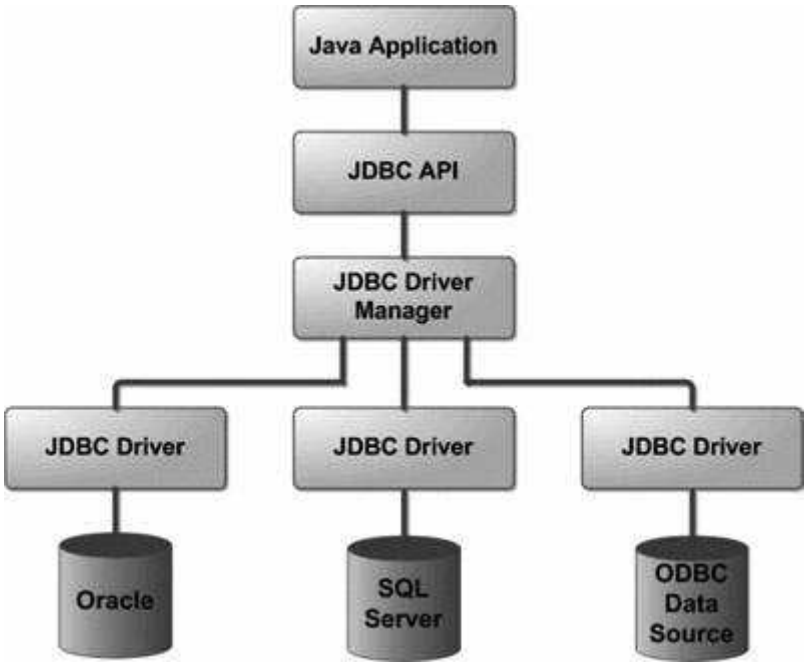
Module-4

Talking to Database, Essential JDBC program, using prepared Statement Object, Interactive SQL tool. JDBC in Action Result sets, Batch updates, Mapping, Basic JDBC data types, Advanced JDBC data types Introduction to EJB, types of EJB

4.1 Talking to Database

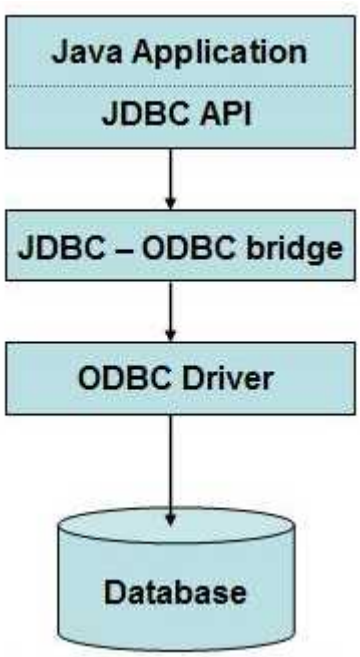
- **Java Database Connectivity(JDBC)** is an **Application Programming Interface(API)** used to connect Java application with Database. JDBC is used to interact with various type of Database such as Oracle, MS Access, My SQL and SQL Server. JDBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows java program to execute SQL statement and retrieve result from database.



Types of JDBC Driver

- JDBC Driver is required to process SQL requests and generate result. The following are the different types of driver available in JDBC.
  - **Type-1 Driver** or **JDBC-ODBC bridge**
  - **Type-2 Driver** or **Native API Partly Java Driver**
  - **Type-3 Driver** or **Network Protocol Driver**
  - **Type-4 Driver** or **Thin Driver**

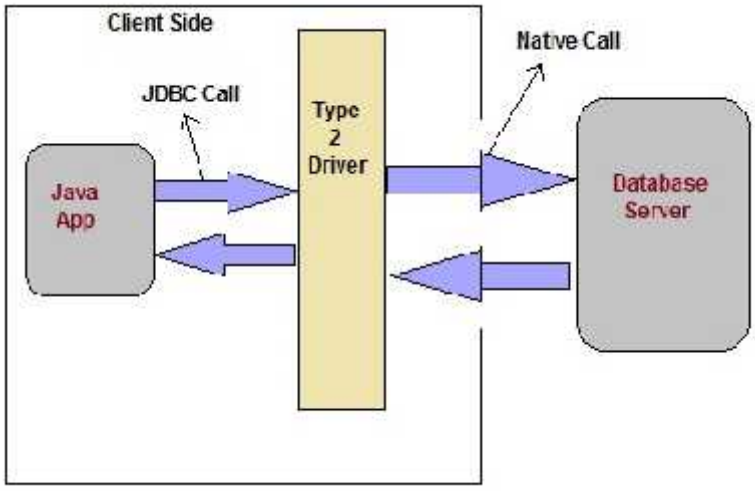
- **Type-1 Driver or JDBC-ODBC bridge**
  - **Type-1 Driver** act as a bridge between JDBC and other database connectivity mechanism(ODBC). This driver converts JDBC calls into ODBC calls and redirects the request to the ODBC driver.



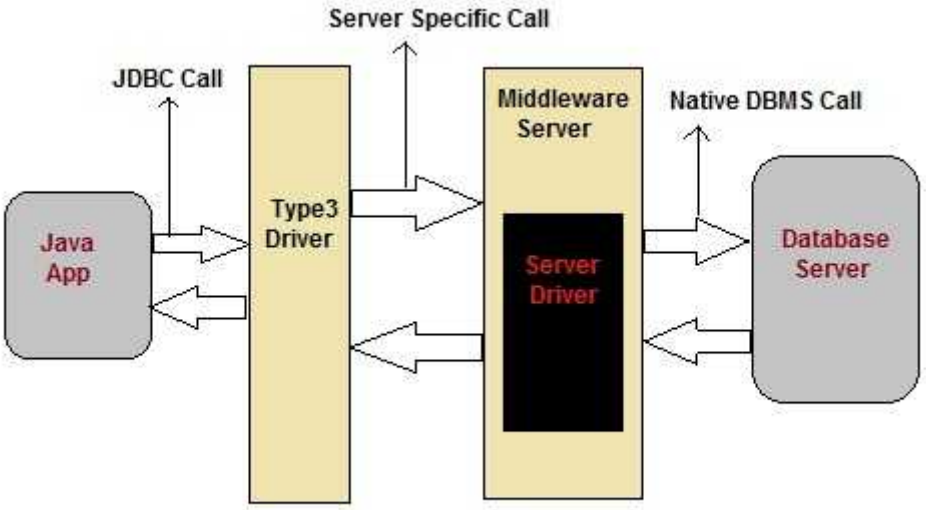
- **Advantage**
  - Easy to use
  - easy connectivity to all database supported by the ODBC Driver
- **Disadvantage**
  - Slow execution time
  - Dependent on ODBC Driver.
  - Uses Java Native Interface(JNI) to make ODBC call.

- **Type-2 Driver** or **Native API Partly Java Driver**
  - This type of driver makes use of Java Native Interface(JNI) call on database specific native client API. These native client API are usually written in C and C++.

  - **Advantage**
    - faster as compared to **Type-1 Driver**
    - Contains additional features
  - **Disadvantage**
    - Requires native library
    - Increased cost of Application



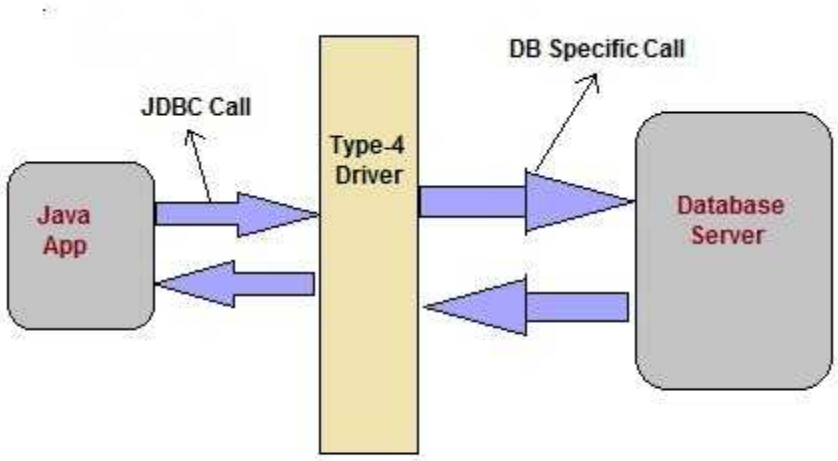
- **Type-3 Driver** or **Network Protocol Driver**
  - This driver translate the JDBC calls into a database server independent and Middleware server-specific calls. Middleware server further translate JDBC calls into database specific calls.



- **Advantage**
  - Does not require any native library to be installed.
  - Database Independency.
  - Provide facility to switch over from one database to another database

- **Disadvantage**
  - Slow due to increase number of network call.

- **Type-4 Driver** or **Thin Driver**
  - This is Driver called Pure Java Driver because. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.



- **Advantages**
  - Does not require any native library.
  - Does not require any Middleware server.
  - Better Performance than other driver.

- **Disadvantage**
  - Slow due to increase number of network call.

JDBC Package

  - This package is also known as JDBC extension API. It provides classes and interface to access server-side data.
  - Important classes and interface of javax.sql package

4.2 Essential JDBC program (steps to establish connection)

1. import package
2. load and register
3. establish connection
4. create connection
5. execute connection
6. process results
7. close connection and statement

- **importing package**

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

- **load and register**

  - `Class.forName( )` is used to load the driver class explicitly.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

- **establish connection**
  - `getConnection( )` method of **DriverManager** class is used to create a connection

```
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","
username","password");
```

- **create connection**
  - `createStatement( )` method is invoked on current Connection object to create a SQL Statement.

```
Statement s=con.createStatement();
```

- **execute statement**
  - executeQuery( ) method of Statement interface is used to execute SQL statements.

```
ResultSet rs=s.executeQuery("select * from user");
```

- **process the results**
  - A ResultSet object maintains a cursor that points to the current row in the result set.
  - rs.next( ) makes rs to point to next tuple.

```
ResultSet rs=s.executeQuery("select * from user");
while(rs.next())
  {
   System.out.println(rs.getString(1)+" "+rs.getString(2));
  }
```

- **close connection and statement**
  - After executing SQL statement you need to close the connection and release the session. The close( ) method of **Connection** interface is used to close the connection.

```
con.close();
stmt.close();
```

4.3 Using Prepared Statement Object

- The PreparedStatement interface is a sub interface of Statement.
- It is used to execute parameterized query.

```
String sql="insert into emp values(?,?,?)";
```

When to use PreparedStatement object

- **Improves performance**: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.
- This means that, the preparedstatement is executed, the DBMS can just run the PreparedStatement' s SQL statement without having to compile it

Creating a PreparedStatement

```
String sql = "select * from people where id=?";
PreparedStatement preparedStatement =
        connection.prepareStatement(sql);
```

Inserting Parameters into a PreparedStatement

- Everywhere you need to insert a parameter into your SQL, you write a question mark (?). For instance.

```
String sql = "select * from people where id=?";
```

- Once a PreparedStatement is created (prepared) for the above SQL statement, you can insert parameters at the location of the question mark. This is done using the many setXXX() methods. Here is an example:

```
preparedStatement.setLong(1, 123);
```

- The first number (1) is the index of the parameter to insert the value for. The second number (123) is the value to insert into the SQL statement.

Executing the PreparedStatement

- Executing the PreparedStatement looks like executing a regular Statement. To execute a query, call the executeQuery() or executeUpdate method. Here is an executeQuery() example:

```
ResultSet result = preparedStatement.executeQuery();
```

4.4 Interactive SQL tool

- The interactive SQL tool is a means of entering and executing SQL statements.
- It will be simple front-end to the JDBC API. It will provide a means of entering and exwcuting SQL statements and at the same time, display areas for viewing results.
- The requirement for the Interactive tool can as follows.
  - To enable user to enter and execute an SQL command.
  - To display the ResultSet from an SQL query.

4.5 JDBC in Action Result Sets

- The SQL statements that read data from a database query, return the data in a result set.
- The SELECT statement is the standard way to select rows from a database and view them in a result set.
- The *java.sql.ResultSet* interface represents the result set of a database query.
- A ResultSet object maintains a cursor that points to the current row in the result set.
- The term "result set" refers to the row and column data contained in a ResultSet object.

- The cursor is movable based on the properties of the ResultSet. These properties are designated when the corresponding Statement that generates the ResultSet is created.
- **Type of ResultSet**

Type	Description
ResultSet.TYPE_FORWARD_ONLY	The cursor can only move forward in the result set.
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
ResultSet.TYPE_SCROLL_SENSITIVE.	The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

4.6 Batch updates

- Batch Processing allows you to group related SQL statements into a batch and submit them with one call to the database.
- Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast.
- The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing.

Methods of Statement interface

- The required methods for batch processing are given below:

Method	Description
void addBatch(String query)	It adds query into batch.
int[] executeBatch()	It executes the batch of queries.

- **Example:**

```
Statement stmt=con.createStatement();
stmt.addBatch("insert into student  values(100,”anu”);
stmt.addBatch("insert into student values(101,”anil”);

stmt.executeBatch( );  //executing the batch
```

4.7 Mapping

- The JDBC driver converts the Java data type to the appropriate JDBC type, before sending it to the database.
- It uses a default mapping for most data types. For example, a Java int is converted to an SQL INTEGER. Default mappings were created to provide consistency between drivers.
- Three sets of method in order to transfer data between a database and a application.
    - **ResultSet** class for retrieving SQL SELECT results as java types.
    - **PreparedStatement** class for sending java types as SQL statement parameters.
    - **CallableStatement** class for retrieving SQL OUT parameters as Java types.

4.8 Basic JDBC data types

- The following is the list of JDBC types

SQL Types	Java programming language types (JDBC)
CHAR	String or char[ ]
VARCHAR	
LONGVARCHAR	
BINARY	Arrays of byte byte[ ]
VARBINARY	
LONGVARBINARY	
BIT	Boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
DOUBLE	double
FLOAT	double or float
DECIMAL	java.Math.BigDecimal
NUMERIC	
DATE	java.util.Date
TIME	
TIMESTAMP	

4.10 Advanced JDBC data types

SQL Types	JDBC
BLOB (Binary Large Object)	Java.sql.blob
CLOB (Character Large Object)	Java.sql.clob
ARRAY	Java.sql.Array
DISTINCT	Java.util.Map
STRUCT	Java.sql.Struct
REF	Java.sql.ref

National Character Set Support

- Some new JDBC data type have been added such as
  - **NCHAR, NVARCHAR, LONGVARCHAR, NCLOB, SQLXM and ROWID**

Note:

JDBC- Statements, PreparedStatement, Callable Statement

- Once a connection is obtained we can interact with the database. The JDBC *Statement*, *CallableStatement*, and *PreparedStatement* interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database.
- They also define methods that help bridge data type differences between Java and SQL data types used in a database.
- The following table provides a summary of each interface's purpose to decide on the interface to use.

Interfaces	**Recommended Use**
Statement	Use for the general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.

Example: Statement stmt = null;
try {
stmt = conn.createStatement( );
. . .}	

PreparedStatement	Use when SQL statements executed many times. The PreparedStatement interface accepts input parameters at runtime.

Example: PreparedStatement pstmt = null;
try {
String SQL = "Update Employees SET age = ? WHERE id = ?";
pstmt = conn.prepareStatement(SQL);
. . . }	

CallableStatement	Use while accessing the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

Example: CREATE OR REPLACE PROCEDURE getEmpName
(EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
SELECT first INTO EMP_FIRST
FROM Employees
WHERE ID = EMP_ID;
END;	

4.11 Introduction to EJB

- (*Enterprise Java Bean*) is used to develop scalable, robust and secured enterprise applications in java.
- To run EJB application, you need an *application server* (EJB Container) such as Jboss, Glassfish, Weblogic, Websphere etc. It performs
  - life cycle management,
  - security,
  - transaction management, and
  - object pooling
- EJB application is deployed on the server, so it is called server side component also

When to use Enterprise Java Bean?

1. **Application needs Remote Access**.
   - In other words, it is distributed.
2. **Application needs to be scalable**.
   - EJB applications supports load balancing, clustering and fail-over.
3. **Application needs encapsulated business logic**.
   - EJB application is separated from presentation and persistent layer.

4.12 Types of Enterprise Bean

- There are 3 types of enterprise bean in java.
  - **Session Bean**
    - Session contains business logic that can invoked by local, remote or webservice client.
  - **Message Driven Bean**
    - It contains the business logic but it is invoked by passing message
  - **Entity Bean**
    - It encapsulates the state that can be persisted in the database. It is deprecated. Now, it is replaced with JPA(Java Persistence API).

4.13 Disadvantage of EJB

- Requires application server
- Requires only java client. For other language client, you need to go for webservice.
- Complex to understand and develop ejb applications.