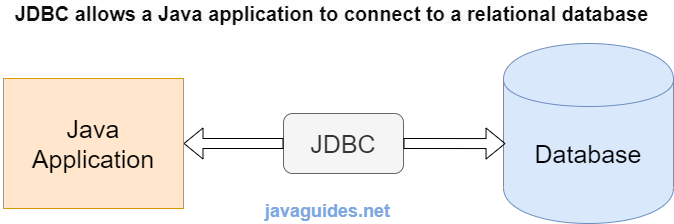
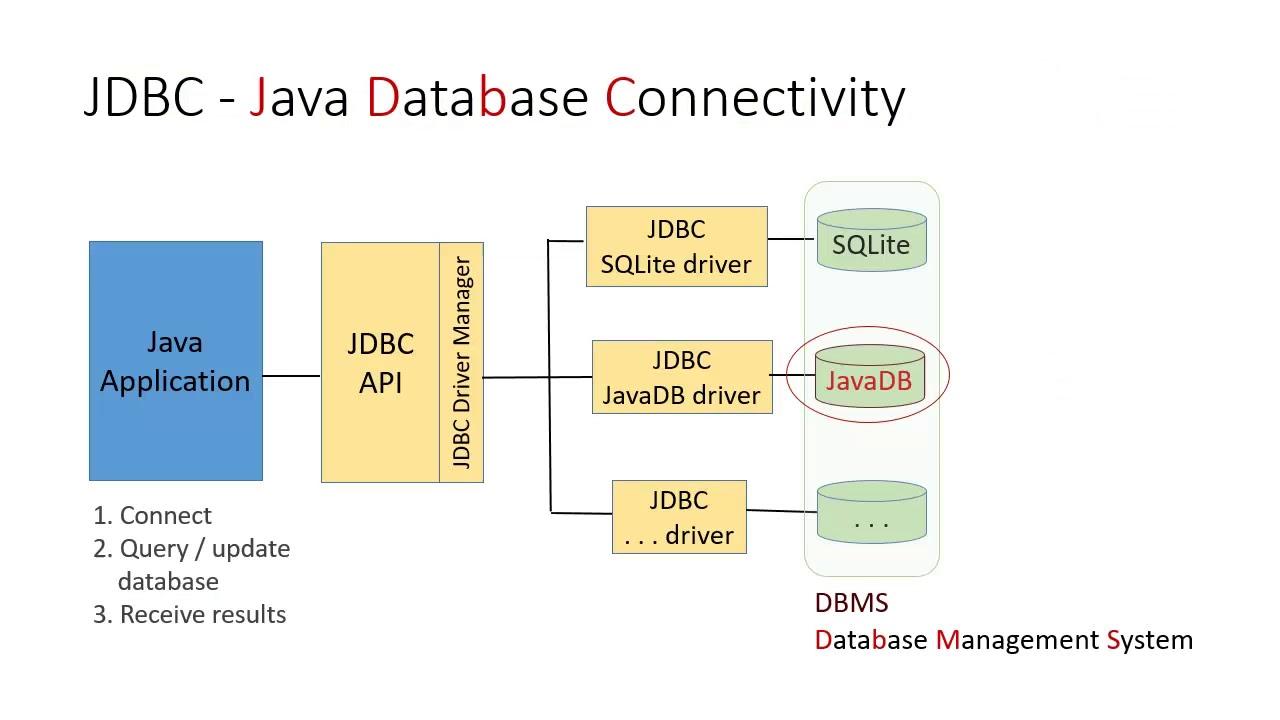
# **1. What is JDBC?**



# **2. JDBC Flow**

1. Connect to a data source, like a database
2. Send queries and update statements to the database
3. Retrieve and process the results received from the database in answer to your query



# **3. JDBC API**

The JDBC API is comprised of two packages:

* [**java.sql**](https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html)
* [**javax.sql**](https://docs.oracle.com/javase/8/docs/api/javax/sql/package-summary.html)

## **Key Classes and Interfaces**

* [**JDBC Connection Interface**](https://www.javaguides.net/2018/10/jdbc-connection-interface.html)
* [**JDBC Statement Interface**](https://www.javaguides.net/2018/10/jdbc-statement-interface.html)
* [**JDBC PreparedStatement Interface**](https://www.javaguides.net/2018/10/jdbc-preparedstatement-interface.html)
* [**JDBC CallableStatement Interface**](https://www.javaguides.net/2018/10/jdbc-callablestatement-interface.html)
* [**JDBC ResultSet Interface with Examples**](https://www.javaguides.net/2018/10/jdbc-resultset-interface-with-examples.html)
* [**JDBC ResultSetMetaData Interface**](https://www.javaguides.net/2018/10/jdbc-resultsetmetadata-interface.html)
* [**JDBC DatabaseMetaData Interface**](https://www.javaguides.net/2018/10/jdbc-databasemetadata-interface.html)
* [**JDBC DriverManager Class**](https://www.javaguides.net/2018/10/jdbc-drivermanager-class.html)

# **4. JDBC Driver Types**

1. **Type 1: JDBC-ODBC bridge driver**
2. **Type 2: Java + Native code driver**
3. **Type 3: All Java + Middleware translation driver**
4. **Type 4: All Java drivers.**

# **5. Fundamental Steps in JDBC**

The fundamental steps involved in the process of connecting to a database and executing a query consist of the following:

1. Import JDBC packages
2. Load and register the JDBC driver **// This step is not required in Java 6 and in JDBC 4.0**
3. Open a connection to the database.
4. Create a statement object to perform a query.
5. Execute the statement object and return a query resultset.
6. Process the resultset.
7. Close the resultset and statement objects. **// This step is not required because we use a try-with-resource statement to auto-close the resources**
8. Close the connection. **// This step is not required because we use a try-with-resource statement to auto-close the resources**

## **Step 1. Import JDBC packages**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

## **Step 2. Open a connection to the database**

**Syntax:** Here are overloaded getConnection() methods available:

Connection java.sql.DriverManager.getConnection(String url) throws SQLException

Connection java.sql.DriverManager.getConnection(String url, String username, String password) throws SQLException

Connection java.sql.DriverManager.getConnection(String url, Properties info) throws SQLException

**Syntax:** Here are overloaded getConnection() methods available:

Connection java.sql.DriverManager.getConnection(String url) throws SQLException

Connection java.sql.DriverManager.getConnection(String url, String username, String password) throws SQLException

Connection java.sql.DriverManager.getConnection(String url, Properties info) throws SQLException

**Example:** The following lines of code illustrate using the getConnection() method to connect to a MySQL database:

try (Connection connection = DriverManager

.getConnection("jdbc:mysql://localhost:3306/mysql\_database?useSSL=false", "root", "

## **Step 3. Create a statement object to perform a query**

**public Statement createStatement() throws SQLException**

**Example: Let's create a Statement object using a connection.createStatement() method:**

**// Step 3:Create a statement using connection object**

**Statement stmt = connection.createStatement();**

## **Step 4. Execute the statement object and return a query resultset**

**public ResultSet executeQuery(String query) throws SQLException**

**Example:**

**// Step 4: Execute the query or update query**

**ResultSet rs = stmt.executeQuery(QUERY));**

## **Step 5. Process the resultset**

**Below snippet shows how to process ResultSet object using while loop:**

**// Step 4: Process the ResultSet object.**

**while (rs.next()) {**

**int id = rs.getInt("id");**

**String name = rs.getString("name");**

**String email = rs.getString("email");**

**String country = rs.getString("country");**

**String password = rs.getString("password");**

**System.out.println(id + "," + name + "," + email + "," + country + "," + password);**

**}**

## **JDBC Statement Create a Table Example**

**In this example, we will create a users table using SQL script:**

**create table users(**

**id int(3) primary key,**

**name varchar(20),**

**email varchar(20),**

**country varchar(20),**

**password varchar(20)**

**);**

Example:-

**Statement statement = connection.createStatement();) {**

**// Step 3: Execute the query or update query**

**statement.execute(createTableSQL);**

## **JDBC Statement Insert Multiple Records Example**

private static final String INSERT\_MULTIPLE\_USERS\_SQL = "INSERT INTO Users " +

"VALUES (3, 'Pramod', 'pramod@gmail.com', 'India', '123')," +

"(4, 'Deepa', 'deepa@gmail.com', 'India', '123')," + "(5, 'Tom', 'top@gmail.com', 'India', '123');";

atement statement = connection.createStatement();) {

int result = statement.executeUpdate(INSERT\_MULTIPLE\_USERS\_SQL);

int result = statement.executeUpdate(INSERT\_MULTIPLE\_USERS\_SQL);

System.out.println("No. of records affected : " + result);

## **JDBC Statement Update Record Example**

private static final String UPDATE\_USERS\_SQL = "update users set name = \"Ram\" where id = 1;";

// Step 3: Execute the query or update query

int result = statement.executeUpdate(UPDATE\_USERS\_SQL);

System.out.println("Number of records affected :: " + result);

## **JDBC Statement Select Records Example**

private static final String SELECT\_QUERY = "select id,name,email,country,password from Users";

// Step 3: Execute the query or update query

ResultSet rs = stmt.executeQuery(QUERY)) {

// Step 4: Process the ResultSet object.

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name");

String email = rs.getString("email");

String country = rs.getString("country");

String password = rs.getString("password");

System.out.println(id + "," + name + "," + email + "," + country + "," + password);

}

## **JDBC Statement Delete a Record Example**

private static final String DELETE\_USERS\_SQL = "delete from users where id = 3;";

// Step 3: Execute the query or update query

int result = statement.executeUpdate(DELETE\_USERS\_SQL);

System.out.println("Number of records affected :: " + result);

## **JDBC Statement Batch Insert Example**

The [**Statement interface**](http://www.javaguides.net/2018/10/jdbc-statement-interface.html) provides below two methods to perform batch operations

1. addBatch(String sql)
2. executeBatch()

### **Statement.addBatch(String sql)**

Adds the given SQL command to the current list of commands for this Statement object. The commands in this list can be executed as a batch by calling the method executeBatch.

### **Statement.executeBatch()**

Submits a batch of commands to the database for execution and if all commands execute successfully, returns an array of update counts. The int elements of the array that is returned are ordered to correspond to the commands in the batch, which are ordered according to the order in which they were added to the batch.

Statement statement = connection.createStatement())

connection.setAutoCommit(false);

statement.addBatch("INSERT INTO Users VALUES (2, 'Pramod', 'pramod@gmail.com', 'India', '123');");

statement.addBatch("INSERT INTO Users VALUES (3, 'A', 'a@gmail.com', 'India', '123');");

statement.addBatch("INSERT INTO Users VALUES (4, 'B', 'b@gmail.com', 'India', '123');");

statement.addBatch("INSERT INTO Users VALUES (5, 'C', 'c@gmail.com', 'India', '123');");

statement.addBatch("INSERT INTO Users VALUES (6, 'D', 'd@gmail.com', 'India', '123');");

int[] updateCounts = statement.executeBatch();

System.out.println(Arrays.toString(updateCounts));

connection.commit();

## **Steps to process Update SQL statement with JDBC**

1. Establishing a connection.
2. Create a statement.
3. Execute the query.
4. Using try-with-resources Statements to Automatically Close JDBC Resources

## **JDBC PreparedStatement - Insert a Record**

private static final String INSERT\_USERS\_SQL = "INSERT INTO users" +

" (id, name, email, country, password) VALUES " +

" (?, ?, ?, ?, ?);";

PreparedStatement preparedStatement = connection.prepareStatement(INSERT\_USERS\_SQL)) {

preparedStatement.setInt(1, 1);

preparedStatement.setString(2, "Tony");

preparedStatement.setString(3, "tony@gmail.com");

preparedStatement.setString(4, "US");

preparedStatement.setString(5, "secret");

System.out.println(preparedStatement);

// Step 3: Execute the query or update query

preparedStatement.executeUpdate();

## **JDBC PreparedStatement Update a Record**

private static final String UPDATE\_USERS\_SQL = "update users set name = ? where id = ?;";

// Step 2:Create a statement using connection object

PreparedStatement preparedStatement = connection.prepareStatement(UPDATE\_USERS\_SQL)

preparedStatement.setString(1, "Ram");

preparedStatement.setInt(2, 1);

// Step 3: Execute the query or update query

preparedStatement.executeUpdate();

## **JDBC PreparedStatement Select Records**

private static final String QUERY = "select id,name,email,country,password from Users where id =?";

PreparedStatement preparedStatement = connection.prepareStatement(QUERY);) {

preparedStatement.setInt(1, 1);

System.out.println(preparedStatement);

// Step 3: Execute the query or update query

ResultSet rs = preparedStatement.executeQuery();

## **JDBC PreparedStatement Batch Insert**

String INSERT\_USERS\_SQL = "INSERT INTO users" + " (id, name, email, country, password) VALUES " +

" (?, ?, ?, ?, ?);";

PreparedStatement preparedStatement = connection.prepareStatement(INSERT\_USERS\_SQL)) {

connection.setAutoCommit(false);

preparedStatement.setInt(1, 20);

preparedStatement.setString(2, "a");

preparedStatement.setString(3, "a@gmail.com");

preparedStatement.setString(4, "India");

preparedStatement.setString(5, "secret");

preparedStatement.addBatch();

preparedStatement.setInt(1, 21);

preparedStatement.setString(2, "b");

preparedStatement.setString(3, "b@gmail.com");

preparedStatement.setString(4, "India");

preparedStatement.setString(5, "secret");

preparedStatement.addBatch();

preparedStatement.setInt(1, 22);

preparedStatement.setString(2, "c");

preparedStatement.setString(3, "c@gmail.com");

preparedStatement.setString(4, "India");

preparedStatement.setString(5, "secret");

preparedStatement.addBatch();

preparedStatement.setInt(1, 23);

preparedStatement.setString(2, "d");

preparedStatement.setString(3, "d@gmail.com");

preparedStatement.setString(4, "India");

preparedStatement.setString(5, "secret");

preparedStatement.addBatch();

int[] updateCounts = preparedStatement.executeBatch();

System.out.println(Arrays.toString(updateCounts));

connection.commit();

connection.setAutoCommit(true);

**CallableStatement**

MySQL stored procedure.

DELIMITER $$

USE `mysql\_database`$$

CREATE PROCEDURE `retreive\_users` ()

BEGIN

select \* from users;

END$$

DELIMITER ;

**retreive\_users**

String sql = "call retreive\_users()";

try (Connection conn = DriverManager.getConnection(jdbcUrl, username, password); CallableStatement stmt = conn.prepareCall(sql); ResultSet rs = stmt.executeQuery();) {

while (rs.next()) {

System.out.println("ID = " + rs.getInt(1) + ", NAME = " + rs.getString(2) + ", Email = " +

rs.getString(3) + ", Country = " + rs.getString(4) + ", Password = " + rs.getString(5));

}

## **CallableStatement with Multiple ResultSet**

stored procedure having multiple select statements.

DELIMITER $$

USE `mysql\_database`$$

BEGIN

select distinct name from users where id = 1;

select distinct email from users;

select count(id) as users\_count from users;

END

DELIMITER ;

String sql = "call retreive\_different\_results()";

try (Connection conn = DriverManager.getConnection(jdbcUrl, username, password); CallableStatement stmt = conn.prepareCall(sql);) {

boolean hasRs = stmt.execute();

System.out.println();

// Get Product Names

if (hasRs) {

try (ResultSet rs = stmt.getResultSet()) {

while (rs.next()) {

System.out.println("NAME = " + rs.getString(1));

}

}

}

// Get Total Price

if (stmt.getMoreResults()) {

try (ResultSet rs = stmt.getResultSet()) {

if (rs.next()) {

System.out.println("Email = " + rs.getString(1));

}

}

}

// Get Max/Min Price

if (stmt.getMoreResults()) {

try (ResultSet rs = stmt.getResultSet()) {

if (rs.next()) {

System.out.println("Users count = " + rs.getInt(1));

}

}

}

