8) System.out.printline("Employee information for Department: " + dname) ;
 9) #sql iterator Emppos(String, String, String, String, double) ;
10) Emppos e = null ;
11) #sql e = { SELECT ssn, fname, minit, lname, salary
12)   FROM EMPLOYEE WHERE Dno = :dnumber} ;
13) #sql { FETCH :e INTO :ssn, :fn, :mi, :ln, :sal} ;
14) while (!e.endFetch()) {
15)   System.out.printline(ssn + " " + fn + " " + mi + " " + ln + " " + sal) ;
16)   #sql { FETCH :e INTO :ssn, :fn, :mi, :ln, :sal} ;
17) } ;
18) e.close() ;
```

10.3. Database Programming with Function Calls: SQL/CLI & JDBC

- Use of function calls
  - **Dynamic** approach for database programming
- Library of functions
  - Also known as **application programming interface (API)**
  - Used to access database
- **SQL Call Level Interface (SQL/CLI)**
  - Part of SQL standard

10.3.1. SQL/CLI: Using Cas the Host Language

- **Environment record**
  - Track one or more database connections
  - Set environment information
- **Connection record**
  - Keeps track of information needed for a particular database connection
- **Statement record**
  - Keeps track of the information needed for one SQL statement

10.3.1. SQL/CLI: Using C as the Host Language (cont'd.)

- **Description record**
  - Keeps track of information about tuples or parameters
- **Handle** to the record
  - C pointer variable makes record accessible to program

Figure 10.10 Program segment CLI1, a C program segment with SQL/CLI.

```
    //Program CLI1:
 0) #include sqlcli.h ;
 1) void printSal() {
 2) SQLHSTMT stmt1 ;
 3) SQLHDBC con1 ;
 4) SQLHENV env1 ;
 5) SQLRETURN ret1, ret2, ret3, ret4 ;
 6) ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env1) ;
 7) if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, env1, &con1) else exit ;
 8) if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz",
      SQL_NTS) else exit ;
 9) if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &stmt1) else exit ;
10) SQLPrepare(stmt1, "select Lname, Salary from EMPLOYEE where Ssn = ?",
      SQL_NTS) ;
11) prompt("Enter a Social Security Number: ", ssn) ;
12) SQLBindParameter(stmt1, 1, SQL_CHAR, &ssn, 9, &fetchlen1) ;
13) ret1 = SQLExecute(stmt1) ;
14) if (!ret1) {
15)   SQLBindCol(stmt1, 1, SQL_CHAR, &lname, 15, &fetchlen1) ;
16)   SQLBindCol(stmt1, 2, SQL_FLOAT, &salary, 4, &fetchlen2) ;
17)   ret2 = SQLFetch(stmt1) ;
18)   if (!ret2) printf(ssn, lname, salary)
19)     else printf("Social Security Number does not exist: ", ssn) ;
20)   }
21) }
```

Figure 10.11 Program segment CLI2, a C program segment that uses SQL/CLI for a query with a **collection of tuples** in its result.

```
   //Program Segment CLI2:
 0) #include sqlcli.h ;
 1) void printDepartmentEmps() {
 2) SQLHSTMT stmt1 ;
 3) SQLHDBC con1 ;
 4) SQLHENV env1 ;
 5) SQLRETURN ret1, ret2, ret3, ret4 ;
 6) ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env1) ;
 7) if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, env1, &con1) else exit ;
 8) if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz",
      SQL_NTS) else exit ;
 9) if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &stmt1) else exit ;
10) SQLPrepare(stmt1, "select Lname, Salary from EMPLOYEE where Dno = ?",
      SQL_NTS) ;
11) prompt("Enter the Department Number: ", dno) ;
12) SQLBindParameter(stmt1, 1, SQL_INTEGER, &dno, 4, &fetchlen1) ;
13) ret1 = SQLExecute(stmt1) ;
14) if (!ret1) {
15)   SQLBindCol(stmt1, 1, SQL_CHAR, &lname, 15, &fetchlen1) ;
16)   SQLBindCol(stmt1, 2, SQL_FLOAT, &salary, 4, &fetchlen2) ;
17)   ret2 = SQLFetch(stmt1) ;
18)   while (!ret2) {
19)     printf(lname, salary) ;
20)     ret2 = SQLFetch(stmt1) ;
21)     }
22)   }
23) }
```

10.3.2. JDBC: SQL Function Calls for Java Programming

- JDBC
  - Java function libraries
- Single Java program can connect to several different databases
  - Called data sources accessed by the Java program
- `Class.forName("oracle.jdbc.driver.OracleDriver")`
  - Load a **JDBC driver** explicitly

10.3.2. JDBC: SQL Function Calls for Java Programming

- **`Connection` object**
- **`Statement` object** has two subclasses:
  - `PreparedStatement` and `CallableStatement`
- Question mark (`?`) symbol
  - Represents a statement parameter
  - Determined at runtime
- **`ResultSet` object**
  - Holds results of query

Figure 10.12 Program segment JDBC1, a Java program segment with JDBC.

```
   //Program JDBC1:
 0) import java.io.* ;
 1) import java.sql.*
   ...
 2) class getEmpInfo {
 3)   public static void main (String args []) throws SQLException, IOException {
 4)     try { Class.forName("oracle.jdbc.driver.OracleDriver")
 5)     } catch (ClassNotFoundException x) {
 6)       System.out.println ("Driver could not be loaded") ;
 7)     }
 8)     String dbacct, passwrd, ssn, lname ;
 9)     Double salary ;
10)     dbacct = readentry("Enter database account:") ;
11)     passwrd = readentry("Enter password:") ;
12)     Connection conn = DriverManager.getConnection
13)       ("jdbc:oracle:oci8:" + dbacct + "/" + passwrd) ;
14)     String stmt1 = "select Lname, Salary from EMPLOYEE where Ssn = ?" ;
15)     PreparedStatement p = conn.prepareStatement(stmt1) ;
16)     ssn = readentry("Enter a Social Security Number: ") ;
17)     p.clearParameters() ;
18)     p.setString(1, ssn) ;
19)     ResultSet r = p.executeQuery() ;
20)     while (r.next()) {
21)       lname = r.getString(1) ;
22)       salary = r.getDouble(2) ;
23)       system.out.printline(lname + salary) ;
24)   } }
25) }
```

Figure 10.13 Program segment JDBC2, a Java program segment that uses JDBC for a query with a **collection of tuples** in its result.

```
    //Program Segment JDBC2:
 0) import java.io.* ;
 1) import java.sql.*
    ...
 2) class printDepartmentEmps {
 3)   public static void main (String args [])
          throws SQLException, IOException {
 4)     try { Class.forName("oracle.jdbc.driver.OracleDriver")
 5)     } catch (ClassNotFoundException x) {
 6)       System.out.println ("Driver could not be loaded") ;
 7)     }
 8)     String dbacct, passwrd, lname ;
 9)     Double salary ;
10)     Integer dno ;
11)     dbacct = readentry("Enter database account:") ;
12)     passwrd = readentry("Enter password:") ;
13)     Connection conn = DriverManager.getConnection
14)       ("jdbc:oracle:oci8:" + dbacct + "/" + passwrd) ;
15)     dno = readentry("Enter a Department Number: ") ;
16)     String q = "select Lname, Salary from EMPLOYEE where Dno = " +
        dno.tostring() ;
17)     Statement s = conn.createStatement() ;
18)     ResultSet r = s.executeQuery(q) ;
19)     while (r.next()) {
20)       lname = r.getString(1) ;
21)       salary = r.getDouble(2) ;
22)       system.out.printline(lname + salary) ;
23)   } }
24) }
```

10.4. Database Stored Procedures and SQL/PSM (More Discussion of Oracle PL/SQL in Part B)

- **Stored procedures**
  - Program modules stored by the DBMS at the database server
  - Can be functions or procedures
- SQL/PSM (**SQL/Persistent Stored Modules**)
  - Extensions to SQL
  - Include general-purpose programming constructs in SQL

10.4. Database Stored Procedures and SQL/PSM

- **Persistent stored modules**
  - Stored persistently by the DBMS
- Useful:
  - When database program is needed by several applications
  - To reduce data transfer and communication cost between client and server in certain situations
  - To enhance modeling power provided by views

10.4. Database Stored Procedures and SQL/PSM

- Declaring stored procedures:

```
CREATE PROCEDURE <procedure name> (<parameters>)
<local declarations>
<procedure body> ;
declaring a function, a return type is necessary,
  so the declaration form is
CREATE FUNCTION <function name> (<parameters>)
RETURNS <return type>
<local declarations>
<function body> ;
```

10.4. Database Stored Procedures and SQL/PSM

- Each parameter has parameter type
  - **Parameter type**: one of the SQL data types
  - **Parameter mode**: `IN`, `OUT`, or `INOUT`
- Calling a stored procedure:

```
CALL <procedure or function name>
(<argument list>) ;
```

SQL/PSM: Extending SQL for Specifying Persistent Stored Modules

- Conditional branching statement:

```
IF <condition> THEN <statement list>
ELSEIF <condition> THEN <statement list>
...
ELSEIF <condition> THEN <statement list>
ELSE <statement list>
END IF ;
```

SQL/PSM (cont'd.)

- Constructs for looping

```
WHILE <condition> DO
        <statement list>
END WHILE ;
REPEAT
        <statement list>
UNTIL <condition>
END REPEAT ;

FOR <loop name> AS <cursor name> CURSOR FOR <query> DO
        <statement list>
END FOR ;
```

Figure 10.14 Declaring a function in SQL/PSM.

```
//Function PSM1:
0) CREATE FUNCTION Dept_size(IN deptno INTEGER)
1) RETURNS VARCHAR [7]
2) DECLARE No_of_emps INTEGER ;
3) SELECT COUNT(*) INTO No_of_emps
4) FROM EMPLOYEE WHERE Dno = deptno ;
5) IF No_of_emps > 100 THEN RETURN "HUGE"
6) ELSEIF No_of_emps > 25 THEN RETURN "LARGE"
7) ELSEIF No_of_emps > 10 THEN RETURN "MEDIUM"
8) ELSE RETURN "SMALL"
9) END IF ;
```

Comparing the Three Approaches

- Embedded SQL Approach
  - Query text checked for syntax errors and validated against database schema at compile time
  - For complex applications where queries have to be generated at runtime
    - Function call approach more suitable

Comparing the Three Approaches (cont'd.)

- Library of Function Calls Approach
  - More flexibility
  - More complex programming
  - No checking of syntax done at compile time
- Database Programming Language Approach
  - Does not suffer from the impedance mismatch problem
  - Programmers must learn a new language

Summary

- Techniques for database programming
  - Embedded SQL
  - SQLJ
  - Function call libraries
  - SQL/CLI standard
  - JDBC class library
  - Stored procedures
  - SQL/PSM