Java Database Connectivity

===============================

where can you store the data?

1. variable

2. file

3. database

where can you store the data perminently?

1. file

2. database

what are the components of a data storage?

1. data

2. medium

3. storage

what data indicates?

what to persist

what medium indicates?

how to persist

what storage indicates?

where to persist

In Java programs, where the data exist?

1. in variables

2. in objects

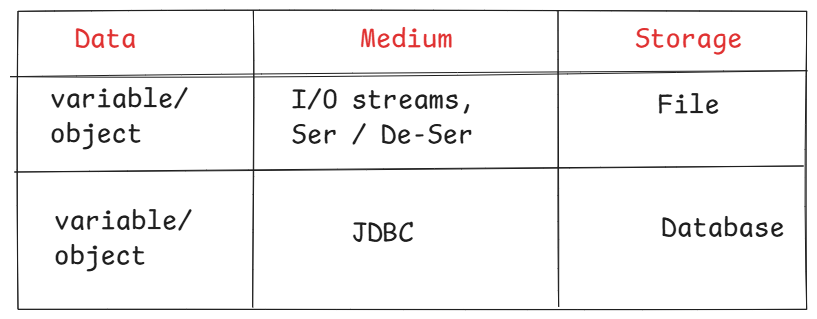
To store/read the data in a file, which Java concepts are used?

output streams & input streams

serialization & de-serialization

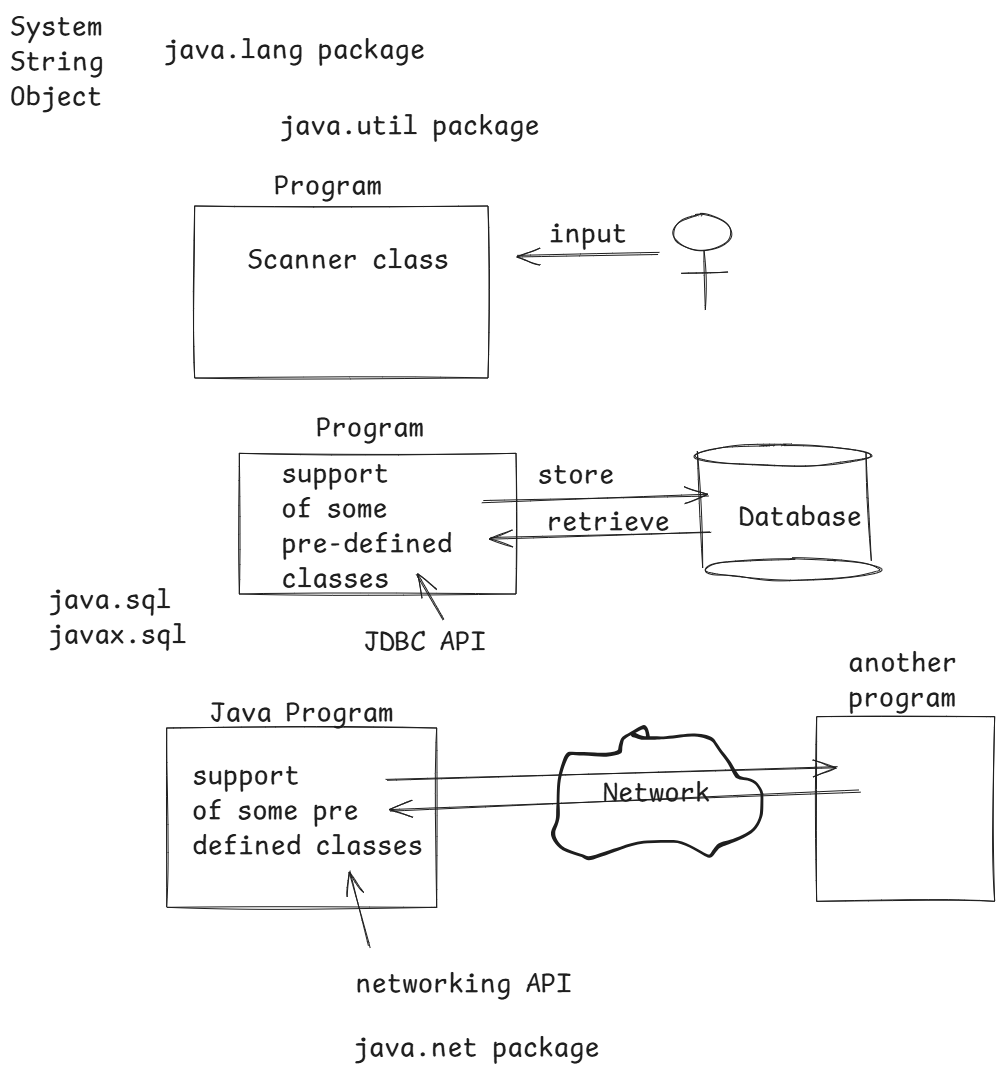
To store/read the data in a database, which Java technology is used?

JDBC



what is JDBC?

* JDBC is an API(Application Programming Interface).
* JDBC API consists a set of pre-defined classes and interfaces, which are used by the Java programs to interact with a Database and can store the data or can retrieve the data.

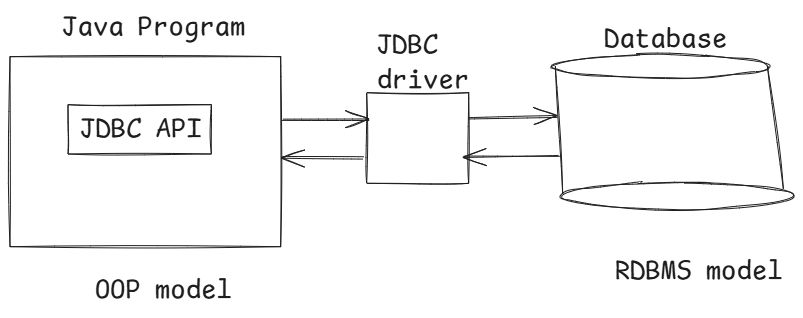


* JDBC API consists two packages

java.sql

javax.sql

* A Java program who wants to interact with a Database, can’t directly communicate with the database. Because, the Java program is developed in OOP model and the Database is developed in RDBMS model.
* So, a translator/mediator is required, to connect the two dissimilar environments.
* This translator is a JDBC Driver.



JDBC Driver:

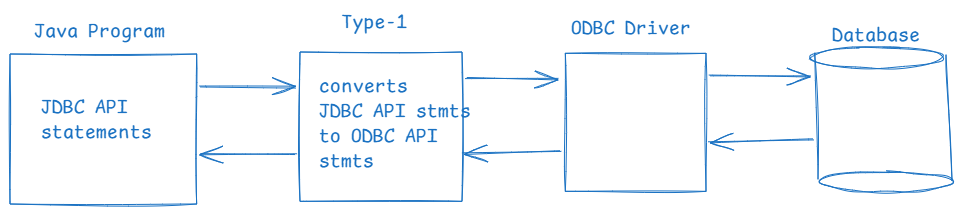
* JDBC driver is a software component which connects Java applications with the databases and transfers the data between the program and a database.
* Mostly database vendors provides the JDBC Driver softwares.
* The JDBC driver softwares, are divided into 4 types.

1. Type 1(JDBC ODBC Bridge Driver)
2. Type 2 (Native API Partially Java Driver)
3. Type 3 (Net Protocol Pure Java Driver)
4. Type 4(Native Protocol Pure Java Driver)

Note: Type 4 drivers are used in the realtime applications.

Type-1: (Bridge Driver)

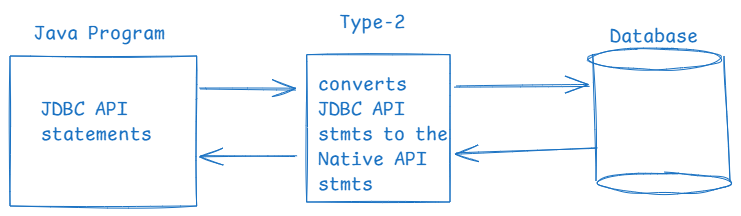
* Type-1 driver uses ODBC Driver to connect to the database.
* The JDBC API statements are converted to ODBC API statements, then ODBC Driver will execute these ODBC API statements on to the database.



* This Type-1 driver is given to connect a Java program with a database and to test the application is working fine or not on a local machine.
* The connectivity is very slow, and it is deprecated from Java 8.

Type-2 : (Native API driver)

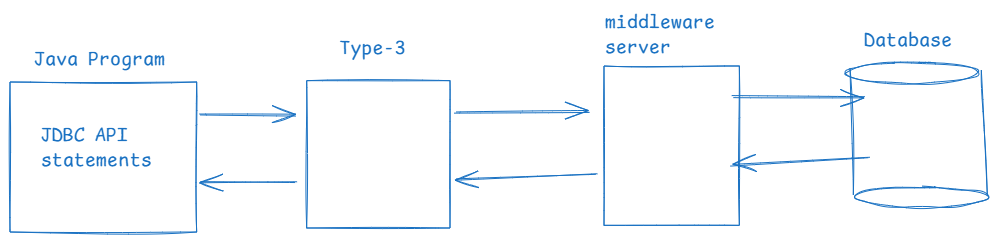
* This Type-2 uses Native API(database specific) to interact with the database.



* When compared to Type-1, Type-2 driver connects faster to the database.
* The drawback is, we require separate Type-2 driver for each Database.

Type-3: ( Net-protocol driver )

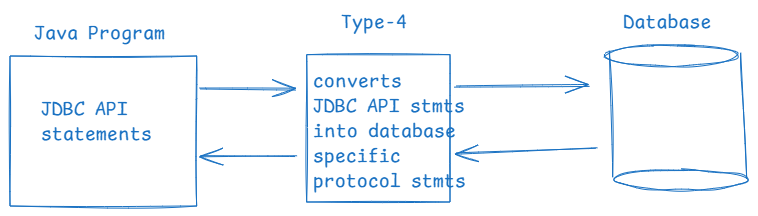
* This Type-3 driver communicates with a middleware server. This middleware server connects to the database using database-specific protocol.



* The advantage is Type-3 driver is completely developed in Java. So, it is pure java driver.
* The disadvantage is again a middleware server software is required/installed.

Type-4: (Thin driver)

* This Type-4 directly converts JDBC API statements/calls into database-specific protocol statements, without using any middleware server.



* This type-4 driver written completely in Java. So, it is platform independent.
* This is the faster driver in JDBC, to connect with the database.
* No need to use any middleware server.
* The disadvantage is, for each database, we have separate type-4 driver. So, if you switch database then you need switch driver also.
* The realtime applications of Java like JDBC applications, Hibernate applications, Spring Boot applications uses this Type-4 drivers.

steps to write a JDBC program:

1. load the driver
2. establish the connection with a database.
3. create the statement
4. execute the queries
5. process the result
6. close the connection with a database.

Q) what operations we can perform on the data?

A) CRUD operations/ CURD operations

C – create(insert new records)

R – Read (retrieve existing records)

U – Update (editing the existing records)

D – Delete ( delete the existing records)

Installing MySQL 8.x version:

1. visit

<https://dev.mysql.com/downloads/mysql/8.0.html>

1. click on Got to Download Page link
2. click on Windows(x86 32-bit)MSI installer

(303.6M) Download button.

4. click on No thanks, just start my download.

5. mysql-installer-community-8.0.39 file is downloaded.

6. Double click on the downloaded file, follow next buttons, then choose radio button full, then click on next and then execute.

7. click on Next buttons, enter the root account password as root, follow the next buttons,

at the last window, uncheck the start MySQL workbench and start MySQL Shell, then finish.

Note:

MySQL shell, is a command line application to connect with MySQL server.

MySQL workbench is a graphical user application to connect with MySQL server.

Q) How can you mysql server is started or not?

A) Goto, windows search 🡪 type services, and open the application 🡪 find MySQL80 application from the list and check the status Running.

MySQL Workbench:

* Goto windows search, type MySQL Workbench and open the application.
* click on the local instance button 🡪 enter the password 🡪 ok.
* click on schemas tab at left side
* type the below query in Query editor at right side, to create a new database called test.

create database test;

* To run this command, select the entire command, then click on execute on (flash icon) on the top.
* In schemas panel at left side, click on refresh icon and verify test database is created or not.

understanding JDBC program steps:

1. load the driver:

----------------

* In this step, we need to load the driver class to connect with a specific database into the JVM.
* To load the driver into JVM, we have to use the below statement.

Class.forName(“fully qualified name”);

* for example,

Class.forName(“oracle.jdbc.driver.OracleDriver”);

The above driver class is loaded to connect with Oracle Database.

* for example,

Class.forName(“com.mysql.cj.jdbc.Driver”);

The above driver class is loaded to connect with MySQL Database.

1. open the connection with database:

------------------------------------

* To open a connection with a database, 3 values are required.

1. database url
2. username
3. password

To connect with test database of MySQL server,

the databse url is,

jdbc:mysql://localhost:3306/test

(or)

jdbc:mysql://127.0.0.1:3306/test

* Use DriverManager class, to open a connection to a database. Call getConnection() static method of DriverManager class.

Connection conn = DriverManager.getConnection( databaseUrl, username, password);

for example,

Connection conn = DriverManager.getConnection( “jdbc:mysql://localhost:3306/test”, “root”, “root”);

3.create statement object:

\* The purpose of creating Statement object is, to send SQL queries to the database from our Java program.

Statement stmt = conn.createStatement();

4.execute the queries:

\* To execute the queries like insert/update/delete/select,….., we have to use Statement object.

\* The methods used to execute the queries are,

1. executeUpdate() : use this method to execute non-select operation. It returns the no of rows effected in the database.

2. executeQuery(): use this method to execute select operation. It returns a ResultSet object with all the rows selected, by executing the query.

5.process the result:

\* suppose, if you execute a select operation then you will get the rows into ResultSet object. Now you have to process the rows one by one, and display them.

6.close the connection:

\* You have to release the resources(memory), once the database operations are completed.

\* You have to close the statement, result set, connection to free the resources.

rs.close();

stmt.close();

conn.close();

First JDBC program to create a new table in the database

Table name : STUDENT

COLUMNS : SID INT,

SNAME VARCHAR(20),

SECTION VARCHAR(20),

MARKS INT

1. launch Eclipse
2. create a new Java project in eclipse.

(project: CreateTableProgram)

1. In src folder, delete module-info.java
2. create a new class in src folder, with package name com.ashokit.jdbc and class name CreateTable.
3. write the below program code.

**package** com.ashokit.jdbc;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** CreateTable {

**public** **static** **void** main(String[] args) **throws** Exception {

//step-1: load the driver

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

System.***out***.println("driver is loaded");

//step-2: open a connection

Connection conn = DriverManager.*getConnection*( "jdbc:mysql://localhost:3306/test", "root", "root");

System.***out***.println("connection is opened with database");

//step-3: create statement object

Statement stmt = conn.createStatement();

System.***out***.println("statement object is created");

//step-4: execute the query

String sqlQuery = "CREATE TABLE STUDENT ("

+ "SID INT, "

+ "SNAME VARCHAR(20), "

+ "SECTION VARCHAR(20), "

+ "MARKS INT, "

+ "PRIMARY KEY(SID, SECTION)"

+ ")" ;

stmt.executeUpdate(sqlQuery);

System.***out***.println("create table query is executed");

//step-5: process the result

System.***out***.println("Table is created");

//step-6: close the connection

stmt.close();

conn.close();

System.***out***.println("connection is closed");

}

}

1. download mysql-connector-j-8.3.0.jar file from google
2. right click on your project 🡪 choose build path 🡪 configure build path 🡪 choose libraries tab 🡪 choose classpath -🡪 click on Add external jars button and select the downloaded mysql-connector-j-8.3.0.jar 🡪open
3. Run the application.

op:

driver is loaded

connection is opened with database

statement object is created

create table query is executed

Table is created

connection is closed

How to check whether table is created in the database or not?

1. open MySQLWorkbench 🡪 connect with local instance 🡪 expand test database 🡪 expand tables

* verify the tabel STUDENT.

//Jdbc program to insert the records with static values

**package** com.ashokit.jdbc;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** InsertRows {

**public** **static** **void** main(String[] args) **throws** Exception {

// step-1: load the driver (this step is optional from Java8)

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

System.***out***.println("driver is loaded");

// step-2: open a connection

Connection conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/test", "root", "root");

System.***out***.println("connection is opened with database");

// step-3: create statement object

Statement stmt = conn.createStatement();

System.***out***.println("statement object is created");

//step-4: execute queries

String insert\_query\_1 = "INSERT INTO STUDENT"

+" VALUES (101, 'Anil', 'J1', 590)";

String insert\_query\_2 = "INSERT INTO STUDENT"

+" VALUES (102, 'Jyothi', 'J1', 487)";

String insert\_query\_3 = "INSERT INTO STUDENT"

+" VALUES (101, 'Ankit', 'J2', 390)";

stmt.executeUpdate(insert\_query\_1);

stmt.executeUpdate(insert\_query\_2);

stmt.executeUpdate(insert\_query\_3);

System.***out***.println("insert queries are executed.");

//step-5: process the result

System.***out***.println("3 rows are inserted");

//step-6: close the connection.

stmt.close();

conn.close();

System.***out***.println("connection is closed");

}

}

Selecting the records:

* When you run a select query from java program,

the result of the query will be stored into a

ResultSet object.

* ResultSet object maintains a cursor and it is by default placed at before the first row of the data, in the ResultSet object.
* you have to call next() method, to move the cursor. It returns true, and moves the cursor to next row, if the next row exists. Otherwise, it returns false.
* To retrieve the values from the columns of a row, call getXxx() methods. Like, getInt() or getString() or getDouble(),…..
* you can provide column index or the column label as a parameter.
* To retrieve the data of all the selected rows, you can use a while loop.

**package** com.ashokit.jdbc;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

**public** **class** SelectRows {

**public** **static** **void** main(String[] args) **throws** Exception {

// step-1: load the driver (this step is optional from Java8)

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

System.***out***.println("driver is loaded");

// step-2: open a connection

Connection conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/test", "root", "root");

System.***out***.println("connection is opened with database");

// step-3: create statement object

Statement stmt = conn.createStatement();

System.***out***.println("statement object is created");

//step-4: execute the query

String sqlQuery = "SELECT \* FROM STUDENT";

ResultSet rs = stmt.executeQuery(sqlQuery);

System.***out***.println("SELECT query is executed");

//step-5: process the result

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

System.***out***.println(rs.getInt("SID") + " " +rs.getString("SNAME")+" " + rs.getString("SECTION") +" "+rs.getInt("MARKS"));

}

//step-6

rs.close();

stmt.close();

conn.close();

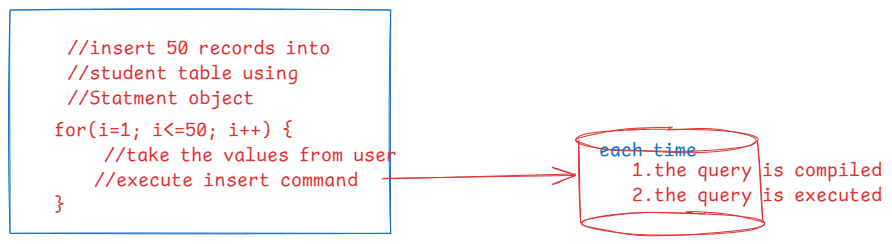
System.***out***.println("connection is closed");

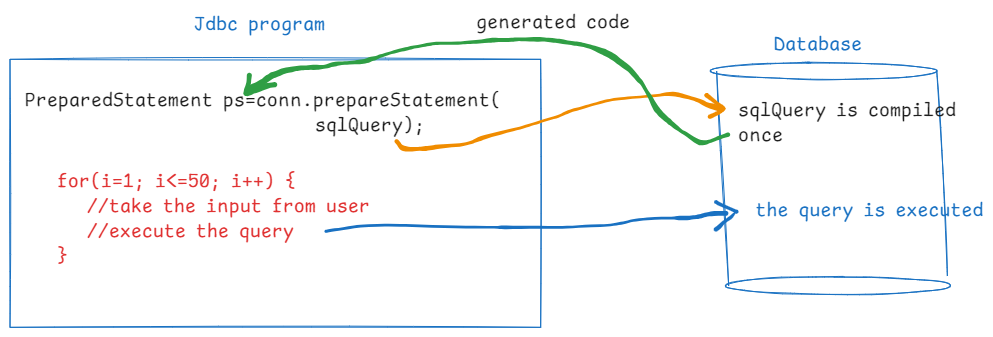
}

}

PreparedStatement in jdbc:

* When you want to execute the same query multiple times with different parameters, you can use Statement object also.
* But, repeatedly each time, the sql query will be compiled in the database and then executed. This will slow down the performance.
* So, to improve the performance of an application, we can use PreparedStatement object.
* A PreparedStatement is compiled once and can be resued multiple times with different parameters.
* This will reduce the repeated compilation of the same SQL query.





String sqlQuery = “INSERT INTO STUDENT VALUES(?, ?, ?, ?)”;

* In sql query, we have to use place holders(?), which are also called positional parameters.

Note: place holder must be the symbol ? only.

PreparedStatement ps = conn.prepareStatement(sqlQuery);

* To execute, the PreparedStatement, first set the values to place holders(?), by calling setXxx() methods.
* Then, call either executeUpdate()/executeQuery() methods.

**package** com.ashokit.jdbc;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.util.Scanner;

**public** **class** Main {

**private** **static** **final** String ***URL*** = "jdbc:mysql://localhost:3306/test";

**private** **static** **final** String ***USERNAME*** = "root";

**private** **static** **final** String ***PASSWORD*** = "root";

**private** **static** Connection *conn*;

**public** **static** **void** main(String[] args) **throws** Exception {

*conn* = DriverManager.*getConnection*(***URL***, ***USERNAME***, ***PASSWORD***);

System.***out***.println("connection is opened");

*insertData*();

*conn*.close();

System.***out***.println("connection is closed");

}

**private** **static** **void** insertData() **throws** Exception {

String sqlQuery = "INSERT INTO STUDENT VALUES(?, ?, ?, ?)";

//create PreparedStatement

PreparedStatement ps = *conn*.prepareStatement(sqlQuery);

//create Scanner

Scanner scan = **new** Scanner(System.***in***);

//insert 3 records

**for** ( **int** i = 1; i <= 3; i++ ) {

System.***out***.println("enter sid : ");

**int** sid = scan.nextInt();

scan.nextLine();

System.***out***.println("enter sname : ");

String sname = scan.nextLine();

System.***out***.println("enter section : ");

String section = scan.nextLine();

System.***out***.println("enter marks : ");

**int** marks = scan.nextInt();

//set the values

ps.setInt(1, sid);

ps.setString(2, sname);

ps.setString(3, section);

ps.setInt(4, marks);

**int** count = ps.executeUpdate();

System.***out***.println( count + " : row inserted");

}

ps.close();

scan.close();

}

}

Inserting an image into the Database:

* For inserting an image into the database, the table must contain a column of type BLOB.
* BLOB --- Binary Large Object
* You can insert or retrieve an image into the database by using PreparedStatement object only. We can’t do it with Statement object.

for ex:

CREATE TABLE IMAGES(

NAME VARCHAR(40),

IMAGE BLOB

);

//PROGRAM FOR INSERTING AN IMAGE

**package** com.ashokit.jdbc;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**public** **class** MainClass {

**private** **static** **final** String ***URL*** = "jdbc:mysql://localhost:3306/test";

**private** **static** **final** String ***USERNAME*** = "root";

**private** **static** **final** String ***PASSWORD*** = "root";

**private** **static** Connection *conn*;

**public** **static** **void** main(String[] args) **throws** Exception {

*conn* = DriverManager.*getConnection*(***URL***, ***USERNAME***, ***PASSWORD***);

System.***out***.println("connection is opened");

*insertImage*();

*conn*.close();

System.***out***.println("connection is closed");

}

**private** **static** **void** insertImage() **throws** Exception {

String sqlQuery = "INSERT INTO IMAGES VALUES(?, ?)";

PreparedStatement ps = *conn*.prepareStatement(sqlQuery);

File file = **new** File("C:\\Users\\WINDOWS\\Downloads\\Cube.gif");

FileInputStream fis = **new** FileInputStream(file);

//set the values

ps.setString(1, "Pocket Cube");

ps.setBinaryStream(2, fis);

//execute

**int** count = ps.executeUpdate();

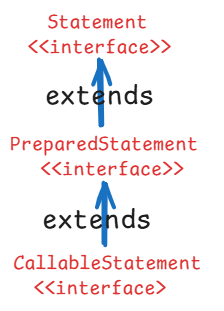
System.***out***.println(count + " : row inserted");

ps.close();

}

}

CallableStatement:



* CallableStatement of Jdbc can perform all the operations that a PreparedStatement can perform.
* CallableStatement can also invoke a procedure or a function of a database.
* Like methods in Java programming, we can also create procedures and functions in a database.

when to create a procedure/function in a database?

1. when there is a frequently used logic, then we can create a procedure/function.

ex: calculating employee bonuses can be placed in a procedure/function of the database and it can be invoked from the Java program whenever it is required.

1. when you want to restrict access to the database tables to users, then create a procedure/fuction and provide access to only the procedure/function. This is a kind of security.
2. when you want to reduce sending mulitiple queries to the database from application to reduce network traffic, then you create a procedure/function in a database.

* After calculation, if there is a single value to return then create a function.
* After calculation, if there is no value to return then create a procedure.

follow the below steps to create a function in mysql database:

1. open mysql workbench
2. click on local instance and login with username and password.
3. expand test database, then right click on Functions and choose create function.
4. write the below code.

CREATE FUNCTION `bonus\_function` (empid int)

RETURNS double deterministic

BEGIN

declare temp double;

declare bonus double;

select sal into temp from emp where empno=empid;

if temp <= 5000 then

set bonus = temp \* 0.20;

else

set bonus = temp \* 0.10;

end if;

RETURN bonus;

END

1. click on apply, again apply then finish.

**package** com.ashokit.jdbc;

**import** java.sql.CallableStatement;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Types;

**import** java.util.Scanner;

**public** **class** DBFunctionCall {

**private** **static** **final** String ***URL*** = "jdbc:mysql://localhost:3306/test";

**private** **static** **final** String ***USERNAME*** = "root";

**private** **static** **final** String ***PASSWORD*** = "root";

**private** **static** Connection *conn*;

**public** **static** **void** main(String[] args) {

**try** {

*conn* = DriverManager.*getConnection*(***URL***, ***USERNAME***, ***PASSWORD***);

System.***out***.println("connection is opened");

*calculateBonus*(*conn*);

}

**catch**(Exception ex) {

System.***out***.println(ex);

}

**finally** {

**try** {

**if**( *conn*!= **null** ) {

*conn*.close();

System.***out***.println("connection is closed");

}

}

**catch**(Exception ex) {

System.***out***.println(ex);

}

}

}

**private** **static** **void** calculateBonus(Connection conn) {

CallableStatement cs = **null**;

**try** {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("enter empno : ");

**int** empno = scan.nextInt();

String str = "{ ? = call bonus\_function(?) }";

//The call statement is compiled and the compiled

//code will be stored into the cs object.

cs = conn.prepareCall(str);

//register out parameter

cs.registerOutParameter(1, Types.***DOUBLE***);

//set the in parameter

cs.setInt(2, empno);

//execute the CallableStatement

cs.execute();

//get the value from out parameter

**double** bonus = cs.getDouble(1);

System.***out***.println("Bonus : " + bonus);

scan.close();

}

**catch**(Exception ex) {

System.***out***.println(ex);

}

**finally** {

**try** {

**if** ( cs != **null** ) {

cs.close();

}

} **catch**(Exception e) {

System.***out***.println(e);

}

}

}

}

creating a procedure:

A procedure doesn’t return a value explicitly, but it can store the values in out/inout parameters.

* A procedure parameters can have any of 3 parameter modes.

in – input

out – output

inout – input and output

* the below procedure takes deptno as input and selects the name and location of that department, stores them in out parameters.

CREATE PROCEDURE `dept\_procedure`(in dno int, out deptname varchar(20), out deptloc varchar(20))

BEGIN

select dname into deptname from dept where deptno = dno;

select loc into deptloc from dept where deptno = dno;

END

**package** com.ashokit.jdbc;

**import** java.sql.CallableStatement;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Types;

**import** java.util.Scanner;

**public** **class** DBProcedureCall {

**private** **static** **final** String ***URL*** = "jdbc:mysql://localhost:3306/test";

**private** **static** **final** String ***USERNAME*** = "root";

**private** **static** **final** String ***PASSWORD*** = "root";

**private** **static** Connection *conn*;

**public** **static** **void** main(String[] args) {

**try** {

*conn* = DriverManager.*getConnection*(***URL***, ***USERNAME***, ***PASSWORD***);

System.***out***.println("connection is opened");

*findDeptDetails*(*conn*);

}

**catch**(Exception ex) {

System.***out***.println(ex);

}

**finally** {

**try** {

**if**( *conn*!= **null** ) {

*conn*.close();

System.***out***.println("connection is closed");

}

}

**catch**(Exception ex) {

System.***out***.println(ex);

}

}

}

**private** **static** **void** findDeptDetails(Connection conn) {

CallableStatement cs = **null**;

**try** {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("enter deptno : ");

**int** deptno = scan.nextInt();

String str = "{ call dept\_procedure(?, ?, ?) }";

//The call statement is compiled and the compiled

//code will be stored into the cs object.

cs = conn.prepareCall(str);

//set the in parameter

cs.setInt(1, deptno);

//register out parameter

cs.registerOutParameter(2, Types.***VARCHAR***);

//register out parameter

cs.registerOutParameter(3, Types.***VARCHAR***);

//execute the CallableStatement

cs.execute();

//get the value from out parameter

String deptName = cs.getString(2);

String location = cs.getString(3);

System.***out***.println("deptname : " + deptName);

System.***out***.println("location : " + location);

scan.close();

}

**catch**(Exception ex) {

System.***out***.println(ex);

}

**finally** {

**try** {

**if** ( cs != **null** ) {

cs.close();

}

} **catch**(Exception e) {

System.***out***.println(e);

}

}

}

}

Q) what is the difference between Statement and PreparedStatement interfaces?

A) When you want to execute same query multiple times with different parameters, don’t use Statement object. Because, it compiles the query each time before execution. So, use PreparedStatement.

When you want to insert/retrieve images, you have to use PreparedStatement object. It can’t be done with Statement object.

Q) what is the difference between PreparedStaement and CallableStatement object?

A) PreparedStatement object can’t invoke procedures/functions of the database. But CallableStatement will do everything that PreparedStatement do, and also it can invoke the procedures/functions of the database.

ResultSetMetaData interface:

* When we are selecting the records from a table, the ResultSet object stores the selected records.
* If you don’t know how many columns are stored or what type of columns they are, or the data types of the columns then it is difficult to read the data of the records from the ResultSet object.
* So, to discover this information dynamically, ResultSetMetaData interface is provided in java.sql package.
* You can create ResultSetMetaData object like below.

ResultSetMetaData rsmd = rs.getMetaData();

* The methods of ResultSetMetaData,

1. getColumnCount(): returns the number of columns in the ResultSet.
2. getColumnName(column): returns the name of the column.
3. getColumnTypeName(column): returns the data type of the column
4. getColumnDisplaySize(column): returns the maximum width of the column.
5. isNullable(column): indicates whether a column can hold null values.

**package** com.ashokit.jdbc;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.ResultSetMetaData;

**import** java.sql.Statement;

**public** **class** MetaDataMain {

**private** **static** **final** String ***URL*** = "jdbc:mysql://localhost:3306/test";

**private** **static** **final** String ***USERNAME*** = "root";

**private** **static** **final** String ***PASSWORD*** = "root";

**private** **static** Connection *conn*;

**public** **static** **void** main(String[] args) {

**try** {

*conn* = DriverManager.*getConnection*(***URL***, ***USERNAME***, ***PASSWORD***);

System.***out***.println("connection is opened");

*findMetaData*(*conn*);

}

**catch**(Exception ex) {

System.***out***.println(ex);

}

**finally** {

**try** {

**if**( *conn*!= **null** ) {

*conn*.close();

System.***out***.println("connection is closed");

}

}

**catch**(Exception ex) {

System.***out***.println(ex);

}

}

}

**private** **static** **void** findMetaData(Connection conn) {

**try**(Statement stmt = conn.createStatement()) {

String str = "SELECT \* FROM EMP";

ResultSet rs = stmt.executeQuery(str);

ResultSetMetaData rsmd = rs.getMetaData();

**int** columnCount = rsmd.getColumnCount();

**for**(**int** i=1; i <= columnCount; i++) {

System.***out***.println("Column " + i + " : " + rsmd.getColumnName(i)

+ ", type " + rsmd.getColumnTypeName(i));

}

rs.close();

}

**catch**(Exception ex) {

System.***out***.println(ex);

}

}

}

Jdbc Transaction:

* A transaction is a group of/sequence of operations, that are executed as a single unit of work.
* A transaction follow a principle that either all operations of the unit are executed or none of them are executed.
* A transaction can be successful or failed.
* For example, when you make online purchase, the operations like deducting product from the inventroy, charging the customer, and generating the order.
* Suppose, in the midway like payment declined error occurs then the operations of the transaction are rolled back, and the transaction is ended with status failed.
* By default, each query is executed as a separate transaction from the jdbc program.
* If you want to execute multiple queries as a single transaction from the jdbc program then you have to apply Jdbc Transaction mechanism.
* We have to use the below methods.

1. setAutoCommit(boolean) : The parameter is true, by default.

* we have to disable the auto commit mode, by providing false as parameter.

conn.setAutoCommit(false);

1. commit() : commits the transaction
2. rollback(): rollbacks the transaction

**package** com.ashokit.jdbc;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.SQLException;

**import** java.util.Scanner;

**public** **class** TransactionTest {

**private** **static** **final** String ***URL*** = "jdbc:oracle:thin:@localhost:1521:xe";

**private** **static** **final** String ***USERNAME*** = "system";

**private** **static** **final** String ***PASSWORD*** = "manager";

**private** **static** Connection *conn*;

**public** **static** **void** main(String[] args) {

**try** {

*conn* = DriverManager.*getConnection*(***URL***, ***USERNAME***, ***PASSWORD***);

System.***out***.println("connection is opened");

*transferMoney*(*conn*);

} **catch**(Exception ex) {

System.***out***.println(ex);

}

**finally** {

**try** {

**if**( *conn* != **null**) {

*conn*.close();

System.***out***.println("connection is closed");

}

} **catch**(SQLException ex) {

System.***out***.println(ex);

}

}

}

**private** **static** **void** transferMoney(Connection conn) {

PreparedStatement ps1 = **null**;

PreparedStatement ps2 = **null**;

**try** {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("enter from account ");

**int** fromAccount = scan.nextInt();

System.***out***.println("enter to account ");

**int** toAccount = scan.nextInt();

System.***out***.println("enter amount to transfer");

**double** amount = scan.nextDouble();

//Disable auto commit mode

conn.setAutoCommit(**false**);

String query1 = "UPDATE ACCOUNTS SET BALANCE = BALANCE - ? WHERE ACCOUNT\_ID=?";

String query2 = "UPDATE ACCOUNTS SET BALANCE = BALANCE + ? WHERE ACCOUNT\_ID=?";

ps1 = conn.prepareStatement(query1);

ps2 = conn.prepareStatement(query2);

ps1.setDouble(1, amount);

ps1.setInt(2, fromAccount);

ps2.setDouble(1, amount);

ps2.setInt(2, toAccount);

ps1.executeUpdate();

ps2.executeUpdate();

//commit the transaction

conn.commit();

System.***out***.println("transaction is committed");

}

**catch**(Exception ex) {

**try** {

//rollback the transaction

conn.rollback();

System.***out***.println("transaction is rolled back");

} **catch**(SQLException e) {

System.***out***.println(e);

}

}

**finally** {

**try** {

**if** (ps1 != **null**) {

ps1.close();

}

**if** (ps2 != **null**) {

ps2.close();

}

} **catch**(Exception ex) {

System.***out***.println(ex);

}

}

}

}

Scrollable ResultSet :

---------------------

* By default, when we create a ResultSet object,

it is a forward only result set.

* It means, the cursor can be moved only in the forward direction.
* If you want to move the cursor in forward and backward direction also, then you have to create a scrollable result set object.
* while creating a statement object, we need to pass two parameters called type and mode.

Statement stmt = conn.createStatement(type, mode);

* type parameter can be,

ResultSet.TYPE\_SCROLL\_SENSITIVE (OR)

ResultSet.TYPE\_SCROLL\_INSENSITIVE (OR)

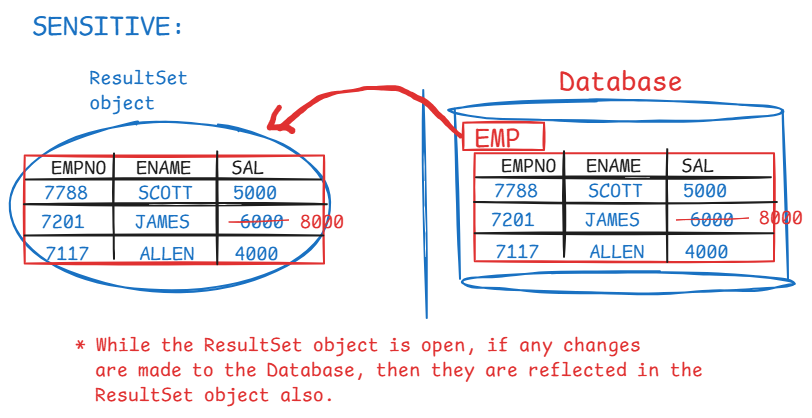
ResultSet.TYPE\_FORWARD\_ONLY (default value)

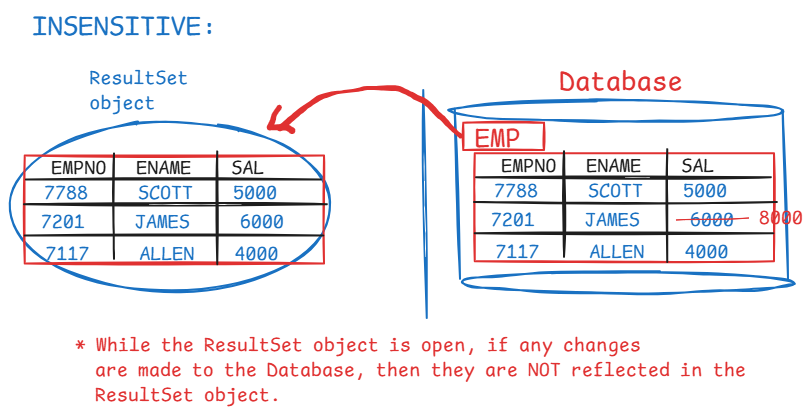
* mode parameter can be,

ResultSet.CONCUR\_READ\_ONLY (default value)

(OR)

ResultSet.CONCUR\_UPDATABLE





* If mode parameter is read only, then we can only read the records from the ResultSet object, but we can’t update the records.
* If mode parameter is updatable, then we can read the records and also we can modify the records. The changes are automatically updated to the database also.

Scrollable result set methods:

1. next()
2. previous()
3. afterLast()
4. beforeFirst()
5. first()
6. last()
7. absolute()

for ex:

rs.absolute(3); //cursor is moved to 3rd row from top.

rs.absolute(-3); // cursor is moved to 3rd row from bottom.

Connection pooling:

* Suppose, if our application is a server-side application and if it is creating a new connection and destroying connection for each request, the problems are,

1. performance: opening a new connection is time-consuming.
2. resource problem: if many clients/users are making request at a time, then the application has to open a new connection for each request. It leads to out of memory errors.

* The solution is connection pooling technique.
* In this technique, a group/limit of connections are opened to a database and stored in memory area called pool.
* The same connections are re-used for all the requests by the application.

How connection pool works?

* A connection pool is created mostly by server admin at the application startup.
* when a user sends request, the application borrows the connection from the pool.
* After executing the database operations/queries, instead of closing the connection, the application returns the connection back to the pool.
* if all the connections are busy/in use, then the new requests will be put on hold and when a connection comes back to the pool, it is served for the waiting request.
* The popular connection pooling libraries used in Java applications are,

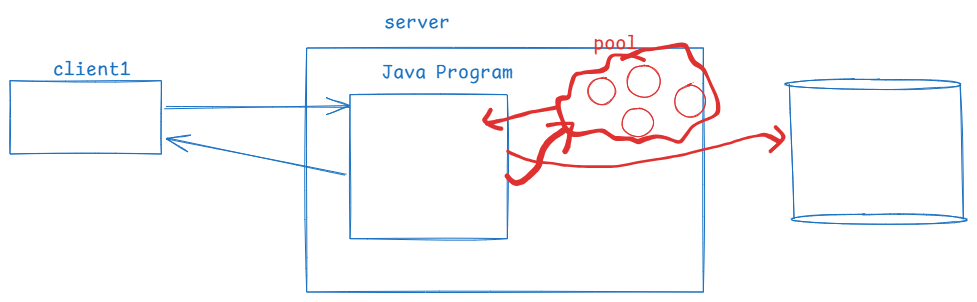
HikarCP : fast, lightweight and efficient

Apache DBCP: widely used connection pooling

implementation.

Who is DataSource?

* DataSource is an interface from javax.sql package.
* A Java application can obtain a connection with a Database, using either DriverManager class or DataSource interface.
* DriverManager can provide direct connection with a database, but it can’t provide a connection from the pool.
* DataSource can provide a connection pool. It means, Java application uses DataSource object to borrow the connection from a pool.



================================================