JDBC

MYSQL

- CREATE USER 'mysql'@'localhost' IDENTIFIED BY 'mysql';
- GRANT SELECT ON * . * TO 'mysql'@'localhost';
- create database lowes;
- GRANT ALL PRIVILEGES ON `lowes` . * TO 'mysql'@'localhost';
- flush priviliges;
- exit;

Open terminal:
— First start using brew
1. brew services restart mysql
2. mysql -u mysql -p mysql
or
mysql -h localhost -P 3306 --protocol=tcp -u mysql -p. (only for Mac, linux connect this way changing the protocol to TCP)

DBeaver Error:
sudo upx -d /Applications/DBeaver.app/Contents/MacOS/dbeaver

JDBC:

Java Application --> JDBC --> JDBC DriverManager -->JDBC Driver -->Oracle
                                --> JDBC Driver -->MySql
                                --> JDBC Driver --> Postgres

Java Database Connectivity
JDBC is a Java API to connect and execute the query with the database.
It is a part of JavaSE (Java Standard Edition).
JDBC API uses JDBC drivers to connect with the database.

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 which 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.

-- JDBC Driver
There are 4 types of JDBC drivers:
1. JDBC-ODBC bridge driver ： In Java 8, the JDBC-ODBC Bridge has been removed.
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

-- Java Database Connectivity with 5 Steps
There are 5 steps to connect any java application with the database using JDBC

1. Load and Register the Driver class
2. Create connection
3. Create statement
4. Execute queries
5. Close connection

1) Register the driver class
The forName() method of Class class is used to load the driver class dynamically.
The Driver class contains a static block which has code to register the driver class which runs automatically whenever it is loaded.
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.
EX: Class.forName("oracle.jdbc.driver.OracleDriver"); // loads the Driver class and register the class

2) Create the connection object
The getConnection() method of DriverManager class is used to establish connection with the database.
EX: Connection con=DriverManager.getConnection(
"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.
EX: 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.
EX: Resultset rs = stmt.executeQuery("select * from emp");
    while(rs.next() {
    System.out.println(rs.getInt(1)+" "+rs.getString(2));
    }

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.
EX : con.close();
Note: Since Java 7, JDBC has ability to use try-with-resources statement to automatically close resources of type Connection,
      ResultSet, and Statement.

```
-- Example to Connect Java Application with Oracle database
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object
Statement stmt=con.createStatement();

//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

//step5 close the connection object
con.close();
```

```
}catch(Exception e){ System.out.println(e);}

}
}
```

———————

SAMPLE PROGRAM:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JbdcDemo {

public static void main(String[] args) {
System.out.println("JDBC Demo...");

Connection con = null;
Statement stmt = null;
ResultSet rs = null;

//STEP 1: Load Driver (MySQL)
try {
Class.forName("com.mysql.cj.jdbc.Driver");
System.out.println("Driver Loaded");
} catch (ClassNotFoundException e) {
System.out.println("Error: Drive not loaded - " + e.getMessage());
e.printStackTrace();
return;
}

//STEP 2: Open Connection to DB
try {
con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbctraining",
"training", "training");
System.out.println(con);
System.out.println("Connected to DB successfully");

//STEP 3: Create Statement
stmt = con.createStatement();
```

```
String query = "SELECT * FROM EMPLOYEE";

//STEP 4: Execute Query
rs = stmt.executeQuery(query);
System.out.println("Query executed successfully. Query: " + query);

//STEP 5: Get ResultSet and access the results
System.out.println("Got results - " + rs);
System.out.println("EmpID \t Name \t Age \t Designation \t Department \t
Country");
while(rs.next()) {
int id = rs.getInt("id");
String name = rs.getString("name");
int age = rs.getInt("age");
String designation = rs.getString("designation");
String department = rs.getString("department");
String country = rs.getString("country");
System.out.printf("%d \t %s \t %d \t %s \t %s \t %s \n", id, name, age, designation,
department, country);
}
} catch (SQLException e) {
System.out.println("Error: " + e.getMessage());
e.printStackTrace();
} finally {
try {
rs.close();
stmt.close();
con.close();
} catch (SQLException e) {
System.out.println("Error while closing the connections. " + e.getMessage());
e.printStackTrace();
}
}

}

}
```

-- DriverManager class
The DriverManager class acts as an interface between user and drivers.

It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.
The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Methods:
1) public static void registerDriver(Driver driver): is used to register the given driver with DriverManager.
2) public static void deregisterDriver(Driver driver): is used to deregister the given driver (drop the driver from the list) with DriverManager.
3) public static Connection getConnection(String url): is used to establish the connection with the specified url.
4) public static Connection getConnection(String url,String userName,String password):

-- Connection interface
A Connection is the session between java application and database.
The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. 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() etc.
By default, connection commits the changes after executing queries.

Methods:
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 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.
7) PreparedStatement prepareStatement(String sql)

-- Statement interface
3 types are statements exist
Statement
PreparedStatement

CallableStatement- for executing stored procedures, PL/SQL queries
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.

Methods:
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.

-- PreparedStatement interface (maintains precompiled query and can be accessed multiple times for subsequent queries, for better performance)
The PreparedStatement interface is a sub interface of Statement. It is used to execute parameterized query. To supply arguments dynamically.
It provides different types of setter methods to set the input parameters for the query.
Example: to insert values into a table with multiple columns

```
String sql = "INSERT INTO employee(FIRST_NAME,LAST_NAME) VALUES (?,?)";
PreparedStatement preparedStatement = connection.prepareStatement(sql);
preparedStatement.setString(1,first_name);
preparedStatement.setString(2,last_name);
int rowsInserted = preparedStatement.executeUpdate();
```

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

Methods:
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.

-- 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.

Methods:
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):

— ResultSet ScrollType
- We can scroll over the rows of data using this feature. The pointer of rs.next() can be moved forward, backward or jump to any position)
- Scrollability, Updatability, Holdability can be based to createStatement
- Types
- ResultSet.*TYPE_SCROLL_INSENSITIVE*. *(does not reflect the db changes)*
- ResultSet.*TYPE_FORWARD_ONLY (cursor moves forward only by 1 step , similar to rs.next()*
- ResultSet.*TYPE_SCROLL_SENSITIVE (reflects db changes*
- Statement stmt = conn.createStatement(ResultSet.*TYPE_SCROLL_INSENSITIVE*, ResultSet.*CONCUR_UPDATABLE*, ResultSet.*CLOSE_CURSORS_AT_COMMIT*);

— RowSet:
- similar to ResultSet

– Scrollability, Updatability is available by default
- directly can access the rows without creating statement
- types - JdbcRowSet(same as ResultSet), CachedRowSet(can work offline without db connectivity)

-- 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.

-- 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.

-- DAO : Data Access Object

Example 2:

```
class jdbcexample{
public static void main(String args[])
{
Class.forName("org.postgresql.Driver");  // forName() method used to register the Driver class. It loads the driver class dynamically
Connection con = DriverManager.getConnection(url, root, password); // connect to the database using getConnection() method of Driver Maanger class which returns a connection object
Statement stmt = con.createStatement(); // create statement object using createStatement() method  present in the Connection interface , statement object is used to execute the queries
ResultSet rs = stmt.executeQuery(sql); // executeQuery() method of Statement interface to create ResultSet object, which is used to pull the records from the database(execute the query)
while(rs.next()) {
System.oout.println(rs.getInt(1) + " " + rs.getString(2));
   con.Close() ; //. By closing connection object statement and ResultSet will be closed automatically.
}
}
```

NOTE:

-- For CREATE, INSERT,UPDATE, DELETE , use stmt.executeUpdate(sql)
-- For INSERT always use PreparedStatement , can be used for UPDATE also if we need to Update multiple fields, pass as parameters dynamically.
-- For SQL SELECT, use stmt.executeQuery(sql);
-- For any process with diff transactions to be done in one shot,(example: Fundtransfer with check balance, withdraw, deposit),
    enable auto commit(false) can be given and after finishing the transaction, we can commit .
    By default, in jdbc sutocommit is true

————————————————————————————

DataWarehousing Concepts:

  6. ETL Implementation Process:
— Generally any Business organization strive to acheive growth in revenue, strength, organization etc.,
Any business generate huge amount of data from various sources (inside the organization, social trends, market research, customer feedback etc),
and by analyzing these data, any business can gain better insights leading to its growth.
— Who needs Business insights?
Business executives, business analysts need these data , better strategies, decisions and implementations can be made
— Problems in achieving insights
The data collected from various sources and stored in different applications cannot be directly visualized (through graphs, plots, reports, dashboards)

This data must be integrated and then processed as required before visualization takes place.
— Business Intelligence :
The perfect solution to achieve business insights
BI is a set of techniques and tools used to gather data from different sources and store into a DW. (Data integration)
This data is used for various analytical purposes and making it easier for end business users to understand this data.
— In BI, to achieve data integration, there are different processes like Data modeling, data warehousing, data profiling, Data cleansing, ETL or ELT.
Among these, ETL is the most preferred effective process followed by many organizations.
 — ETL :
Extract, Transform, Load is a process of extracting data from various sources, transforming the data as per the requirements and loading into the target DW
— DW
A DW is a single, complete and consistent store of data , obtained from various sources and made available to the end users(business analysts, users etc)
 in a way they understand and use it in business context.
A DW helps business executives to organize, analyze and use the data for decision making
DW types -

1. Information Processing DW - for basic querying, basic statistical analytics, reporting using cross tabs, charts, tables etc
2. Analytical Processing DW - for OLAP data with summary data using pivots, slice& dice, pull up, pulls-down
3. Data Mining - finding hidden patterns, associations in the data, classifications and prediction of data

— Informatica is the best ETL tool for achieving data integration, for data migration.
—The ETL process utilize the SCD types (Slowly changing dimensions)
 Type 1 - Overwriting the old value.
No history maintained while updating the record
Type 2 - Creating a new additional record
Maintains history while updating by adding a new record with start_date and end_date
Type 3 - Adding a new column
Partial history maintained by just adding a new column with previous data .
Example current_value, previous_value
Type 4 - Using historical table
Show current value in dimension table but track all changes in separate table
— Data visualization
The final step after loading the DW using ETL, generate graphical representation of the data for the end users

7. Informatica Power center Client

Power center Repository Manager:

Power center Designer :

Load and join source data, cleanse the data, design the mapping based on the specifications and create a load target meta data

Source Analyzer, Mapplet Designer, Mapping Designer, Target designer

Transformations: source_qualifier, joiner, filter, router, aggregator, lookup, union, java etc

Power center Workflow Manager : Create workflows with sessions which intern includes mappings, configure session properties and execute the workflows

Task developer, workout designer, workflow designer

Power center Workflow Monitor: Monitor the execution of the workflows fix the errors and verify if the data is processed.

8. Healthcare domain terms:

Claim:

A request submitted to a health insurance company by an insurance policy holder or by the

provider to obtain the services that are covered in the health insurance policy.

Deductible:

The amount you must pay each year before your plan starts to pay for covered medical expenses.

Coinsurance: cost sharing requirement

You pay a percentage and the insurance company pays the remaining percentage for covered medical expenses after deductible is met.

You pay 20% of coinsurance until your out of pocket limit is reached for the year.

Out of pocket limit:

Copay:

The flat fee you must pay for each medical service.

Explanation of Benefits: EOB

Details about the insurance claim about for what services and what portion was paid to the providers and what portion of the payment is your responsibility.

Benefit Data: the list of healthcare services , the plan is covered and the maximum cost , it will cover.

PBM:

Pharmacy Benefit Managers are the middlemen who manage the prescription drug benefits programs. They regulate the drugs prices, and the check for the correct drugs to be given to the patients etc.,

9. Datawarehouse concepts

— Datawarehouse:

A DW is a single, complete and consistent store of data , obtained from various sources and made available to the end users(business analysts, users etc)
Note: DW is not loaded every time a new data is added to database.
— Properties of DW: subject oriented, Integrated, Time-variant, Non volatile
— OLAP vs OLTP
OLAP [Datawarehouse] : historical data, for analysing the business, read data from it, primitive detailed data, operations(rollup, drill down, pivot, slice, dice), millions of data
OLTP [Database]: current data, for running the business, write data into it, summarized data (insert, delete, commit, rollback), less data
— Datamart:
It is a smaller version of DW, deals with single subject area, limited to few sources, occupies less memory, shorter time to implement.
— DW Architecture:
Data sources ->ETL -> Staging area -> ETL -> DW [metadata, aggregate data, raw data} -> Datamarts -> End Users
— Dimension:
A collection of reference information about a measurable event.
The tables that involved dimensions are called dimension tables.
Dividing a DW project into dimensions provides structured information for analysis and reporting.
Ex: Product, Customer
— Fact:
A fact is a measure that can be summed, averaged or manipulated. Contains dimension tables info.
It has 2 kinds of data . Dimension key, measure
Every dimension table is linked to a fact table
Ex: Fact table
Product(dimension) ->productID (dimension key)
-> number of units sold (measure)
— Schemas:
A schema gives logical description of the entire database. It gives details about the keys, constraints, which keys are linked between the tables etc.,
A database uses relational model. A DW uses star, snowflake and constellation schemas.

— Star Schema:

Every dimension is represented as one dimension table and the fact table at the center contains all the dimensions keys along with measure attributes like number of units sold, revenue etc.,

— Snowflake Schema:

Every dimension are normalized and split into multiple dimension tables. The fact table at the center with all dimension keys and measures.

— Fact Constellation or Galaxy Schema:

Contains multiple fact tables sharing dimension tables. It is viewed as

collection of stars. The shared dimensions are called conformed dimensions.

10. SQL:
DDL - Data definition Language : CREATE, ALTER, DROP, TRUNCATE, COMMENT, RENAME
DML - Data Manipulation Language : INSERT, DELETE, UPDATE, LOCK, CALL, EXPLAIN PLAN
DQL - Data Query Language : SELECT
DCL - Data Control Language : GRANT, REVOKE
TCL - Transaction Control Language: COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION

https://www.tutorialspoint.com/e-f-codd-s-12-rules-for-rdbms

Database Management Systems
- DBMS Overview
- RDBMS Overview

MySQL
- History & Background
  - Developed by MySQL AB - Swedish Company
  - David Axmark, Allan Larsson, Michael Widenius
  - 1995 -> Initial Release
  - SunMicrosystem Acquired MySQL 2008
  - Oracle acquired Sun at 2010
  - LAMP / WAMP
  - Owned by Oracle
  - Forked MySQL -> MariaDB
  - 5.7.x -> RDBMS stable release
  - 8.x -> NoSQL like document store with JSON support
- Storage Engines
  - MyISAM -> Faster Read
  - InnoDB -> ACID compliant, Support Tranctions and Referential Integrity
- Installation and Config changes
- Connect using MySQL client
- Connect using MySQL Workbench
- SQL
  - DDL
    - CREATE SCHEMA
    - CREATE TABLE
    - ALTER TABLE
    - DROP TABLE
    - CONSTRAINTS - PRIMARY & FOREIGN KEY
  - DML
    - SELECT

        - WHERE
        - GROUP BY
        - HAVING
        - ORDER BY
        - JOIN
    - INSERT
    - UPDATE
    - DELETE
    - TRUNCATE
- DCL
    - GRANT
    - REVOKE

— Mysql scripts:

CREATE TABLE employee (id **int auto_increment**, name **text**, age **int**,
designation **text**, department **text**, country **text**,
salary **double**, doj **date**, created_time **datetime**, modified_time **datetime**,
PRIMARY KEY(id));

INSERT INTO employee (name, age, designation, department, country, salary, doj,
created_time) **VALUES**
('Ramu',25,'Developer','IT','India',50000,'2020-05-21','2022-12-31 01:15:00');

create table user (id **int auto_increment**, first_name **text**, last_name **text**, email
text, username **text**, **password text**, **PRIMARY KEY**(id));

INSERT INTO user(first_name, last_name, email, username, **password**) **values**
('Hari','Kumar','harikumar@gmail.com', 'harikumar','Hari123');

CREATE TABLE employee_new (id **int auto_increment**, name **text**, age **int**,
gender **text**, contractor **text**, skills **text**, designation **text**,
department **text**, address **text**, country **text**, **PRIMARY**
KEY(id));

insert into employee_new (name, age, gender, contractor, skills, designation,
department, address, country) **values**
('Raju', 23,'male','yes','java','developer','IT', 'Banglore',

'India');