# Java JDBC

* JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database.

# JDBC Driver

1. [JDBC Drivers](https://www.javatpoint.com/jdbc-driver)
   1. [Native-API driver](https://www.javatpoint.com/jdbc-driver#driver2)
   2. [Network Protocol driver](https://www.javatpoint.com/jdbc-driver#driver3)
   3. [Thin driver](https://www.javatpoint.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. Native-API driver (partially java driver) 2. Network Protocol driver (fully java driver) 3. Thin driver (fully java driver) |

### 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.

# Java Database Connectivity with 5 Steps

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

### Example to register the mysqlDriver class

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

1. Class.forName("**com.mysql.jdbc.Driver**");

### 2) Create the connection object

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

**public** **static** Connection getConnection(String url,String name,String password)

1. **throws** SQLException

### Example to establish connection with the mysql database

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

# 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.

560.8K

Vodafone has launches Pro II - The UK’s fastest WiFi tech

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. }

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

# DriverManager class

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.

# Connection interface

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.

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

**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.

# Statement interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

### Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

|  |
| --- |
| **1) public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet. |
| **2) public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc. |
| **3) public boolean execute(String sql):** is used to execute queries that may return multiple results. |
| **4) public int[] executeBatch():** is used to execute batch of commands. |

### Example of Statement interface

Let’s see the simple example of Statement interface to insert, update and delete the record.

1. **import** java.sql.\*;
2. **class** FetchRecord{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. Class.forName("com.mysql.jdbc.Driver");
5. Connection con=DriverManager.getConnection(
6. "jdbc:mysql://localhost:3306/sonoo","root","root");
7. Statement stmt=con.createStatement();
9. //stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");
10. //int result=stmt.executeUpdate("update emp765 set name='Vimal',salary=10000 where id=33");
11. **int** result=stmt.executeUpdate("delete from emp765 where id=33");
12. System.out.println(result+" records affected");
13. con.close();
14. }}

# 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.

### 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("com.mysql.jdbc.Driver");
6. Connection con=DriverManager.getConnection(
7. "jdbc:mysql://localhost:3306/sonoo","root","root");
8. Statement stmt=con.createStatement();
9. ResultSet rs=stmt.executeQuery("select \* from emp765");
11. //getting the record of 3rd row
12. rs.absolute(3);
13. System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));
15. con.close();
16. }}

# PreparedStatement interface

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.

Play Videox[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

#### 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("com.mysql.jdbc.Driver");
6. Connection con=DriverManager.getConnection(
7. "jdbc:mysql://localhost:3306/sonoo","root","root");
8. PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
9. stmt.setInt(1,101);//1 specifies the first parameter in the query
10. stmt.setString(2,"Ratan");
12. **int** i=stmt.executeUpdate();
13. System.out.println(i+" records inserted");
15. con.close();
17. }**catch**(Exception e){ System.out.println(e);}
19. }
20. }

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

# Java ResultSetMetaData Interface

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("com.mysql.jdbc.Driver");
6. Connection con=DriverManager.getConnection(
7. "jdbc:mysql://localhost:3306/sonoo","root","root");
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

# Java DatabaseMetaData interface

DatabaseMetaData interface provides methods to get meta data of a database such as database product name, database product version, driver name, name of total number of tables, name of total number of views etc.

## Commonly used methods of DatabaseMetaData interface

* **public String getDriverName()throws SQLException:**it returns the name of the JDBC driver.
* **public String getDriverVersion()throws SQLException:**it returns the version number of the JDBC driver.
* **public String getUserName()throws SQLException:**it returns the username of the database.
* **public String getDatabaseProductName()throws SQLException:**it returns the product name of the database.
* **public String getDatabaseProductVersion()throws SQLException:**it returns the product version of the database.
* **public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)throws SQLException:**it returns the description of the tables of the specified catalog. The table type can be TABLE, VIEW, ALIAS, SYSTEM TABLE, SYNONYM etc.

### How to get the object of DatabaseMetaData:

The getMetaData() method of Connection interface returns the object of DatabaseMetaData. Syntax:

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

### Simple Example of DatabaseMetaData interface :

1. **import** java.sql.\*;
2. **class** Dbmd{
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. DatabaseMetaData dbmd=con.getMetaData();
10. System.out.println("Driver Name: "+dbmd.getDriverName());
11. System.out.println("Driver Version: "+dbmd.getDriverVersion());
12. System.out.println("UserName: "+dbmd.getUserName());
13. System.out.println("Database Product Name: "+dbmd.getDatabaseProductName());
14. System.out.println("Database Product Version: "+dbmd.getDatabaseProductVersion());
16. con.close();
17. }**catch**(Exception e){ System.out.println(e);}
18. }
19. }

Output:Driver Name: Oracle JDBC Driver

Driver Version: 10.2.0.1.0XE

Database Product Name: Oracle

Database Product Version: Oracle Database 10g Express Edition

Release 10.2.0.1.0 –Production

# Java CallableStatement Interface

CallableStatement interface is used to call the **stored procedures and functions**.

We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

### What is the difference between stored procedures and functions.

The differences between stored procedures and functions are given below:

Play Videox[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

|  |  |
| --- | --- |
| **Stored Procedure** | **Function** |
| is used to perform business logic. | is used to perform calculation. |
| must not have the return type. | must have the return type. |
| may return 0 or more values. | may return only one values. |
| We can call functions from the procedure. | Procedure cannot be called from function. |
| Procedure supports input and output parameters. | Function supports only input parameter. |
| Exception handling using try/catch block can be used in stored procedures. | Exception handling using try/catch can't be used in user defined functions. |

### How to get the instance of CallableStatement?

The prepareCall() method of Connection interface returns the instance of CallableStatement. Syntax is given below:

1. **public** CallableStatement prepareCall("{ call procedurename(?,?...?)}");

The example to get the instance of CallableStatement is given below:

1. CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");

It calls the procedure myprocedure that receives 2 arguments.

### Full example to call the stored procedure using JDBC

To call the stored procedure, you need to create it in the database. Here, we are assuming that stored procedure looks like this.

1. create or replace procedure "INSERTR"
2. (id IN NUMBER,
3. name IN VARCHAR2)
4. is
5. begin
6. insert into user420 values(id,name);
7. end;
8. /

The table structure is given below:

1. create table user420(id number(10), name varchar2(200));

In this example, we are going to call the stored procedure INSERTR that receives id and name as the parameter and inserts it into the table user420. Note that you need to create the user420 table as well to run this application.

1. **import** java.sql.\*;
2. **public** **class** Proc {
3. **public** **static** **void** main(String[] args) **throws** Exception{
5. Class.forName("com.mysql.jdbc.Driver");
6. Connection con=DriverManager.getConnection(
7. "jdbc:mysql://localhost:3306/sonoo","root","root");
8. CallableStatement stmt=con.prepareCall("{call insertR(?,?)}");
9. stmt.setInt(1,1011);
10. stmt.setString(2,"Amit");
11. stmt.execute();
13. System.out.println("success");
14. }
15. }

Now check the table in the database, value is inserted in the user420 table.

### Example to call the function using JDBC

In this example, we are calling the sum4 function that receives two input and returns the sum of the given number. Here, we have used the **registerOutParameter** method of CallableStatement interface, that registers the output parameter with its corresponding type. It provides information to the CallableStatement about the type of result being displayed.

The **Types** class defines many constants such as INTEGER, VARCHAR, FLOAT, DOUBLE, BLOB, CLOB etc.

Let's create the simple function in the database first.

1. create or replace function sum4
2. (n1 in number,n2 in number)
3. **return** number
4. is
5. temp number(8);
6. begin
7. temp :=n1+n2;
8. **return** temp;
9. end;
10. /

Now, let's write the simple program to call the function.

1. **import** java.sql.\*;
3. **public** **class** FuncSum {
4. **public** **static** **void** main(String[] args) **throws** Exception{
6. Class.forName("com.mysql.jdbc.Driver");
7. Connection con=DriverManager.getConnection(
8. "jdbc:mysql://localhost:3306/sonoo","root","root");
10. CallableStatement stmt=con.prepareCall("{?= call sum4(?,?)}");
11. stmt.setInt(2,10);
12. stmt.setInt(3,43);
13. stmt.registerOutParameter(1,Types.INTEGER);
14. stmt.execute();
16. System.out.println(stmt.getInt(1));
18. }
19. }

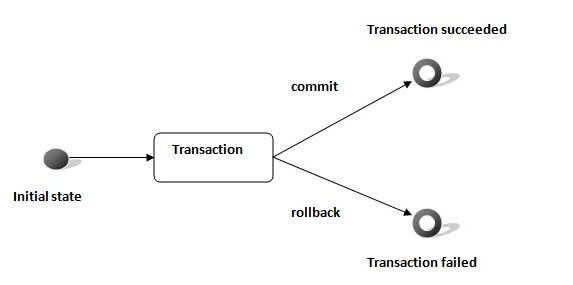
Output: 53

# Transaction Management in JDBC

Transaction represents **a single unit of work**.

#### Advantage of Transaction Mangaement

**fast performance** It makes the performance fast because database is hit at the time of commit.



In JDBC, **Connection interface** provides methods to manage transaction.

|  |  |
| --- | --- |
| **Method** | **Description** |
| void setAutoCommit(boolean status) | It is true bydefault means each transaction is committed bydefault. |
| void commit() | commits the transaction. |
| void rollback() | cancels the transaction. |

### Simple example of transaction management in jdbc using Statement

Let's see the simple example of transaction management using Statement.

1. **import** java.sql.\*;
2. **class** FetchRecords{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. Class.forName("com.mysql.jdbc.Driver");
5. Connection con=DriverManager.getConnection(
6. "jdbc:mysql://localhost:3306/sonoo","root","root");
7. con.setAutoCommit(**false**);
9. Statement stmt=con.createStatement();
10. stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");
11. stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");
13. con.commit();
14. con.close();
15. }}

If you see the table emp400, you will see that 2 records has been added.