### ****Lab 1:**** Overview of the features of SQL and SQL\*Plus

**What is SQL?**

* **SQL**: Stands for **Structured Query Language**.
* **Purpose**: It is a programming language used for managing and manipulating relational databases.
* **Key Functions**:
  + **Querying Data**: Retrieve specific information from databases (e.g., using the SELECT statement).
  + **Inserting Data**: Add new records to tables (e.g., using the INSERT statement).
  + **Updating Data**: Modify existing records (e.g., using the UPDATE statement).
  + **Deleting Data**: Remove records from tables (e.g., using the DELETE statement).
  + **Defining Structures**: Create and modify the structure of database tables and schemas (e.g., using CREATE and ALTER statements).
  + **Controlling Access**: Set permissions for users and roles to access and modify data.

**What is SQL\*Plus?**

* **SQL\*Plus**: A command-line tool that comes with Oracle Database.
* **Purpose**: It allows users to interact with the Oracle Database using SQL commands.
* **Features**:
  + **Interactive Interface**: Users can type SQL commands directly into the command line to execute queries or scripts.
  + **Script Execution**: Users can run SQL scripts, which are files containing multiple SQL statements.
  + **Formatting Output**: SQL\*Plus can format the output of SQL queries for better readability.
  + **Database Administration**: It provides various commands for database management and administration tasks.

**What is a Database?**

* **Need for Information**: Organizations require information to make good decisions.
  + **Example**: A library tracks books, members, due dates, and fines.
  + **Example**: A company keeps records of employees, departments, and salaries.
* **Data**: The collection of facts about various entities in an organization is called data.
* **Database**:
  + A database is an organized collection of related data.
  + To manage databases, we use Database Management Systems (DBMS).
  + **DBMS**: A program that stores, retrieves, and modifies data in the database when requested by users.

**Relational Database Concept**

* **Relational Database**:
  + Uses tables (like spreadsheets) to store data.
  + Data can be accessed and changed using Structured Query Language (SQL).
* **History**:
  + Dr. E. F. Codd created the relational model for databases in 1970.
  + Relational Database Management Systems (RDBMS) became popular due to their user-friendliness and flexibility.
* **Tables**:
  + A table is the basic structure in an RDBMS that holds all relevant data about something real (e.g., employees, invoices, customers).

**Oracle Databases**

* **Oracle 7**: A relational database management system.
* **Oracle 8 and Oracle 9i**: These are object-relational database management systems.
* **Oracle 9i**:
  + Includes everything needed to develop, deploy, and manage internet applications.
  + Contains two main products:
    - **Oracle 9i Application Server**: Runs the applications.
    - **Oracle 9i Database**: Stores all the data.

**Relating Multiple Tables**

* **Single Entity per Table**: Each table holds data about one specific thing (entity).
  + **Example**: The EMP table has information only about employees.
* **Combining Tables**: Sometimes, we need data from more than one table to answer questions.
  + **Example**: To find the department location of an employee, we need data from both the EMP table and the DEPT table.
* **Foreign Keys**: An RDBMS (Relational Database Management System) links data between tables using foreign keys.

**Guidelines for Primary and Foreign Keys**

* **Primary Key Rules**:
  + Cannot have duplicate values.
  + Generally, cannot be changed.
* **Foreign Key Rules**:
  + Based on data values; they are logical, not physical pointers.
  + Must match an existing primary key or unique key value, or it can be null.

**SQL (Structured Query Language)**

* **Query Language**: Most database systems use SQL to manage data.
  + SQL is a nonprocedural language, meaning you tell it what data you want, not how to get it.
* **High-Level Language**: SQL operates at a high level, leaving the details of data access to the DBMS.
* **Communication**: In Oracle, SQL is used to communicate with the Oracle Server from various applications.
* **Features of SQL**:
  + Besides querying, SQL can define data structure, insert or modify data, and set security constraints.
* **Advantages of SQL**:
  + Efficient and easy to learn.
  + Can fully define, retrieve, and manipulate data in tables.

**Scope of SQL**

* **Users**: SQL is used by various roles, including:
  + System administrators
  + Database administrators
  + Security administrators
  + Application programmers
  + Decision support personnel
  + Other end users

**Language Components**

* **Data Retrieval**: SQL has a query language for retrieving data.
* **Data Definition Language (DDL)**: Commands for defining and modifying database schemas, deleting relations, and creating indices.
* **Interactive Data Manipulation Language (DML)**: Commands to insert, delete, and modify data in the database.
* **Embedded DML**: SQL can be embedded in programming languages like Cobol, Pascal, and C.
* **View Definition**: Commands for creating views (virtual tables).
* **Authorization**: Commands to set access rights for data and views.
* **Integrity**: Commands to enforce rules that data must meet; updates violating these rules are not allowed.
* **Transaction Control**: Commands to manage the start and end of transactions and control data access.

This summary breaks down the concepts into simpler terms while retaining the essential information. If you need further clarification or details, let me know!

Here's how you can create the EMP table in SQL with the characteristics you've provided:

CREATE TABLE EMP (

EMPNO INT NOT NULL, -- Employee number, primary key

ENAME VARCHAR(50), -- Employee name

JOB VARCHAR(50), -- Employee job role

MGR INT, -- Manager employee number (can be NULL)

HIREDATE DATE, -- Hire date

SAL DECIMAL(10, 2), -- Salary

COMM DECIMAL(10, 2), -- Commission (can be NULL for non-salesman roles)

DEPTNO INT, -- Department number, foreign key

PRIMARY KEY (EMPNO), -- Defining EMPNO as the primary key

FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO) -- Defining DEPTNO as a foreign key referencing the DEPT table

);

**Explanation:**

* EMPNO: This is the primary key of the EMP table and uniquely identifies each employee. The NOT NULL constraint ensures that every employee must have a unique EMPNO.
* ENAME: Stores the employee's name. We assume a maximum length of 50 characters for names.
* JOB: Stores the job title of the employee.
* MGR: Represents the manager's employee number (optional, so it can be NULL).
* HIREDATE: Stores the date when the employee was hired.
* SAL: Stores the employee's salary as a decimal number, with up to 10 digits and 2 decimal places.
* COMM: Stores the commission earned by employees (only non-null for salesmen). It is a decimal field, but can be NULL for employees without a commission.
* DEPTNO: A foreign key referring to the DEPTNO in the DEPT table. This assumes the existence of a DEPT table with a primary key DEPTNO.

This table structure supports the primary key (EMPNO), foreign key (DEPTNO), and allows NULL values in fields like COMM and MGR.

**Example SQL Script to Populate the EMP Table:**

INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)

VALUES

(1001, 'John Smith', 'Manager', NULL, '2020-01-15', 7500.00, NULL, 10),

(1002, 'Jane Doe', 'Salesman', 1001, '2021-06-23', 4500.00, 800.00, 20),

(1003, 'Robert Brown', 'Clerk', 1002, '2019-03-19', 3200.00, NULL, 30),

(1004, 'Alice Green', 'Analyst', 1001, '2022-08-12', 6000.00, NULL, 20),

(1005, 'David Wilson', 'Salesman', 1002, '2018-10-05', 4800.00, 500.00, 30),

(1006, 'Emily White', 'Manager', NULL, '2023-02-28', 8200.00, NULL, 40),

(1007, 'Daniel Clark', 'Clerk', 1006, '2020-11-18', 3100.00, NULL, 40),

(1008, 'Sophia Hill', 'Analyst', 1006, '2021-09-30', 5700.00, NULL, 20),

(1009, 'Michael King', 'Salesman', 1001, '2019-12-22', 5100.00, 700.00, 10),

(1010, 'Olivia Adams', 'Clerk', 1009, '2017-05-10', 2900.00, NULL, 20),

(1011, 'Lucas Baker', 'Analyst', 1004, '2020-04-02', 6400.00, NULL, 30),

(1012, 'Emma Scott', 'Manager', NULL, '2016-07-27', 9100.00, NULL, 40),

(1013, 'James Evans', 'Salesman', 1012, '2022-10-15', 4700.00, 600.00, 30),

(1014, 'Charlotte Turner', 'Clerk', 1004, '2021-12-05', 3000.00, NULL, 20);

**Explanation:**

* **EMPNO**: Unique employee number.
* **ENAME**: Random employee names.
* **JOB**: Jobs like 'Manager', 'Salesman', 'Clerk', 'Analyst', etc.
* **MGR**: Manager employee number (NULL for employees with no manager).
* **HIREDATE**: Random hire dates (in YYYY-MM-DD format).
* **SAL**: Salary in decimal format (e.g., 7500.00).
* **COMM**: Commission (NULL for non-salesman roles).
* **DEPTNO**: Department numbers, which are foreign keys referencing the DEPT table (make sure valid department numbers exist in the DEPT table).

You can tweak the values or add more random data if needed. This script inserts 14 entries into your EMP table with various employee details.

**SQL Commands for SQL**

Here are the commands again with minor adjustments for clarity and best practices specific to SQL:

-- Create DEPT Table

CREATE TABLE DEPT (

DEPTNO INT NOT NULL, -- Department number, primary key

DNAME VARCHAR(50) NOT NULL, -- Department name

LOC VARCHAR(50), -- Location

PRIMARY KEY (DEPTNO) -- Defining DEPTNO as the primary key

) ENGINE=InnoDB; -- Specify InnoDB as the storage engine

-- Create SALGRADE Table

CREATE TABLE SALGRADE (

GRADE INT NOT NULL, -- Salary grade, primary key

LOSAL DECIMAL(10, 2) NOT NULL, -- Low salary limit

HISAL DECIMAL(10, 2) NOT NULL, -- High salary limit

PRIMARY KEY (GRADE) -- Defining GRADE as the primary key

) ENGINE=InnoDB; -- Specify InnoDB as the storage engine

-- Create JOB\_HISTORY Table

CREATE TABLE JOB\_HISTORY (

EMPNO INT NOT NULL, -- Employee number, foreign key

JOB VARCHAR(50) NOT NULL, -- Job title

START\_DATE DATE NOT NULL, -- Start date of the job

END\_DATE DATE, -- End date of the job (can be NULL for current jobs)

PRIMARY KEY (EMPNO, START\_DATE), -- Composite primary key

FOREIGN KEY (EMPNO) REFERENCES EMP(EMPNO) -- Defining EMPNO as a foreign key referencing EMP table

) ENGINE=InnoDB; -- Specify InnoDB as the storage engine

**Key Considerations for SQL:**

1. **Storage Engine**: Specifying ENGINE=InnoDB ensures that you are using a transactional storage engine, which supports foreign keys and transactions. This is a best practice for relational databases in SQL.
2. **NULL Constraints**: Make sure to define NOT NULL where necessary, especially for foreign keys and essential columns.
3. **Data Types**: The data types used (like INT, VARCHAR, DECIMAL, and DATE) are compatible with SQL.
4. **Foreign Key Constraints**: Ensure that the EMP table is created before the JOB\_HISTORY table to avoid foreign key constraint errors.

**Execution Steps:**

* You can execute these SQL commands in a SQL client (like SQL Workbench, phpMyAdmin, or directly in a SQL command-line interface).
* After creating these tables, you can proceed to insert data as previously discussed.

If you need any specific examples or further modifications, feel free to ask!

**Basic Data Retrieval in SQL**

The basic structure of an SQL query in SQL consists of three main clauses: SELECT, FROM, and WHERE.

SELECT \* | {[DISTINCT] column | expression [alias], …}

FROM table

[WHERE condition];

*Note: The keywords and clauses enclosed in square brackets are optional.*

**Examples:**

i. **Selecting all columns from a table**

SELECT \*

FROM DEPT;

ii. **To select distinct job titles in the EMP table**

sql

SELECT DISTINCT JOB

FROM EMP;

*Note: The DISTINCT clause suppresses duplicate values.*

iii. **To select all employees whose salary is greater than 2200**

sql

SELECT \*

FROM EMP

WHERE SAL > 2200;

iv. **To display the name and department number of employees who were hired before May 12, 1981**

sql

SELECT ENAME, DEPTNO

FROM EMP

WHERE HIREDATE < '1981-05-12';

v. **To display the name and job of employees using literal character strings and the concatenation function**

sql

SELECT CONCAT(ENAME, ' is a ', JOB) AS "Employee Details"

FROM EMP;

vi. **Retrieving data from multiple tables: To select employee name, job, and department name**

sql

SELECT E.ENAME, E.JOB, D.DNAME

FROM EMP E

JOIN DEPT D ON E.DEPTNO = D.DEPTNO;

**Key Adjustments for SQL:**

1. **Date format:** SQL uses YYYY-MM-DD for date literals (e.g., '1981-05-12').
2. **String concatenation:** SQL uses the CONCAT() function instead of ||.
3. **JOIN syntax:** While older implicit join syntax (using commas in the FROM clause) is still supported, using JOIN ... ON is the recommended method for clarity and maintainability.

**SQL Commands Overview**

**1. DESCRIBE: To display the structure of a table**

sql

DESCRIBE EMP;

*Example:*

sql

DESC EMP;

*Note: DESCRIBE or DESC shows the columns, their data types, and other table structure information.*

**2. Saving and Retrieving SQL Files**

**Saving SQL Commands**

* In SQL, the SQL commands are typically written and saved in an external file (e.g., .sql files) using a text editor.
* To run SQL scripts, you can use SQL from the command line or an IDE like SQL Workbench.

**Running SQL Commands from a File**

bash

SQL> SOURCE /path/to/filename.sql;

*Example:*

bash

SQL> SOURCE D:/DATA/FINDSAL.sql;

*Note: The SOURCE command reads and executes the contents of a file. The default file extension for SQL scripts is .sql.*

**3. Spooling Output**

In SQL, there is no direct equivalent of the SPOOL command. However, you can redirect query results to a file using the command line:

bash

SQL -u username -p -e "SELECT \* FROM EMP" > outputfile.txt

*This writes the query results to outputfile.txt. Alternatively, output can be redirected in SQL Workbench or any other SQL client tool with export options.*

**4. Editing SQL Files**

* SQL does not have an internal editor like SQL\*Plus. Instead, you can use any external text editor (e.g., Notepad, VS Code) to edit SQL scripts and then run them using the SOURCE command.

**5. EXIT: Leaves SQL**

sql

EXIT;

*Use EXIT or QUIT to leave the SQL command line interface.*

**Summary of Key SQL Differences:**

* **DESCRIBE:** Works the same as in SQL\*Plus to show the structure of a table.
* **SAVE/GET Commands:** SQL uses external editors for writing scripts, and the SOURCE command to execute them.
* **SPOOL:** No direct equivalent in SQL, but query results can be redirected to files using command-line options.
* EDIT: Use external editors to edit .sql files.