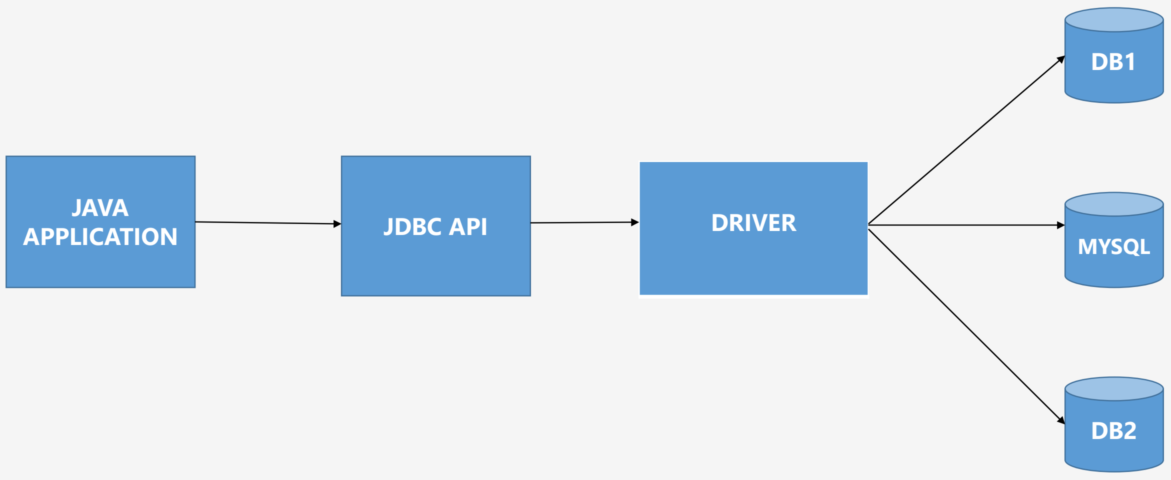
JDBC

JAVA DATABASE CONNECTIVITY

INTRODUCTION TO JDBC

1. JDBC stands for Java Database Connectivity.
2. JDBC is used to communicate with the database.
3. JDBC API is the one API which is used to connect with the database.
4. JDBC API uses the Drivers to communicate.
5. JDBC is the standard java API for independent database connection and it is Driver dependent.
6. It is composed of number of classes and interfaces that represent a connection to the database.
7. JDBC acts like a translator or it is API which translates java programming language to SQL and vice-versa.

JDBC ARCHITECTURE



STEPS TO CONNECT

1.Load or register the driver

2.Establish Connection

3.Create Statement

4.Execute Statement

5.Close

JDBC
CONCEPTUAL COMPONENTS

1. **Driver Manager:** Loads database drivers and manages connections between the application and the driver
2. **Driver:** Translates API calls into operations for specific database
3. **Connection:** Session between application and data source
4. **Statement:** SQL statement to perform query or update
5. **Result Set:** Logical set of columns and rows of data returned by executing a statement

JDBC
DRIVERMANAGER

1.Driver Manager manages the set of Java Database Connectivity (JDBC)drivers that are available for an application to use

2.Checks with each driver to determine if it can handle the specified URL

3.Driver Manager class can not be instantiated

- All methods of Driver Manager are static

JDBC
DRIVERS

1. JDBC Driver is a software component that enables java application to interact with the database
2. To connect with individual databases JDBC API requires drivers for each databases
3. There are 4 types of JDBC drivers:
   - JDBC-ODBC bridge driver
   - Native-API driver (partially java driver)
   - Network Protocol driver (fully java driver)
   - Thin driver (fully java driver)

How to Load or Register the Driver?

There are 2 ways to Load or Register the Driver:

1.**Using the Class. forName() method** –The forName() method of the class named Class accepts a class name as a String parameter and loads it into the memory, as Soon as it is loaded into the memory it gets registered automatically

2. **Using registerDriver() Method**: The registerDriver() method of the DriverManager class accepts an object of the diver class as a parameter and, registers it with the JDBC driver manager.

JDBC
CONNECTION

1. Required to communicate with a database via JDBC
2. Three separate methods:

   public static Connection getConnection(String url)

   public static Connection getConnection(String url, Properties info)

   public static Connection getConnection(String url, String user, String password)

```
{// Load the driver class
    System.out.println("Loading Class driver");
    Class.forName("com.mysql.cj.jdbc.Driver");
    // Define the data source for the driver
    String sourceURL = "jdbc:mysql://localhost:3306/employedb"
    // Create a connection through the DriverManager class
    System.out.println("Getting Connection");
    Connection connection = DriverManager.getConnection(sourceURL);
}
```

JDBC
STATEMENT

1. Statements in JDBC abstract the SQL statements
2. Primary interface to the tables in the database
3. Used to create, retrieve, update & delete data (CRUD) from a table

   Syntax: Statement statement = connection.createStatement();
4. Three types of statements each reflecting a specific SQL statements
   - Statement
   - PreparedStatement
   - CallableStatement

JDBC
STATEMENT AND PREPARED STATEMENT

STATEMENT	PREPARED STATEMENT
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.
This interface cannot be used for retrieving data from database.	This interface can be used for retrieving data from database.

Callable Statement

1. The **CallableStatement** interface provides methods to execute the stored procedures.
2. Create an object of the **CallableStatement** (interface) using the **prepareCall()** method of the **Connection** interface. This method accepts a string variable representing a query to call the stored procedure and returns a **CallableStatement** object

```
CallableStatement stmt=connection.prepareCall(“call myProcedure(?,?,?)”);
```

3. You can set values to the parameters of the procedure call using the setter methods. These accepts 2 arguments one is integer value representing index and another is int, float, String representing the value you need to pass.

JDBC
METHODS TO EXECUTE STATEMENT

There are 3 Methods to execute statement

execute():

- This method is used for all the commands like DDL,DML,DQL.
- Return type of execute method is **Boolean.**
- Return true when DQL commands are used.
- Return false when other than DQL commands are used.

executeUpdate():

- This method is used for DML commands
- Return type is **int.**
- DML commands like update,insert,delete.

executeQuery():

- This method is used for DQL commands
- Return type is Resultset.
- This method is used for Select query.

JDBC
RESULTSET

1. ResultSet interface represents the result set of a database query.
2. The SQL statements that read data from a database query, return the data in a result set.
3. A ResultSet object maintains a cursor that points to the current row in the result set.
4. The term "result set" refers to the row and column data contained in a ResultSet object.

JDBC
PROPERTY FILE

1. A property file is one type of the file which organizing the data in the form of (key, value) pair.
2. Properties class object organizes the data in the form of (key, value) pair and it displays in the same order in whichever order it is added.
3. Property file always resides in secondary memory.
4. Each parameter is stored as a pair of strings
5. Main advantage of Properties file is that they are outside the source code and can change any time.
6. **public void load(FileInputStream);** is used for loading the content of property file into properties class object by opening the properties file in read mode with the help of FileInputStream class.

```
public class User{
public connection getConnection(){
Driver driver=new Driver();
DriverManager.registerDriver(driver);
FileInputStream fileinputstream=new FileInputStream("dbconfig.properties");
Properties properties=new Properties();
Connection connection=DriverManager.getConnection(properties.getproperty("url"),properties.getproperties("username"),properties.getproperties("password");
return connection;
}
```

JDBC
STATEMENT EXAMPLE

```
//1.load or register driver
String className="com.mysql.cj.jdbc.Driver";
Class.forName(className);
//2.establish connection
String url="jdbc:mysql://localhost:3306/studentdb";
String username="root";
String pwd="root";
Connection connection=DriverManager.getConnection(url,username,pwd);
//3.create statement
Statement statement=connection.createStatement();
//4.execute statement
String query="INSERT INTO STUDENT VALUES(4,'SHRAVAN',50,'HYDERABAD','VIJAY')";
statement.execute(query);
//5.CLOSE
connection.close();
```

JDBC

PREPARED STATEMENT CODE EXAMPLE

```
String className="com.mysql.cj.jdbc.Driver";
String url="jdbc:mysql://localhost:3306/studentdb";
String username="root";
String pwd="root";
//1.LOAD OR REGISTER DRIVER
Class.forName(className);
//2.ESTABLISH CONNECTION
Connection connection=DriverManager.getConnection(url, username, pwd);
//3.CREATE STATEMENT
PreparedStatement preparedStatement = connection.prepareStatement("select * from student");
ResultSet resultSet = preparedStatement.executeQuery();
while (resultSet.next()) {
System.out.println(resultSet.getInt(1));
System.out.println(resultSet.getString(2));
System.out.println(resultSet.getInt(3));
System.out.println(resultSet.getString(4));
System.out.println(resultSet.getString(5));
}
preparedStatement.execute();
preparedStatement.close();
connection.close();
```

BATCH EXECUTION

- Instead of executing a single query we can execute batch of queries . It makes the performance fast
- Batch processing groups multiple queries into one unit and passes it in a single strike to a database.
- To avoid the frequent communication with the database which results to a low performance we do make use of BatchExecution.
- JDBC provides two methods to perform batchExecution:

  1.addBatch()

  2.executeBatch()
- **addBatch():** It is used to add the objects to the particular batch
- **executeBatch():** It is used to execute the multiple objects at single time

Example for BatchExecution

//Create the objects

```
for(User u:list){
preparedStatement.setInt(1, u.getUserID());
preparedStatement.setString(2, u.getUserName());
preparedStatement.setString(3, u.getUserEmail());
preparedStatement.setString(4, u.getPassword());
preparedStatement.addBatch();
System.out.println("inserted");
}
preparedStatement.executeBatch();
```

THANK YOU