CS 377
Database Systems
SQL Programming

Li Xiong
Department of Mathematics and Computer Science
Emory University
A SQL Query Joke

- A SQL query walks into a bar and sees two tables. He walks up to them and says 'Can I join you?'
Database Programming

- Database access through interactive interface (SQL*Plus)
- Database access through application programs or web applications
Database Programming Approaches

- **Embedded SQL**: database commands are embedded in a general-purpose programming language (language extension)
  - SQLJ

- **Database API**: library of database functions available to the host language (API)
  - JDBC

- **New language**
  - PL/SQL
Comparison

- **Embedded SQL:**
  - programs tend to be shorter
  - SQL syntax can be checked at compile time.
  - static SQL which performs better than dynamic SQL
  - requires a preprocessing step

- **Database API:**
  - Flexible and dynamic SQL for more complex applications

- **New language**
  - Fast, tightly integrated, minimizes impedance mismatch
  - Product specific
Impedance Mismatch

- Incompatibilities between a host programming language and the database model
  - Attribute type mismatch and incompatibilities; requires binding for each language
  - set vs. record-at-a-time processing; requires iterators to loop over query results and manipulate individual values
Sequence of Interaction in Database Programming

1. Opens a connection to database server
2. Interacts with database by submitting queries, updates, and other commands
3. Terminates the connection to database
Database Programming Approaches

- **Embedded SQL**: database commands are embedded in a general-purpose programming language (language extension)
  - SQLJ
- **Database API**: library of database functions available to the host language (API)
  - JDBC
- **New language**
  - PL/SQL
Embedded SQL in Java

- SQLJ: an language extension for embedding SQL in Java
- An embedded SQL statement is distinguished from the host language statements by special prefix
- A preprocessor SQLJ translator is required to convert SQL statements into Java before the application program is compiled
Figure 13.8
Program segment J2A, a Java program segment that uses a named iterator to print employee information in a particular department.

```java
//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.println("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.println(e.ssn + " " + e.fname + " " + e.minit + " " + e.lname + " " + e.salary) ;
15) } ;
16) e.close() ;
```
Oracle SQLJ

```bash
% sqlj emp.sqlj -user=scott/tiger
% java emp
```
Database Programming Approaches

- **Embedded SQL**: database commands are embedded in a general-purpose programming language (language extension)
  - SQLJ
- **Database API**: library of database functions available to the host language (API)
  - JDBC
- **New language (SQL extension)**
  - PL/SQL
PL/SQL

- PL/SQL (Procedural Language/Structured Query Language) is Oracle's proprietary server-based procedural extension to the SQL database language.
- Gives user the power of general programming language
DECLARE
    v_Counter BINARY_INTEGER := 1;
BEGIN
    LOOP
        INSERT INTO MyTable
            VALUES (v_Counter, 'Loop index');
        v_Counter := v_Counter + 1;
        IF v_Counter > 50 THEN
            EXIT;
        END IF;
    END LOOP;
END; /

Database Programming Approaches

- **Embedded SQL**: database commands are embedded in a general-purpose programming language (language extension)
  - SQLJ

- **Database API**: library of database functions available to the host language (API)
  - JDBC

- **New language**
  - PL/SQL
JDBC

- JDBC: Java library for SQL function calls
- Consists of a set of classes and interfaces written in Java
- Allows the programmer to send SQL statements to a database server for execution and retrieve query results
JDBC use JDBC Drivers to support communication transparency with specific DBMS.

A JDBC driver converts requests from Java programs to a protocol that a particular DBMS can understand.

A Java program with JDBC functions can access any relational DBMS that has a JDBC driver.

Different types of JDBC driver.
JDBC Driver

Calling Java Application

JDBC API

JDBC Driver Manager

Native-Protocol driver
(Type 4 Driver)

direct calls using
specific database protocol

Database
Before writing/running JDBC programs

- Set up CLASSPATH to include Oracle's JDBC libraries on Math/CS server
  $ORACLE_HOME/jdbc/lib/ojdbc14.jar
  (/oracle/app/oracle/product/10.2.0/jdbc/lib/ojdbc14.jar)
Writing JDBC programs

1. **Import JDBC library** *(java.sql.*)*
2. **Load appropriate JDBC driver** *(via DriverManager.registerDriver)*
3. **Create a connection object** *(via DriverManager.getConnection)*
4. **Create a statement object from the** Statement class:
   1. Statement
   2. PreparedStatement
   3. CallableStatement
5. **Execute SQL statement** *(referenced by an object)* via JDBC’s `executeQuery`
6. **Process query results** *(returned in an object of type ResultSet)*
Step 2. Load JDBC Driver

- Use generic java function for loading a class (must use try/catch block)
  
  Class.forName("oracle.jdbc.driver.OracleDriver")

- Use DriverManager class (defined in java.sql package) that keeps track of installed drivers
  
  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
Step 3. Connect to the Database

Connection conn = DriverManager.getConnection(url, user, password);

- The DriverManager.getConnection create a Connection object that maintains all the state information of a connection to the database server.
- Connect URL for Math/CS Oracle Server for CS377
  "jdbc:oracle:thin:@cs377.mathcs.emory.edu:1521:cs377"
  - Name of machine where the SQL server is running: cs377.mathcs.emory.edu
  - Port: 1521 (Oracle's default port)
  - Database space name: cs377
Step 4. Creating SQL Statement

Connection conn = DriverManager.getConnection (url, user, password);
Statement stmt = SQLconnection.createStatement();

- createStatement in Connection class creates a Statement object for sending SQL statements to the database server.
5. Executing SQL Statements - Queries

Connection conn = DriverManager.getConnection(url, user, password);
Statement stmt = SQLconnection.createStatement();
ResultSet rset = stmt.executeQuery("SELECT * FROM cs377000.employee");

- executeQuery(String sql) in Statement class
  - Executes a given SQL statement
  - returns a single ResultSet object

- Retrieve data from ResultSet (Step 6)
5. Executing SQL Statements - Updates

```java
int count = stmt.executeUpdate( "INSERT INTO
cs377000.Employee(fname) VALUES ( 'John' ) " );
```

- **executeUpdate(String sql)** in Statement class
  - executes a given SQL statement, which may be an INSERT, UPDATE, or DELETE statement or an SQL DDL
  - returns an update count
Prepared Statement

- Prepared Statement - precompiled and can be executed multiple times in an efficient manner.

- Step 4. Prepare a statement
  - `prepareStatement(String sql)` of Connection class – prepare a statement and returns PreparedStatement

- Step 5. Executing SQL statements
  - `ResultSet executeQuery()` of PreparedStatement class - Executes the SQL query in this PreparedStatement object and returns the ResultSet
  - `int executeUpdate()` of PreparedStatement class – Executes SQL INSERT, UPDATE or DELETE statement; and returns update count.
Step 6. Process Query Results

```java
ResultSet rset = stmt.executeQuery("SQL-query");
while ( rset.next () )
{
    stringVar = rset.getString(index);
    intVar = rset.getInt(index);
}
```

- A ResultSet object maintains a cursor to current row
- Initially the cursor is positioned before the first row
- Use cursor methods to iterate through tuples (rows)
  - `next()` moves the cursor forward one row. `next()` if there are still more tuples and returns `TRUE`
- Use *getter* methods to retrieve attribute (column) values from the current row
  - `getString(int columnIndex)`
  - `getString(String columnLabel)`
  - …
# Common Java and SQL Type Equivalence

<table>
<thead>
<tr>
<th>Java method</th>
<th>SQL Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>getInt</td>
<td>INTEGER</td>
</tr>
<tr>
<td>getLong</td>
<td>BIG INT</td>
</tr>
<tr>
<td>getFloat</td>
<td>REAL</td>
</tr>
<tr>
<td>getDouble</td>
<td>FLOAT</td>
</tr>
<tr>
<td>getBignum</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>getBoolean</td>
<td>BIT</td>
</tr>
<tr>
<td>getString</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>getString</td>
<td>CHAR</td>
</tr>
<tr>
<td>getDate</td>
<td>DATE</td>
</tr>
<tr>
<td>getTime</td>
<td>TIME</td>
</tr>
<tr>
<td>getTimestamp</td>
<td>TIME STAMP</td>
</tr>
<tr>
<td>getObject</td>
<td>any type</td>
</tr>
</tbody>
</table>
Getting Meta Data about Result Set

```java
ResultSet rset = stmt.executeQuery("SQL-query");
ResultSetMetaData metaData = rset.getMetaData();
```

- **getMetaData() of ResultSet**
  - creates a `ResultSetMetaData` object that contains meta information about the result set

- **Getter methods of ResultSetMetaData**
  - `int get_Column_Count()` - returns the number of columns in the table
  - `String get_Column_Name(int columnIndex)` - returns the name of column "columnIndex"
  - `String get_Column_Type_Name(int columnIndex)` - returns the type of column "columnIndex"
Demo

- Employee.java
- MetaData.java