Unit 13:
Database Programming using JDBC

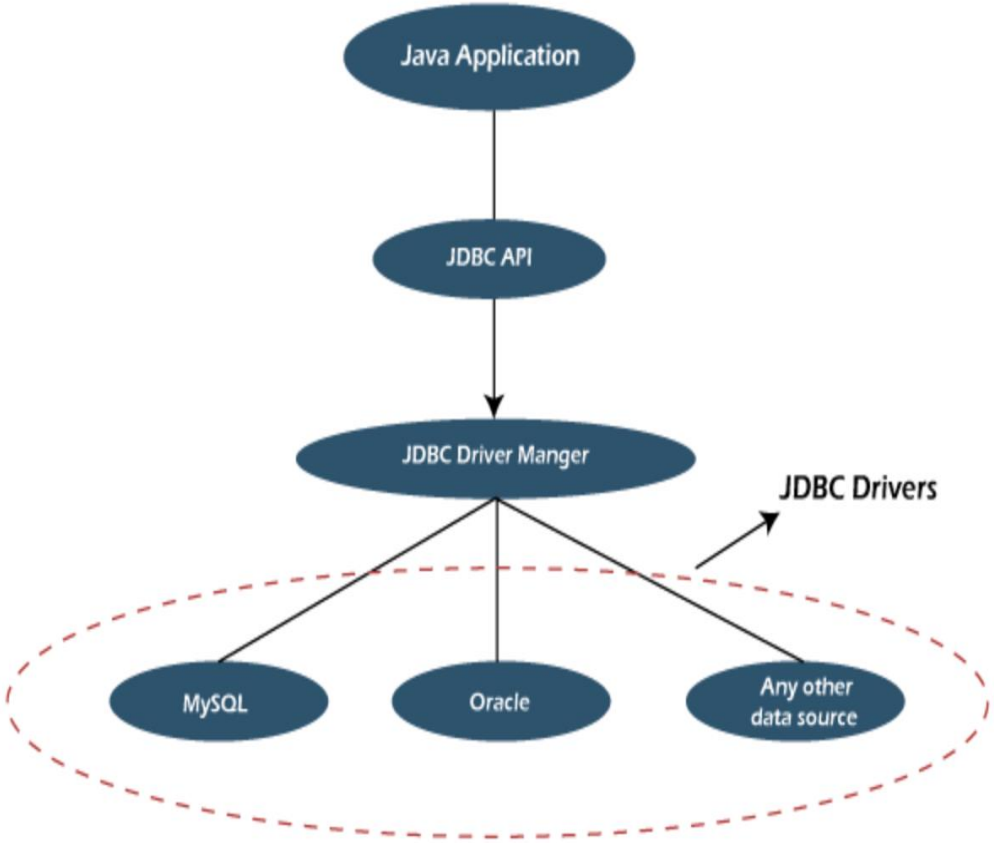
Er. Shankar pd. Dahal
pdsdahal@gmail.com

The Design of JDBC :

- **Java Database Connectivity** (JDBC) is an Application Programming Interface (API), from Sun microsystem that is used by the Java application to communicate with the relational databases from different vendors.

- JDBC and database drivers work in tandem to access spreadsheets and databases. **Design of JDBC** defines the components of JDBC, which is used for connecting to the database.

Components of JDBC :



1. JDBC API:

It provides various methods and interfaces for easy communication with the database.It provides two packages as follows which contains the java SE and java EE platforms to exhibit WORA(write once run everywhere) capabilities.

1. java.sql.*;
2. javax.sql.*;

2. JDBC Driver manager:

It loads database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.

3. JDBC Test suite:

It is used to test the operation(such as insertion, deletion, updation) being performed by JDBC Drivers.

4. JDBC-ODBC Bridge Drivers:

It connects database drivers to the database.This bridge translates JDBC method call to the ODBC function call.It makes the use of

sun.jdbc.odbc package that includes native library to access ODBC characteristics.

SQL

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

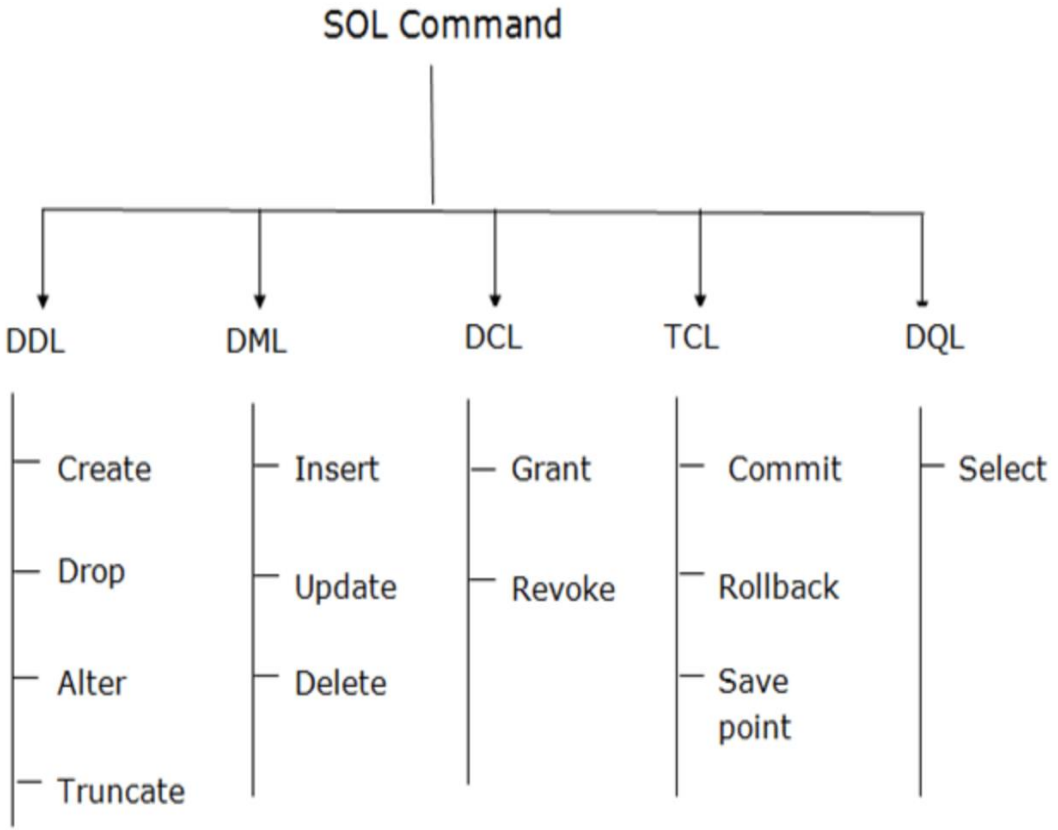
SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.

- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.

1. Data Definition Language (DDL)
2. Data Manipulation Language(DML)
3. Data Control Language(DCL)
4. Transaction Control Language(TCL)
5. Data Query Language(DQL)



1. Data Definition Language (DDL)

CREATE : It is used to create a new database and table.

Syntax :

a. CREATE DATABASE *databasename*;

b. CREATE TABLE *table_name* (
column1 datatype,
column2 datatype,
column3 datatype,
....
);

DROP: It is used to delete objects such as the table, view, index, database.
A drop statement cannot be rolled back.

Syntax:

a. DROP TABLE table_name;
b. DROP DATABASE *databasename*;

ALTER :
Alter command is used for altering the table structure, such as :

- to add a column to existing table
- to change datatype of any column or to modify its size.
- to drop a column from the table.

Syntax:

1. Add new column :
ALTER TABLE table_name
ADD column_name datatype;

2. Modify an existing Column :

ALTER TABLE table_name
MODIFY COLUMN column_name datatype;

3. Drop column name :
ALTER TABLE table_name
DROP COLUMN column_name;

TRUNCATE :
- command removes all the records from a table. But this command will not destroy the table's structure.

Syntax:
TRUNCATE TABLE table_name

2. Data Manipulation Language(DML)

INSERT:
Insert command is used to insert data into a table.

Syntax:
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN) VALUES (value1, value2, value3,...valueN);

UPDATE :
It is used to update any record of data in a table.

Syntax:
UPDATE table_name SET column_name = new_value WHERE some_condition;

Delete:
It is used to delete data from a table.

Syntax :
DELETE FROM table_name WHERE condition;

Data Query Language(DQL)

SELECT :

It is used to retrieve data from a table. It is the most used SQL query. We can retrieve complete table data, or partial by specifying conditions using the WHERE clause.

Syntax:
SELECT column1, column2, columnN FROM table_name;

Steps to develop/ create JDBC Application (JDBC Configuration)

Steps For Connectivity Between Java Program and Database

1. Import the JDBC packages
2. Loading and Registering a driver
3. Create Connection
4. Create a statement
5. Execute the Statement
6. Processing Result Set
7. Close the connection

1. Import the JDBC packages:

Import all the packages that contain the JDBC classes needed for database programming.

E.g.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
Import java.sql.Statement;
```

2. Loading and Registering a driver

- Provide the code to register your installed driver with your program.
- 2 ways to register driver :

1. Class.ForName()
2. DriverManager.registerDriver() method

Class.ForName() – It loads Jdbc driver directly.

However, Class.ForName() method is valid only for JDK- Compliant JVM. It is not valid for Microsoft JVM. In this case you can use DriverManager.registerDriver() .

3. Create Connection:

Once you loaded the driver, you can establish a connection to the database with static *getConnection*() method of the JDBC Driver Manager class.

Java provide three overloaded DriverManager.getConnection() methods :

DriverManager. *getConnection*(String url)
DriverManager. *getConnection*(String url, String username, String password)
DriverManager. *getConnection*(String url, Properties info)

- The *getConnection*() method return an object of the JDBC connection class.
- e.g.

Connection con = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/world", "Admin", "Admin");

4. Create a statement :

- createStatement() method of JDBC connection object returns an object of JDBC statement class.

e.g.

Statement statement = con.createStatement();

5. Execute Statement :

- When you execute Statement objects, it generates ResultSet objects, which is a table of data representing a database ResultSet.

e.g.

ResultSet rs = statement.executeQuery(sql);

6. Processing Result Set :

- If you want to process the ResultSet to pull data out of the ResultSet and iterate though it. You can use the next() method of your ResultSet object to loop through the results. This method iterate through the result set row by row, detecting the end of the ResultSet when it is reached.
- e.g.
- **while** (rs.next()) {

  }

7. Close the connection :

- Finally, to end the database session you need to close all the database resources which immediately release the resources it's using.

e.g.

con.close();

JDBC Statements :

The statement interface is used to create SQL basic statements in Java it provides methods to execute queries with the database. There are different types of statements that are used in JDBC as follows:

1. Create Statement/ Statement
2. Prepared Statement
3. Callable Statement

1. Create a Statement:

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

Syntax:

```
Statement statement = connection.createStatement();
```

Implementation: Once the Statement object is created, there are three ways to execute it.

1. ***boolean execute(String SQL):*** If the ResultSet object is retrieved, then it returns true else false is returned.

2. int executeUpdate(String SQL): Returns number of rows that are affected by the execution of the statement, used when you need a number for INSERT, DELETE or UPDATE statements.

3. ResultSet executeQuery(String SQL): Returns a ResultSet object. Used similarly as SELECT is used in SQL.

2. Prepared Statement : - Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.

Syntax:

```
String query = "INSERT INTO people(name, age)VALUES(?, ?)";
PreparedStatement pstmt = con.prepareStatement(query);
pstmt.setString(columnNo,”Value");
ptstmt.setInt(columnNo,Value);
```


Implementation: Once the PreparedStatement object is created, there are three ways to execute it:

1. execute(): This returns a boolean value and executes a static SQL statement that is present in the prepared statement object.
2. executeQuery(): Returns a ResultSet from the current prepared statement.

3. executeUpdate(): Returns the number of rows affected by the DML statements such as INSERT, DELETE, and more that is present in the current Prepared Statement.

3. Callable Statement - Use this when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

Syntax:

```
CallableStatement cstmt = con.prepareCall("{call Procedure_name(?, ?}");
```

Implementation: Once the callable statement object is created

execute() is used to perform the execution of the statement.

Difference between Statement and PreparedStatement :

Statement	PreparedStatement
It is used when SQL query is to be executed only once.	It is used when SQL query is to be executed multiple times.
You can not pass parameters at runtime.	You can pass parameters at runtime.
Used for CREATE, ALTER, DROP statements.	Used for the queries which are to be executed multiple times.
Performance is very low.	Performance is better than Statement.
It is base interface.	It extends statement interface.
Used to execute normal SQL queries.	Used to execute dynamic SQL queries.
We can not used statement for reading binary data.	We can used Preparedstatement for reading binary data.
It is used for DDL statements.	It is used for any SQL Query.
We can not used statement for writing binary data.	We can used Preparedstatement for writing binary data.
No binary protocol is used for communication.	Binary protocol is used for communication.

Difference between CallableStatement and PreparedStatement :

CallableStatement	PreparedStatement
It is used when the stored procedures are to be executed.	It is used when SQL query is to be executed multiple times.
You can pass 3 types of parameter IN, OUT, INOUT.	You can pass any type of parameters at runtime.
Used to execute functions.	Used for the queries which are to be executed multiple times.
Performance is very high.	Performance is better than Statement.
Used to call the stored procedures.	Used to execute dynamic SQL queries.
It extends PreparedStatement interface.	It extends Statement Interface.
No protocol is used for communication.	Protocol is used for communication.

ResultSet

- The ResultSet interface in the java.sql package in Java represents the result of a database query as a tabular data set.
- It allows us to retrieve data from a database, store it in memory, and insert, update and delete data in the database.
- The ResultSet object is not thread-safe, which means that it cannot be shared between multiple threads. Each thread must create its own ResultSet instance.

Types of ResultSet :

- A ResultSet object maintains a cursor that points to its current row of data.
- Based on the cursor movement, the following are the two types 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.

Methods:

Method	Description
next()	Moves the cursor to the next row of data
previous()	Moves the cursor to the previous row of data
first()	Moves the cursor to the first row of data
last()	Moves the cursor to the last row of data
getRow()	Returns the current row number
beforeFirst()	Moves the cursor before the first row of data
afterLast()	Moves the cursor after the last row of data
isBeforeFirst()	Returns true if the cursor is before the first row
isAfterLast()	Returns true if the cursor is after the last row
isFirst()	Returns true if the cursor is on the first row
isLast()	Returns true if the cursor is on the last row
getInt()	Retrieves the value of a column as int

Methods	Description
getLong()	Retrieves the value of a column as long
getFloat()	Retrieves the value of a column as float
getDouble()	Retrieves the value of a column as double
getString()	Retrieves the value of a column as string
getBoolean()	Retrieves the value of a column as boolean
getDate()	Retrieves the value of a column as a java.sql.Date object
getTime()	Retrieves the value of a column as a java.sql.Time object
getTimestamp()	Retrieves the value of a column as a java.sql.Timestamp object
getObject()	Retrieves the value of a column as an object
findColumn()	Returns the index of a given column name