Module 2- RDBMS & Database

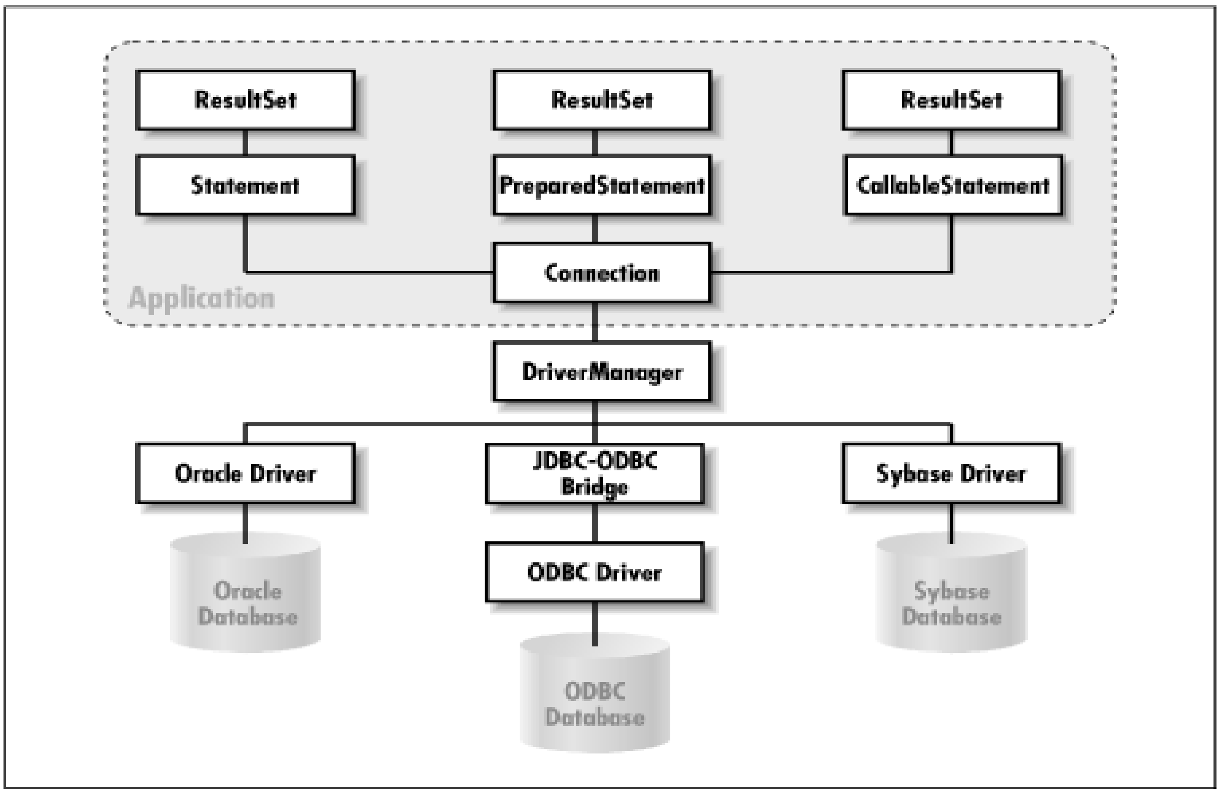
❖ What is JDBC (Java Database Connectivity)?

- JDBC Stands for the Java Database Connectivity.
- The Role of the JDBC is to most important with respect to java database application.
- JDBC API provides connectivity and Data access across the range of relational database.

❖ Importance of JDBC in Java Programming

- It allows Java programs to connect to and interact with databases.
- It provides a standard API for accessing different relational databases.
- JDBC is platform-independent and works across various operating systems.
- It supports popular databases like MySQL, Oracle, PostgreSQL, SQL Server, etc.

❖ JDBC Architecture: Driver Manager, Driver, Connection, Statement, and ResultSet.



- JDBC Architecture describes the interaction of JDBC API with java application and java applet.
- JDBC API consists of several call level interfaces for interaction with JDBC Driver Manager and JDBC driver for defining database.

- **Driver Manager**
  - DriverManager class belongs to java.sql package. It consists of static method to manage JDBC Drivers.
  - Each and every driver must register with DriverManager class. There are many JDBC Drivers used for different JDBC servers.

- **Driver**
  - JDBC Drivers specification classifies JDBC drivers into four groups.
  - Groups are referred to as JDBC Drivers types and address a specific need for Communicating with various DBMS
  - Types of driver
    1) Type - 1 :- JDBC to ODBC Driver
    2) Type - 2 :- NATIVE API Driver
    3) Type - 3 :- JDBC Drivers/NET Protocol Driver
    4) Type - 4 :- JDBC Driver/Native Protocol Driver

- **Connection**
  - Before executing any SQL statement it is mandatory to establish a connection with a database.
  - To do this the **getConnection()** method of the DriverManager class is to be invoked which is used to find a specific driver.

- **Statement**
  - In order to interact with the database, the SQL statement must be executed.
  - This requires that a statement object needs to be created to manage SQL statements.
  - To do this createStatement() method of the Connection class is to be invoked.

    Statement st= cn.createStatement();

- **ResultSet**
  - A ResultSet is the collection of results retrieved from the query.
  - There are various method available in the ResultSet class to iterate through these results.
  - A while loop can be used to fetch rows from a ResultSet.
    rs.next();

❖ Overview of JDBC Driver Types:

1) Type 1 Driver – JDBC – ODBC Bridge

- The JDBC type – 1 driver , also known as the **JDBC-ODBC bridge** is a database driver that employs the ODBC driver to connect to the database.
- The driver converts JDBC method calls into ODBC method calls.
- The bridge is usually used when there is no pure java driver available for a particular database.
- Almost any database , for which ODBC driver is installed , can be accessed.
- **Comparision:**
  - Type 1 is not platform independent
  - Performance is slow
  - Common use only for testing

2) Type – 2 Driver – Native API Driver

- Also known as the Native-API driver , is a database driver implementation that uses the client-side libraries of the database.
- The driver convert JDBC method calls into native calls of the database API.
- The type – 2 driver is not written entirely in java as it interfaces with non-java code that makes the final database calls.
- **Comparision:**
  - Type 1 is not platform independent
  - Better performance than type 1 since no JDBC to ODBC transmission is needed.
  - Common use for legacy apps.

3) Type 3 Driver – Network Protocol Driver

- Type 3 driver makes use of middle-tier between the calling program and the database.
- The middle-tier(App. Server) convert JDBC calls directly or indirectly into vendor specific database protocol.
- Follows a three tier communication approach

  Client->JDBC Driver->Middleware->Any DB
- **Comparision:**
  - Type 3 is platform independent.
  - Type 3 driver performance is high.
  - Use for enterprise networks.

4) Type 4 Driver – Native-Protocol Driver

- Also known as Direct to Database Pure java driver.
- Type 4 driver convert JDBC calls directly into vendor specific database protocol.
- Type 4 driver is written in completely java and is hence platform independent.
- It is installed inside the Java Virtual Machine of the client.
- **Comparision:**
  - Type 4 is platform independent.
  - Type 4 driver performance is high.
  - use for web & enterprise apps.

❖ Step-by-Step Process to Establish a JDBC Connection

1) Import the JDBC packages

- Import necessary JDBC classes from java.sql package.
- Enables the use of JDBC interfaces like Connection, Statement, and ResultSet.

2) Register the JDBC driver

- This second step using JDBC is to load the JDBC-ODBC bridge driver.
- This is done by the forName static method of the Class object.

  Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);

3) Open a connection to the database

- To do this the getConnection() method of the DriverManager class is to be invoked which is used to find a specific driver.
- The DriverManager class searches for registered drivers which can process the database.
- Connection cn = DriverManager.getConnection (“jdbc:odbc:dbnm”,””,””);

4) Create a statement

- To do this createStatement() method of the Connection class is to be invoked.
- Statement st= cn.createStatement();
- The statement class provides methods for executing SQL statements and retrieving the results from the statement execution.

5) Execute SQL queries

- The SQL statement can be executed by invoking executeQuery() method
- String s = “Select * from emp”;
- ResultSet rs = st.executeQuery(s);
- The above statement send the query to the database and return the result of the query as a ResultSet.

6) Process the result set

- A ResultSet is the collection of results retrieved from the query.
- There are various method available in the ResultSet class to iterate through these results.
- One of the method is next() which places the pointer to the next record.

7) Close the connection

- Always close ResultSet, Statement, and Connection to avoid memory leaks.
- Frees up database resources and ensures efficient performance.

❖ Overview of JDBC Statements:

➢ Statement:

- Used to execute simple SQL queries (like SELECT, INSERT, UPDATE, DELETE).
- Does not support parameters.
- Suitable for static SQL queries.
- **Difference:**
  - ✓ Query type is Static SQL
  - ✓ Compiled every time it's run
  - ✓ Parameters are Not supported

➢ PreparedStatement:

- A PreparedStatement can be used to execute a dynamic sql statement with IN parameter.
- The preparedStatement object is created to use in preparedStatement() in connection class.
- Parameters are set using setXXX() methods (e.g., setInt(), setString()).
- **Difference:**
  - ✓ Query type is Dynamic SQL with input parameters
  - ✓ Compiled once, reused multiple times
  - ✓ Parameters are Supported using ? placeholders

➢ CallableStatement:

- A callable statement is used to execute stored procedure object in the RDBMS.
- A procedure without parameter can be executed only in the callable statement.
- A callable statement can also contain IN parameter.
- An OUT parameter has to be register prior to execute the store procedure.
- Syntax : registerOutParameter(int index,Type T)
- **Difference:**
  - ✓ Query type is Used for stored procedure calls
  - ✓ Compiled once, reused for stored procedure calls
  - ✓ Parameters are Supported (IN, OUT, INOUT parameters)

❖ JDBC CRUD Operations

➢ INSERT

- The INSERT command is used to add new records (rows) into a database table.
- It requires specifying the table name and the values for each column.
- Example: INSERT INTO Students (ID, Name, Age) VALUES (1, 'John', 20);

➢ UPDATE

- The update command use is used to modify existing records in a table.
- It requires specifying the table name, the column to update and a condition to identify which row to change.
- Example: UPDATES Students SET Age = 21 WHERE ID =1;

➢ SELECT

- the select command is used to retrieve data from a database table.
- It can fetch specific columns or all columns(*).
- It may include conditions (WHERE) sorting (ORDER BY), etc.
- Example: SELECT ID,Name,age FROM Student WHERE age >18;

➢ DELETE

- The delete command is used to remove existing records from a table.
- It requires specifying the table name and a condition to avoid deleting all rows.
- Example:DELETE FROM student WHERE id = 1;

❖ What is ResultSet in JDBC?

- Resultset object stores the data of the table by executing the query.
- It is used to access the data of the table from database.
- There are six methods of ResultSet object that are used to position the virtual cursor. They are **first()** , **last()** , **previous()**, **next(),absolute()** , **relative()** , and **getRow().**
- The executeQuery() when called on statement, PreparedStatement, and Callable Statement it return object of type resultset.

❖ Navigating through ResultSet (first, last, next, previous)

- These methods are useful when we need to scroll through data in both directions or access specific rows directly.
- This kind of navigation is helpful in applications like GUIs, reports, or when implementing custom pagination.
- **Next():**Moves the cursor to the next row. Returns `false` if there are no more rows.
- **Previous():**Moves the cursor to the previous row.
- **First():**Moves the cursor to the first row of the ResultSet.
- **Last():**Moves the cursor to the last row of the ResultSet.

❖ Working with ResultSet to retrieve data from SQL queries

- ResultSet object is used to store and access the data returned from executing a SQL SELECT query.
- When a query is executed using the executeQuery() method of the Statement or PreparedStatement interface, it returns a ResultSet containing the rows of the result.

1) **Execute a Query:**

```
ResultSet rs = statement.executeQuery
("SELECT * FROM employees");
```

2) Iterate through the ResultSet:

- Use the next() method to move the cursor to the next row

```
while (rs.next()) {
    // Retrieve data from current row
}
```

3) Retrieve Column Data:

- Use getter methods like getInt(), getString(), getDouble(), etc., to access column values. You can use either column names or column indexes.

```
int id = rs.getInt("id");
String name = rs.getString("name");
```

4) Close the ResultSet:

- It is important to close the ResultSet, Statement, and Connection to free up resources:

```
rs.close();
statement.close();
connection.close();
```

❖ What is DatabaseMetaData?

- ➢ In JDBC, DatabaseMetaData is an interface that provides comprehensive information about the database and its capabilities.
- ➢ It allows developers to retrieve metadata (data about the data), such as database version, supported features, table structures, column details, and more.

❖ Importance of Database Metadata in JDBC

- ➢ **Database Information**: Retrieve product name, version, driver info, and user name.
- ➢ **Schema and Table Info**: Get information about tables, columns, primary keys, indexes, etc.

- **Portability**: Helps write database-independent code by checking features supported by the database.
- **Dynamic Queries**: Useful for generating dynamic SQL queries based on actual schema.
- **Feature Detection**: Check support for joins, transactions, stored procedures, etc.
- It is especially useful in applications like database admin tools, report generators, and dynamic form builders.

❖ Methods provided by DatabaseMetaData

- **getDatabaseProductName():** Returns the name of the database (e.g., MySQL, Oracle).
- **getDatabaseProductVersion():** Returns the version of the database.
- **getDriverName():** Returns the name of the JDBC driver.
- **getURL():** Returns the database URL used to connect.
- **getUserName():** Returns the username used for the connection.
- **getMaxConnections():** Returns the maximum number of concurrent connections allowed.

❖ What is ResultSetMetaData?

- In JDBC, ResultSetMetaData is an interface that provides information about the structure of a ResultSet.
- It allows you to analyze the number of columns in a result set and get details like column names, types, and sizes.

❖ Importance of ResultSet Metadata

- ResultSetMetaData is important because it allows programs to dynamically understand the structure of a query result, without knowing the schema in advance
- **Dynamic SQL applications:** when queries change at runtime.
- **Generic data processing**: like building report generators or data viewers.
- **Validation tools**: to inspect or verify column types and names

❖ Methods in ResultSetMetaData

- **getColumnCount():** Returns the total number of columns in the ResultSet.
- **getColumnName(int column):** Returns the name of the specified column (1-based index).
- **getColumnType(int column):** Returns the SQL type of the specified column as an integer constant from java.sql.Types.
- **getColumnLabel(int column):** Returns the alias (label) used in the query for the column.

❖ What is a CallableStatement?

- In JDBC**,** CallableStatement is an interface used to execute stored procedures in a database.
- Stored procedures are precompiled SQL routines stored in the database that can perform complex operations, accept parameters, and return results.
- CallableStatement extends the PreparedStatement interface and allows calling stored procedures with IN, OUT, or INOUT parameters.

❖ How to call stored procedures using CallableStatement in JDBC

1) Write the SQL call syntax:

- Use ? as placeholders for parameters.

```
{call procedure_name(?, ?, ?)}
```

2) Prepare the CallableStatement:

```
CallableStatement cstmt = conn.prepareCall("{call getEmployeeById(?)}");
```

3) **Set parameters** using setXXX() for IN parameters and registerOutParameter() for OUT parameters.

4) **Execute the procedure** using execute() or executeQuery() depending on the procedure type.

5) **Retrieve output** using getXXX() methods for OUT parameters.

❖ Working with IN and OUT parameters in stored procedures

➢ Stored procedures can accept three types of parameters:

- **IN**: Input values passed from Java to the procedure.
- **OUT**: Output values returned from the procedure to Java.
- **INOUT**: Both input and output.