Ders # 7

DB Programming: SQL Programming Techniques

From Elmasri/Navathe textbook Ch9,26

Sciore textbook, Ch 9-10

Outline:

- Database Programming Approaches
- Embedded SQL
- JDBC
- Stored Procedures, SQL/PSM
- Summary

Objective:

❑Various techniques for accessing and manipulating a database via programs in general-purpose languages s.a. Java,C, etc.

❑Modern application programming architectures and techniques s.a. JAVA

Database Programming

- 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 (increasingly thru web applications)

Database Programming Approaches

1. **Embedded commands:**
   - Database commands are embedded in a general-purpose programming language
2. **Library of database functions:**
   - Available to the host language for database calls; known as an *API*
     - *API* standards for Application Program Interface
3. **A brand new, full-fledged 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

Basic Steps in Database Programming

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

1-) Embedded SQL

- Most SQL statements can be embedded in a general-purpose *host* programming language such as ADA,COBOL, C, Java
- An embedded SQL statement is distinguished from the host language statements by enclosing it between **EXEC SQL** and a matching **END-EXEC** (or **EXEC SQL BEGIN** and **EXEC SQL END**); semicolon also indicates the end.
- Syntax may vary with language
- *Shared variables* (used in both languages)
  - usually prefixed with a colon (:) in SQL;
  - used without (:) in the host program.

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

```
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 ;
```

Figure 13.1
C program variables used in the embedded SQL examples E1 and E2.

Variable Declaration:

- Variables inside **DECLARE** are shared and can appear (while prefixed by a colon) in SQL statements
- **SQLCODE and SQLSTATE:** to communicate errors/exceptions between the database and the program
- **SQLSTATE:** String of five characters
  - '00000' = no error or exception; Other values indicate various errors
  - For example, '02000' indicates 'no more data' when using SQLSTATE
- SQLCODE variable
  - 0 = statement executed successfully
  - 100 = no more data available in query result
  - < 0 = indicates some error has occurred

Connecting to a Database:

- Connection (multiple connections: possible but only one is active)

```
CONNECT TO server-name AS connection-name
AUTHORIZATION user-account-info;
```

- Change from an active connection to another one

```
SET CONNECTION connection-name;
```

- Disconnection: `DISCONNECT connection-name;`

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

```
    //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)   }
```

Figure 13.2
Program segment E1, a C program segment with embedded SQL.

Retrieving Multiple Tuples with Embedded SQL Using Cursors

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

Figure 13.3
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 ;
```

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: where to move cursor
  - Added right after fetch using value: `NEXT`, `PRIOR`, `FIRST`, `LAST`, `ABSOLUTE` *i*, and `RELATIVE` *i*
  - `Ex: Fetch NEXT ...`, `Fetch FIRST...`, Fetch `LAST...`

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

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
    - a point-and-click operation translates to certain SQL query
- Dynamic update is relatively simple; dynamic query can be complex
  - because the type and number of retrieved attributes are unknown at compile time
- 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;

- No syntax check or other types of checks are possible at copile time..
- Unable to know the type or number of attributes to be retrievedby the SQL query at the compile time.
- PREPARE is useful in case the dynamic SQL is to be executed in the code repeatedly.

Embedded SQL in Java: SQLJ

- 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**

```
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 13.5
Importing classes needed for including SQLJ in Java programs in Oracle, and establishing a connection and default context.

Figure 13.6
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 ;
```

SQLJ: Retrieving Single Tuple

```
    //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)
```

Figure 13.7
Program segment J1, a Java program segment with SQLJ.

Retrieving Multiple Tuples in SQLJ Using Iterators

- **Iterator**
  - Object associated with a collection (set or multiset) of records in a query result (table)
  - Two types: named iterator and positional iterator
- **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
- A **`FETCH`** operation: retrieves the next tuple in a query result:

```
fetch iterator-var into program-var
```

Retrieving Multiple Tuples in SQLJ Using Named Iterators

Figure 13.8
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() ;
```

Retrieving Multiple Tuples in SQLJ Using Positional Iterators (last slide covered)

Figure 13.9
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() ;
```

2-) Database Programming with Function Calls

- Embedded SQL provides static database programming
- **API**: Dynamic database programming with a library of functions
  - Advantage:
    - No preprocessor needed (thus more flexible)
  - Disadvantage:
    - SQL syntax checks to be done at run-time
- Example:
  - SQL/CLI
    - A part of the SQL standard
    - Provides easy access to several databases within the same program
    - Certain libraries (e.g., **sqlcli.h** for C) have to be installed and available
    - SQL statements are dynamically created and passed as string parameters in the calls
  - **JDBC**
    - **SQL connection function calls for Java programming**
    - **A Java program with JDBC functions can access any relational DBMS that has a JDBC driver**
    - **JDBC allows a program to connect to several databases (known as data sources)**

```
    //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 13.10
Program segment CLI1, a C program segment with SQL/CLI.

Figure 13.11
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)  }
```

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 connect 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

Figure 13.12
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) }
```

```
    //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.println(lname + salary) ;
23)   } }
24) }
```

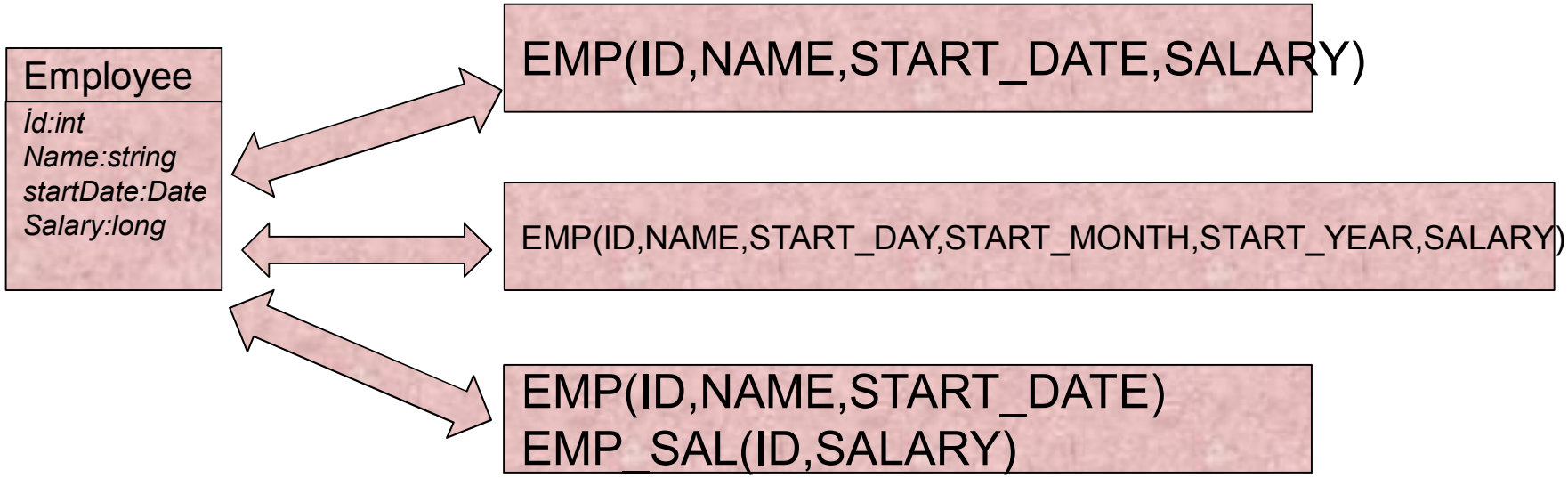
Figure 13.13
Program segment JDBC2, a Java program segment that uses JDBC for a query with a collection of tuples in its result.

Tanımlar

- Empedans Uyumsuzluğu *(Impedance Mismatch):*
  - *Java programları yapı birimi: nesneler*
  - *VT temel veri birimi: kayıtlar*
- Nesne-Yönelimli Programlarda, View-Model ayrımının
  - View: Kullanıcı arayüzü kapsulü
  - Model: VT erişim ve sorgulama kapsulü
- View-Model ayrımının 2 temel avantajı:
  - Basit programlama
  - Esnek kullanım *(aynı model farklı view'larda kullanılabilir)*
- *View-Model bağıntısı ~ Client-Server bağıntısı*

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
    - **Class representation**
    - Relationships
    - Inheritence
  - set vs. record-at-a-time processing
    - need special iterators to loop over query results and manipulate individual values



3-)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
  - Enhance the modeling power of views
- Disadvantages:
  - Every DBMS has its own syntax and this can make the system less portable

Stored Procedure Constructs

- SQL/PSM:
  - Part of the SQL standard for writing persistent stored modules
- SQL + stored procedures/functions + additional programming constructs
  - E.g., branching and looping statements
  - Enhance the power of SQL
- 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);

SQL/PSM: Example#1

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

SELECT COUNT (*) INTO TOT_EMPS
    FROM 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;
```

Summary

- 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
  - JDBC
  - Stored procedure and functions