**LAB MANUAL**

**of**

**Database Management System Laboratory**

**(CS238)**

**Bachelor of Technology (CSE)**

**By**

**SAHIL PANDYA (21124097)**

**Second Year, Semester 4**

**Course In-charge: Prof. Sushma Vankhede**

**Department of Computer Science and Engineering**

**School Engineering and Technology**

**Navrachana University, Vadodara**

**Spring Semester**

**(2022-2023)**

**TABLE OF CONTENT**

**1.** [**SQL PRACTICAL 1**](#_b70cy275e989) **3**

**2.** [**SQL PRACTICAL 2**](#_fhgbo4jkjbg8) **14**

**3.** [**SQL PRACTICAL 3**](#_15kqnweb8hqf) **25**

**4.** [**SQL PRACTICAL 4(JOINS)**](#_g426bj3zmvm9) **33**

**5.** [**SQL PRACTICAL 5**](#_rmxp1oeaqkwy) **35**

**6.** [**SQL PRACTICAL 6**](#_6s5s2tk5tzvf) **40**

**7.** [**SQL PRACTICAL 7**](#_pru541ysxqem) **50**

**8.** [**SQL PRACTICAL 8**](#_lfckqhkq0eqk) **58**

**9.** [**SQL PRACTICAL 9**](#_h9g2avg7rrqq) **66**

**10.** [**SQL PRACTICAL 10(PL/SQL)**](#_g6yvzxldqa9) **71**



# SQL PRACTICAL 1



**Aim: study DDL-create and DML-insert commands**

1. Create tables according to the following definition.
   * CREATE TABLE DEPOSIT (ACTNO VARCHAR2(5), CNAME VARCHAR2(18) , BNAME VARCHAR2(18), AMOUNT NUMBER (8,2), ADATE DATE);
   * CREATE TABLE BRANCH (BNAME VARCHAR2(18), CITY VARCHAR2(18));
   * CREATE TABLE CUSTOMERS (CNAME VARCHAR2 (19), CITY VARCHAR2(18));
   * CREATE TABLE BORROW (LOANNO VARCHAR2(5), CNAME VARCHAR2(18), BNAME VARCHAR2(18), AMOUNT NUMBER (8,2));
2. Insert the data as shown below: DEPOSIT

| **DEPOSIT:**  **ACTNO** | **CNAME** | **BNAME** | **AMOUNT** | **ADATE** |
| --- | --- | --- | --- | --- |
| 100 | ANIL | VRCE | 10000.00 | 1-MAR-2015 |
| 101 | SUNIL | AJNI | 50000.00 | 4-JAN-2016 |
| 102 | MEHUL | KAROLBAGH | 35000.00 | 17-NOV-2018 |
| 104 | MADHURI | CHANDI | 12000.00 | 17-DEC-2018 |
| 105 | PRAMOD | M.G.ROAD | 30000.00 | 27-MAR-2020 |
| 106 | NEHA | BKC | 875.00 | 18-FEB-2016 |
| 107 | SANDIP | ANDHERI | 20000.00 | 31-MAR-2021 |
| 108 | SHIVANI | VIRAR | 10000.00 | 5-SEP-2022 |
| 109 | KRANTI | NEHRU PLACE | 50000.00 | 2-JUL-2021 |
| 110 | SHREYA | DADAR | 500.00 | 17-JUL-2017 |
| 111 | MINU | POWAI | 70000.00 | 10-AUG-2019 |

**BRANCH**

| **BRANCH:**  **BNAME** | **CITY** |
| --- | --- |
| VRCE | NAGPUR |
| AJNI | NAGPUR |
| KAROLBAGH | DELHI |
| CHANDI | DELHI |
| DHARAMPETH | NAGPUR |
| M.G.ROAD | BANGLORE |
| ANDHERI | MUMBAI |
| VIRAR | MUMBAI |
| NEHRU PLACE | DELHI |

| POWAI | MUMBAI |
| --- | --- |
| BKC | MUMBAI |
| DADAR | MUMBAI |

**CUSTOMERS**

| **CUSTOMERS: CNAME** | **CITY** |
| --- | --- |
| ANIL | CALCUTTA |
| SUNIL | DELHI |
| MEHUL | BARODA |
| MANDAR | PATNA |
| MADHURI | NAGPUR |
| PRAMOD | NAGPUR |
| SANDIP | SURAT |
| SHIVANI | MUMBAI |
| KRANTI | MUMBAI |
| NAREN | MUMBAI |

**BORROW**

| **BORROW: LOANNO** | **CNAME** | **BNAME** | **AMOUNT** |
| --- | --- | --- | --- |
| 201 | ANIL | VRCE | 10000.00 |
| 206 | MEHUL | AJNI | 50000.00 |
| 311 | SUNIL | DHARAMPETH | 30000.00 |
| 321 | MADHURI | ANDHERI | 20000.00 |
| 375 | PRAMOD | VIRAR | 80000.00 |
| 481 | KRANTI | NEHRU PLACE | 30000.00 |

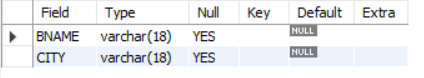
**From the above given tables perform the following queries:**

1. **Display branch, borrow, customer.**

**OUTPUT**:

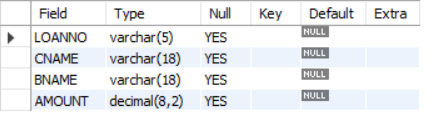
use mydb;

describe BRANCH;



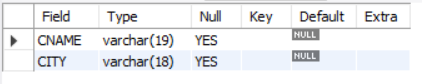
use mydb;

describe BORROW;



use mydb;

describe CUSTOMER;

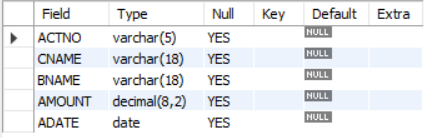


1. **Describe deposit.**

OUTPUT:

use mydb;

describe deposit;

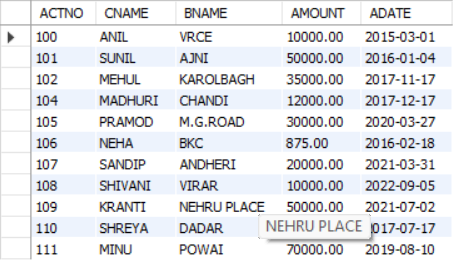


1. **Display all data from table DEPOSIT**.

**OUTPUT:**

use mydb;

select \* from deposit;

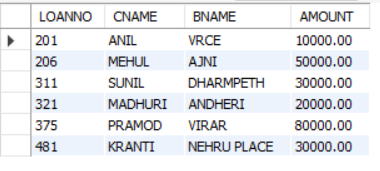


1. **Display all data from table BORROW.**

**OUTPUT:**

use mydb;

select \* from borrow;

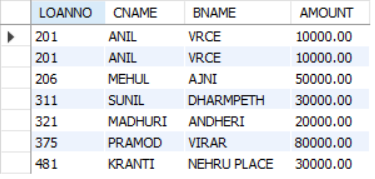


1. **Display all data from table CUSTOMERS.**

**OUTPUT:**

use mydb;

select \* from customer;

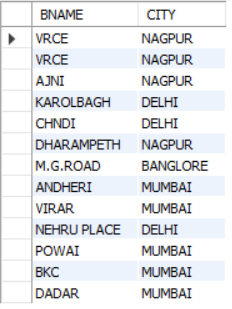


1. **Display all data from table BRANCH.**

**OUTPUT:**

use mydb;

select \* from branch;

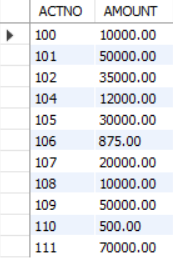


1. **Give account no and amount of depositors.**

**OUTPUT:**

use mydb;

select ACTNO, AMOUNT from deposit;

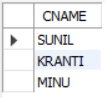


1. **Give name of depositors having amount greater than 40000.**

**OUTPUT:**

use mydb;

select CNAME from deposit WHERE AMOUNT>40000;



1. **Give name of customers who opened account after date '1-12-2015'.**

**OUTPUT:**

use mydb;

select CNAME FROM DEPOSIT WHERE ADATE > 2015-12-1;



1. **Write a query to get the records from deposit tables in which branch of the employee is VIRAR and amount is 10000.**

**OUTPUT:**

use mydb;

select \* FROM DEPOSIT WHERE AMOUNT=10000 AND BNAME='VIRAR';



1. **Write a query to get the records from BRANCH tables in which CITY of the employee is BOMBAY and branch name is POWAI.**

**OUTPUT**:

use mydb;

select \* FROM BRANCH WHERE CITY='MUMBAI' AND BNAME='POWAI';



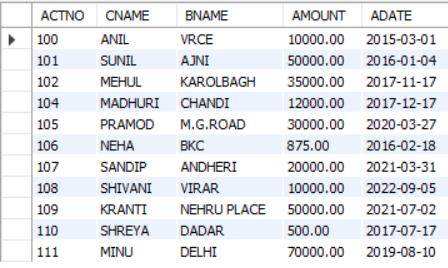
1. **SQL "AND" example with "UPDATE" statement**

**Write a query to update the records in DEPOSIT tables in which BNAME of the employee is POWAI, and the first name is MINU. For that employee, set the updated value of the location as Delhi.**

**OUTPUT:**

USE mydb;

UPDATE DEPOSIT SET BNAME = 'DELHI' WHERE CNAME = 'MINU' AND BNAME = 'POWAI';

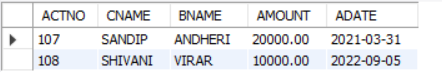


1. **Write a query to get the records from deposit tables in which name of the employee is Sandip or location is VIRAR.**

**OUTPUT:**

use mydb;

SELECT \* FROM DEPOSIT WHERE CNAME = 'SANDIP' OR BNAME = 'VIRAR';



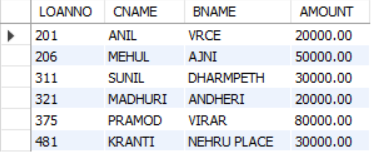
1. **SQL "OR" example with SQL UPDATE**

**Write a query to update the records in tables in BORROW in which name of the employee is anil, or the branch name is VRCE. For that employee, set the updated value of the amount as 20000.**

**OUTPUT:**

USE mydb;

UPDATE BORROW SET AMOUNT = 20000 WHERE CNAME = 'ANIL' AND BNAME='VRCE';

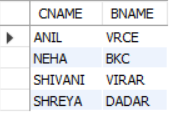


1. **Display name and branch name of all customers whose balance is less than equal to 10000.**

**OUTPUT:**

USE mydb;

SELECT CNAME,BNAME FROM DEPOSIT WHERE AMOUNT <= 10000;



1. **Display customers whose branch is either VIRAR or POWAI, and amount less than 60000.**

**OUTPUT:**

USE mydb;

SELECT \* FROM DEPOSIT WHERE AMOUNT<60000 AND BNAME='VIRAR' OR BNAME='POWAI';

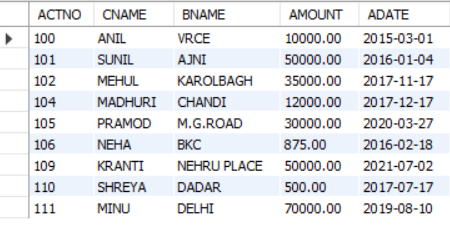


1. **Display the details of all the Customers who does not have account in the Virar, Andheri and Powai branch.**

**OUTPUT:**

USE mydb;

SELECT \* FROM DEPOSIT WHERE BNAME!='VIRAR' AND BNAME!='ANDHERI' AND BNAME!='POWAI';



1. **Display the details of all the customers who created account from 1-JAN-2020 to 31- DEC-2020.**

**OUTPUT:**

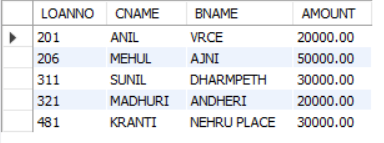
SELECT \* FROM deposit where ADATE>"2020-01-01" and ADATE<'2020-12-31';



1. **Display the details of all the customers who borrowed loan more than 19999 and less than 50001.**

**OUTPUT:**

SELECT \* FROM BORROW where AMOUNT>19999 and AMOUNT<50001;



1. **Display the details of all the employees whose name starts with letter ‘A’.**

**OUTPUT:**

SELECT \* from deposit where CNAME like 'A%';



**CONCLUSION**

The provided tasks involve creating and querying tables in a database using SQL commands. The tasks cover various fundamental concepts in database management, including table creation, data retrieval, manipulation, and analysis. By completing these tasks, individuals can gain practical experience and develop skills in database design, data organization, and SQL query writing. Overall, a thorough understanding of these concepts and commands is crucial for effective database management and optimization of database performance.



## **SQL PRACTICAL 2**



Following tables are used for queries given:

# EMPLOYEE



# DEPARTMENT

| **DEPTNO** | **DNAME** | **LOC** |
| --- | --- | --- |
| 10 | ACCOUNTING | NEW YORK |
| 20 | RESEARCH | DALLAS |
| 30 | SALES | CHICAGO |
| 40 | OPERATIONS | BOSTON |

# 

# 

# SALGRADE

| **GRADE** | **LOSAL** | **HISAL** |
| --- | --- | --- |
| 1 | 700 | 1200 |
| 2 | 1201 | 1400 |
| 3 | 1401 | 2000 |
| 4 | 2001 | 3000 |
| 5 | 3001 | 9999 |

**Questions:**

**1. Create table Employee, Department and salgrade.**

**Query and Output**

create table EMPLOYEE (EMPNO int, ENAME varchar(64), JOB varchar(64), MGR int, HIREDATE date, SAL int, COMM int, DEPTNO int);

create table DEPARTMENT (DEPTNO int, DNAME varchar(64), LOC varchar(64));

create table SALGRADE (GRADE int, LOSAL int, HISAL int);

**Insert DATA**

insert into EMPLOYEE values (7369, "SMITH", "CLERK", 7902, "1980-12-17", 800, NULL, 20);

insert into EMPLOYEE values (7499, "ALLEN", "SALESMAN", 7698, "1981-02-20", 1600, 300, 30);

insert into EMPLOYEE values (7521, "WARD", "SALESMAN", 7698, "1981-02-22", 1250, 500, 30);

insert into EMPLOYEE values (7566, "JONES", "MANAGER", 7839, "1981-04-02", 2975, NULL, 20);

insert into EMPLOYEE values (7654, "MARTIN", "SALESMAN", 7698, "1981-09-28", 1250, 1400, 30);

insert into EMPLOYEE values (7698, "BLAKE", "MANAGER", 7839, "1981-05-01", 2850, NULL, 30);

insert into EMPLOYEE values (7782, "CLARK", "MANAGER", 7839, "1981-06-09", 2450, NULL, 10);

insert into EMPLOYEE values (7788, "SCOTT", "ANALYST", 7566, "1982-12-09", 3000, NULL, 20);

insert into EMPLOYEE values (7839, "KING", "PRESIDENT", NULL, "1981-11-17", 5000, NULL, 10);

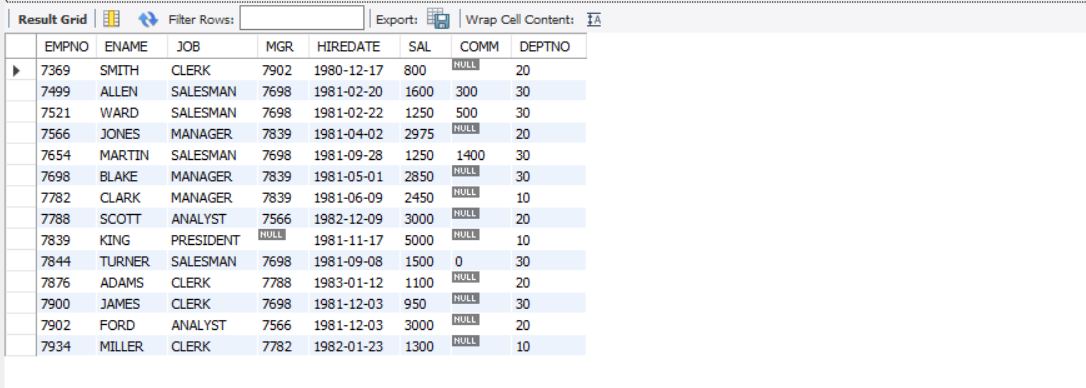
insert into EMPLOYEE values (7844, "TURNER", "SALESMAN", 7698, "1981-09-08", 1500, 0, 30);

insert into EMPLOYEE values (7876, "ADAMS", "CLERK", 7788, "1983-01-12", 1100, NULL, 20);

insert into EMPLOYEE values (7900, "JAMES", "CLERK", 7698, "1981-12-03", 950, NULL, 30);

insert into EMPLOYEE values (7902, "FORD", "ANALYST", 7566, "1981-12-03", 3000, NULL, 20);

insert into EMPLOYEE values (7934, "MILLER", "CLERK", 7782, "1982-01-23", 1300, NULL, 10);



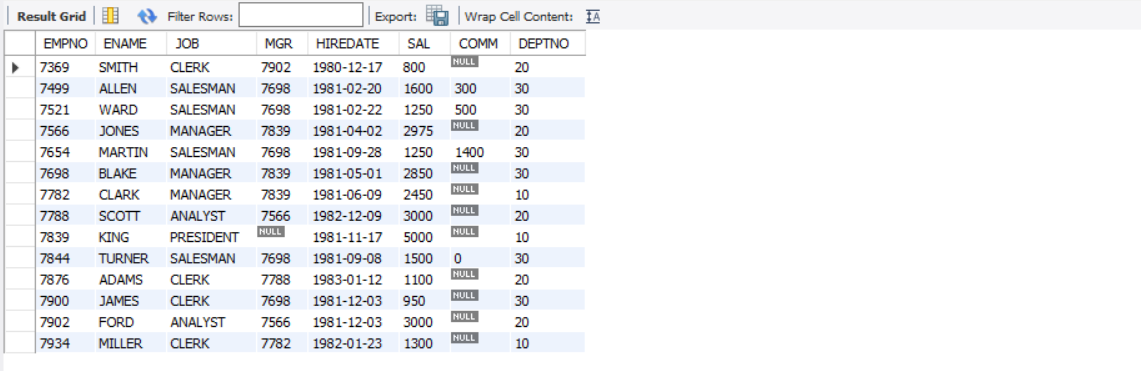
**INSERT DATA**

insert into DEPARTMENT values (10, "ACCOUNTING", "NEW YORK");

insert into DEPARTMENT values (20, "RESEARCH", "DALLAS");

insert into DEPARTMENT values (30, "SALES", "CHICAGO");

insert into DEPARTMENT values (40, "OPERATIONS", "BOSTON");



**INSERT DATA**

insert into SALGRADE values (1, 700, 1200);

insert into SALGRADE values (2, 1201, 1400);

insert into SALGRADE values (3, 1401, 2000);

insert into SALGRADE values (4, 2001, 3000);

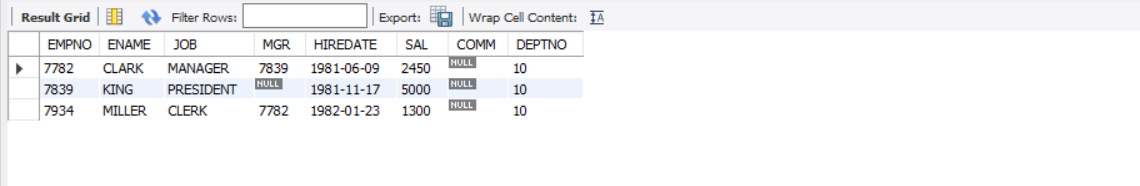
insert into SALGRADE values (5, 3001, 9999);



**2. Display all the records in emp table where employee belongs to deptno 10?**

**Query and Output**

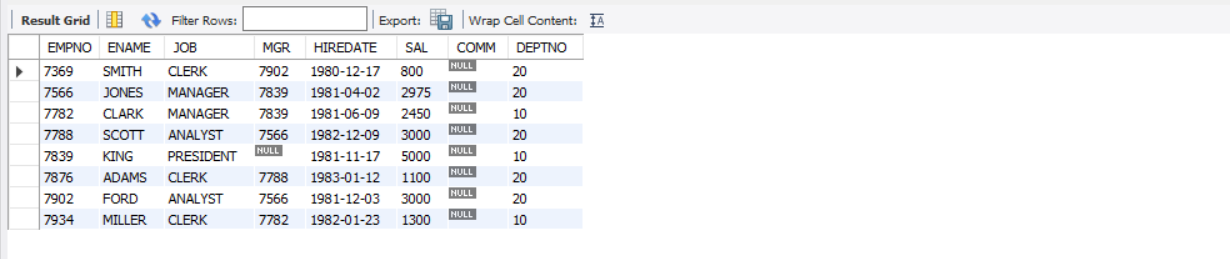
select \* from employee where deptno=10;



**3. Display all the records in emp table where employee does not belong to deptno 30?**

**Query and Output**

select \* from employee where deptno!=30;



**4. Display total number of records in Emp table?**

**Query and Output**

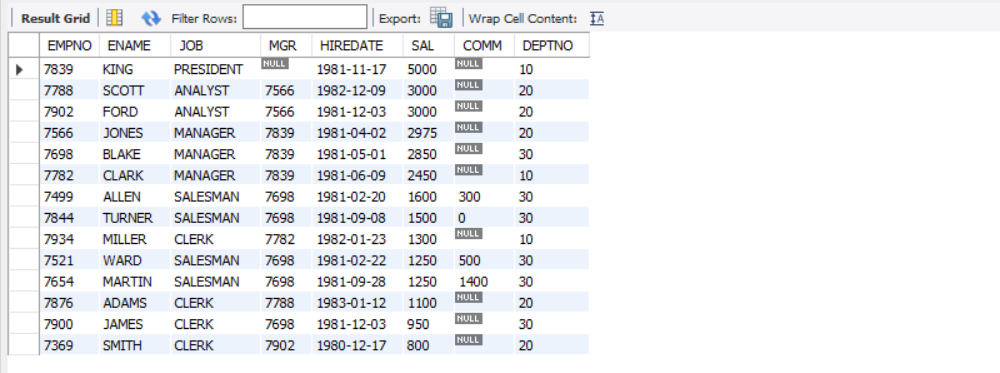
select count(EMPNO) from employee;



**5. Display emp table with salary descending order?**

**Query and Output**

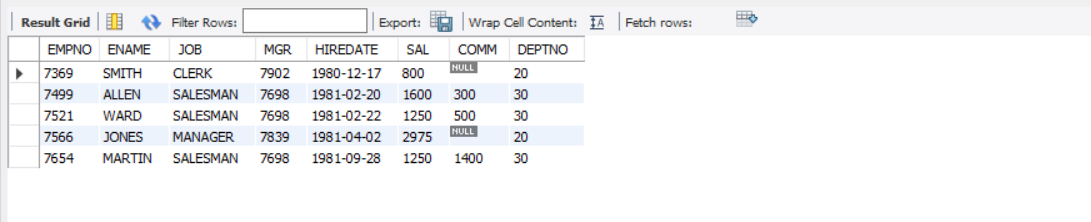
select \* from employee order by sal desc;



**6. Display first five records in employee table?**

**Query and Output**

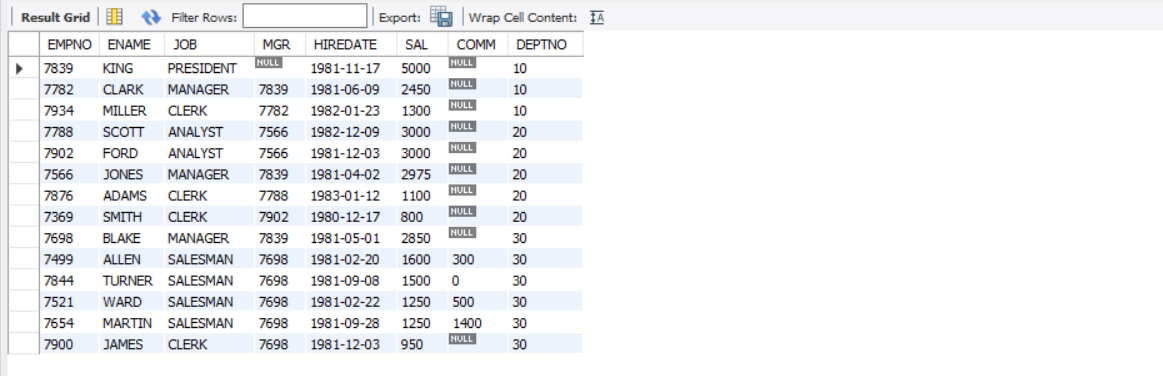
select \* from employee limit 5;



**7. Display all the records in emp table order by ascending deptno, descending salary?**

**Query and Output**

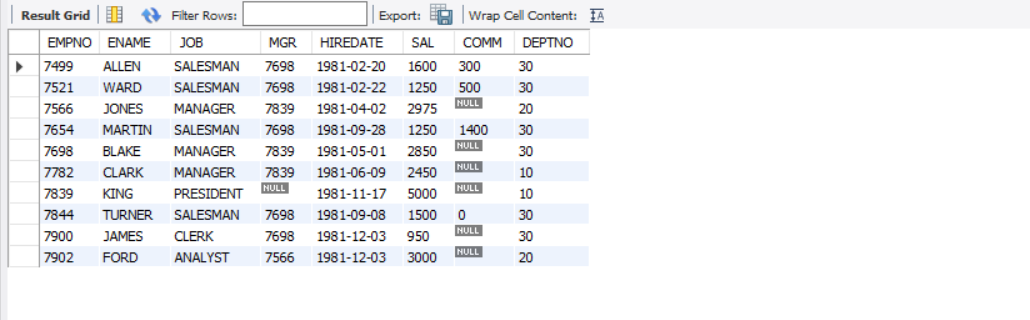
select \* from employee order by deptno, sal desc;



**8. Display all employees those who were joined in year 1981?**

**Query and Output**

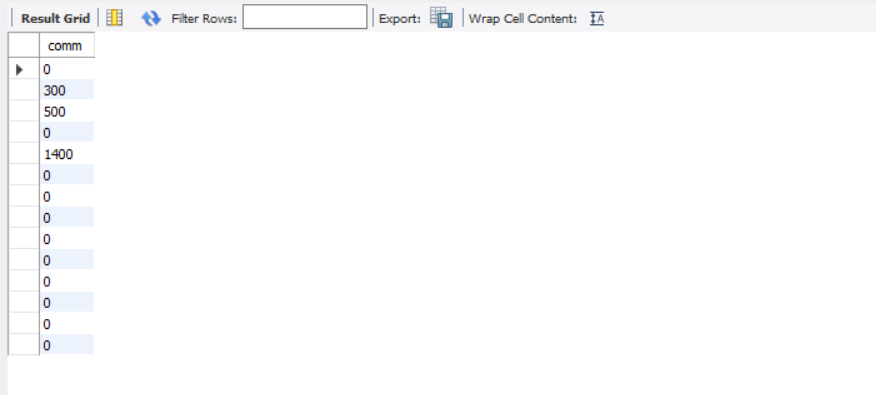
select \* from employee where year(hiredate)=1981;



**9. Display COMM in emp table. Display zero in place of null.**

**Query and Output**

select COALESCE(comm, 0) as comm from employee

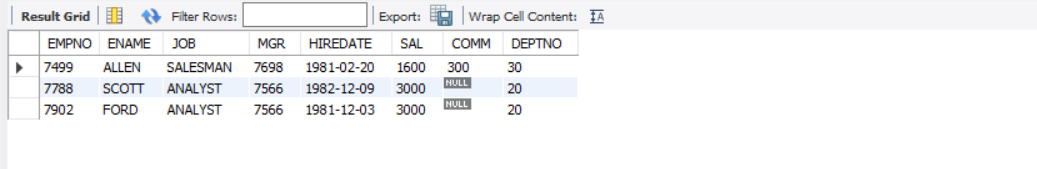


**10) Display the records in emp table where MGR in 7698,7566 and sal should be**

**greater then 1500.**

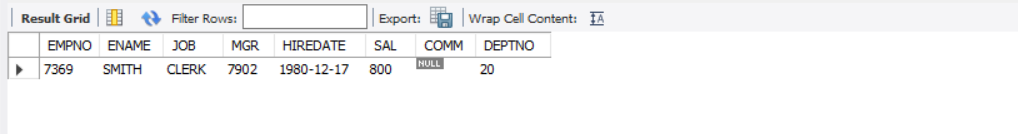
**Query and Output**

select \* from employee where ((mgr=7698 or mgr=7566) and sal>1500);



**11) Display all employees where employees hired before 01-JAN-1981**

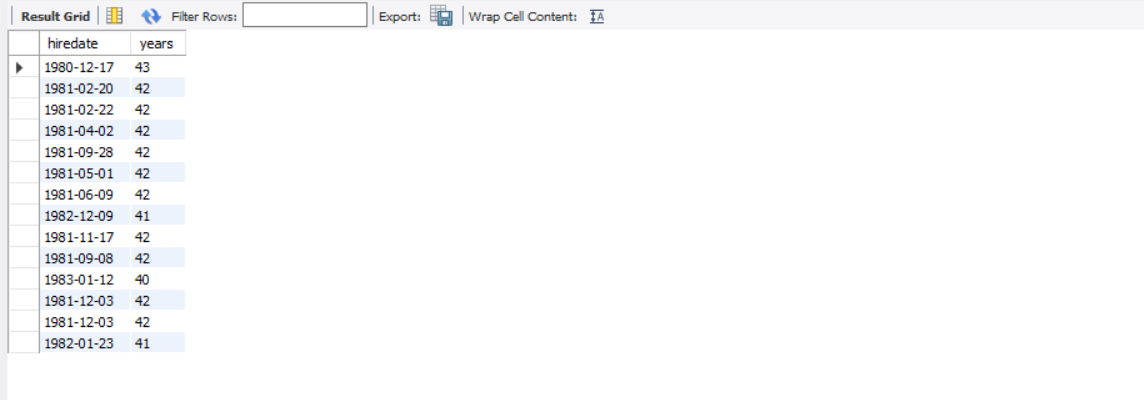
**select \* from employee where hiredate<"1981-01-01";**



**12) Display all employees with how many years they have been servicing in the company?**

**Query and Output**

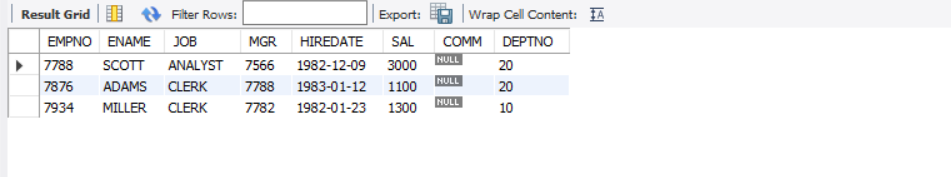
select hiredate, (year(date(sysdate()))-year(hiredate)) as years from employee;



**13) Display all employees those were not joined in 1981?**

**Query and Output**

select \* from employee where year(hiredate)>"1981";



**14. Display all employees where their hiredate belongs to third quarter?**

**Query and Output**

USE mydb;

select \* from EMPLOYEE where quarter(hiredate)=3;

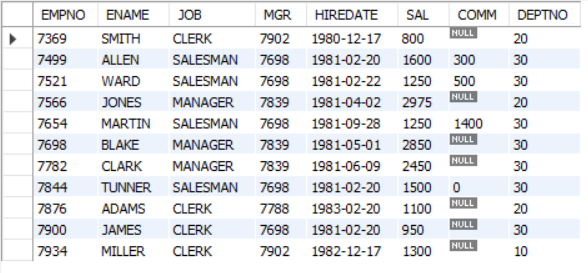


**15. Display all employees where their salary is less then the Ford’s salary?**

**Query and Output**

USE mydb;

select \* from employee where sal <(select sal from employee where ename='FORD');



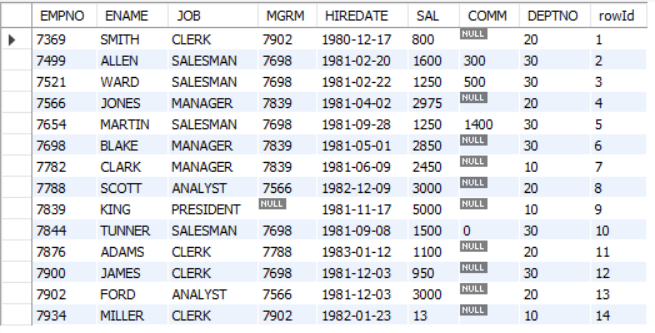
**16. Display all the records in EMP table along with the rowid?**

**Query and Output**

USE mydb;

SELECT \*, ROW\_NUMBER() OVER (ORDER BY empNO)

as rowId FROM EMPloyee;

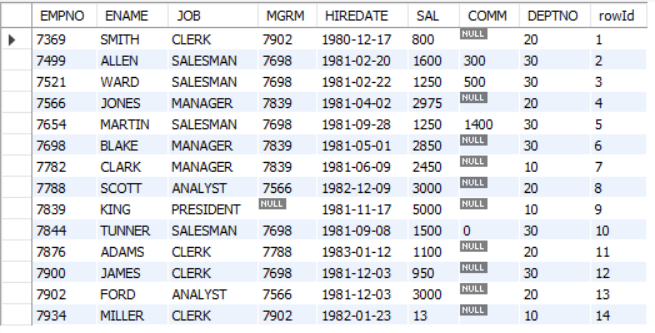


**17. Display all records in EMP table those were joined before SCOTT joined?**

**Query and Output**

USE mydb;

select \* from employee where hiredate <(select hiredate from employee where ename='SCOTT');

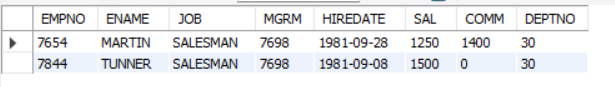


**18. Display all employees those who were joined in third quarter of 1981?**

**Query and Output**

USE mydb;

select \* from EMPLOYEE where quarter(hiredate)=3 and year(hiredate) = '1981';

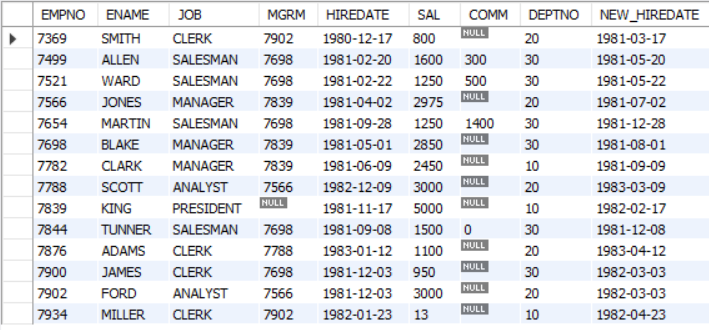


**19. Add 3 months with hiredate in EMP table and display the result?**

**Query and Output**

SELECT \*, DATE\_ADD(hiredate, INTERVAL 3 MONTH) as NEW\_HIREDATE

FROM EMPLOYEE;



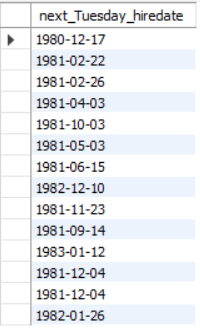
**20. Display the date for next TUESDAY in hiredate column?**

**Query and Output**

SELECT DATE\_ADD(date(hiredate),

INTERVAL (7 + (WEEKDAY(DATE(hiredate)) - 2)) % 7 DAY)

AS next\_Tuesday\_hiredate from EMPLOYEE;



**CONCLUSION**

understanding the concepts of employee, department, and salary grade tables, as well as the SELECT statement and its variations, is essential for anyone working with databases. These tables provide the foundation for organizing and managing employee data, while the SELECT statement allows for the retrieval and manipulation of this data. By using the SELECT statement to filter and sort employees based on criteria such as hire date or salary grade, individuals can gain valuable insights into employee performance and organizational trends.



## SQL PRACTICAL 3



**AIM: To perform various data manipulation commands & aggregate functions on all created tables.**

**Theory:**

**What is Function:**

SQL has many built-in functions for performing calculations on data.

**SQL Aggregate Function.**

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

* AVG() - Returns the average value
* COUNT() - Returns the number of rows
* FIRST() - Returns the first value
* LAST() - Returns the last value
* MAX() - Returns the largest value
* MIN() - Returns the smallest value
* SUM() - Returns the sum

**PROGRAM EXECUTION:**

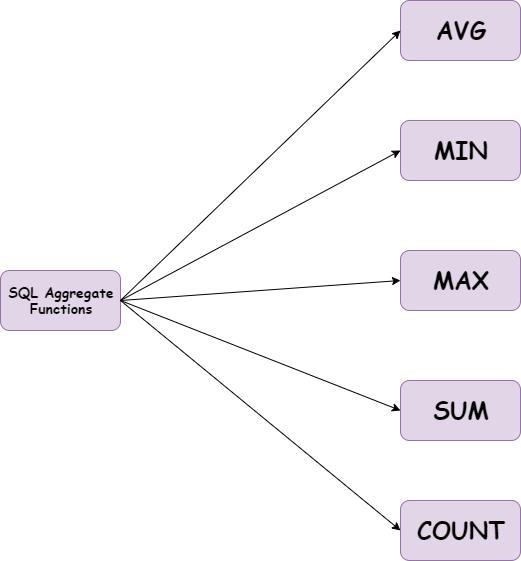
**1. Create a following table: sales\_master**

| **Col\_name** | **Datatype** | **Size** | **Attribute** |
| --- | --- | --- | --- |
|  |  |  |  |
| Salesman\_no | varchar | 6 | primary key |
|  |  |  |  |
| Sal\_name | varchar | 20 | Not null |
|  |  |  |  |
| City | varchar | 40 | Not null |
|  |  |  |  |
| pincode | int | 6,2 | not null, cannot be 0 |
|  |  |  |  |
| Sal\_amt | float | 6,2 | not null, cannot be 0 |
|  |  |  |  |
| target\_to\_get | float | 6,2 | not null, cannot be 0 |



1. **Create a following table: sales\_order**

| **Col\_name** | **Datatype** | **Size** | **Attribute** |
| --- | --- | --- | --- |
| S\_order\_no | varchar | 6 | primary key |
| S\_rder\_date | date |  | Not null |
| Salesman\_no | Int | 40 | Foreign Key |
| Order\_status | varchar | 30 | Status like in-process, |
|  |  |  | fulfilled, cancelled |
| Dely\_type | varchar | 20 | Part, full, default-pending |
| S\_order\_qunity | int |  | cannot be less than 2 |



**TABLE QUERY**

create table sales\_master (

salesman\_no varchar(6) primary key,

sal\_name varchar(20) not null,

city varchar(40) not null,

pincode int not null,

sal\_amt float not null,

target\_to\_get float not null,

CHECK (pincode>0),

CHECK (sal\_amt!=0),

CHECK (target\_to\_get!=0)

);

create table sales\_order (

s\_order\_no varchar(6) primary key,

s\_order\_date date not null,

salesman\_no varchar(6),

order\_status varchar(30),

dely\_type varchar(20) default "pending",

s\_order\_quantity int,

CHECK (order\_status="in-process" or order\_status="fulfilled" or order\_status="cancelled"),

CHECK (dely\_type="part" or dely\_type="full" or dely\_type="pending"),

CHECK (s\_order\_quantity>2),

CONSTRAINT f\_k foreign key (salesman\_no) REFERENCES sales\_master(salesman\_no)

);

insert into sales\_master values ("S101", "NAME01", "AHMEDABAD", 300001, 2000, 4000);

insert into sales\_master values ("S102", "NAME02", "BANGLORE", 300002, 2200, 4400);

insert into sales\_master values ("S103", "NAME03", "CALCUTTA", 300003, 2400, 4800);

insert into sales\_master values ("S104", "NAME04", "DELHI", 300004, 2600, 5200);

insert into sales\_master values ("S105", "NAME05", "ELLORA", 300005, 2800, 5600);

insert into sales\_master values ("S106", "NAME06", "FARIDABAD", 300006, 3000, 6000);

insert into sales\_master values ("S107", "NAME07", "GWALIOR", 300007, 3200, 6400);

insert into sales\_master values ("S108", "NAME08", "HYDERABAD", 300008, 3400, 6800);

insert into sales\_master values ("S109", "NAME09", "INDORE", 300009, 3600, 7200);

insert into sales\_master values ("S110", "NAME10", "JAIPUR", 300010, 3800, 7600);

insert into sales\_master values ("S111", "NAME11", "KOTA", 300011, 4000, 8000);

insert into sales\_master values ("S112", "NAME12", "LUCKNOW", 300012, 4200, 8400);

insert into sales\_master values ("S113", "NAME13", "MUMBAI", 300013, 4400, 8800);

insert into sales\_master values ("S114", "NAME14", "NIZAMABAD", 300014, 4600, 9200);

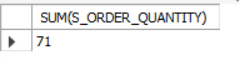
insert into sales\_master values ("S115", "NAME15", "OOTY", 300015, 4800, 9600);

1. **List total number of sales\_order from sales\_order table.**

**OUTPUT:**

USE mydb;

SELECT SUM(S\_ORDER\_QUANTITY) FROM SALES\_ORDER;



1. **List total sale amount from Mumbai City.**

**OUTPUT:**

USE mydb;

SELECT SUM(SAL\_AMT) FROM SALES\_MASTER WHERE CITY='MUMBAI';



1. **Give maximum order quantity from Salesman number ‘ S101’.**

**OUTPUT:**

USE mydb;

SELECT MAX(S\_ORDER\_QUANTITY) FROM SALES\_ORDER WHERE SALESMAN\_NO = 'S101';



1. **Count total number of Salesman..**

**OUTPUT:**

USE mydb;

SELECT COUNT(SALESMAN\_NO) FROM SALES\_MASTER;

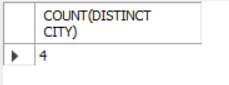


5. **Count total number of cities.**

**OUTPUT:**

USE mydb;

SELECT COUNT(DISTINCT CITY) FROM SALES\_MASTER;

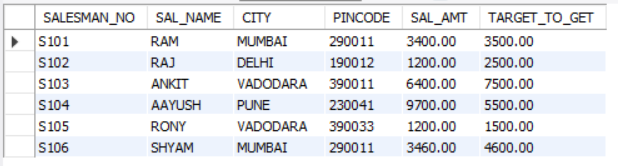


**6**. **Create table sales\_person from sales\_master table with all the columns.**

**OUTPUT:**

USE mydb;

CREATE TABLE SALES\_PERSON SELECT \* FROM SALES\_MASTER;



**7. Display the name of Salesman whose name starts with ‘A’ and its 6 character long.**

**OUTPUT:**

USE mydb;

SELECT \* FROM SALES\_MASTER WHERE length(SAL\_NAME)=6 AND SAL\_NAME LIKE 'A%';



**8. Write a query to set corresponding fields as Null, if we delete the record from sales master table.**

**OUTPUT:**

SELECT \* FROM SALES\_MASTER WHERE length(SAL\_NAME)=6 AND SAL\_NAME LIKE 'A%';



**9. Delete the detail of supplier whose salesman number is ‘S105’.**

**OUTPUT:**

USE mydb;

DELETE FROM SALES\_MASTERS WHERE SALESMAN\_NO = 'S105';



**10. Update the table such as way the if we Delete the records of ‘S103’ from salesmaster table then details of ‘S103’ should get deleted from all the tables.**

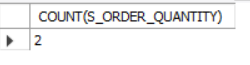
**OUTPUT:**

**11. Count the number of orders which are in-process.**

**OUTPUT:**

USE mydb;

SELECT COUNT(S\_ORDER\_QUANTITY) FROM SALES\_ORDER WHERE ORDER\_STATUS = 'IN-PROCESS';



**12. Display the number of orders fulfilled between in the month of January 2023.**

**OUTPUT:**

USE mydb;

SELECT S\_ORDER\_QUANTITY FROM SALES\_ORDER WHERE MONTH(S\_ORDER\_DATE)= '01' AND YEAR(S\_ORDER\_DATE) = '2023' AND ORDER\_STATUS = 'FULFILLED';

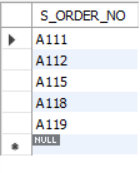


**13. Display the order numbers who have ordered more than 10 quantities.**

**OUTPUT:**

USE mydb;

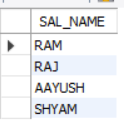
SELECT S\_ORDER\_NO FROM SALES\_ORDER WHERE S\_ORDER\_QUANTITY > 10;



**14. Display the name of salesman who do not based in ’Vadodara’.**

USE mydb;

SELECT SAL\_NAME FROM SALES\_MASTER WHERE CITY != 'VADODARA';



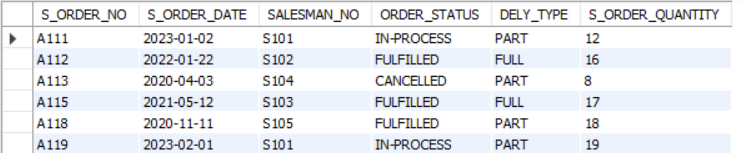
**15. Delete the records of whoes order has been cancelled in the year ‘2020’**

**OUTPUT:**

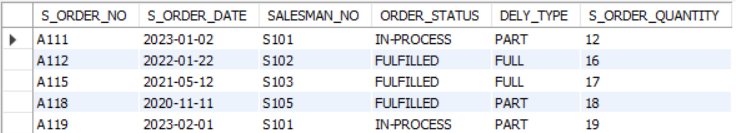
USE mydb;

DELETE FROM SALES\_ORDER WHERE ORDER\_STATUS = 'CANCELLED' AND YEAR(S\_ORDER\_DATE) = '2020';

Before:



After:



**CONCLUSION**

The ability to manipulate data and use aggregate functions on created tables is crucial for effective database management. These commands and functions are particularly important for tables such as sale master and sale orders, which require careful management of sales data to identify trends and patterns. Overall, a thorough understanding of these concepts is essential for optimizing the performance and effectiveness of any database system.



## SQL PRACTICAL 4



**CREATE TABLE**

CREATE TABLE Employee1 (

EmpID numeric,

EmpFName varchar(20),

EmpLName varchar(20),

Age numeric(65),

EmailId varchar(30),

PhoneNo numeric(10),

Address varchar(50)

);

**Insert data**

insert into project values (444, 4, 2, "Project4", "2019-04-16");

insert into project values (555, 5, 4, "Project5", "2019-05-23");

insert into project values (666, 6, 1, "Project6", "2019-01-12");

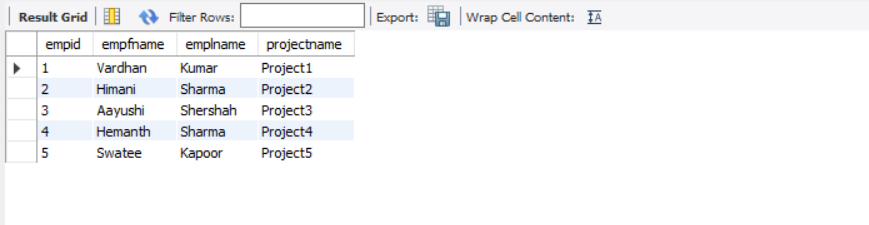
insert into project values (777, 7, 2, "Project7", "2019-07-25");

insert into project values (888, 8, 3, "Project8", "2019-08-20");

**INNER JOIN**

**Query and Output**

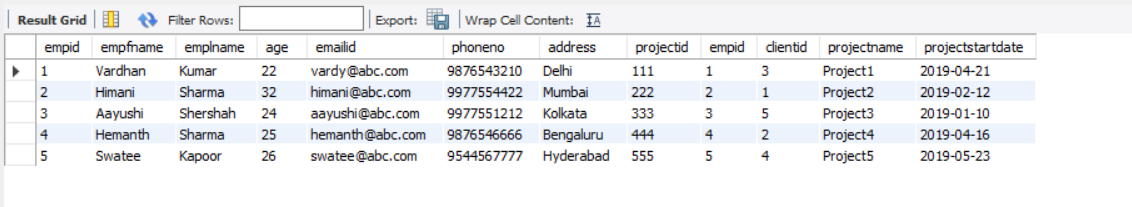
SELECT employee.empid, employee.empfname, employee.emplname, project.projectname FROM employee INNER JOIN project ON employee.empid=project.empid;



**LEFT JOIN**

**Query and Output**

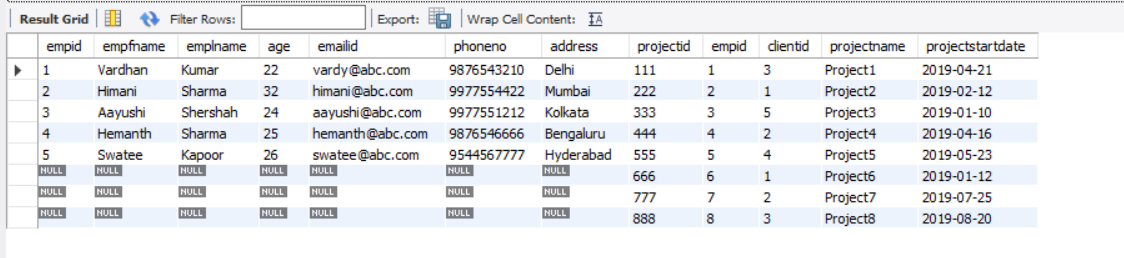
select \* from employee left join project on employee.empid=project.empid



**RIGHT JOIN**

**Query and Output**

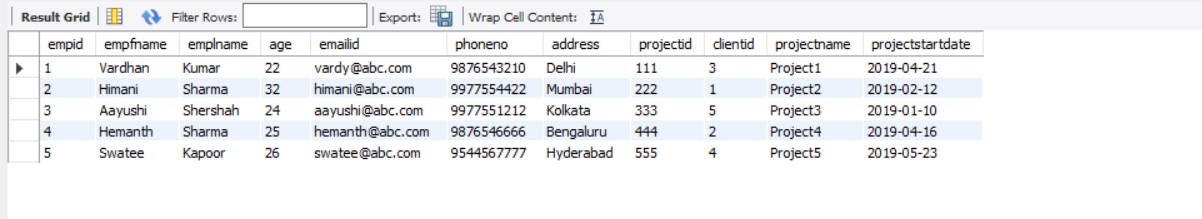
select \* from employee right join project on employee.empid=project.empid



**NATURAL JOIN**

**Query and Output**

select \* from employee natural join project



**CONCLUSION**

Understanding the different types of joints in DBMS is crucial for effective database management. These joints allow for the combination of data from multiple tables, enabling individuals to retrieve and analyze data in a meaningful way. The different types of joints, including inner join, outer join, and cross join, provide flexibility and versatility in managing different types of data relationships. By using the appropriate type of join, individuals can ensure that the data they retrieve is accurate and relevant to their needs. Overall, a thorough understanding of the different types of joints is essential for optimizing the performance and effectiveness of any database system.



## SQL PRACTICAL 5



**Q1. Create the Employee table and insert the data given above.**

**Query and Output**

create table Employee\_lab5

(

employee\_id integer,

name varchar(50),

months integer,

salary integer

);

insert into Employee\_lab5 values(12228, 'Rose', 15, 1968),

(33645,'Angela', 1, 3443);

insert into Employee\_lab5 values(45692,'Frank',17, 1608),

(56118,'Patrick',7,1345),

(59725,'Lisa',11,2330),

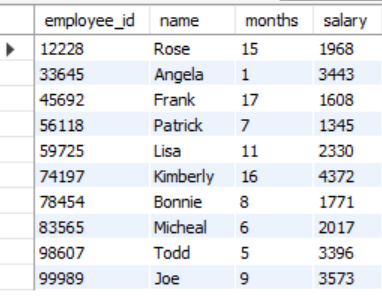
(74197, 'Kimberly', 16,4372),

(78454, 'Bonnie', 8, 1771),

(83565, 'Micheal', 6, 2017),

(98607, 'Todd', 5, 3396),

(99989, 'Joe', 9, 3573);

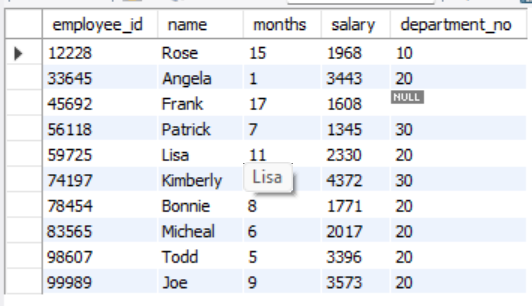


**Q2. Add the new column department\_no with integer data type.**

**Query and Output**

Alter table employee\_lab5

add department\_no integer;



**Query and Output**

SET SQL\_SAFE\_UPDATES=0;

update employee\_lab5

set department\_no = 10

where name = 'Angela'or name='bonnnie'or name= 'joe'or name='Rose';

**Q.4 Update the department\_no as 20 for Michael, Todd and Lisa.**

**Query and Output**

SET SQL\_SAFE\_UPDATES=0;

update employee\_lab5

set department\_no = 20

where name = 'Micheal'or name='Todd'or name= 'lisa';

**Q5. Update the department\_no as 30 for Kimberly and Patrick.**

**Query and Output**

SET SQL\_SAFE\_UPDATES=0;

update employee\_lab5

set department\_no = 30

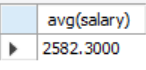
where name = 'Kimberly'or name='Patrick';

**Q6. Find out the average salary of all the employees.**

**Query and Output**

select avg(salary)

from employee\_lab5;

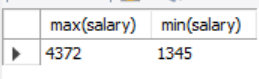


**Q7. Find the maximum salary and minimum salary from each department.**

**OUTPUT:**

select max(salary), min(salary)

from employee\_lab5;



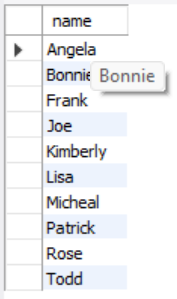
**Q8. Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table in alphabetical order.**

**OUTPUT:**

select name

from employee\_lab5

ORDER BY name asc;



**Q9. Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than 2000 per month who have been employees for less than 10 months. Sort your result by ascending employee\_id.**

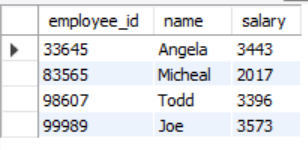
**OUTPUT:**

select employee\_id, name, salary

from employee\_lab5

where salary> 2000 and months<10

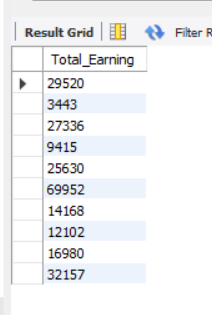
ORDER BY employee\_id asc;



**Q10. We define an employee's total earnings to be their monthly ‘salary x months’ worked, and the maximum total earnings to be the maximum total earnings for any employee in the Employee table. Write a query to find the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings.**

**OUTPUT:**

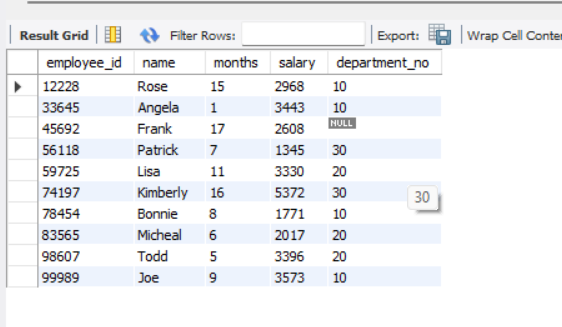
Select (salary\*months) as Total\_Earning from employee\_lab5;



**Q11. Increase the salary of each employee by 1000 who have worked for more than 10 months.**

**OUTPUT:**

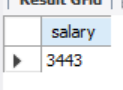
update employee set salary=salary+1000 where months>10;



**Q12. Find out the third largest salary of the employee.**

**OUTPUT:**

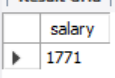
select salary from employee\_lab5 order by salary desc limit 2,1;



**Q13. Find out thirdlast smallest salary of the employee.**

**OUTPUT:**

select salary from employee\_lab5 order by salary asc limit 2,1;



**CONCLUSION**

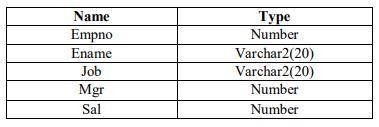
The employee table and functions such as MAX(), ascending, and descending are essential concepts for effective database management. The employee table serves as the foundation for organizing and managing employee data, while the MAX() function allows for the retrieval of the highest value in a specified column. Additionally, sorting data in ascending or descending order provides valuable insights into employee performance and organizational trends. By using these functions effectively, individuals can gain valuable insights into employee data and optimize the performance of their database system. Overall, a thorough understanding of the employee table and these functions is crucial for effective database management.



## SQL PRACTICAL 6



1. **Create a table called Employee with the following structure.**



create table employee (

empno int,

ename varchar(20),

job varchar(20),

mgr int,

sal int

);

**1. Add a column commission with domain to the Employee table.**

**Query and Output**

ALTER TABLE Employee

ADD commission DECIMAL(10,2) CHECK (commission >= 0);

insert into employee values ( 201, "Alpha Beta", "Manager", 1, 3500, 100);

insert into employee values ( 202, "Gamma Theta", "Clerk", 2, 2000, 75);

insert into employee values ( 203, "Omega Eta", "Clerk", 1, 1900, 70);

insert into employee values ( 204, "Zeta Delta", "Manager", 3, 3000, 90);

insert into employee values ( 205, "Omicron Sigma", "Clerk", 4, 2200, 80);

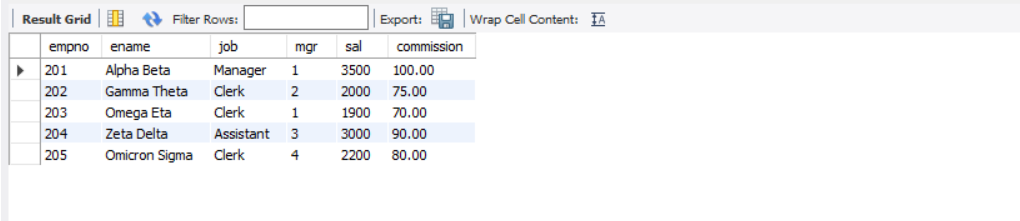
select \* from employee



**2. UPDATE EMPLOYEE SET JOB = "SAL\_MANAGER" WHERE SAL = "11000";**

**Query and Output**

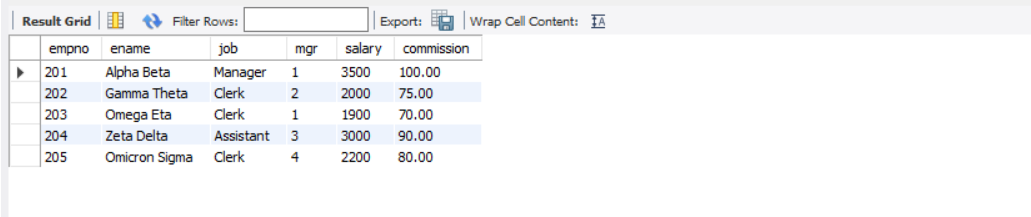
update employee set job="Assistant" where empno=204;



**3. ALTER TABLE employee RENAME COLUMN sal TO salary;**

**Query and Output**

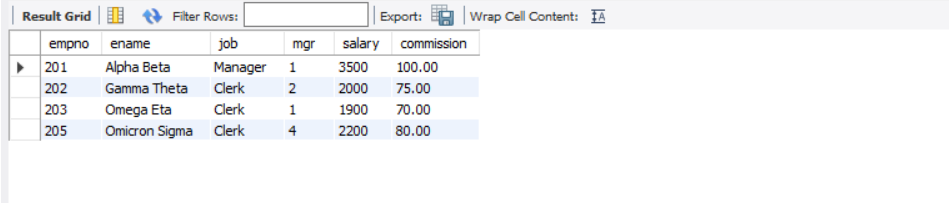
ALTER TABLE EMPLOYEE RENAME COLUMN Sal to Salary;



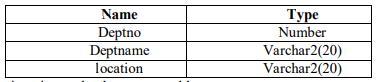
**4. delete from employee where empno=204;**

**Query and Output**

UPDATE EMPLOYEE SET JOB = "SAL\_MANAGER" WHERE SAL = "11000";



1. **Create department table with the following structure.**



create table department (

deptno int,

deptname varchar(20),

location varchar(20)

);

**1. Add column designation to the department table.**

**Query and Output**

ALTER TABLE department add designation varchar(20);

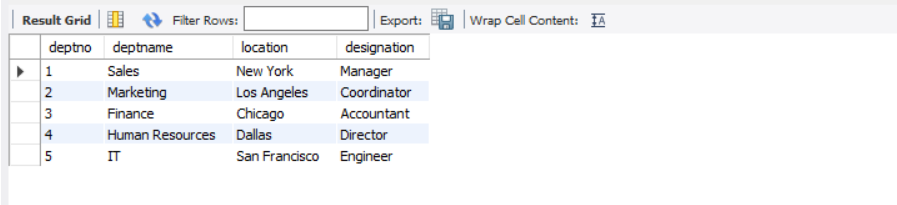
INSERT INTO department VALUES (1, 'Sales', 'New York', 'Manager');

INSERT INTO department VALUES (2, 'Marketing', 'Los Angeles', 'Coordinator');

INSERT INTO department VALUES (3, 'Finance', 'Chicago', 'Accountant');

INSERT INTO department VALUES (4, 'Human Resources', 'Dallas', 'Director');

INSERT INTO department VALUES (5, 'IT', 'San Francisco', 'Engineer');



**2. List the records of emp table grouped by deptno.**

**Query and Output**

SELECT mgr, COUNT(\*) as numofemp FROM employee GROUP BY mgr;



**3. Update the record where deptno is 9.**

**Query and Output**

update department set deptname="Software" where deptno=2;

select \* from department;

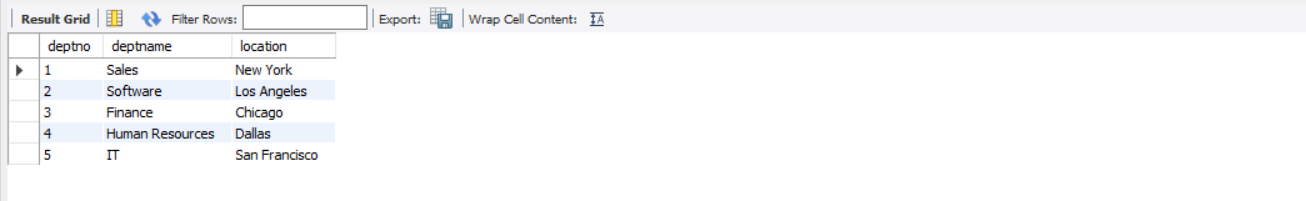


**4. Delete any column data from the table.**

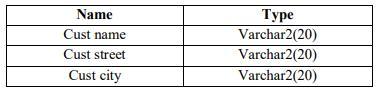
**Query and Output**

alter table department drop column designation;

select \* from department;



**3. Create a table called Customer table**



create table customer (

custname varchar(20),

custstreet varchar(20),

custcity varchar(20)

);

insert into customer values ("Alpha Beta", "Street 1", "Delhi");

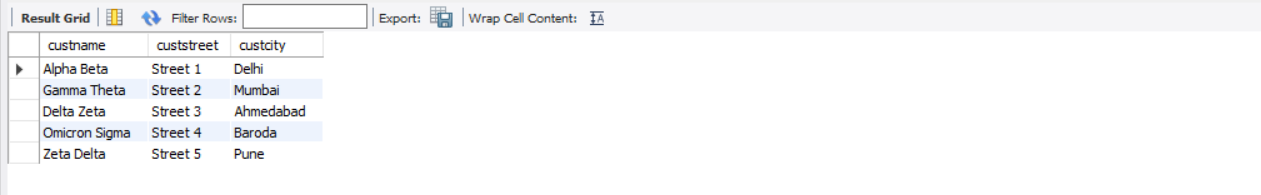
insert into customer values ("Gamma Theta", "Street 2", "Mumbai");

insert into customer values ("Delta Zeta", "Street 3", "Ahmedabad");

insert into customer values ("Omicron Sigma", "Street 4", "Baroda");

insert into customer values ("Zeta Delta", "Street 5", "Pune");

select \* from customer;



**1. Add salary column to the table.**

**Query and Output**

ALTER TABLE customer ADD salary int;

select \* from customer;



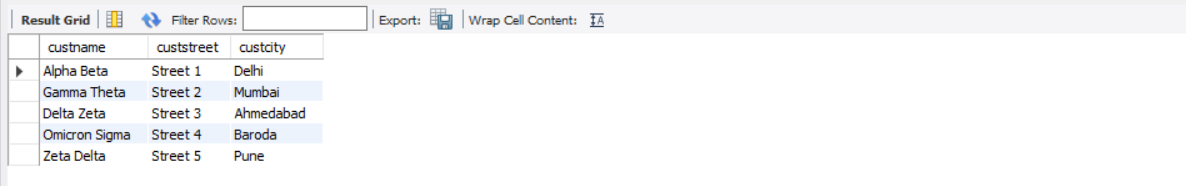
**2. Drop salary column of the customer table.**

**Query and Output**

ALTER TABLE customer ADD CONSTRAINT chk\_salary CHECK (salary > 0);

ALTER TABLE customer DROP COLUMN salary;

select \* from customer;

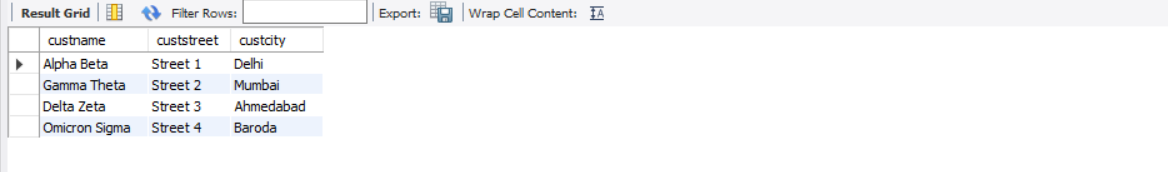


**3. Delete the rows of customer table whose cust\_city is pune.**

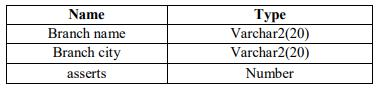
**Query and Output**

delete from customer where custcity="Pune";

select \* from customer;



**4. BRANCH**



create table branch (

branchname varchar(20),

branchcity varchar(20),

asserts decimal(6,2),

primary key (branchname)

);

**1. Add and drop a column to the branch table.**

**Query and Output**

ALTER TABLE branch MODIFY COLUMN asserts DECIMAL(16, 2);

ALTER TABLE branch ADD COLUMN manager VARCHAR(255) NOT NULL;

ALTER TABLE branch DROP COLUMN manager;

**2. Insert values to the table.**

**Query and Output**

insert into branch values ("Branch 1", "Ahmedabad", 2313);

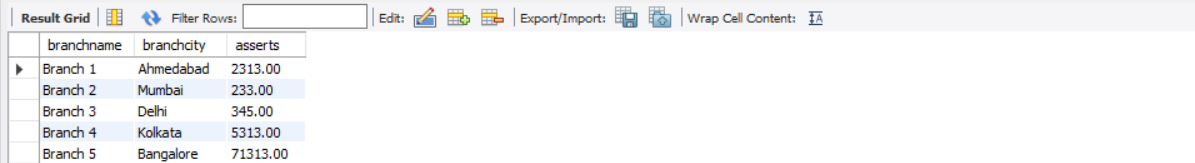
insert into branch values ("Branch 2", "Mumbai", 233);

insert into branch values ("Branch 3", "Delhi", 345);

insert into branch values ("Branch 4", "Kolkata", 5313);

insert into branch values ("Branch 5", "Bangalore", 71313);

select \* from branch;



**3. Modify branch name.**

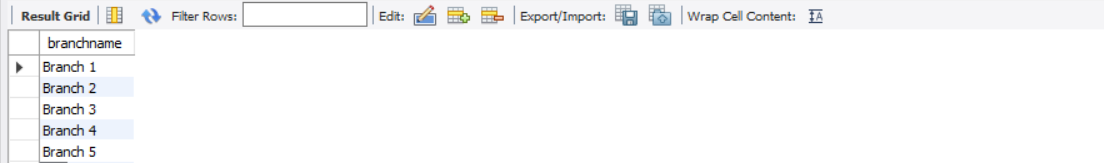
**Query and Output**

ALTER TABLE branch MODIFY COLUMN branchname VARCHAR(64);

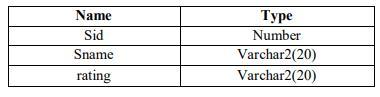
ALTER TABLE branch DROP COLUMN branchcity;

ALTER TABLE branch DROP COLUMN asserts;

select \* from branch;



**5. Sailor**



create table sailor (

sid int,

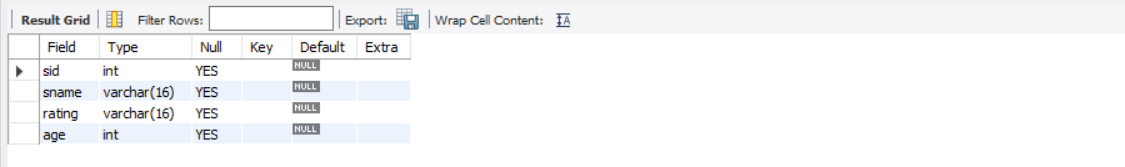
sname varchar(16),

rating varchar(16)

);

ALTER TABLE sailor ADD age INT;

describe sailor;



insert into sailor values (10, "Alpha", "A", 56);

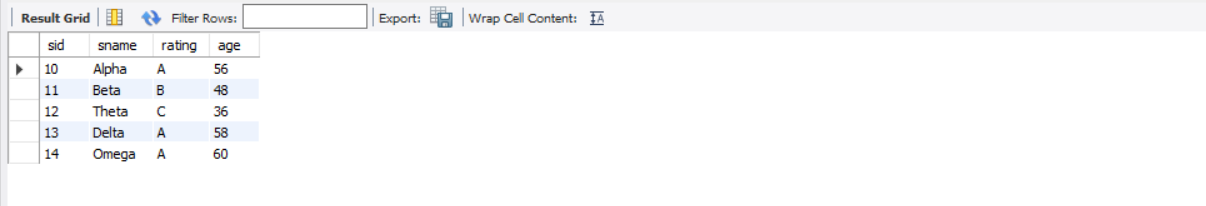
insert into sailor values (11, "Beta", "B", 48);

insert into sailor values (12, "Theta", "C", 36);

insert into sailor values (13, "Delta", "A", 58);

insert into sailor values (14, "Omega", "A", 60);

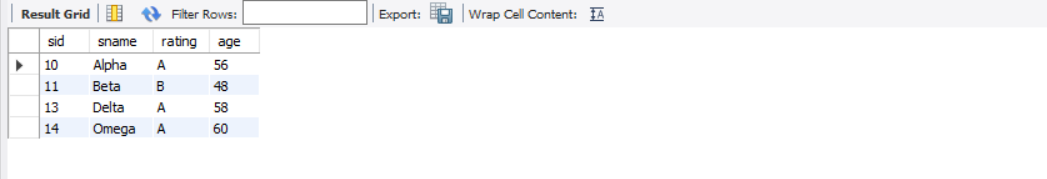
select \* from sailor;



**1. Delete the row with rating >8.**

**Query and Output**

delete from sailor where rating>"B";

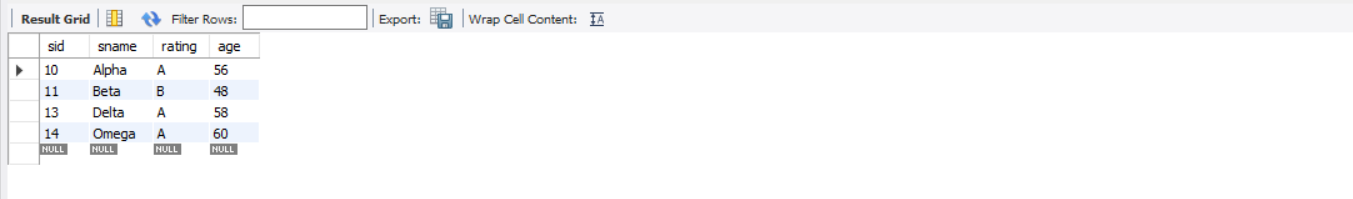


**2. Update the column details of sailor.**

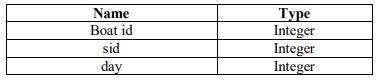
**Query and Output**

ALTER TABLE sailor MODIFY COLUMN sname VARCHAR(32);

insert into sailor values ();



**6. reserves**



create table reserves (

boatid int,

sid int,

day\_ int

);

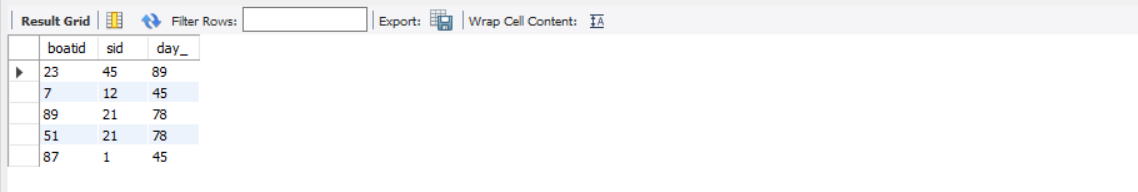
insert into reserves values (23, 45, 89);

insert into reserves values (7, 12, 45);

insert into reserves values (89, 21, 78);

insert into reserves values (51, 21, 78);

insert into reserves values (87, 1, 45);

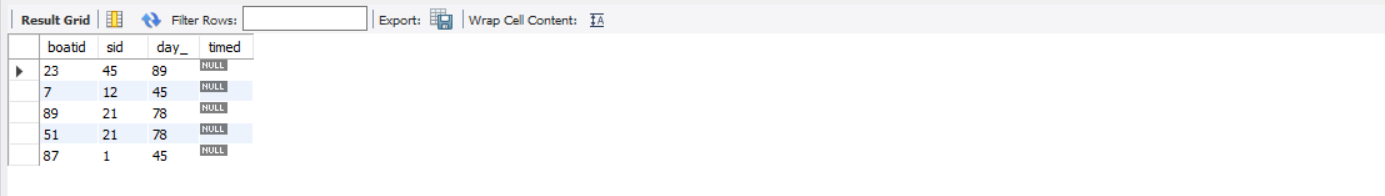


**1. Add column time to the reserves table.**

**Query and Output**

ALTER TABLE reserves ADD timed time;

select \* from reserves;



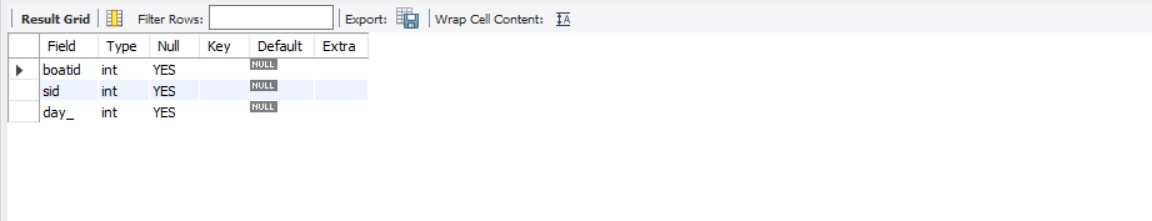
**2. Alter the column day data type to date.**

**Query and Output**

ALTER TABLE reserves MODIFY COLUMN day\_ DATE;

ALTER TABLE reserves DROP COLUMN timed;

describe reserves;



**3. Delete the row of the table with some condition.**

**Query and Output**

Delete from reserves where boatid=7;



**CONCLUSION**

The employee table, department table, consumer table, branch table, and sailor table are all fundamental concepts in database management. The employee table serves as the foundation for managing employee data, while the department table provides a means of organizing and managing department data. Similarly, consumer, branch, and sailor tables are essential for managing data related to customers, branches, and sailors, respectively. By effectively managing and organizing data in these tables, individuals can gain valuable insights into organizational performance and make informed decisions. Overall, a comprehensive understanding of these tables is essential for optimizing the performance and effectiveness of any database system.



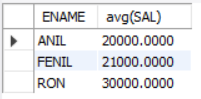
## SQL PRACTICAL 7



**1. By using the group by clause, display the enames who belongs to deptno 10 along with average salary.**

**OUTPUT:**

SELECT ENAME,avg(SAL) FROM EMPLOYEE1 WHERE DEP\_NO = 10 GROUP BY ENAME;



**2. Display lowest paid employee details under each department.**

**OUTPUT:**

SELECT DEP\_NO, ENAME, SAL

FROM employee1

WHERE SAL = (

SELECT MIN(SAL)

FROM EMPLOYEE1

)

ORDER BY DEP\_NO;



**3. Display number of employees working in each department and their department number.**

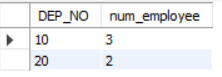
**OUTPUT:**

SELECT DEP\_NO, COUNT(\*) as num\_employee

FROM employee1

GROUP BY DEP\_NO

ORDER BY DEP\_NO



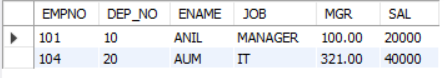
**4. List all employees which start with either A or C.**

**OUTPUT:**

SELECT \*

FROM employee1

WHERE ENAME LIKE 'A%' OR ENAME LIKE 'C%';



**5. Display only these ename of employees where the maximum salary is greater than or equal to 5000.**

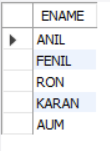
**OUTPUT:**

SELECT ENAME

FROM employee1

GROUP BY ENAME

HAVING MAX(SAL) >= 5000



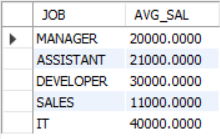
1. **1. Calculate the average salary for each different job.**

**OUTPUT:**

SELECT JOB, AVG(SAL) AS AVG\_SAL

FROM employee1

GROUP BY JOB



**2. Show the average salary of each job excluding manager.**

**OUTPUT:**

SELECT JOB, AVG(SAL) AS AVG\_SAL

FROM employee1

WHERE JOB != 'MANAGER'

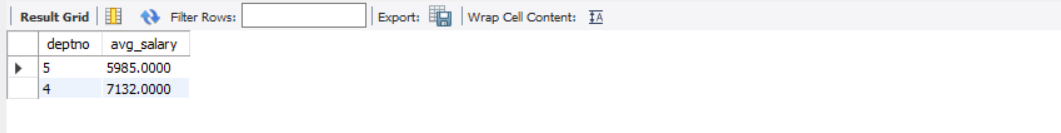
GROUP BY JOB



**3. Show the average salary for all departments employing more than three people.**

**OUTPUT:**

SELECT deptno, AVG(sal) as avg\_salary FROM employees GROUP BY deptno HAVING COUNT(\*) > 2;



**4. Display employees who earn more than the lowest salary in department 30**

**OUTPUT:**

SELECT \* FROM employees WHERE deptno = 5 AND sal > (SELECT MIN(sal) FROM employees WHERE deptno = 5);



**5. Show that value returned by sign (n) function.**

**OUTPUT:**

select ename, sign(sal) as salary\_sign from employees;



1. **1. Show that two substring as single string.**

**OUTPUT:**

SELECT CONCAT(ename, ' ', sal) AS ename\_plus\_sal FROM employees;

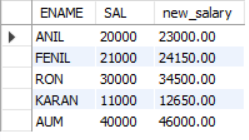


**2. List all employee names, salary and 15% rise in salary.**

**OUTPUT:**

SELECT ENAME, SAL, (SAL \* 1.15) AS new\_salary

FROM employee1;



**3. Display lowest paid emp details under each manager**

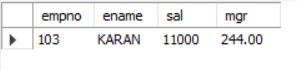
**OUTPUT:**

Use lab;

SELECT empno, ename, sal, mgr FROM employee1

WHERE SAL = ( SELECT MIN(SAL) FROM employee1 WHERE mgr = mgr )

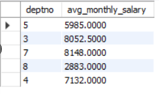
ORDER BY mgr;



**4. Display the average monthly salary bill for each deptno.**

**OUTPUT:**

SELECT deptno, AVG(sal) AS avg\_monthly\_salary FROM employees GROUP BY deptno;



**5. Show the average salary for all departments employing more than two people.**

**OUTPUT:**

SELECT DEP\_NO, AVG(SAL) as avg\_salary

FROM employee1

GROUP BY DEP\_NO

HAVING COUNT(\*) > 2;



**6. By using the group by clause, display the eid who belongs to deptno 05 along with average salary.**

**OUTPUT:**

SELECT EMPNO, AVG(SAL) as avg\_salary

FROM employee1

WHERE DEP\_NO = '10'

GROUP BY EMPNO, DEP\_NO;



1. **1. Count the number of employees in department 20**

**OUTPUT:**

SELECT COUNT(\*) as num\_employees FROM employee1 WHERE DEP\_NO = '20';



**2. Find the minimum salary earned by clerk.**

**OUTPUT:**

SELECT MIN(sal) FROM employees WHERE job = 'CLERK';



**3. Find minimum, maximum, average salary of all employees.**

**OUTPUT:**

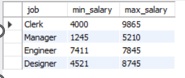
SELECT MIN(sal) AS min\_salary, MAX(sal) AS max\_salary, AVG(sal) AS avg\_salary FROM employees;



**4. List the minimum and maximum salaries for each job type.**

**OUTPUT:**

SELECT job, MIN(sal) AS min\_salary, MAX(sal) AS max\_salary FROM employees GROUP BY job;



**5. List the employee names in descending order.**

**OUTPUT:**

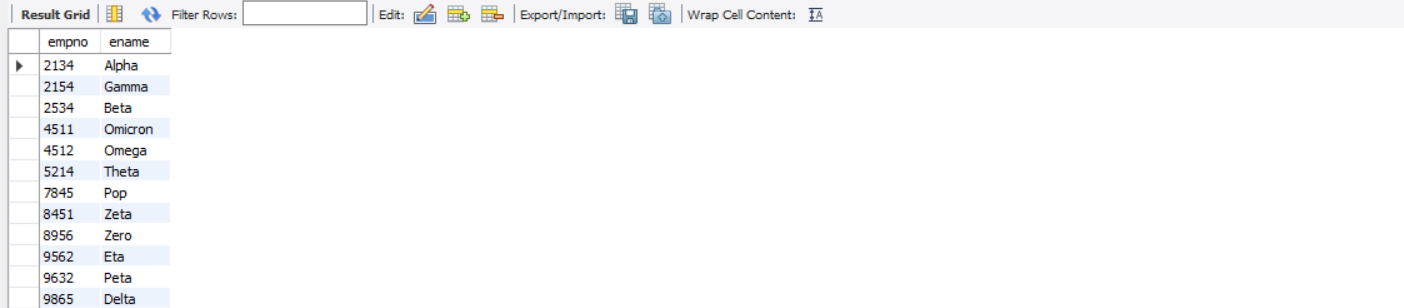
SELECT ename FROM employees ORDER BY ename DESC;



**6. List the employee id, names in ascending order by empid.**

**OUTPUT:**

SELECT empno, ename FROM employees ORDER BY empno ASC;



**CONCLUSION**

The provided list of tasks covers a wide range of fundamental concepts in database management, including data retrieval, manipulation, and analysis. By completing these tasks, individuals can gain practical experience in using SQL commands such as GROUP BY, INSERT, and built-in functions, among others, to manage and organize data in tables such as employee, department, and sailor. These tasks also require individuals to analyze data using aggregate functions, sorting, and filtering, providing valuable insights into organizational performance. Overall, a thorough understanding of these concepts and commands is essential for effective database management and optimization of database performance.



## SQL PRACTICAL 8



**A. Consider the schema for University Database:**

STUDENT (USN, SName, Address, Phone,Gender) SEMSEC (SSID, Sem, Sec)

CLASS (USN, SSID)

SUBJECT (Subcode, Title, Sem, Credits)

IAMARKS (USN, Subcode, SSID, Test1, Test2, Test3, Final\_IA)

**Q.1 Consider the above database and draw the Entity Relationship Model.**

**Q.2 Draw the schema diagram for the above database.**

**Q.3 Write SQL queries to**

**1) List all the student details studying in fourth semester 'C' section.**

**OUTPUT:**

SELECT S.\*, SS.SEM, SS.SEC

FROM STUDENT S, SEMSEC SS, CLASS C

WHERE S.USN = C.USN AND

SS.SSID = C.SSID AND

SS.SEM = 4 AND SS.SEC='C';



**2) Compute the total number of male and female students in each semester and in each section.**

**OUTPUT:**

SELECT SS.SEM, SS.SEC, S.GENDER, COUNT(S.GENDER) AS COUNT

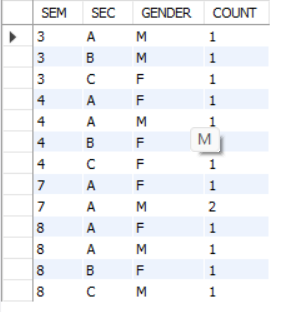
FROM STUDENT S, SEMSEC SS, CLASS C

WHERE S.USN = C.USN AND

SS.SSID = C.SSID

GROUP BY SS.SEM, SS.SEC, S.GENDER

ORDER BY SEM;



**3)Create a view of Test1 marks of student USN ' 1BI15CS101' in all subjects.**

**OUTPUT:**

CREATE VIEW STUDENT\_TEST1\_MARKS\_V

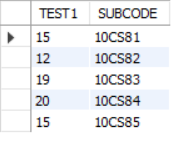
AS

SELECT TEST1, SUBCODE

FROM IAMARKS

WHERE USN = '1BI15CS101';

SELECT \* FROM STUDENT\_TEST1\_MARKS\_V;



**4) Categorize students based on the Following criterion: If Final\_IA = 17 to 20 then CAT ='Outstanding' If Final IA = 12 to 16 then CAT =`Average' If Final\_IA< 12 then CAT = 'Weak'**

**OUTPUT:**

SELECT S.USN,S.SNAME,S.ADDRESS,S.PHONE,S.GENDER, IA.SUBCODE,

(CASE

WHEN IA.FINALIA BETWEEN 17 AND 20 THEN 'OUTSTANDING'

WHEN IA.FINALIA BETWEEN 12 AND 16 THEN 'AVERAGE'

ELSE 'WEAK'

END) AS CAT

FROM STUDENT S, SEMSEC SS, IAMARKS IA, SUBJECT SUB

WHERE S.USN = IA.USN AND

SS.SSID = IA.SSID AND

SUB.SUBCODE = IA.SUBCODE AND

SUB.SEM = 8;



**5) Give these details only for 8th semester A, B, and C section students.**

**OUTPUT:**

SELECT I.USN, S.SName, S.Address, S.Phone, S.Gender, I.Final\_IA,

CASE

WHEN I.Final\_IA >= 70 AND I.Final\_IA <= 100 THEN 'Outstanding'

WHEN I.Final\_IA >= 40 AND I.Final\_IA <= 70 THEN 'Average'

ELSE 'Weak'

END AS CAT

FROM IAMARKS I

INNER JOIN STUDENT S ON I.USN = S.USN

INNER JOIN CLASS C ON I.USN = C.USN

INNER JOIN SEMSEC SS ON C.SSID = SS.SSID

WHERE SS.Sem = 5 AND SS.Sec IN ('A', 'B');



**B. Consider the schema for Movie Database:**

ACTOR (Act\_id, Act Name, Act\_Gender)

DIRECTOR (Dir id, Dir Name, Dir Phone)

MOVIES (Mov\_id, Moe Title, Mov\_Year, Mov\_Lang, DIR ID

MOVIE CAST (Act\_id, Mov\_id, Role)

RATING (Mov\_id, Rev Stars)

Write SQL queries to

1. **Create the tables as per above schema and set the different keys as per requirement.**

**OUTPUT:**

**ACTOR TABLE:**

CREATE TABLE ACTOR (

ACT\_ID INTEGER PRIMARY KEY,

ACT\_NAME VARCHAR(20),

ACT\_GENDER CHAR(1));

**DIRECTOR TABLE:**

CREATE TABLE DIRECTOR(

DIR\_ID INTEGER PRIMARY KEY,

DIR\_NAME VARCHAR(20),

DIR\_PHONE INTEGER);

**MOVIES TABLE:**

CREATE TABLE MOVIES(

MOV\_ID INTEGER PRIMARY KEY,

MOV\_TITLE VARCHAR(25),

MOV\_YEAR INTEGER,

MOV\_LANG VARCHAR(15),

DIR\_ID INTEGER,

FOREIGN KEY (DIR\_ID) REFERENCES DIRECTOR(DIR\_ID));

**MOVIES CAST TABLE:**

CREATE TABLE MOVIE\_CAST(

ACT\_ID INTEGER,

MOV\_ID INTEGER,

ROLE VARCHAR(10),

PRIMARY KEY (ACT\_ID,MOV\_ID),

FOREIGN KEY (ACT\_ID) REFERENCES ACTOR(ACT\_ID),

FOREIGN KEY (MOV\_ID) REFERENCES MOVIES(MOV\_ID));

**RATING TABLE:**

CREATE TABLE RATING(

MOV\_ID INTEGER PRIMARY KEY,

REV\_STARS VARCHAR(25),

FOREIGN KEY (MOV\_ID) REFERENCES MOVIES(MOV\_ID));

1. **Insert 5 records in each table.**

**OUTPUT:**

ACTOR TABLE:

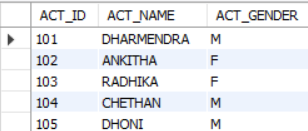
INSERT INTO ACTOR VALUES(101,'DHARMENDRA','M');

INSERT INTO ACTOR VALUES(102,'ANKITHA','F');

INSERT INTO ACTOR VALUES(103,'RADHIKA','F');

INSERT INTO ACTOR VALUES(104,'CHETHAN','M');

INSERT INTO ACTOR VALUES(105,'DHONI','M');



**DIRECTOR TABLE:**

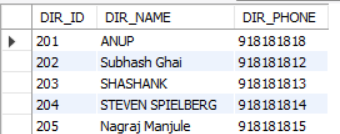
INSERT INTO DIRECTOR VALUES(201,'ANUP',918181818);

INSERT INTO DIRECTOR VALUES(202,'Subhash Ghai',918181812);

INSERT INTO DIRECTOR VALUES(203,'SHASHANK',918181813);

INSERT INTO DIRECTOR VALUES(204,'STEVEN SPIELBERG',918181814);

INSERT INTO DIRECTOR VALUES(205,'Nagraj Manjule',918181815);



**MOVIES TABLE:**

INSERT INTO MOVIES VALUES(1001,'MANASU',2017,'KANNADA',201);

INSERT INTO MOVIES VALUES(1002,'RRR',2020,'TELUGU',202);

INSERT INTO MOVIES VALUES(1003,'KALIYONA',2008,'KANNADA',201);

INSERT INTO MOVIES VALUES(1004,'WAR HORSE',2011,'ENGLISH',204);

INSERT INTO MOVIES VALUES(1005,'HOME',2012,'ENGLISH',205);



**MOVIES CAST TABLE:**

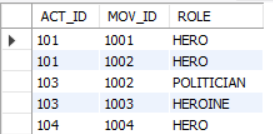
INSERT INTO MOVIE\_CAST VALUES(101,1002,'HERO');

INSERT INTO MOVIE\_CAST VALUES(101,1001,'HERO');

INSERT INTO MOVIE\_CAST VALUES(103,1003,'HEROINE');

INSERT INTO MOVIE\_CAST VALUES(103,1002,'POLITICIAN');

INSERT INTO MOVIE\_CAST VALUES(104,1004,'HERO');



**RATING TABLE:**

INSERT INTO RATING VALUES(1001,4);

INSERT INTO RATING VALUES(1002,2);

INSERT INTO RATING VALUES(1003,5);

INSERT INTO RATING VALUES(1004,4);

INSERT INTO RATING VALUES(1005,3);



1. **List the titles of all movies directed by 'Subhash Ghai'.**

**OUTPUT:**

USE LAB8;

SELECT MOV\_TITLE

FROM MOVIES

WHERE DIR\_ID = (SELECT DIR\_ID

FROM DIRECTOR

WHERE DIR\_NAME='Subhash Ghai');



1. **Display the name of all actors whose name starts with 'D'.**

**OUTPUT:**

USE LAB8;

SELECT ACT\_NAME FROM ACTOR

WHERE ACT\_NAME LIKE 'D%';



1. **Find the movie names where one or more actors acted in two or more movies.**

**OUTPUT:**

USE LAB8;

SELECT MOV\_TITLE

FROM MOVIES M,MOVIE\_CAST MC

WHERE M.MOV\_ID=MC.MOV\_ID AND ACT\_ID IN (SELECT ACT\_ID

FROM MOVIE\_CAST GROUP BY ACT\_ID

HAVING COUNT(ACT\_ID)>1)

GROUP BY MOV\_TITLE

HAVING COUNT(\*)>1;



1. **Update rating of all movies directed by `Nagraj Manjule' to 5.**

**OUTPUT:**

USE LAB8;

UPDATE RATING

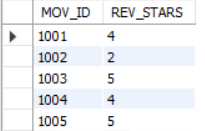
SET REV\_STARS=5

WHERE MOV\_ID IN (SELECT MOV\_ID FROM MOVIES

WHERE DIR\_ID IN (SELECT DIR\_ID

FROM DIRECTOR

WHERE DIR\_NAME='Nagraj Manjule'));



1. **Display Movie title and movie language directed id 205.**

**OUTPUT:**

use lab8;

SELECT MOV\_TITLE,MOV\_LANG FROM MOVIES WHERE DIR\_ID = 205;



**8.Display the details of all female Actors whose Name have the second last letter 'K';**

**OUTPUT:**

USE LAB8;

SELECT \* FROM ACTOR WHERE ACT\_GENDER = 'F' AND ACT\_NAME LIKE '%K\_';



**9. Display all the movies released in year 2020.**

**OUTPUT:**

USE LAB8;

SELECT \* FROM MOVIES WHERE MOV\_YEAR = 2020;



**10.Display the name of all actors who have done role of 'Politician';**

**OUTPUT:**

USE LAB8;

SELECT \* FROM MOVIE\_CAST WHERE ROLE = 'POLITICIAN';



**CONCLUSION**

The provided tasks involve working with two different database schemas, University Database and Movie Database, and require individuals to apply fundamental concepts in database management. The tasks involve designing an Entity Relationship Model and schema diagram, as well as writing SQL queries to perform data retrieval, manipulation, and analysis. By completing these tasks, individuals can gain practical experience and develop skills in database design, data organization, and SQL query writing. Overall, a thorough understanding of these concepts and commands is crucial for effective database management and optimization of database performance.

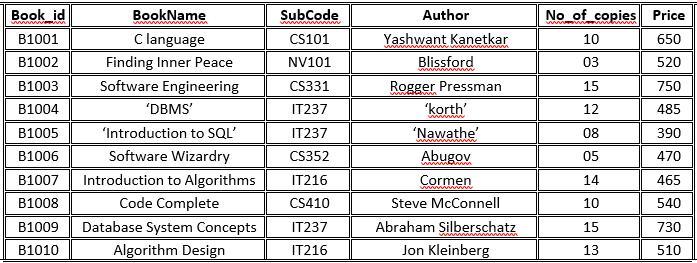


## SQL PRACTICAL 9



**Consider the structure of ‘Book’ Table as**

**Book\_Master {Bookid, BookName, SubCode, Autor, No\_of\_copies}**



Write SQL queries for the following:

1. **Add new column ‘Price’ in Book\_master Table.**

**OUTPUT:**

USE LAB9;

ALTER TABLE Book\_master ADD Price DECIMAL(8, 2);

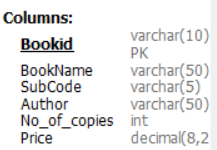


1. **Modify the datatype of ‘Book\_id’ attribute as varchar(10).**

**OUTPUT:**

USE LAB9;

ALTER TABLE books MODIFY COLUMN Bookid VARCHAR(10);



1. **Insert the 10 rows in Book\_master Table as mentioned in above table.**

**OUTPUT:**

INSERT INTO Books (BookID, BookName, SubCode, Author, No\_of\_copies, Price)

VALUES

('B1001', 'C language', 'CS101', 'Yashwant Kanetkar', 10, 650),

('B1002', 'Finding Inner Peace', 'NV101', 'Blissford', 3, 520),

('B1003', 'Software Engineering', 'CS331', 'Rogger Pressman', 15, 750),

('B1004', 'DBMS', 'IT237', 'Korth', 12, 485),

('B1005', 'Introduction to SQL', 'IT237', 'Nawathe', 8, 390),

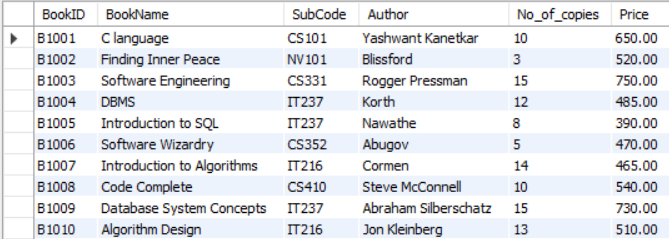
('B1006', 'Software Wizardry', 'CS352', 'Abugov', 5, 470),

('B1007', 'Introduction to Algorithms', 'IT216', 'Cormen', 14, 465),

('B1008', 'Code Complete', 'CS410', 'Steve McConnell', 10, 540),

('B1009', 'Database System Concepts', 'IT237', 'Abraham Silberschatz', 15, 730),

('B1010', 'Algorithm Design', 'IT216', 'Jon Kleinberg', 13, 510);

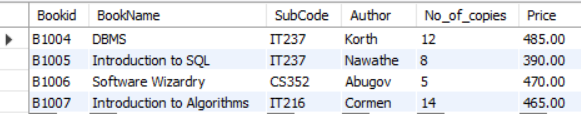


1. **Display the details of book whose price is less than 500.**

**OUTPUT:**

USE LAB9;

SELECT \* FROM Books WHERE Price < 500;



1. **Display the book id and book name of books whose Author is ‘Abugov’.**

**OUTPUT:**

USE LAB9;

SELECT Bookid, BookName FROM Books WHERE Author = 'Abugov';

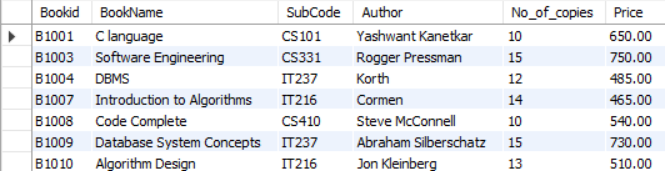


1. **Display the details of books whose 10 or more copies are available in the database.**

**OUTPUT:**

USE LAB9;

SELECT \* FROM Books WHERE No\_of\_copies >= 10;

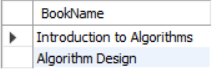


1. **Display the Name of books which are available for subject code ‘IT216’.**

**OUTPUT:**

USE LAB9;

SELECT BookName FROM Books WHERE SubCode = 'IT216';

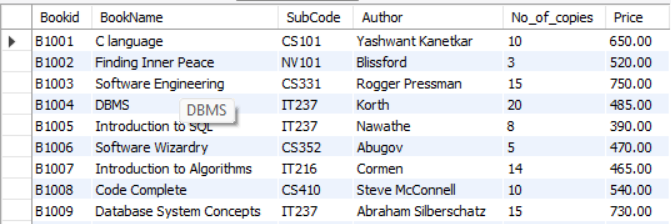


1. **Modify the No\_of \_copies of ‘DBMS’ book to 20.**

**OUTPUT:**

USE LAB9;

UPDATE Books SET No\_of\_copies = 20 WHERE BookName = 'DBMS';

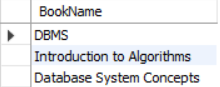


1. **Display the name of books whose book name ends with character ‘s’;**

**OUTPUT:**

USE LAB9;

SELECT BookName FROM Books WHERE BookName LIKE '%s';



1. **Modify the price of books as given bellow:**
   * **Increase it by 50 rupees whose price < 400,**
   * **Increase it by 70 rupees if price >=400 and price <700.**
   * **Increase it by 100 rupees if price >=700 and price<=1000.**

**OUTPUT:**

USE LAB9;

UPDATE Books

SET Price =

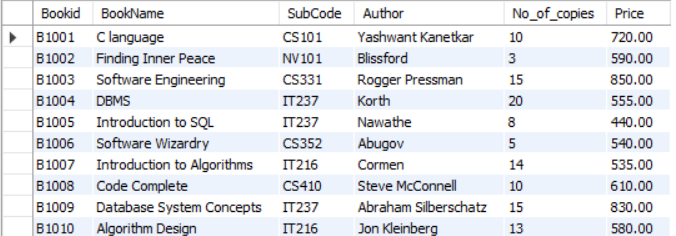
CASE

WHEN Price < 400 THEN Price + 50

WHEN Price >= 400 AND Price < 700 THEN Price + 70

WHEN Price >= 700 AND Price <= 1000 THEN Price + 100

END;



**CONCLUSION**

The provided tasks involve querying and modifying data in a Book\_master table using SQL commands. The tasks cover various fundamental concepts in database management, including adding and modifying columns, inserting data, and filtering and sorting data using conditions. By completing these tasks, individuals can gain practical experience and develop skills in database design, data organization, and SQL query writing. Overall, a thorough understanding of these concepts and commands is crucial for effective database management and optimization of database performance.



## SQL PRACTICAL 10



1. **Write a PL/SQL program to check whether a given number is positive, negative or zero.**

**CODE:**

**declare**

**num1 integer:= &num;**

**begin**

**if num1 < 0 then**

**dbms\_output.put\_line('negative number');**

**ELSIF num1 = 0 THEN**

**DBMS\_OUTPUT.PUT\_LINE('equal to zero');**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE('positive number');**

**END IF;**

**END;**

**/**

**OUTPUT:**



1. **Write a PL/SQL program to convert a temperature in scale Fahrenheit to Celsius and vice versa.**

**CODE:**

**DECLARE**

**temp1 NUMBER := &input\_a\_temp;**

**t\_scale CHAR := '&input\_temp\_scale';**

**new\_temp NUMBER;**

**new\_scale CHAR;**

**BEGIN**

**IF t\_scale != 'C'**

**AND**

**t\_scale != 'F' THEN**

**dbms\_output.Put\_line ('The scale you input is not a valid scale');**

**new\_temp := 0;**

**new\_scale := 'C';**

**ELSE**

**IF t\_scale = 'C' THEN**

**new\_temp := ( ( 9 \* temp1 ) / 5 ) + 32;**

**new\_scale := 'F';**

**ELSE**

**new\_temp := ( ( temp1 - 32 ) \* 5 ) / 9;**

**new\_scale := 'C';**

**END IF;**

**END IF;**

**dbms\_output.Put\_line ('The new temperature in scale '**

**||new\_scale**

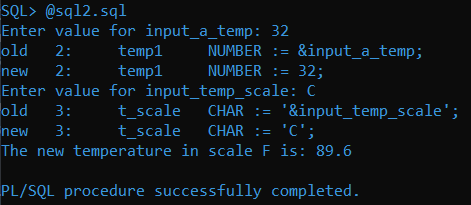
**||' is: '**

**||new\_temp);**

**END;**

**/**

**OUTPUT:**



1. **Write a Pl/SQL Program to print the following pattern.**

**1**

**12**

**123**

**1234**

**12345**

**CODE:**

**DECLARE**

**n INTEGER;**

**i INTEGER;**

**j INTEGER;**

**BEGIN**

**n:=&n;**

**FOR i IN 1..n LOOP**

**FOR j IN 1..i LOOP**

**dbms\_output.put(j);**

**END LOOP;**

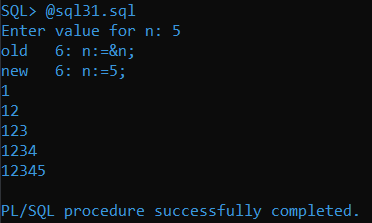
**DBMS\_OUTPUT.NEW\_LINE;**

**END LOOP;**

**END;**

**/**

**OUTPUT:**



1. **Write a program in PL/SQL to print 1st N numbers.**

**CODE:**

**DECLARE**

**n INTEGER;**

**i INTEGER;**

**BEGIN**

**n:=&n;**

**FOR i IN 1..n LOOP**

**dbms\_output.put(i);**

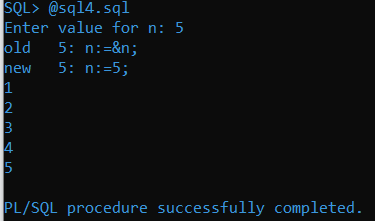
**DBMS\_OUTPUT.NEW\_LINE;**

**END LOOP;**

**END;**

**/**

**OUTPUT:**



1. **Write a program in PL/SQL to print the prime numbers between 1 to 50.**

**CODE:**

**DECLARE**

**i NUMBER(3);**

**j NUMBER(3);**

**BEGIN**

**dbms\_output.Put\_line('The prime numbers are:');**

**dbms\_output.new\_line;**

**i := 2;**

**LOOP**

**j := 2;**

**LOOP**

**EXIT WHEN( ( MOD(i, j) = 0 )**

**OR ( j = i ) );**

**j := j + 1;**

**END LOOP;**

**IF( j = i )THEN**

**dbms\_output.Put(i||' ');**

**END IF;**

**i := i + 1;**

**exit WHEN i = 50;**

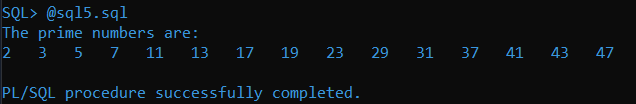
**END LOOP;**

**dbms\_output.new\_line;**

**END;**

**/**

**OUTPUT:**



1. **Write a PL/SQL block to increase the salary of Ram by 5000 if his salary is greater than or equal to 50000 and increase by 2000 if its less than 50000.**

**CODE:**

**DECLARE**

**salary INTEGER;**

**BEGIN**

**salary:=&salary;**

**IF(salary>=50000)THEN**

**salary:=salary+5000;**

**ELSE**

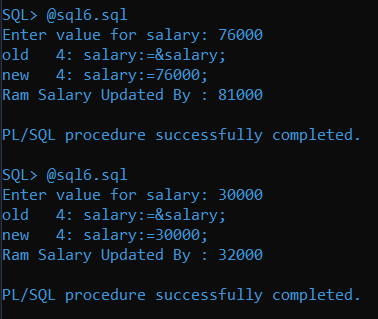
**salary:=salary+2000;**

**END IF;**

**dbms\_output.Put\_line('Ram Salary Updated By : '||salary);**

**END;**

**OUTPUT:**



1. **Write PL/SQL program to accept the age of person and decide whether the person is eligible to vote or not.**

**CODE:**

**DECLARE**

**age INTEGER;**

**BEGIN**

**age:=&age;**

**IF(age>=18)THEN**

**dbms\_output.Put\_line('You are eligible for vote. ');**

**ELSE**

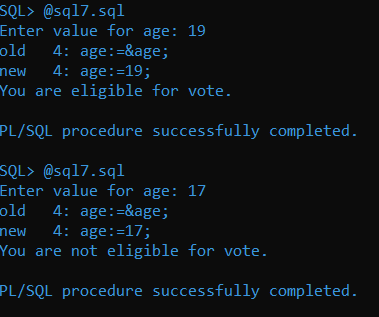
**dbms\_output.Put\_line('You are not eligible for vote.');**

**END IF;**

**END;**

**/**

**OUTPUT:**



1. **Write a PL/SQL block to create stored procedure to calculate addition of two numbers and execute it.**

**CODE:**

**Declare**

**Var1 integer;**

**Var2 integer;**

**Var3 integer;**

**Begin**

**Var1:=&var1;**

**Var2:=&var2;**

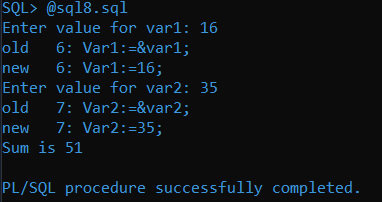
**Var3:=var1+var2;**

**Dbms\_output.put\_line('Sum is '||var3);**

**End;**

**/**

**OUTPUT:**



1. **Write a PLSQL program to create Function to calculate multiplication of two numbers and then write a PL/SQL block to execute it.**

**CODE:**

**Declare**

**Var1 integer;**

**Var2 integer;**

**Var3 integer;**

**Begin**

**Var1:=&var1;**

**Var2:=&var2;**

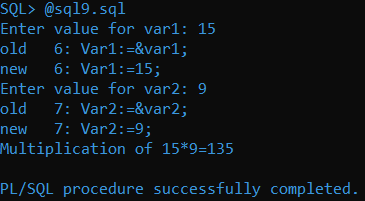
**Var3:=var1\*var2;**

**Dbms\_output.put\_line('Multiplication of '||var1||'\*'||var2||'='||var3);**

**End;**

**/**

**OUTPUT:**



1. **State the difference between Stored Procedure and Functions.**

**ANS:**

| **Functions** | **Procedures** |
| --- | --- |
| A function has a return type and returns a value. | A procedure does not have a return type. But it returns values using the OUT parameters. |
| You cannot use a function with Data Manipulation queries. Only Select queries are allowed in functions. | You can use DML queries such as insert, update, select etc… with procedures. |
| A function does not allow output parameters | A procedure allows both input and output parameters. |
| You cannot manage transactions inside a function. | You can manage transactions inside a procedure. |
| You cannot call stored procedures from a function | You can call a function from a stored procedure. |
| You can call a function using a select statement. | You cannot call a procedure using select statements. |

**CONCLUSION**

The provided tasks involve writing PL/SQL programs to perform various tasks such as checking numbers, temperature conversion, and printing patterns and numbers. The tasks also involve creating stored procedures and functions for performing calculations and decision-making processes. By completing these tasks, individuals can gain practical experience and develop skills in PL/SQL programming, which is essential for database management and optimization of database performance. Overall, a thorough understanding of PL/SQL programming is crucial for effective database management and development of efficient database systems.

**- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -**