* If you don’t have any constructor – default constructor will be added by compiler to byte code.
* If you have one – then default constructor will not be added by compiler to byte code. You can add programmatically.

Open Workbench – SQL Editor [[ on terminal mysql -uroop -p ]

1. **Create Database**

create database practiceDB;

1. **Open Database / Make database Default**

use practiceDB;

1. **Create Table**

create table EMPLOYEE

(

EMP\_ID INT PRIMARY KEY,

EMP\_NAME VARCHAR(30) NOT NULL,

SALARY DOUBLE NOT NULL,

DEPT\_IT INT NULL

);

1. **RENAME COLUMN DEPT\_IT TO DEPT\_ID**

ALTER TABLE EMPLOYEE RENAME COLUMN DEPT\_IT TO DEPT\_ID;

1. **INSERTING Record**

INSERT INTO EMPLOYEE(EMP\_ID, EMP\_NAME, SALARY, DEPT\_ID) VALUE(1,'NITIN',2300.00,101);

--continue inserting multiple records for demo

1. **Display records**

SELECT \* FROM EMPLOYEE;

**Connecting to Database From Java**

* **Attached Database Driver to your classpath**

**String jdbcURL = “jdbc:mysql://locahost:3306/practiceDB”;**

**Connection con=DriverManager.getConnection(jdbcURL, username, password);**

**Statement**

**To execute the queries [ SQL ]**

**ResultSet executeQuery(“select statement”) ;**

**int executeUpdate(“update/insert/delete statement”);**

**boolean execute(“all queries”) // it return true – means it was select query and if false means it was update/delete**

[getResultSet()](https://docs.oracle.com/javase/8/docs/api/java/sql/Statement.html#getResultSet--) & [getUpdateCount()](https://docs.oracle.com/javase/8/docs/api/java/sql/Statement.html#getUpdateCount--) will be called respectively.

To avoid SQL INJECTTION than can do some malicious action on queries – we prefer to use preparedstatement

PreparedStatement is pre-compiled statement

**Data Access Object Pattern**

**CRUD OPERATION**

**C = CREATE / INSERT**

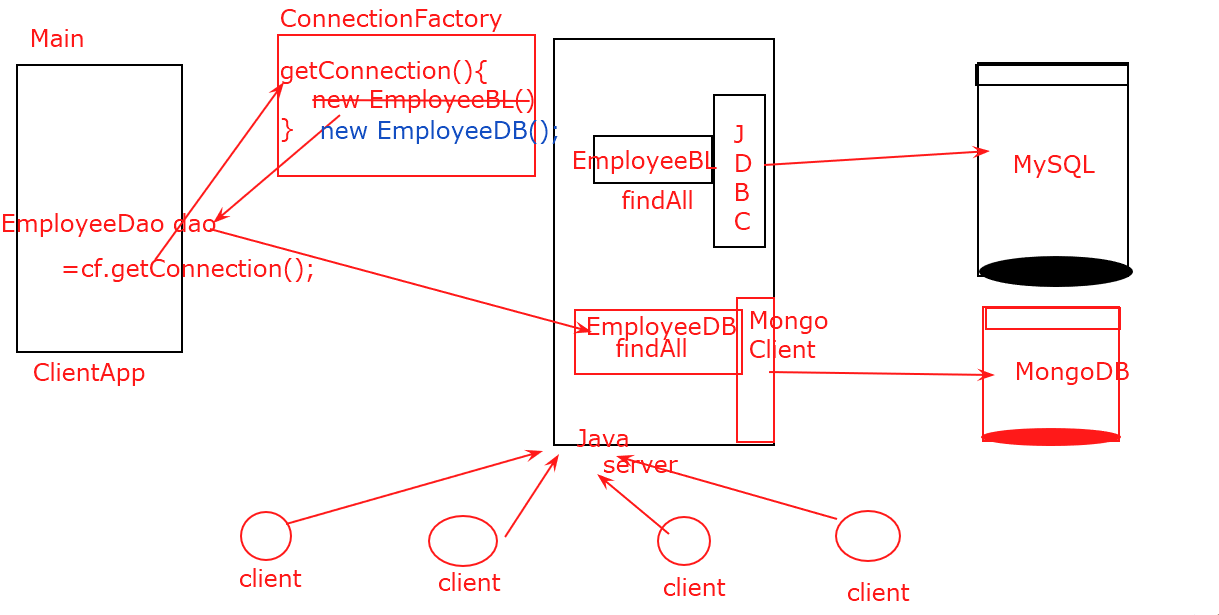
**R = READ / SELECT**

**U = UPDATE**

**D = DELETE**

**Singleton Design Pattern – must produce only one instance per JVM**

**ConnectionFactory class as listed in design below is Singleton class**



**Try…With Resource Statement**

**---- databases**

**---- File System**

**A class that implements AutoCloseable or its sub-interfaces are Resource Statement.**

**try**(Connection con=DriverManager.*getConnection*(jdbcURL, "root", "admin#123");

PreparedStatement st=con.prepareStatement("select \* from employee where emp\_id=?"))

{

}

}**catch**(SQLException se) {

**throw** **new** HRException(se.getMessage(),se);

}finally{

try{

con.close();

st.close()

}catch(SQLException){}

}

Highlighted statement above were no longer needed -if you use try with resource statement- It ensures that resource open, are closed automatically at the end.

try(resource statement)

{

}catch(…..){}

**Database Transaction Management**

Workbench Connection 1 / Terminal 1

1. Disable Autocommit ; set autocommit=0;
2. **update employee set salary = 2000 where emp\_id=5;**
3. **select \* from employee ; // you will notice the salary is changed**

Workbench Connection 2 / Terminal 2

1. **select \* from employee ; // you will notice the salary is not changed**

Workbench Connection 1 / Terminal 1

1. **rollback;**
2. **select \* from employee ; // you will notice the salary is reverted. // undone**

**\*\*\* Repeat the above steps [ 2 to 6 ] with “commit” at the place “rollback”. And go back to terminal2 – you will notice the updated salary is shown.**

**JDBC transaction management**

setAutoCommit(boolean status)

**commit()**

**rollback();**