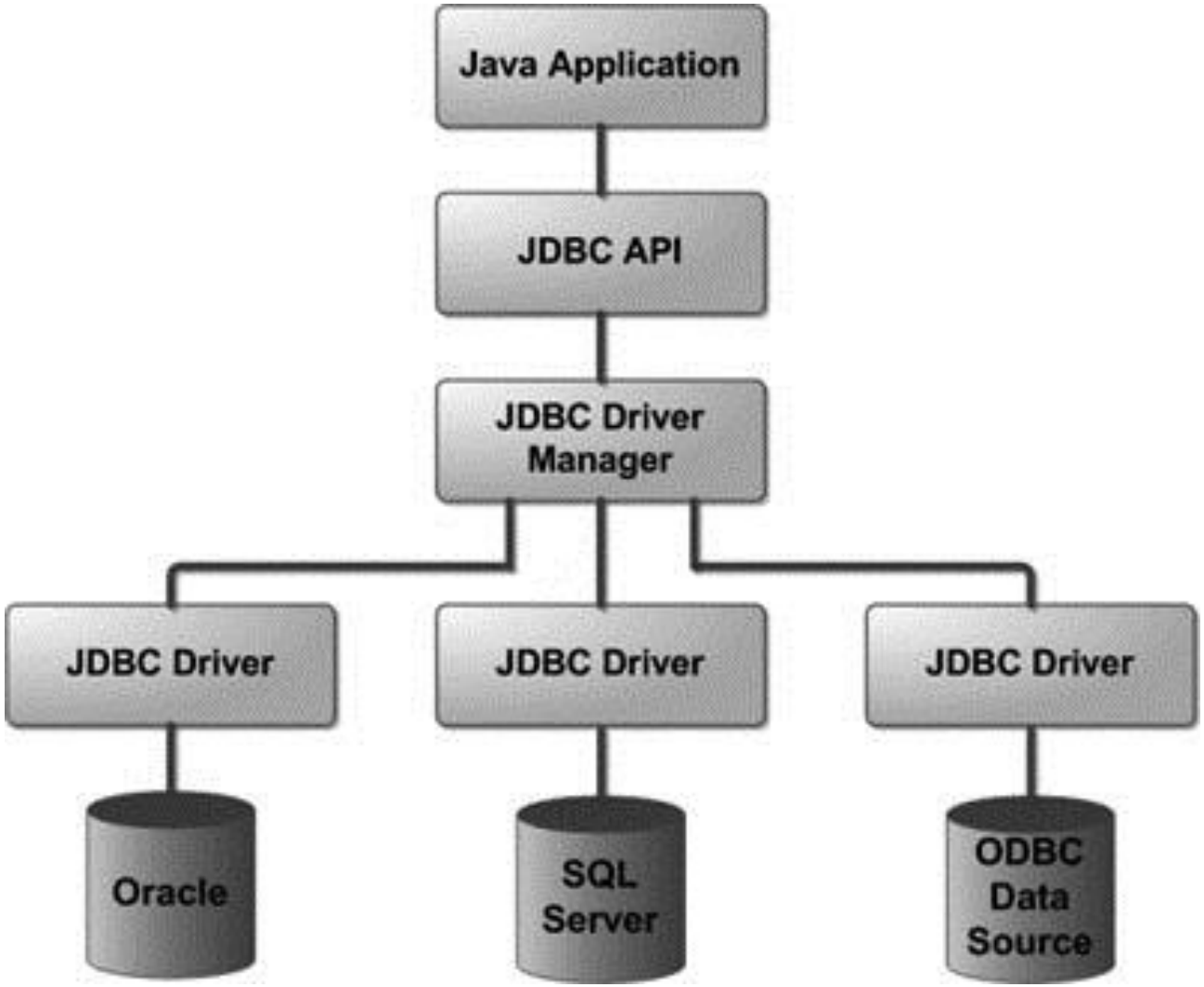
Java Database Connectivity (JDBC)

Introduction

- A set of standard Java APIs for database-independent connectivity between the Java programming language and a wide range of databases (MySQL, Oracle, DB2, MS SQL Server, and more...)

JDBC Architecture

- JDBC API & JDBC Driver API



JDBC Components

- DriverManager
- Driver
- Connection
- Statement
- ResultSet
- SQLException

JDBC Functionalities

1. Making a connection to a database
2. Creating SQL statements
3. Executing SQL queries in the database
4. Viewing and modifying the resulting records

Creating a Simple JDBC Application

- General steps
  1. Import the packages
  2. Register the JDBC driver
  3. Open a connection
  4. Execute a query
  5. Extract data from the result set
  6. Clean up the environment

Step 1: Import the Packages

- Include the packages containing the JDBC classes needed for database programming

```
import java.sql.*;
```

Step 2: Register the JDBC Driver

- Initialize a driver, in order to establish a communication channel with the database

```
Class.forName("com.mysql.jdbc.Driver");
```

Step 3: Open a Connection

- Create a *Connection* object, which represents a physical connection with the database

```
static final String USER = "username";
static final String PASS = "password";
conn = DriverManager.getConnection(DB_URL,USER,PASS);
```

Step 4: Execute a Query

- Using *Statement* or *PreparedStatement* for building and submitting an SQL statement to the database

```
stmt = conn.createStatement();
String sql;
sql = "SELECT id, name, hometown FROM t_student";
ResultSet rs = stmt.executeQuery(sql);
```

Step 4: Execute a Query (cont.)

- For *UPDATE*, *INSERT* or *DELETE* statement

```
stmt = conn.createStatement();
String sql;
sql = "DELETE FROM t_student WHERE id = 1";
int num = stmt.executeUpdate(sql);
```

Step 5: Extract Data from Result Set

- Use *ResultSet.getXXX()* to retrieve data from the result set

```
while(rs.next()){
    //Retrieve by column name
    int id  = rs.getInt("id");
    String name = rs.getString(“name");
    String hometown = rs.getString(“hometown");
}
```

Step 6: Clean Up the Environment

- Explicitly close all database resources

```
rs.close();
stmt.close();
conn.close();
```

Data Types

SQL	*JDBC/Java*	*setXXX*	*updateXXX*
VARCHAR	java.lang.String	setString	updateString
CHAR	java.lang.String	setString	updateString
NUMERIC	java.math.BigDecimal	setBigDecimal	updateBigDecimal
INTEGER	int	setInt	updateInt
FLOAT	float	setFloat	updateFloat
DOUBLE	double	setDouble	updateDouble
DATE	java.sql.Date	setDate	updateDate
TIME	java.sql.Time	setTime	updateTime
TIMESTAMP	java.sql.Timestamp	setTimestamp	updateTimestamp

Executing Queries

- Send query (SELECT) to database
  - executeQuery: returns one ResultSet object
- Send update (INSERT, DELETE, UPDATE) to database
  - executeUpdate: returns an integer representing the number of rows affected by the SQL statement

Three Types of Statements

Interface	*Recommended Use*
Statement Used to implement simple SQL statements with no parameters.	• Use for general-purpose access to your database. • Useful when you are using static SQL statements at runtime. • The Statement interface cannot accept parameters.
PreparedStatement (Extends Statement.) Used for precompiling SQL statements that might contain input parameters.	• Use when you plan to use the SQL statements many times. • The PreparedStatement interface accepts input parameters at runtime.
CallableStatement (Extends PreparedStatement.) Used to execute stored procedures that may contain both input and output parameters.	• Use when you want to access database stored procedures. • The CallableStatement interface can also accept runtime input parameters.

The Statement Object

- Using *createStatement( )* method

```
Statement stmt = null;
try {
   stmt = conn.createStatement( );
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   stmt.close();
}
```

The Statement Object (cont.)

- Three execute methods
  - boolean execute(String SQL)
  - int executeUpdate(String SQL)
    - Returning the numbers of rows affected by the execution
  - ResultSet executeQuery(String SQL)
    - Returning a ResultSet object

The PreparedStatement Object

```
PreparedStatement pstmt = null;
try {
   String SQL = “UPDATE t_student SET age = ? WHERE id = ?";
   pstmt = conn.prepareStatement(SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   pstmt.close();
}
```

The PreparedStatement Object (cont.)

- Parameters are represented by "?" (parameter marker)
- Supply values for every parameter before execution: use setXXX() methods
- Each parameter marker is referred to by its ordinal position: the first marker represents position 1, the next position 2, and so forth

```
stmt.setInt(1, 35);  // This would set age
stmt.setInt(2, 102); // This would set ID
```

- Statement object's methods for interacting with the database (execute, executeQuery, executeUpdate) also work with the PreparedStatement object

The CallableStatement Object

```
CallableStatement cstmt = null;
try {
   String SQL = "{call procedureName(?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

The CallableStatement Object (cont.)

Parameter	*Description*
IN	A parameter whose value is unknown when the SQL statement is created. You bind values to IN parameters with the setXXX() methods.
OUT	A parameter whose value is supplied by the SQL statement it returns. You retrieve values from the OUT parameters with the getXXX() methods.
INOUT	A parameter that provides both input and output values. You bind variables with the setXXX() methods and retrieve values with the getXXX() methods.

```
int studentID= 102;
stmt.setInt(1, studentID);
stmt.registerOutParameter(2, java.sql.Types.VARCHAR);
```

Lab

1. Using Statement
2. Using PreparedStatement
3. Using CallableStatement with stored procedures

Self-Learning Session

- Topic: retrieving and modifying values from result sets
- Key issues
  - ResultSet types
  - ResultSet concurrency
  - Cursors
  - Updating rows in ResultSet objects
  - Using Statement objects for batch updates
  - Inserting rows in ResultSet objects