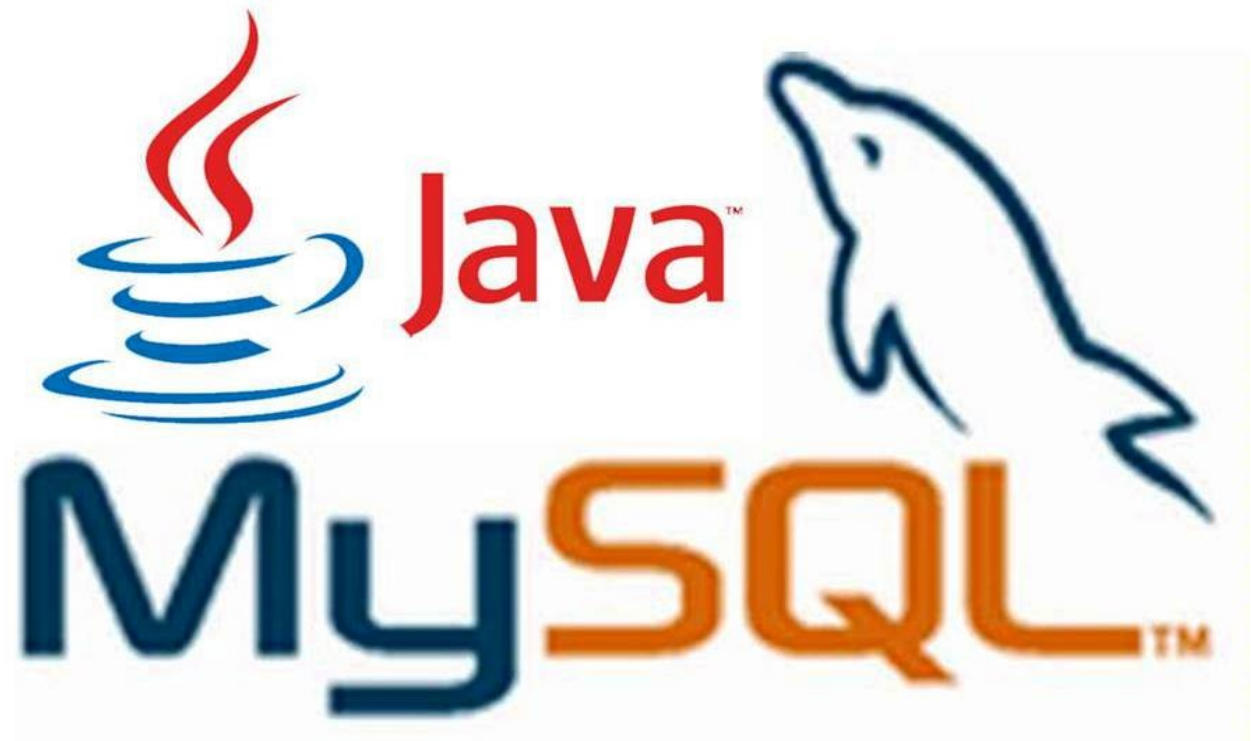


# JDBC

Team Emertxe

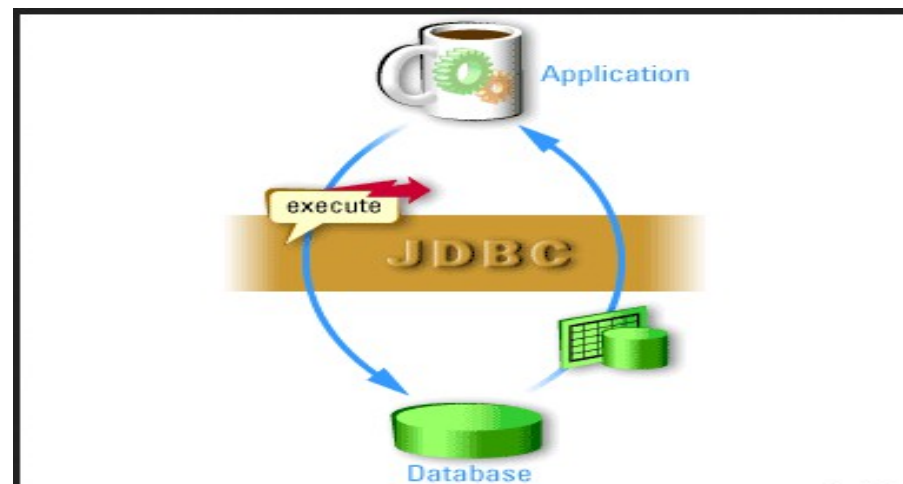




What is JDBC?

# Definition

- JDBC is a Java-based data access technology (Java Standard Edition platform) from Oracle Corporation.
- This technology is an API for the Java programming language that defines how a client may access a database.
- It provides methods for querying and updating data in a database.



# JDBC History



# JDBC History



- Before JDBC, ODBC API was used to connect and execute query to the database.
- But ODBC API uses ODBC driver that is written in C language which is platform dependent and unsecured.
- That is why Sun Micro System has defined its own API (JDBC API) that uses JDBC driver written in Java language.

# JDBC History

- Sun Microsystems released JDBC as part of JDK 1.1 on February 19, 1997.
- The JDBC classes are contained in the Java package `java.sql` and `javax.sql`
- The latest version, JDBC 4.2, and is included in Java SE 8.

What is API



# API



- The Java API is the set of classes included with the Java Development Environment. These classes are written using the Java language and run on the JVM. The Java API includes everything from collection classes to GUI classes.
- JDBC is also an API.



JDBC Drivers



# JDBC Drivers


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



## Type 1: JDBC-ODBC Bridge Driver



- The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. This is now discouraged because of thin driver.



## Type 2: Native-API Driver



- The Native API driver uses the client-side libraries of the database.
- It is not written entirely in Java.



## Type 3: Network Protocol Driver



- The Network Protocol driver uses middle ware (application server).
- It is fully written in Java.

## Type 4: Thin Driver

- The thin driver converts JDBC calls directly into the vendor-specific database protocol.
- That is why it is known as thin driver.
- It is fully written in Java language.
- Better performance than all other drivers.
- No software is required at client side or server side.
- *Disadvantage:* Drivers depends on the Database.

# Steps to Connect to the Database in Java



# Steps to Connect to Database



*There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:*

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection



# Registering the Driver



- The `forName()` method of `Class` class is used to register the driver class.
- `Class.forName("com.mysql.jdbc.Driver");`

# Creating Connection Object



- The getConnection() method of DriverManager class is used to establish connection with the database.
- Connection  
`con=DriverManager.getConnection( "jdbc:mysql://localhost:3306","root","password");`

# Creating Statement Object



- The `createStatement()` method of `Connection` interface is used to create statement. The object of statement is responsible to execute queries with the database.
- `Statement stmt=con.createStatement();`

# Execute Query

- The `executeQuery()` method of `Statement` interface is used to execute queries to the database. This method returns the object of `ResultSet` that can be used to get all the records of a table.
- *`ResultSet rs=stmt.executeQuery("select * from emp");`*

```
while(rs.next()){
```

```
System.out.println(rs.getInt(1)+" "+rs.getString(2));
```

```
}
```

# Closing Connection



- By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.
- `con.close();`

Connecting to the MySQL Database

# Connect to MySQL Database



- Driver class: `com.mysql.jdbc.Driver`.
- Connection URL: `jdbc:mysql://localhost:3306/db_name`
- Username: The default username for the mysql database is root.
- Password: Given by the user at the time of installing the mysql database
- *Example:*

```
Connection con=DriverManager.getConnection(  
"jdbc:mysql://localhost:3306/sonoo","root","root");
```

## Loading the .jar

- Download the MySQL connector.jar from [mysql.com](https://mysql.com)
- Paste the mysqlconnector.jar in the lib folder of source directory.
- Set the classpath



# DriverManager class



- The DriverManager class acts as an interface between user and drivers.
- It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.
- *Connection con = null;  
con=DriverManager.getConnection( "jdbc:mysql://localhost:3306","root","password");*
- *DriverManager.registerDriver().*

# Connection Interface



- A Connection is the session between Java application and database.
- The Connection interface is a factory of Statement and PreparedStatement.
- Object of Connection can be used to get the object of Statement and PreparedStatement.

# Connection Interface



## *Methods of Connection interface:*

- `public Statement createStatement():` creates a statement object that can be used to execute SQL queries.
- `public void commit():` saves the changes made since the previous commit/rollback permanent.
- `public void close():` closes the connection and Releases a JDBC resources immediately.

# Statement Interface



- The Statement interface provides methods to execute queries with the database.
- The statement interface is a factory of ResultSet.
- It provides factory method to get the object of ResultSet.

# Statement Interface

## *Methods of Statement interface:*

- `public ResultSet executeQuery(String sql):` is used to execute SELECT query. It returns the object of ResultSet.
- `public int executeUpdate(String sql):` is used to execute specified query, it may be create, drop, insert, update, delete etc.
- `public boolean execute(String sql):` is used to execute queries that may return multiple results.

# Statement Interface

*Example:*

- *Statement stmt=con.createStatement();  
int result=stmt.executeUpdate("delete from table where id=xy");  
System.out.println(result+" records affected");  
con.close();*

# ResultSet interface

- The object of ResultSet maintains a cursor pointing to a particular row of data.
- Initially, cursor points to before the first row.

# ResultSet Interface

## *Methods of ResultSet interface:*

- **public boolean next():** is used to move the cursor to the one row next from the current position.
- **public boolean previous():** is used to move the cursor to the one row previous from the current position.
- **public boolean first():** is used to move the cursor to the first row in result set object.
- **public boolean last():** is used to move the cursor to the last row in result set object.



# ResultSet Interface

*Methods of ResultSet interface:*

- **public int getInt(int columnIndex):** is used to return the data of specified column index of the current row as int.
- **public int getInt(String columnName):** columnName): is used to return the data of specified column name of the current row as int.
- **public String getString(int columnIndex):** is used to return the data of specified column index of the current row as String.
- **public String getString(String columnName):** is used to return the data of specified column name of the current row as String.

# ResultSet Interface

*Example:*

- ```
ResultSet rs=stmt.executeQuery("select * from table");  
//getting the record of 3rd row  
rs.absolute(3);  
  
System.out.println(rs.getString(1)+" "+rs.getString(2)+"  
"+rs.getString(3));  
  
con.close();
```

# PreparedStatement Interface



- The PreparedStatement interface is a sub interface of Statement.
- It is used to execute parameterized query.
- Example of parameterized query:
  - `String sql="insert into emp values(?,?,?)";`

# PreparedStatement Interface



## *Methods of PreparedStatement:*

- *public void setInt(int paramIndex, int value): sets the integer value to the given parameter index.*
- *public void setString(int paramIndex, String value): sets the String value to the given parameter index.*
- *public void setFloat(int paramIndex, float value): sets the float value to the given parameter index.*

# PreparedStatement Interface



## *Methods of PreparedStatement:*

- `public void setDouble(int paramIndex, double value):` sets the double value to the given parameter index.
- `public int executeUpdate():` executes the query. It is used for create, drop, insert, update, delete etc.
- `public ResultSet executeQuery():` executes the select query. It returns an instance of `ResultSet`.

# PreparedStatement Interface



*Example:*

- PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");  
stmt.setInt(1,101); // 1 specifies the first parameter in the query  
stmt.setString(2,"Ratan");  
int i=stmt.executeUpdate();  
System.out.println(i+" records inserted");

# Questions

Write a program to implement CRUD in a table.

Write a program to implement PreparedStatement?

# Stay connected

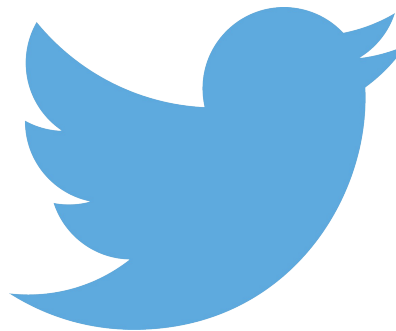


**About us:** Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies,  
No-1, 9th Cross, 5th Main,  
Jayamahal Extension,  
Bangalore, Karnataka 560046  
T: +91 80 6562 9666  
E: [training@emertxe.com](mailto:training@emertxe.com)



<https://www.facebook.com/Emertxe>



<https://twitter.com/EmertxeTweet>



**slideshare**  
Present Yourself

<https://www.slideshare.net/EmertxeSlides>





Thank You