**JDBC in Java | Java Database Connectivity**

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database, and processing the results. It is a part of **JavaSE** (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

* JDBC-ODBC Bridge Driver
* Native Driver
* Network Protocol Driver
* Thin Driver

We have discussed the above four drivers in the next chapter.

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.



The current version of JDBC is 4.3. It is the stable release since 21st September, 2017. It is based on the X/Open SQL Call Level Interface. The **java.sql** package contains classes and interfaces for JDBC API. A list of popular *interfaces* of JDBC API are given below:

* Driver interface
* Connection interface
* Statement interface
* PreparedStatement interface
* CallableStatement interface
* ResultSet interface
* ResultSetMetaData interface
* DatabaseMetaData interface
* RowSet interface

A list of popular *classes* of JDBC API are given below:

* DriverManager class
* Blob class
* Clob class
* Types class

**Why Should We Use JDBC?**

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But ODBC API uses ODBC driver that is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

Do You Know

* How to connect Java application with Oracle and Mysql database using JDBC?
* What is the difference between Statement and PreparedStatement interface?
* How to print total numbers of tables and views of a database using JDBC?
* How to store and retrieve images from Oracle database using JDBC?
* How to store and retrieve files from Oracle database using JDBC?

**What is API?**

JDBC also provides support for handling database metadata that allows us to retrieve information about the database, such as its tables, columns, and indexes. We can use the **DatabaseMetaData** interface to obtain this information that can be useful for dynamically generating SQL queries or for database schema introspection.

Another important feature of JDBC is its support for batch processing that allows us to group multiple SQL statements into a batch and execute them together. It can improve performance by reducing the number of round trips between the application and the database.

**Topics in Java JDBC Tutorial**

[2) JDBC Drivers](https://www.tpointtech.com/jdbc-driver)

In this JDBC tutorial, we will learn four types of JDBC drivers, their advantages and disadvantages.

[3) 5 Steps to connect to the Database](https://www.tpointtech.com/steps-to-connect-to-the-database-in-java)

In this JDBC tutorial, we will see the five steps to connect to the database in Java using JDBC.

[4) Connectivity with Oracle using JDBC](https://www.tpointtech.com/example-to-connect-to-the-oracle-database)

In this JDBC tutorial, we will connect a simple Java program with the Oracle database.

[5) Connectivity with MySQL using JDBC](https://www.tpointtech.com/example-to-connect-to-the-mysql-database)

In this JDBC tutorial, we will connect a simple Java program with the MySQL database.

[6) Connectivity with Access without DSN](https://www.tpointtech.com/connectivity-with-access-without-dsn)

Let's connect java application with access database with and without DSN.

[7) DriverManager class](https://www.tpointtech.com/DriverManager-class)

In this JDBC tutorial, we will learn what does the DriverManager class and what are its methods.

[8) Connection interface](https://www.tpointtech.com/Connection-interface)

In this JDBC tutorial, we will learn what is Connection interface and what are its methods.

[9) Statement interface](https://www.tpointtech.com/Statement-interface)

In this JDBC tutorial, we will learn what is Statement interface and what are its methods.

[10) ResultSet interface](https://www.tpointtech.com/ResultSet-interface)

In this JDBC tutorial, we will learn what is ResultSet interface and what are its methods. Moreover, we will learn how we can make the ResultSet scrollable.

[11) PreparedStatement Interface](https://www.tpointtech.com/PreparedStatement-interface)

In this JDBC tutorial, we will learn what is benefit of PreparedStatement over Statement interface. We will see examples to insert, update or delete records using the PreparedStatement interface.

[12) ResultSetMetaData interface](https://www.tpointtech.com/ResultSetMetaData-interface)

In this JDBC tutorial, we will learn how we can get the metadata of a table.

[13) DatabaseMetaData interface](https://www.tpointtech.com/DatabaseMetaData-interface)

In this JDBC tutorial, we will learn how we can get the metadata of a database.

[14) Storing image in Oracle](https://www.tpointtech.com/storing-image-in-oracle-database)

Let's learn how to store image in the Oracle database using JDBC.

[15) Retrieving image from Oracle](https://www.tpointtech.com/retrieving-image-from-oracle-database)

Let's see the simple example to retrieve image from the Oracle database using JDBC.

[16) Storing file in Oracle](https://www.tpointtech.com/storing-file-in-oracle-database)

Let's see the simple example to store file in the Oracle database using JDBC.

[17) Retrieving file from Oracle](https://www.tpointtech.com/retrieving-file-from-oracle-database)

Let's see the simple example to retrieve file from the Oracle database using JDBC.

[18) CallableStatement](https://www.tpointtech.com/CallableStatement-interface)

Let's see the code to call stored procedures and functions using CallableStatement.

[19) Transaction Management using JDBC](https://www.tpointtech.com/transaction-management-in-jdbc)

Let's see the simple example to use transaction management using JDBC.

[20) Batch Statement using JDBC](https://www.tpointtech.com/batch-processing-in-jdbc)

Let's see the code to execute batch of queries.

[21) JDBC RowSet](https://www.tpointtech.com/jdbc-rowset)

Let's see the working of new JDBC RowSet interface.

# JDBC Driver

17 Mar 2025 |  3 min read

1. [JDBC Drivers](https://www.tpointtech.com/jdbc-driver)
   1. [JDBC-ODBC bridge driver](https://www.tpointtech.com/jdbc-driver#driver1)
   2. [Native-API driver](https://www.tpointtech.com/jdbc-driver#driver2)
   3. [Network Protocol driver](https://www.tpointtech.com/jdbc-driver#driver3)
   4. [Thin driver](https://www.tpointtech.com/jdbc-driver#driver4)

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

### 1) JDBC-ODBC bridge driver

|  |
| --- |
| The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver. |



#### In Java 8, the JDBC-ODBC Bridge has been removed.

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

### Advantages:

* easy to use.
* can be easily connected to any database.

### Disadvantages:

* Performance degraded because JDBC method call is converted into the ODBC function calls.
* The ODBC driver needs to be installed on the client machine.

### 2) Native-API driver

|  |
| --- |
| The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java. |



### Advantage:

* performance upgraded than JDBC-ODBC bridge driver.

### Disadvantage:

* The Native driver needs to be installed on the each client machine.
* The Vendor client library needs to be installed on client machine.

### 3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.



### Advantage:

* No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

### Disadvantages:

* Network support is required on client machine.
* Requires database-specific coding to be done in the middle tier.
* Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

### 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. |



### Advantage:

* Better performance than all other drivers.
* No software is required at client side or server side.

### Disadvantage:

* Drivers depend on the Database.

# 5 Steps to Connect Database in Java

24 Mar 2025 |  3 min read

1. [5 Steps to connect to the database in java](https://www.tpointtech.com/steps-to-connect-to-the-database-in-java)
   1. [Register the driver class](https://www.tpointtech.com/steps-to-connect-to-the-database-in-java#step1)
   2. [Create the connection object](https://www.tpointtech.com/steps-to-connect-to-the-database-in-java#step2)
   3. [Create the Statement object](https://www.tpointtech.com/steps-to-connect-to-the-database-in-java#step3)
   4. [Execute the query](https://www.tpointtech.com/steps-to-connect-to-the-database-in-java#step4)
   5. [Close the connection object](https://www.tpointtech.com/steps-to-connect-to-the-database-in-java#step5)

|  |
| --- |
| There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:   * Register the Driver class * Create connection * Create statement * Execute queries * Close connection |



### 1) Register the driver class

|  |
| --- |
| The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class. |

### Syntax of forName() method

1. **public** **static** **void** forName(String className)**throws** ClassNotFoundException

#### Note: Since JDBC 4.0, explicitly registering the driver is optional. We just need to put vender's Jar in the classpath, and then JDBC driver manager can detect and load the driver automatically.

### Example to register the OracleDriver class

Here, Java program is loading oracle driver to esteblish database connection.

1. Class.forName("oracle.jdbc.driver.OracleDriver");

### 2) Create the connection object

|  |
| --- |
| The **getConnection()** method of DriverManager class is used to establish connection with the database. |

### Syntax of getConnection() method

1. 1) **public** **static** Connection getConnection(String url)**throws** SQLException
2. 2) **public** **static** Connection getConnection(String url,String name,String password)
3. **throws** SQLException

### Example to establish connection with the Oracle database

1. Connection con=DriverManager.getConnection(
2. "jdbc:oracle:thin:@localhost:1521:xe","system","password");

### 3) Create the 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. |

### Syntax of createStatement() method

1. **public** Statement createStatement()**throws** SQLException

### Example to create the statement object

1. Statement stmt=con.createStatement();

### 4) Execute the 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. |

### Syntax of executeQuery() method

1. **public** ResultSet executeQuery(String sql)**throws** SQLException

### Example to execute query

1. ResultSet rs=stmt.executeQuery("select \* from emp");
3. **while**(rs.next()){
4. System.out.println(rs.getInt(1)+" "+rs.getString(2));
5. }

### 5) Close the connection object

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

### Syntax of close() method

1. **public** **void** close()**throws** SQLException

### Example to close connection

1. con.close();

#### Note: Since Java 7,8 JDBC has ability to use try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement.

It avoids explicit connection closing step.

**Java Database Connectivity with Oracle**

24 Mar 2025 |  2 min read

|  |
| --- |
| To connect java application with the oracle database, we need to follow 5 following steps. In this example, we are using Oracle 10g as the database. So we need to know following information for the oracle database:   1. **Driver class:**The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**. 2. **Connection URL:**The connection URL for the oracle10G database is **jdbc:oracle:thin:@localhost:1521:xe** where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these information from the tnsnames.ora file. 3. **Username:**The default username for the oracle database is **system**. 4. **Password:**It is the password given by the user at the time of installing the oracle database. |

|  |
| --- |
| Create a Table  Before establishing connection, let's first create a table in oracle database. Following is the SQL query to create a table. |

1. create table emp(id number(10),name varchar2(40),age number(3));

Example to Connect Java Application with Oracle database

In this example, we are connecting to an Oracle database and getting data from **emp** table. Here, **system** and **oracle** are the username and password of the Oracle database.

1. **import** java.sql.\*;
2. **class** OracleCon{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. *//step1 load the driver class*
6. Class.forName("oracle.jdbc.driver.OracleDriver");
8. *//step2 create  the connection object*
9. Connection con=DriverManager.getConnection(
10. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
12. *//step3 create the statement object*
13. Statement stmt=con.createStatement();
15. *//step4 execute query*
16. ResultSet rs=stmt.executeQuery("select \* from emp");
17. **while**(rs.next())
18. System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
20. *//step5 close the connection object*
21. con.close();
23. }**catch**(Exception e){ System.out.println(e);}
25. }
26. }

[download this example](https://images.tpointtech.com/src/jdbc/OracleCon.zip)

The above example will fetch all the records of emp table.

To connect java application with the Oracle database ojdbc14.jar file is required to be loaded.

[download the jar file jdbc14.jar](https://images.tpointtech.com/src/jdbc/ojdbc14.jar)

**Two ways to load the jar file:**

1. paste the ojdbc14.jar file in jre/lib/ext folder
2. set classpath

**1) paste the ojdbc14.jar file in JRE/lib/ext folder:**

|  |
| --- |
| Firstly, search the ojdbc14.jar file then go to JRE/lib/ext folder and paste the jar file here. |

**2) set classpath:**

|  |
| --- |
| There are two ways to set the classpath:   * temporary * permanent |

**How to set the temporary classpath:**

|  |
| --- |
| Firstly, search the ojdbc14.jar file then open command prompt and write: |

1. C:>set classpath=c:\folder\ojdbc14.jar;.;

**How to set the permanent classpath:**

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to ojdbc14.jar by appending ojdbc14.jar;.; as C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;

**Java Database Connectivity with MySQL**

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:**The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:**The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
3. **Username:**The default username for the mysql database is **root**.
4. **Password:**It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

1. create database sonoo;
2. use sonoo;
3. create table emp(id **int**(10),name varchar(40),age **int**(3));

Example to Connect Java Application with mysql database

In this example, sonoo is the database name, root is the username and password both.

1. **import** java.sql.\*;
2. **class** MysqlCon{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. Class.forName("com.mysql.jdbc.Driver");
6. Connection con=DriverManager.getConnection(
7. "jdbc:mysql://localhost:3306/sonoo","root","root");
8. *//here sonoo is database name, root is username and password*
9. Statement stmt=con.createStatement();
10. ResultSet rs=stmt.executeQuery("select \* from emp");
11. **while**(rs.next())
12. System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
13. con.close();
14. }**catch**(Exception e){ System.out.println(e);}
15. }
16. }

[download this example](https://images.tpointtech.com/src/jdbc/MysqlCon.zip)

The above example will fetch all the records of emp table.

To connect java application with the mysql database, **mysqlconnector.jar** file is required to be loaded.

[download the jar file mysql-connector.jar](https://images.tpointtech.com/src/jdbc/mysql-connector.jar)

**Two ways to load the jar file:**

1. Paste the mysqlconnector.jar file in jre/lib/ext folder
2. Set classpath

**1) Paste the mysqlconnector.jar file in JRE/lib/ext folder:**

|  |
| --- |
| Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here. |

**2) Set classpath:**

|  |
| --- |
| There are two ways to set the classpath:   * temporary * permanent |

**How to set the temporary classpath**

|  |
| --- |
| open command prompt and write: |

1. C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.;

**How to set the permanent classpath**

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;.; as C:\folder\mysql-connector-java-5.0.8-bin.jar;.;

# Connectivity with access without dsn and with dsn

Accessing a **Microsoft Access database** **without using a Data Source Name** (DSN) is often preferred in scenarios where portability, simplicity, or avoiding system-level configuration is essential. It can be achieved through a **DSN-less** connection, which allows a Java program to directly specify the connection parameters, bypassing the need for a pre-configured DSN.

In Java, connectivity with a Microsoft Access database is typically done using the JDBC-ODBC Bridge or a third-party library like **UCanAccess**. As the JDBC-ODBC bridge has been deprecated in newer versions of Java, UCanAccess is a popular alternative that offers robust support for Access databases.

## Key Components of DSN-less Connection

1. **Driver Specification:** A JDBC driver must be specified explicitly. For Access databases, the UCanAccess JDBC driver is commonly used.
2. **Database Path:** The absolute or relative file path to the Access database file (.mdb or .accdb) is directly provided in the connection URL.
3. **Dependencies:** UCanAccess requires certain libraries (ucanaccess.jar, commons-lang.jar, etc.) to be included in the project.
4. **Connection URL Format:** The connection URL includes the database file path and optional configurations.

## Advantages of DSN-less Connection

1. **Portability:** No need for DSN configuration on every system where the application runs.
2. **Ease of Setup:** Eliminates reliance on administrative tools or registry settings for configuration.
3. **Dynamic Connections:** Database paths can be easily changed programmatically.

### Example of DSN-less Connection Using Java and UCanAccess

**Database File: example.accb**

|  |  |
| --- | --- |
| **ID** | **Name** |
| 1 | Alice |
| 2 | Bob |
| 3 | Charlie |

**File Name: AccessDatabaseConnection.java**

1. **import** java.sql.Connection;
2. **import** java.sql.DriverManager;
3. **import** java.sql.ResultSet;
4. **import** java.sql.Statement;
5. **public** **class** AccessDatabaseConnection {
6. **public** **static** **void** main(String[] args) {
7. *// Path to the Microsoft Access database file*
8. String dbFilePath = "C:/databases/example.accdb";
9. *// Connection URL for UCanAccess driver*
10. String jdbcURL = "jdbc:ucanaccess://" + dbFilePath;
11. *// Database operations*
12. **try** (Connection connection = DriverManager.getConnection(jdbcURL)) {
13. System.out.println("Connection established successfully.");
14. *// Create a statement*
15. Statement statement = connection.createStatement();
16. *// Execute a query to retrieve data from Employees table*
17. String query = "SELECT ID, Name FROM Employees";
18. ResultSet resultSet = statement.executeQuery(query);
19. *// Process the result set*
20. System.out.println("Employee Details:");
21. **while** (resultSet.next()) {
22. System.out.println("ID: " + resultSet.getInt("ID"));
23. System.out.println("Name: " + resultSet.getString("Name"));
24. }
25. *// Close resources*
26. resultSet.close();
27. statement.close();
28. } **catch** (Exception e) {
29. e.printStackTrace();
30. }
31. }
32. }

**Output:**

*Connection established successfully.*

*Employee Details:*

*ID: 1*

*Name: Alice*

*ID: 2*

*Name: Bob*

*ID: 3*

*Name: Charlie*

**Explanation**

The provided code demonstrates a simple way to connect to an Access database file without using a DSN. It starts by specifying the file path of the database, which is directly embedded into the JDBC URL.

The UCanAccess JDBC driver is used to establish the connection. After successfully connecting, the program executes an SQL query on a specified table, retrieves the results, and prints them to the console. The try-with-resources statement ensures that the database connection is closed automatically, avoiding potential memory leaks.

This approach minimizes external dependencies and makes the application configuration-free for database connectivity.

### Steps to Set Up and Run the Code

**1. Download UCanAccess:**

* Visit the official [UCanAccess](http://ucanaccess.sourceforge.net/site.html) website.
* Download the latest version of the UCanAccess library.

**2. Add Dependencies**

Include the following JAR files in your project:

* ucanaccess-.jar
* commons-lang-.jar
* commons-logging-.jar
* hsqldb-.jar
* jackcess-.jar

#### Note: If using an IDE like Eclipse or IntelliJ, add these files to the project's build path.

**3. Prepare the Database**

Ensure the .accdb or .mdb file exists and has a valid schema (tables, columns, etc.).

**4. Run the Program**

* Compile and execute the Java program.
* Ensure the file path to the Access database is correct.
* Check the console output for query results or connection issues.

## UCanAccess Features and Configuration Options

* **Read/Write Operations:** UCanAccess supports all common SQL operations, such as SELECT, INSERT, UPDATE, and DELETE.
* **Data Integrity:** Handles data types, constraints, and relationships defined in the Access database.
* **Encryption Support:** It can connect to password-protected Access databases by appending ;jackcessOpener=YourOpenerClass to the connection URL.
* **Logging:** It supports detailed logging of SQL queries and transactions.

## Customizing the Connection URL

The connection URL can be customized for various purposes:

**1. Read-Only Mode**

1. String jdbcURL = "jdbc:ucanaccess://" + dbFilePath + ";openExclusive=true";

It ensures no other application can access the database simultaneously.

**2. Password-Protected Database**

1. String jdbcURL = "jdbc:ucanaccess://" + dbFilePath + ";jackcessOpener=YourOpenerClass";

Replace YourOpenerClass with the appropriate handler for decrypting the database.

**3. Memory Settings**

1. Add ;memory=**true** to improve performance **for** in-memory operations.

## Best Practices for DSN-less Connections

1. **Use Relative Paths:** For portability, use relative file paths instead of hardcoded absolute paths.  
   String dbFilePath = "./data/database.accdb";
2. **Error Handling:** Implement detailed error logging to identify issues during connection or query execution.
3. **Connection Pooling:** For high-performance applications, integrate a connection pooling library to manage database connections efficiently.
4. **Thread Safety:** Ensure thread-safe operations when accessing the database concurrently.

## Conclusion

Connecting to an Access database without a DSN is straightforward and provides flexibility for Java applications. The UCanAccess library simplifies this process by offering a robust JDBC driver with extensive functionality.

By avoiding DSN configurations, developers can achieve greater portability and reduced setup complexity, making this approach ideal for lightweight and easily deployable solutions.

# DriverManager class

29 Aug 2024 |  2 min read

The DriverManager class is the component of JDBC API and also a member of the java.sql package. The DriverManager class acts as an interface between users and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. It contains all the appropriate methods to register and deregister the database driver class and to create a connection between a Java application and the database. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver(). Note that before interacting with a Database, it is a mandatory process to register the driver; otherwise, an exception is thrown.

## Methods of the DriverManager Class

|  |  |
| --- | --- |
| **Method** | **Description** |
| **1) public static synchronized void registerDriver(Driver driver):** | is used to register the given driver with DriverManager. No action is performed by the method when the given driver is already registered. |
| **2) public static synchronized void deregisterDriver(Driver driver):** | is used to deregister the given driver (drop the driver from the list) with DriverManager. If the given driver has been removed from the list, then no action is performed by the method. |
| **3) public static Connection getConnection(String url) throws SQLException:** | is used to establish the connection with the specified url. The SQLException is thrown when the corresponding Driver class of the given database is not registered with the DriverManager. |
| **4) public static Connection getConnection(String url,String userName,String password) throws SQLException:** | is used to establish the connection with the specified url, username, and password. The SQLException is thrown when the corresponding Driver class of the given database is not registered with the DriverManager. |
| **5) public static Driver getDriver(String url)** | Those drivers that understand the mentioned URL (present in the parameter of the method) are returned by this method provided those drivers are mentioned in the list of registered drivers. |
| **6) pubic static int getLoginTimeout()** | The duration of time a driver is allowed to wait in order to establish a connection with the database is returned by this method. |
| **7) pubic static void setLoginTimeout(int sec)** | The method provides the time in seconds. sec mentioned in the parameter is the maximum time that a driver is allowed to wait in order to establish a connection with the database. If 0 is passed in the parameter of this method, the driver will have to wait infinitely while trying to establish the connection with the database. |
| **8) public static Connection getConnection(String URL, Properties prop) throws SQLException** | A connection object is returned by this method after creating a connection to the database present at the mentioned URL, which is the first parameter of this method. The second parameter, which is "prop", fetches the authentication details of the database (username and password.). Similar to the other variation of the getConnection() method, this method also throws the SQLException, when the corresponding Driver class of the given database is not registered with the DriverManager. |

# Java Connection interface

22 Mar 2025 |  2 min read

A Connection is a session between a Java application and a database. It helps to establish a connection with the database.

The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData, i.e., an object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback(), setAutoCommit(), setTransactionIsolation(), etc.

#### By default, connection commits the changes after executing queries.

### Commonly used methods of Connection interface:

**1) public Statement createStatement():** creates a statement object that can be used to execute SQL queries.

**2) public Statement createStatement(int resultSetType,int resultSetConcurrency):** Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

**3) public void setAutoCommit(boolean status):** is used to set the commit status. By default, it is true.

**4) public void commit():** saves the changes made since the previous commit/rollback is permanent.

**5) public void rollback():** Drops all changes made since the previous commit/rollback.

**6) public void close():** closes the connection and Releases a JDBC resources immediately.

## Connection Interface Fields

There are some common Connection interface constant fields that are present in the Connect interface. These fields specify the isolation level of a transaction.

**TRANSACTION\_NONE**: No transaction is supported, and it is indicated by this constant.

**TRANSACTION\_READ\_COMMITTED**: It is a constant which shows that the dirty reads are not allowed. However, phantom reads and non-repeatable reads can occur.

**TRANSACTION\_READ\_UNCOMMITTED**: It is a constant which shows that dirty reads, non-repeatable reads, and phantom reads can occur.

**TRANSACTION\_REPEATABLE\_READ**: It is a constant which shows that the non-repeatable reads and dirty reads are not allowed. However, phantom reads and can occur.

**TRANSACTION\_SERIALIZABLE**: It is a constant which shows that the non-repeatable reads, dirty reads as well as the phantom reads are not allowed.

# ResultSet interface

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

#### By default, ResultSet object can be moved forward only and it is not updatable.

But we can make this object to move forward and backward direction by passing either TYPE\_SCROLL\_INSENSITIVE or TYPE\_SCROLL\_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

1. Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE,
2. ResultSet.CONCUR\_UPDATABLE);

### Commonly used methods of ResultSet interface

|  |  |
| --- | --- |
| **1) public boolean next():** | is used to move the cursor to the one row next from the current position. |
| **2) public boolean previous():** | is used to move the cursor to the one row previous from the current position. |
| **3) public boolean first():** | is used to move the cursor to the first row in result set object. |
| **4) public boolean last():** | is used to move the cursor to the last row in result set object. |
| **5) public boolean absolute(int row):** | is used to move the cursor to the specified row number in the ResultSet object. |
| **6) public boolean relative(int row):** | is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative. |
| **7) public int getInt(int columnIndex):** | is used to return the data of specified column index of the current row as int. |
| **8) public int getInt(String columnName):** | is used to return the data of specified column name of the current row as int. |
| **9) public String getString(int columnIndex):** | is used to return the data of specified column index of the current row as String. |
| **10) public String getString(String columnName):** | is used to return the data of specified column name of the current row as String. |

### Example of Scrollable ResultSet

Let’s see the simple example of ResultSet interface to retrieve the data of 3rd row.

1. **import** java.sql.\*;
2. **class** FetchRecord{
3. **public** **static** **void** main(String args[])**throws** Exception{
5. Class.forName("oracle.jdbc.driver.OracleDriver");
6. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
7. Statement stmt=con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);
8. ResultSet rs=stmt.executeQuery("select \* from emp765");
10. *//getting the record of 3rd row*
11. rs.absolute(3);
12. System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));
14. con.close();
15. }}

# PreparedStatement interface

29 Mar 2025 |  2 min read

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

Let's see the example of parameterized query:

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

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

### Why use PreparedStatement?

**Improves performance**: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

#### How to get the instance of PreparedStatement?

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

1. **public** PreparedStatement prepareStatement(String query)**throws** SQLException{}

### Methods of PreparedStatement interface

The important methods of PreparedStatement interface are given below:

|  |  |
| --- | --- |
| **Method** | **Description** |
| 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. |
| 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. |

### Example of PreparedStatement interface that inserts the record

First of all create table as given below:

1. create table emp(id number(10),name varchar2(50));

Now insert records in this table by the code given below:

1. **import** java.sql.\*;
2. **class** InsertPrepared{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. Class.forName("oracle.jdbc.driver.OracleDriver");
7. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
9. PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
10. stmt.setInt(1,101);*//1 specifies the first parameter in the query*
11. stmt.setString(2,"Ratan");
13. **int** i=stmt.executeUpdate();
14. System.out.println(i+" records inserted");
16. con.close();
18. }**catch**(Exception e){ System.out.println(e);}
20. }
21. }

[download this example](https://images.tpointtech.com/src/jdbc/InsertPrepared.zip)

### Example of PreparedStatement interface that updates the record

1. PreparedStatement stmt=con.prepareStatement("update emp set name=? where id=?");
2. stmt.setString(1,"Sonoo");*//1 specifies the first parameter in the query i.e. name*
3. stmt.setInt(2,101);
5. **int** i=stmt.executeUpdate();
6. System.out.println(i+" records updated");

[download this example](https://images.tpointtech.com/src/jdbc/UpdatePrepared.zip)

### Example of PreparedStatement interface that deletes the record

1. PreparedStatement stmt=con.prepareStatement("delete from emp where id=?");
2. stmt.setInt(1,101);
4. **int** i=stmt.executeUpdate();
5. System.out.println(i+" records deleted");

[download this example](https://images.tpointtech.com/src/jdbc/DeletePrepared.zip)

### Example of PreparedStatement interface that retrieve the records of a table

1. PreparedStatement stmt=con.prepareStatement("select \* from emp");
2. ResultSet rs=stmt.executeQuery();
3. **while**(rs.next()){
4. System.out.println(rs.getInt(1)+" "+rs.getString(2));
5. }

[download this example](https://images.tpointtech.com/src/jdbc/RetrievePrepared.zip)

### Example of PreparedStatement to insert records until user press n

1. **import** java.sql.\*;
2. **import** java.io.\*;
3. **class** RS{
4. **public** **static** **void** main(String args[])**throws** Exception{
5. Class.forName("oracle.jdbc.driver.OracleDriver");
6. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
8. PreparedStatement ps=con.prepareStatement("insert into emp130 values(?,?,?)");
10. BufferedReader br=**new** BufferedReader(**new** InputStreamReader(System.in));
12. **do**{
13. System.out.println("enter id:");
14. **int** id=Integer.parseInt(br.readLine());
15. System.out.println("enter name:");
16. String name=br.readLine();
17. System.out.println("enter salary:");
18. **float** salary=Float.parseFloat(br.readLine());
20. ps.setInt(1,id);
21. ps.setString(2,name);
22. ps.setFloat(3,salary);
23. **int** i=ps.executeUpdate();
24. System.out.println(i+" records affected");
26. System.out.println("Do you want to continue: y/n");
27. String s=br.readLine();
28. **if**(s.startsWith("n")){
29. **break**;
30. }
31. }**while**(**true**);
33. con.close();
34. }}

Java ResultSetMetaData Interface

13 Feb 2025 |  1 min read

The metadata means data about data i.e. we can get further information from the data.

If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

Commonly Used Methods Of Resultsetmetadata Interface

|  |  |
| --- | --- |
| **Method** | **Description** |
| public int getColumnCount()throws SQLException | It returns the total number of columns in the ResultSet object. |
| public String getColumnName(int index)throws SQLException | It returns the column name of the specified column index. |
| public String getColumnTypeName(int index)throws SQLException | It returns the column type name for the specified index. |
| public String getTableName(int index)throws SQLException | It returns the table name for the specified column index. |

How to get the object of ResultSetMetaData:

|  |
| --- |
| The getMetaData() method of ResultSet interface returns the object of ResultSetMetaData. Syntax: |

1. **public** ResultSetMetaData getMetaData()**throws** SQLException

**Example of ResultSetMetaData interface :**

1. **import** java.sql.\*;
2. **class** Rsmd{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. Class.forName("oracle.jdbc.driver.OracleDriver");
6. Connection con=DriverManager.getConnection(
7. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
9. PreparedStatement ps=con.prepareStatement("select \* from emp");
10. ResultSet rs=ps.executeQuery();
11. ResultSetMetaData rsmd=rs.getMetaData();
13. System.out.println("Total columns: "+rsmd.getColumnCount());
14. System.out.println("Column Name of 1st column: "+rsmd.getColumnName(1));
15. System.out.println("Column Type Name of 1st column: "+rsmd.getColumnTypeName(1));
17. con.close();
18. }**catch**(Exception e){ System.out.println(e);}
19. }
20. }

**Output:**

*Total columns: 2*

*Column Name of 1st column: ID*

*Column Type Name of 1st column: NUMBER*