Advanced Programming

Code: SWEG2033

Chapter four

Java Database Connectivity

Objective:

- To introduce database systems, and how to develop database applications using Java.

Introduction

- **Database** is an organized collection of data(file) having rule to access it.
- **Relational database** is a set of tables containing data fitted into predefined categories. Each table (which is sometimes called a *relation*) contains one or more data categories in columns.
- To using relational database programs, a standard language called Structured Query Language (SQL) was developed.
- **SQL** is the industry-standard approach to accessing relational databases.
- **SQL(Structured Query Language)**: a language used with relational database to perform ***queries & to manipulate data***. **SQL** is a standard for database access that has been adopted by virtually all database vendors.
- In database programming, a request for records in a database is called a *query*.

Introduction

- **DBMS** provides a mechanism to store, retrieve, organize and modify data for many users on database.
- **E.g., of RDBMSs:** SQL Server, Oracle, Sybase, DB2, MySQL, etc.
- **Integrity Constraints** : Integrity constraints provide a way of ensuring that changes made to the database by authorized users do not result in a loss of data consistency.

Introduction

- Three types of constraints on table columns: domain constraints, primary key constraints, and foreign key constraints.
  - *Domain constraints* specify the permissible values for an attribute.
  - The *foreign key constraints* define the relationships among relations.
  - A **primary key** (PK) is a single column or combination of columns (called a compound **key**) that uniquely identifies each row in a table.

JDBC

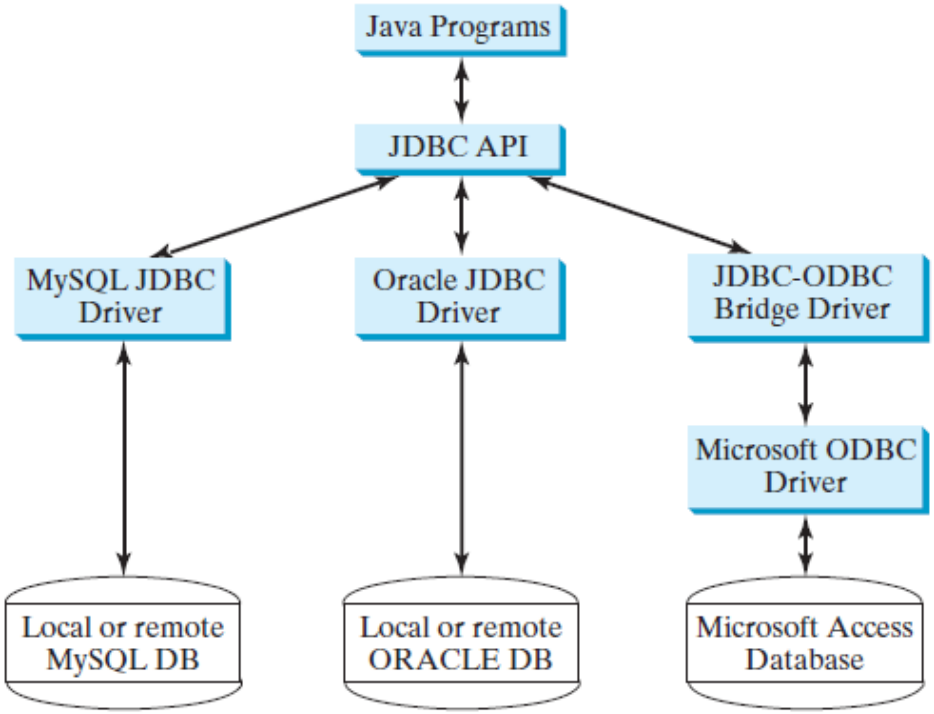
Java Database Connectivity

- Java Database Connectivity (JDBC) is a set of classes that can be used to develop client/server applications that work with databases developed by Microsoft, Sybase, Oracle, IBM, and other sources.
- With JDBC, you can use the same methods and classes in Java programs to read and write records and perform other kinds of database access.
- A class called a driver acts as a bridge to the database source. There are drivers for each of the popular databases.

JDBC

- JDBC provides a standard library for accessing relational database.
- JDBC provides Java programmers with a uniform interface for accessing and manipulating a wide range of relational databases.
- Using the JDBC API, applications written in the Java can execute SQL statements, retrieve results, present data in a user-friendly interface, and propagate changes back to the database.
- The JDBC API can also be used to interact with multiple data sources in a distributed, heterogeneous environment.

JDBC


Java Programs
JDBC API
MySQL JDBC Driver
Oracle JDBC Driver
JDBC-ODBC Bridge Driver
Microsoft ODBC Driver
Local or remote MySQL DB
Local or remote ORACLE DB
Microsoft Access Database

JDBC

- In General, you explore JDBC in the following ways:
  - Using JDBC drivers to work with different relational databases
  - Accessing a database with Structured Query Language (SQL)
  - Reading records from a database using SQL and JDBC
  - Adding records to a database using SQL and JDBC
  - Creating a new Java DB database and reading its records

JDBC

- JDBC consists two parts :
  - JDBC API a purely java based API
  - JDBC Driver Manager Which communicates with Vendor specific driver that perform the real communication with database.

Developing Database Applications

- The JDBC API is a Java application program interface to generic SQL databases.
- The JDBC API consists of classes and interfaces for:
  - Establishing/making connections with databases,
  - Crating and Sending SQL statements to databases,
  - Executing that SQL query in the database
  - Processing/Viewing the results of the SQL statements, and
  - Obtaining database metadata.
- N.B: These JDBC classes all are part of the java.sql package.
- Four key interfaces are needed to develop any database application using Java: **Driver**, **Connection**, **Statement**, and **ResultSet**.
- The JDBC driver vendors provide implementation for them. Programmers use the interfaces.

Developing Database Applications

- A JDBC application:
  - Loads an appropriate driver using the **Driver** interface,
  - Connects to the database using the **Connection** interface,
  - Creates and executes SQL statements using the **Statement** interface, and
  - Processes the result using the **ResultSet** interface if the statements return results.
- **Note** that some statements, such as SQL data definition statements and SQL data modification statements, **do not return results**.

Developing Database Applications

- Seven Basic Steps in Using JDBC

1. Load the driver
2. Define the connection URL
3. Establish the Connection
4. Crate statement object
5. Execute a query
6. Process the results
7. Close the connection

Developing Database Applications

- 1. **Loading drivers**
- An appropriate driver must be loaded using the statement shown below before connecting to a database.
  - Class.forName("**JDBCDriverClass**");

Database	**Driver Class**
Access	**sun.jdbc.odbc.JdbcOdbcDriver**
MySQL	**com.mysql.jdbc.Driver**
Oracle	**oracle.jdbc.driver.OracleDriver**

```
try {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Class.forName("org.gjt.mm.mysql.Driver");
} catch { ClassNotFoundException cnfe) {
  System.out.println("Error loading driver: " cnfe);
}
```

Developing Database Applications

- 2. **Define the connection URL**

```
String host = "dbhost.yourcompany.com";
String dbName = "someName";
int port = 1234;
String oracleURL = "jdbc:oracle:thin:@" + host +
                   ":" + port + ":" + dbName;
String mysqlURL = "jdbc:mysql://" + host  +
                  ":" + port + "/" + dbName;
```

Developing Database Applications

- **3. Establishing connections**.
- To connect to a database, use the static method getConnection(databaseURL) in the DriverManager class, as follows:
  - Connection connection = DriverManager.getConnection(databaseURL);
  - Example:
    - *Connection connection = DriverManager.getConnection ("**jdbc:odbc:ExampleMDBDataSource**");*

```
String username = "jay_debesee";
String password = "secret";
Connection connection =
   DriverManager.getConnection(oracleURL,
                               username,
                               password);
```

Developing Database Applications

Database	URL Pattern
Access	**jdbc:odbc:dataSource**
MySQL	**jdbc:mysql://hostname/dbname**
Oracle	**jdbc:oracle:thin:@hostname:port#:oracleDBSID**

Optionally, look up information about the database

```
DatabaseMetaData dbMetaData = connection.getMetaData();
String productName =
  dbMetaData.getDatabaseProductName();
System.out.println("Database: " + productName);
String productVersion =
  dbMetaData.getDatabaseProductVersion();
System.out.println("Version: " + productVersion);
```

Developing Database Applications

4. Creating Statements

- If a **Connection** object can be envisioned as a cable linking your program to a database, an object of **Statement** can be viewed as a cart that delivers SQL statements for execution by the database and brings the result back to the program
- Once a **Connection** object is created, you can create statements for executing SQL statements as follows:
  - *Statement statement = connection.createStatement();*

Developing Database Applications

5. Executing Statements

- SQL DDL or update statement can be executed using **executeUpdate(String sql)**
- SQL query statement can be executed using **executeQuery(String sql)**.
- The result of the query is returned in **ResultSet**.
- For example, the following code executes the SQL statement **create table Temp (col1 char(5), col2 char(5))**:
  - *statement.executeUpdate("**create table Temp (col1 char(5), col2 char(5))**");*

Developing Database Applications

- The next code executes the SQL query **select firstName, mi, lastName from Student where lastName = 'Smith'**:

// Select the columns from the Student table

ResultSet resultSet = statement.executeQuery

("select firstName, mi, lastName from Student where lastName "

+ **" = 'Smith'"**);

```
String query = "SELECT col1, col2, col3 FROM sometable";
ResultSet resultSet = statement.executeQuery(query);
```

Developing Database Applications

- **6. Processing ResultSet**
- The **ResultSet** maintains a table whose current row can be retrieved.
- The initial row position is **null**.
- You can use the **next** method to move to the next row and the various **get** methods to retrieve values from a current row.
- For example, the code given below displays all the results from the preceding SQL query.

```
// Iterate through the result and print the student names
while (resultSet.next())
System.out.println(resultSet.getString(1) + " " +
resultSet.getString(2) + ". " + resultSet.getString(3));
```

Developing Database Applications

7. Close the connection

Connection.close();