Practical-1

Aim: Creation and Accessing of the tables (without constraints)

Theory:

Creating a basic table involves naming the table and defining its columns and each column's data type.

The SQL **CREATE TABLE** statement is used to create a new table.

Basic syntax of CREATE TABLE:

```
CREATE TABLE table_name (Column-1 datatype(size),
                         Column-2 datatype(size), .....
                          Column-N datatype(size));
```

The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

Syntax for INSERTION :

```
INSERT INTO TABLE_NAME VALUES(column-1,column-2,..column-n)
VALUES(value-1,value-  2,..value-n);
```

THE SQL **SELECT** statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

Syntax for SELECT:

```
select column1, column2,...columnN from table_name;
```

Here, column1, column2...are the fields of a table whose values you want to fetch. If you want to fetch all the fields use '*'.

The SQL **WHERE** clause is used to specify a condition while fetching the data from single table or joining with multiple tables. If the given condition is satisfied then only it returns specific value from the table. The WHERE clause is not only used in SELECT statement, but it is also used in UPDATE, DELETE statement, etc., which we would examine in subsequent chapters.

The basic syntax of SELECT statement with WHERE clause is as follows:

```
select column1, column2,...columnN from table_name where condition;
```

The SQL **UPDATE** Query is used to modify the existing records in a table. You can use WHERE clause with UPDATE query to update selected rows otherwise all the rows would be affected.

Syntax for UPDATING:

update table_name set column_name=value where condition;

The SQL **DELETE** Query is used to delete the existing records from a table. You can use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

Syntax for DELETION:

delete from table_name where condition;

The SQL **DROP TABLE** statement is used to remove a table definition and all data, indexes, triggers, constraints, and permission specifications for that table.

Syntax of DROPING TABLE:

drop table table_name;

Program: # 1

CREATE STUDENT TABLE AND INSERT DATA INTO THAT

Mysql> CREATE TABLE STUDENT1 (SNO int(5), SNAME varchar(15), SDOJ date);
OUTPUT:-TABLE CREATED

MySQL> INSERT INTO STUDENT1 VALUES (7001, 'MANIKANTA','1992-01-01');

OUTPUT:-1 row inserted
MySQL> INSERT INTO STUDENT1 VALUES (7002, 'ANURAJ', '1993-05-01');

OUTPUT:-1 row inserted
MySQL> INSERT INTO STUDENT1 VALUES (7003, 'AKSHITH','1995-01-04');

OUTPUT:-1 row inserted
MySQL> INSERT INTO STUDENT1 VALUES (7004, 'DIVYA','1994-08-05');

OUTPUT:-1 row inserted

MySQL> UPDATE STUDENT1 SET SNAME='Mani' WHERE SNAME='MANIKANTA';
OUTPUT:-1 row updated.

MYSQL> ALTER TABLE STUDENT1 ADD BRANCH VARCHAR(6);
OUTPUT:-Table altered.

MYSQL> UPDATE STUDENT1 SET BRANCH='IT' WHERE SNO=7001;
OUTPUT:-1 row updated.

MYSQL> SELECT * FROM STUDENT1;
OUTPUT:

SNO	SNAME	DOJ	BRANCH
7001	MANIKANTA	1992-01-01	IT
7002	ANURAJ	1993-05-01	NULL
7003	AKSHITH	1995-01-04	NULL
7004	DIVYA	1994-08-05	NULL

MySQL> DELETE FROM STUDENT1 WHERE SNO=7002;

OUTPUT:-1 row deleted.

MySQL> SELECT * FROM STUDENT1;

OUTPUT:-

SNO	SNAME	DOJ	BRANCH
7001	MANIKANTA	1992-01-01	IT
7003	AKSHITH	1995-01-04	NULL
7004	DIVYA	1994-08-05	NULL

MYSQL> ALTER TABLE STUDENT1 DROP BRANCH ;
OUTPUT:-Table altered.

MYSQL> RENAME TABLE STUDENT1 TO STUD;
OUTPUT:-Table renamed.

MySQL> DROP TABLE STUD;

OUTPUT:-Table dropped.

Practical-2

Aim: Simple SQL Queries using Select and Where Clause

Theory:
The **SELECT** Statement is used for retrieval of information from tables

Syntax:

SELECT <Col-List> FROM <Table-name>
[WHERE <Condition>]
[GROUP BY <Col-name(s)>]
[HAVING <Condition>]
[ORDER BY <Expression>];

Range Searching: Between operation is used for range searching

DISTINCT: The Distinct Clause is used with select to suppress duplicate values if any in a column.

Matching Patterns:

- The LIKE operator is used only with CHAR & VARCHAR 2 to match a pattern
- '%' represents a sequence of zero or more characters
- '_' (underscore) stands for any single character
- Both '%' & '_' are used with the LIKE operator to specify a pattern

Program:

1) List all information about all employee from table employee.

MYSQL>select * from emp;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

14 rows selected.

2) List all information about all departments from dept table.

MYSQL>select * from dept;

OUTPUT:

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

3) List all employee names along with their salaries from employee table.

MYSQL> select ename, sal from emp;

OUTPUT:

ENAME	SAL
SMITH	800
ALLEN	1600
WARD	1250
JONES	2975
MARTIN	1250
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
TURNER	1500
ADAMS	1100
JAMES	950
FORD	3000
MILLER	1300

4) List all Department no, employee number and their manager's number of all employee from employee table.

MYSQL> select deptno, empno, mgr from emp;

OUTPUT:

DEPTNO	EMPNO	MGR
20	7369	7902
30	7499	7698
30	7521	7698
20	7566	7839
30	7654	7698
30	7698	7839
10	7782	7839

20	7788	7566
10	7839	
30	7844	7698
20	7876	7788
30	7900	7698
20	7902	7566
10	7934	7782

5) **List dept names and location from dept table** .

MYSQL> select dname, loc from dept;

OUTPUT:

DNAME	LOC
ACCOUNTING	NEW YORK
RESEARCH	DALLAS
SALES	CHICAGO
OPERATIONS	BOSTON

6) **List empno, commission and hire date from emp table**.

MYSQL> select empno, comm, hiredate from emp;

OUTPUT:

EMPNO	COMM	HIREDATE
7369		17-DEC-80
7499	300	20-FEB-81
7521	500	22-FEB-81
7566		02-APR-81
7654	1400	28-SEP-81
7698		01-MAY-81
7782		09-JUN-81
7788		09-DEC-82
7839		17-NOV-81
7844	0	08-SEP-81
7876		12-JAN-83
7900		03-DEC-81
7902		03-DEC-81
7934		23-JAN-82

7) List employee belonging to the dept 20.

SQL> select ename from emp where deptno=20;

OUT PUT:

ENAME
SMITH
JONES
SCOTT

ADAMS
FORD

8) List name and salary of employee whose salary is more than 1000/-

SQL> select ename,sal from emp where sal>1000;

OUTPUT:

ENAME	SAL
ALLEN	1600
WARD	1250
MARTIN	1250
TURNER	1500
JONES	2975
BLAKE	2855
CLARK	2450
SCOTT	3000
ADAMS	1100
FORD	3000
MILLER	1300

9) List employee number and names of managers.

SQL> select empno,ename from emp where JOB='MANAGER';

OUTPUT:

EMPNO	ENAME
7566	JONES
7698	BLAKE
7782	CLARK

10) **List names of clerks working in department 20**.

SQL> select ename from emp where job='CLERK'and deptno=20;

OUTPUT:

ENAME
SMITH
ADAMS

11) **List names of analysts and salesman**.

SQL> select ename from emp where job='ANALYST' or job='SALESMAN';

OUTPUT:

ENAME
ALLEN

WARD
MARTIN
TURNER
SCOTT
FORD
6 rows selected.

12) **List details of employee who are joined before the end of Sep 81**.

SQL> select * from emp where HIREDATE<'1981-09-30';

OUTPUT:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7698	BLAKE	MANAGER	7839	01-MAY-81	2855		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10

8 rows selected.

13) **List all employee names who are not managers**.

SQL> select ename from emp where job<>'MANAGER';

OUTPUT:

ENAME

ALLEN
WARD
MARTIN
TURNER
SMITH
SCOTT
ADAMS
JAMES
FORD
MILLER
10 rows selected.

14) **List names of employee whose employ no 7369, 7521,7839,7934,7788**.

SQL> select ename from emp where empno in (7369,7521,7839,7934,7788);

OUTPUT:

ENAME

SCOTT
MILLER
WARD
SMITH

15) **List employ details not belonging to dept no's 10, 30, 40**.

SQL>select * from emp where deptno NOT IN(10,30,40)

OUTPUT:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7902	FORD	ANALYST	7566	03-DEC-81	3000		20

5 rows selected.

16) **List employ details of ename & sal whose sal is between 1000 and 2000**.

SQL> select ename,sal from emp where sal>1000 and sal<2000;

OUTPUT:

ENAME	SAL
ALLEN	1600
WARD	1250
MARTIN	1250
TURNER	1500
ADAMS	1100
MILLER	1300

6 rows selected.

17) **List the names of employee who are joined before 30-jun-81 and after dec-81**.

SQL> select ename from emp where hiredate<1981-06-30' or hiredate>'1981-12-31';

OUTPUT:
ENAME

ALLEN
WARD
SMITH
JONES
BLAKE
CLARK

SCOTT
ADAMS
MILLER
9 rows selected.

18) **List the distinct jobs available in the emp table**.

SQL> select distinct job from emp;

OUTPUT:

JOB

ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN
5 rows selected.

19) **List the employee names who are not eligible for commission**.

SQL> select ename from emp where comm is NULL;

OUTPUT:

ENAME

KURODA
JONES
BLAKE
CLARK
SCOTT
ADAMS
JAMES
FORD
MILLER
9 rows selected.

20) **List names of employees & designation who does not report to anybody**.

SQL> select ename,job from emp where mgr is NULL;

OUTPUT:

ENAME JOB
--- ---
KING PRESIDENT
1 row selected.

21) **List names of employee not assigned to any department**.

SQL> select ename from emp where deptno is NULL;

OUTPUT:

ENAME

MARTIN

1 row selected

22) **List all employees who are eligible for commission**.

SQL> select ename from emp where comm is not NULL;

OUTPUT:
ENAME

ALLEN
WARD
MARTIN
TURNER
4 rows selected

23) List details of employee whose salary is greater than 2000 and commission is null.

SQL> select * from emp where sal >2000 and comm is NULL;

OUTPUT:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7698	BLAKE	MANAGER	7839	01-MAY-81	2855	30	
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000	20	
7902	FORD	ANALYST	7566	03-DEC-81	3000	20	
7839	KING	PRESIDENT		17-NOV-81	5000	10	

6 rows selected.

24) **List the employee whose names starts with 's'**.

SQL> select ename from emp where ename like 'S%';

OUTPUT:
ENAME

SMITH
SCOTT
2 rows selected.

25) **List names of employee end with letter 's'**.

SQL> select ename from emp where ename like '%S';

OUTPUT:

ENAME

JONES
ADAMS
JAMES

3 rows selected.

26) **List names of employee whose names have exactly five characters**.

SQL> select ename from emp where ename like '_____';

OUTPUT:

ENAME

ALLEN
JONES
BLAKE
CLARK
SCOTT
ADAMS
JAMES

7 rows selected.

27) **List employee names whose are having 'I' as a second character**.

SQL> select ename from emp where ename like '_I%';

OUTPUT:

ENAME

MILLER
1 rows selected.

28) List the name, Sal and PF amount of all employee. (pf=10% of sal).

SQL> select ename,sal,sal*0.1 as pf from emp;

OUTPUT:

ENAME	SAL	PF
ALLEN	1600	160
WARD	1250	125
MARTIN	1250	125
TURNER	1500	150
JONES	2975	297.5
BLAKE	2855	285.5
CLARK	2450	245

SCOTT	3000	300
ADAMS	1100	110
JAMES	950	95
FORD	3000	300
MILLER	1300	130

12 rows selected.

29) **List names of employee who are more than 2 years old in the organization**.

SQL> select ename from emp where datediff(sysdate(),hiredate)>2*365;

OUTPUT:

ENAME

ALLEN
WARD
MARTIN
TURNER
JONES
BLAKE
CLARK
SCOTT
ADAMS
JAMES
FORD
MILLER

12 rows selected.

Practical-3

Aim: Simple SQL Queries on Aggregate Functions, Group by and Order by.

Theory:

ORDERING Results of a Query:

- SQL uses the **ORDER BY** Clause to impose an order on the result of a query
- ORDER BY Clause is used with SELECT statement

Syntax:

```
SELECT [DISTINCT] <column-list>/<exp>
FROM <table> [<table>]
[WHERE <Condition>]
GROUP BY<column/expression>
[HAVING <condition>];
[ORDER BY <columns>][ASC/DESC]
```

- One or more columns and/or expressions can be specified in ORDER BY clause
- Ascending is the default. Multiple columns are ordered one with in another. ORDER BY must always be last clause in SELECT.

ORDERING OUTPUT by COLUMN numbers

In place of column names, we can use numbers to indicate the fields being used to order the output. Their numbers will refer, not to the order of the columns in the table, but to their order in the output.

Aggregate Functions:

The aggregate functions produce a single value for an entire group or table.

- COUNT: Determine the number of rows non NULL column values
- SUM: Determine the sum of all selected columns
- MAX: Determine the largest of all selected values of a column
- MIN: Determine the smallest of all selected values of a column
- AVG: Determine the average of all selected values of a column

COUNT: Syntax: COUNT (*/ [Distinct]/ALL/column name)

SUM: Syntax: SUM ([Distinct]/ALL] column name)

MAX: Syntax: MAX (column-name)

MIN: Syntax: MIN (column-name)

AVG: Syntax: AVG ([Distinct/ALL] column-name)

GROUPING the Result of a query: GROUP BY clause is used to divide the rows in a table into smaller groups

Groups within groups: GROUP BY clause can be used to provide results for groups within groups.

HAVING Clause: is used to specify which group is to be displayed, that is restrict the groups that you return as the basis of aggregate functions.

Program:

1) **List employee details in ascending order of salary**.

```
SQL>  select *  from emp  order by sal;
```

OUTPUT:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7934	MILLER	CLERK	7782	23-JAN-82	1300		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7698	BLAKE	MANAGER	7839	01-MAY-81	2855		30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10

13 rows selected.

2) List employee names, sal, pf, hra, da, gross and order the result in ascd order of gross. (pf= 10% of sal, da= 50% of sal, hra=30% of sal).

```
SQL> select empno, ename, sal, sal*0.1 pf, sal*0.3 hra, sal*0.5 da, sal+sal*0.1+sal*0.3+sal*0.5
        gross from emp order by 7;
```

Output:

EMPNO	ENAME	SAL	PF	HRA	DA	GROSS
7900	JAMES	950	95	285	475	1805
7876	ADAMS	1100	110	330	550	2090
7521	WARD	1250	125	375	625	2375
7654	MARTIN	1250	125	375	625	2375
7934	MILLER	1300	130	390	650	2470
7844	TURNER	1500	150	450	750	2850
7499	ALLEN	1600	160	480	800	3040
7782	CLARK	2450	245	735	1225	4655

7698	BLAKE	2850	285	855	1425	5415
7566	JONES	2975	297.5	892.5	1487.5	5652.5
7788	SCOTT	3000	300	900	1500	5700
7902	FORD	3000	300	900	1500	5700
7839	KING	5000	500	1500	2500	9500

13 rows selected.

3) **List employee names and hiredate in descending order of hiredate**.

SQL> select ename, hiredate from emp order by hiredate desc;

OUTPUT:

ENAME

ADAMS
SCOTT
MILLER
JAMES
FORD
KING
MARTIN
TURNER
CLARK
BLAKE
JONES
WARD
ALLEN

13 rows selected.

4) List the employee names and sal, job, department number in descending order of deptno.

SQL> select ename, sal, job, deptno from emp order by deptno desc

Output:

ENAME	SAL	JOB	DEPTNO
ALLEN	1600	SALESMAN	30
WARD	1250	SALESMAN	30
MARTIN	1250	SALESMAN	30
JAMES	950	CLERK	30
TURNER	1500	SALESMAN	30
BLAKE	2850	MANAGER	30
JONES	2975	MANAGER	20
SCOTT	3000	ANALYST	20

```
ADAMS        1100 CLERK          20
FORD         3000 ANALYST        20
CLARK        2450 MANAGER        10
KING         5000 PRESIDENT      10
MILLER       1300 CLERK          10

13 rows selected.
```

5) List the number of employees working with company

```
SQL> select count(*) as noofemp  from emp;
        OUTPUT:
          NOOFEMP
        ----------
             14
```

6) List the number of jobs available in emp-table.

```
SQL> select distinct count(*) as No_Of_Jobs  from emp;
      OUTPUT:
         No_Of_Jobs
          ----------
            5
```

7) **List the total salaries payable to employees**.

```
SQL> select sum(sal) from emp;
      Output:
        SUM(SAL)
      ----------
         28230
```

8) **List the max salaries of employee working as a analyst**.

```
SQL> select max(sal)  from emp  where job='ANALYST';
      OUTPUT:
        MAX(SAL)
      ----------
          3000
```

9) **List minimum salary from emp-table**.

```
SQL> select min(sal)  from emp;
      OUTPUT:
        MIN(SAL)
      ----------
```

950

10) **List the average salary and no of employees working in dept 20**.

SQL> select avg(sal),count(*) from emp where deptno=20;

OUTPUT:

AVG(SAL)	COUNT(*)
2518.75	4

11) **List the dept no's and number of employees in each dept**.

SQL> select deptno,count(*) as noofemp from emp group by deptno;

DEPTNO	NOOFEMP
10	3
20	4
30	6

12) **List the dept no, and the total salary payable in each dept**.

SQL> select deptno, sum(sal) from emp group by deptno;

DEPTNO	SUM(SAL)
10	8750
20	10075
30	9405

13) List the jobs and no of emp in each job. The result should be in descending order of the number of employees.

SQL> select job, count(*) from emp group by job order by 2 desc;

Output:

JOB	COUNT(*)
SALESMAN	4
CLERK	3
MANAGER	3
ANALYST	2
PRESIDENT	1

14) List the total salary, maximum , minimum and average salary of the emp job wise

SQL> select job,sum(sal),min(sal),max(sal) from emp group by job;

OUTPUT:

```
JOB       SUM(SAL)   MIN(SAL)   MAX(SAL)
---------    ----------      ----------      ----------
ANALYST       6000      3000      3000
CLERK          3350      950      1300
MANAGER       8280      2450      2975
PRESIDENT     5000      5000      5000
SALESMAN      5600      1250      1600
```

15) **List the average salary from each job excluding managers**.

```
SQL>  select job,avg(sal)   from emp   where (job!='MANAGER')  group by job;
```

OUTPUT:

```
JOB       AVG(SAL)
--------- ----------
ANALYST       3000
CLERK    1116.66667
PRESIDENT     5000
SALESMAN      1400
```

16) List the total salary, maximum , minimum and average salary of the emp job wise for dept 20 only.

```
SQL> select job,sum(sal),min(sal),max(Sal),avg(sal) from emp  where(deptno=20)
 group by job;
```

OUTPUT:

```
JOB       SUM(SAL)   MIN(SAL)   MAX(SAL)   AVG(SAL)
---------    ----------   ---------- ----------    ----------
ANALYST       6000      3000       3000     3000
CLERK          1100      1100      1100      1100
MANAGER       2975      2975      2975      2975
```

17) **List the average monthly salary for each job type with in department**.

```
SQL> select job,deptno,avg(sal)  from emp  group by deptno,job;
```

OUTPUT:

```
JOB        DEPTNO   AVG(SAL)
--------- ---------- ----------
CLERK          10      1300
MANAGER        10      2450
PRESIDENT      10      5000
ANALYST        20      3000
CLERK          20      1100
MANAGER        20      2975
CLERK          30      950
MANAGER        30      2855
SALESMAN       30      1400
```

18) List all the total, average salary for all the departments employing more than '5' people.

```
SQL> select deptno,sum(sal),avg(sal)  from emp  group by deptno  having count(*)>5;
        OUTPUT:
          DEPTNO   SUM(SAL)   AVG(SAL)
           ---------- ---------- ----------
            30      9405     1567.5
```

19) List jobs of all employee where max salary is >=3000.

```
SQL> select job  from emp  group by job  having max(sal)>3000;
        OUTPUT:
        JOB
        ---------
        PRESIDENT
```

20) **List the job, total sal, maximum, minimum salary and average sal of the employees job wise for dept 20 and display only those rows having avg salary greater than 1000**.

```
SQL> select job, sum(sal),max(sal), min(sal), avg(sal) from emp
        where deptno=20 group by job having avg(sal) >1000;
        Output:
        JOB       SUM(SAL)   MAX(SAL)   MIN(SAL)   AVG(SAL)
        --------- ---------- ---------- ---------- ----------
        ANALYST       6000       3000       3000       3000
        CLERK         1100       1100       1100       1100
        MANAGER       2975       2975       2975       2975
```

Practical-4:

Aim: Simple SQL Queries on Built in Functions

Program:

1) **Find the absolute value of column or value passed**.

```
SQL> select ABS(-36.7) FROM DUAL;
Output:
ABS(-36.7)
----------
    36.7
```

2) **Find the smallest integer greater than or equal to n**.

```
SQL>select CEIL(35.3) from dual;
Output:
CEIL(35.3)
----------
      36
```

3) **Find the largest integer less than or equal to n**.

```
SQL> select FLOOR(35.3) from dual;
Output:
FLOOR(35.3)
-----------
       35
```

4) **Find the mod value of given two numbers**.

```
SQL> select MOD(5,2) from dual;
Output:
 MOD(5,2)
----------
       1
```

5) **Write a function that returns the value of m power n**.

```
SQL> select POWER(2,3)   from dual;
Output:
POWER(2,3)
----------
       8
```

6) **Find the square root of a given value**.

```
SQL> select SQRT(25)  from dual;


Output:
 SQRT(25)
----------
```

5

7) **Find the sign of the given number**.

```
SQL> select sign(-5)   from dual;

Output:
SIGN(-5)
----------
      -1
```

8) **Write a function to truncate a given value to 'n' decimal places**.

```
SQL> select TRUNC(3.9257,2)   from dual;
Output:
TRUNC(3.9257,2)
---------------
        3.92
```

9) **Write a function to round a given value to 'n' decimal places**.

```
SQL> select ROUND(3.92,1)   FROM DUAL;
Output:
ROUND(3.92,1)
-------------
        3.9
```

10) **Find the exponent value of a given number**.

```
SQL> select EXP(2)  from dual;
Output:
   EXP(2)
----------
 7.3890561
```

11) **Write a function to convert a given string into upper case**.

```
SQL> select upper('ram')  from dual;

Output:
UPP
---
RAM
```

12) **Write a function to convert a given string into lower case**.

```
SQL> select lower('STOP') from dual;

Output:
LOWER
----
```

stop

13) **Write a function to concatenate two strings**.

```
SQL>select concat('Ap','ple') from dual;
Output:
CONCA
-----
Apple
```

14) **Write a function to find ASCII value of given character**.

```
SQL> select ASCII('A') from dual;
Output:
ASCII(
-
65
```

15) **Write a function to find length of given string**.

```
SQL> select length('APPLE') from dual;

Output:
LENGTH
----
5
```

16) **Write a function to replace a character in a given string with a specified string**.

```
SQL> select replace('sam','s','r') from dual;

OUTPUT:
REP
---
ram
```

17) **Function to remove spaces on left side of string**.

```
SQL>  select LTRIM('     ABC') from dual;

OUTPUT:
LTRIM
-----
ABC
```

18) **Function to remove spaces on right side of string**.

```
SQL> SELECT RTRIM('ABC***','*') FROM DUAL;

OUTPUT:
RTRIM
------
```

ABC

19) **Using RPAD & LPAD place the '*' for ENAMES in the employee table**.

```
SQL> select RPAD(ename,10,'*')  from emp;

RPAD(ENAME)
----------
ALLEN*****
WARD******
JONES*****
MARTIN****
BLAKE*****
CLARK*****
SCOTT*****
KING******
TURNER****
ADAMS*****
JAMES*****
FORD******
MILLER****

SQL> select LPAD(ename,10,'*')   from emp;

LPAD(ENAME)
----------
*****ALLEN
******WARD
*****JONES
****MARTIN
*****BLAKE
*****CLARK
*****SCOTT
******KING
****TURNER
*****ADAMS
*****JAMES
******FORD
****MILLER
```

20) **Find the substring of a given string from 3rd character**.

```
SQL> select SUBSTR('DBMSLAB',3,2) FROM DUAL;

OUTPUT:
SU
--
MS
```

21) **Using translate function change letter 'O' to 'A' from employee table**.

SQL> select ename,TRANSLATE(ename,'O','A') from emp;

Output:

```
ENAME      TRANSLATE(
---------- ----------
ALLEN      ALLEN
WARD       WARD
JONES      JANES
MARTIN     MARTIN
BLAKE      BLAKE
CLARK      CLARK
SCOTT      SCATT
KING       KING
TURNER     TURNER
ADAMS      ADAMS
JAMES      JAMES
FORD       FARD
MILLER     MILLER
```

22) Add five months using add_months function to the hiredate of all employees in employee table.

SQL> select HIREDATE,DATE_ADD(HIREDATE,INTERVAL 5 MONTH) from emp;

Output:

```
HIREDATE        DATE_ADD
---------       ---------
20-FEB-81       20-JUL-81
22-FEB-81       22-JUL-81
02-APR-81       02-SEP-81
28-SEP-81       28-FEB-82
01-MAY-81       01-OCT-81
09-JUN-81       09-NOV-81
19-APR-87       19-SEP-87
17-NOV-81       17-APR-82
08-SEP-81       08-FEB-82
23-MAY-87       23-OCT-87
03-DEC-81       03-MAY-82
03-DEC-81       03-MAY-82
23-JAN-82       23-JUN-82
```

23) **Determine the last day of the sysdate**.

SQL> select sysdate,last_day(sysdate) from dual;

Output:

```
SYSDATE   LAST_DAY(
--------- ---------
27-OCT-11 31-OCT-11
```

24) **Determine highest experienced people among all employees in employee table**.

SQL>select ename, datediff (sysdate(),hiredate)/365 experience from emp order by 2

Output:

```
ENAME      EXPERIENCE
---------- ----------
ADAMS      24.4958933
SCOTT      24.5899793
MILLER     29.8292266
JAMES      29.9663234
FORD       29.9663234
KING       30.0120223
MARTIN     30.1491191
TURNER     30.2028825
CLARK      30.4501943
BLAKE      30.555033
JONES      30.6356782
WARD       30.7485814
ALLEN      30.7539578

13 rows selected.
```

25) **Convert a given string to date**.

SQL> select STR_TO_DATE('APR/12/1981','%M/%d/%Y'), STR_TO_DATE('04/12/81','%m/%d/%y') from dual;

Output:

```
STR_TO_DATE()         STR_TO_DATE()
-------------         ---------------
1981-04-12      1981-04-12
```

26) **Find the standard deviation and variance of the salary**.

SQL> select STDDEV(sal),VARIANCE(sal) from emp;

Output:

```
STDDEV(SAL)              VARIANCE(SAL)
-----------                        -------------
 1170.19613              1369358.97
```

Practical-5:

Aim: Creation of tables with constraints

Theory:

Data constraints: Besides the cell name, cell length and cell data type there are other parameters i.e. other data constrains that can be passed to the DBA at check creation time. The constraints can either be placed at column level or at the table level.

i. **Column Level Constraints:** If the constraints are defined along with the column definition, it is called a column level constraint.

ii. **Table Level Constraints:** If the data constraint attached to a specify cell in a table reference the contents of another cell in the table then the user will have to use table level constraints.

Column Constraints

1. NOT NULL: Prevents a column from accepting NULL values.
2. UNIQUE: Ensures uniqueness of the value.
3. PRIMARY KEY: Ensures column to be unique & not null, but only one column per table.
4. AUTO_INCREMENT : used along with the primary key to generate sequence for a column
5. ENUM: Prevents a column from accepting values other than in the enumeration
6. SET: Prevents a column from accepting values other than from the set, can take multiple values.
7. CHECK: Controls the value of a column being inserted.
8. DEFAULT: Assigns a default value, at the time of insertion.

Table Constraints

9. FOREIGN KEY (REFERENCES): Assigns a foreign key constraint to maintain a "Referential Integrity".

Program:

CREATE TABLES WITH VARIOUS SPECIFIED CONSTRAINTS AND INSERT DATA.

Null Value

```
MYSQL>    CREATE TABLE  STUD( ROLLNO INT(6) NOT NULL,
                                  NAME VARCHAR(10),
                                  BRANCH VARCHAR(6));
Query OK, 0 rows affected (0.21 sec)

MYSQL> DESC STUD;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| rollno | int(6)      | NO   |     | NULL    |       |
| name   | varchar(10) | YES  |     | NULL    |       |
| branch | varchar(6)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)


MYSQL> INSERT    INTO    STUD VALUES(5001, 'ABILASH', 'IT');
QUERY OK, 1 ROW AFFECTED (0.05 SEC)

MYSQL> INSERT    INTO    STUD VALUES(5002, 'AKSHITHA', 'IT');
QUERY OK, 1 ROW AFFECTED (0.03 SEC)

MYSQL> SELECT  *  FROM  STUD;


+--------+----------+--------+
| rollno | name     | branch |
+--------+----------+--------+
|   5001 | ABILASH  | IT     |
|   5002 | AKSHITHA | IT     |
+--------+----------+--------+
2 rows in set (0.00 sec)

MYSQL> INSERT  INTO  STUD VALUES(NULL, 'AKI', 'IT');
ERROR 1048 (23000): COLUMN 'ROLLNO' CANNOT BE NULL
```

1. **UNIQUE:**

```
MYSQL> CREATE TABLE STUD1(ROLLNO INT(6) UNIQUE,
                                  NAME VARCHAR(10),
                                  BRANCH VARCHAR(6));
QUERY OK, 0 ROWS AFFECTED (0.30 SEC)
```

```
MYSQL> DESC STUD1;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| rollno | int(6)      | YES  | UNI | NULL    |       |
| name   | varchar(10) | YES  |     | NULL    |       |
| branch | varchar(6)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

MYSQL> INSERT INTO STUD1 VALUES(5001, 'ABILASH', 'IT');
QUERY OK, 1 ROW AFFECTED (0.04 SEC)

MYSQL> INSERT INTO STUD1 VALUES(5002, 'AKSHITHA', 'IT');
QUERY OK, 1 ROW AFFECTED (0.03 SEC)

MYSQL> INSERT INTO STUD1 VALUES(NULL, 'AKI', 'IT');
QUERY OK, 1 ROW AFFECTED (0.02 SEC)

MYSQL> SELECT * FROM STUD1;
+--------+----------+--------+
| rollno | name     | branch |
+--------+----------+--------+
|   5001 | ABILASH  | IT     |
|   5002 | AKSHITHA | IT     |
|   NULL | AKI      | IT     |
+--------+----------+--------+
3 rows in set (0.00 sec)

MYSQL> INSERT INTO STUD1 VALUES(5001, 'ABI', 'IT');
ERROR 1062 (23000): DUPLICATE ENTRY '5001' FOR KEY 'ROLLNO'
```

2. PRIMARY KEY:

```
MYSQL> CREATE TABLE STUD2(ROLLNO INT(6) PRIMARY KEY,
                          NAME VARCHAR(10),
                          BRANCH VARCHAR(6));
QUERY OK, 0 ROWS AFFECTED (0.23 SEC)

MYSQL> DESC STUD2;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| rollno | int(6)      | NO   | PRI | NULL    |       |
| name   | varchar(10) | YES  |     | NULL    |       |
```

```
| branch | varchar(6)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)


MYSQL> INSERT INTO STUD2 VALUES(5001, 'ABILASH', 'IT');
QUERY OK, 1 ROW AFFECTED (0.04 SEC)

MYSQL> INSERT INTO STUD2 VALUES(5002, 'AKSHITHA', 'IT');
QUERY OK, 1 ROW AFFECTED (0.04 SEC)

MYSQL> SELECT * FROM STUD2;
+--------+----------+--------+
| rollno | name     | branch |
+--------+----------+--------+
|   5001 | ABILASH  | IT     |
|   5002 | AKSHITHA | IT     |
+--------+----------+--------+
2 rows in set (0.00 sec)

MYSQL> INSERT INTO STUD2 VALUES(NULL, 'AKI', 'IT');
ERROR 1048 (23000): COLUMN 'ROLLNO' CANNOT BE NULL

MYSQL> INSERT INTO STUD2 VALUES(5001, 'ABI', 'IT');
ERROR 1062 (23000): DUPLICATE ENTRY '5001' FOR KEY 'PRIMARY'
```

3. AUTO INCREMENT:

```
MYSQL> CREATE TABLE STUD3(ROLLNO INT(6) PRIMARY KEY AUTO_INCREMENT,
                          NAME VARCHAR(10),
                          BRANCH VARCHAR(6));
QUERY OK, 0 ROWS AFFECTED (0.24 SEC)

MYSQL> DESC STUD3;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| rollno | int(6)      | NO   | PRI | NULL    | auto_increment |
| name   | varchar(10) | YES  |     | NULL    |                |
| branch | varchar(6)  | YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

MYSQL> INSERT INTO STUD3 VALUES(5001, 'ABILASH', 'IT');
QUERY OK, 1 ROW AFFECTED (0.02 SEC)

MYSQL> INSERT INTO STUD3 VALUES(5002, 'AKSHITHA', 'IT');
```

```
QUERY OK, 1 ROW AFFECTED (0.05 SEC)

MYSQL> SELECT * FROM STUD3;
+--------+----------+--------+
| rollno | name     | branch |
+--------+----------+--------+
|   5001 | ABILASH  | IT     |
|   5002 | AKSHITHA | IT     |
+--------+----------+--------+
2 rows in set (0.00 sec)

MYSQL> INSERT INTO STUD3 VALUES(5001, 'ABI', 'IT');
ERROR 1062 (23000): DUPLICATE ENTRY '5001' FOR KEY 'PRIMARY'

MYSQL> INSERT INTO STUD3 VALUES(NULL, 'AKI', 'IT');
QUERY OK, 1 ROW AFFECTED (0.04 SEC)


MYSQL> INSERT INTO STUD3 VALUES(NULL, 'ABI', 'IT');
QUERY OK, 1 ROW AFFECTED (0.04 SEC)

MYSQL> SELECT * FROM STUD3;

+--------+----------+--------+
| rollno | name     | branch |
+--------+----------+--------+
|   5001 | ABILASH  | IT     |
|   5002 | AKSHITHA | IT     |
|   5003 | AKI      | IT     |
|   5004 | ABI      | IT     |
+--------+----------+--------+
4 rows in set (0.00 sec)
```

4. ENUMERATOR:

```
MYSQL> CREATE TABLE STUD5(ROLLNO INT(6) PRIMARY KEY ,
                          NAME VARCHAR(10),
                          BRANCH ENUM ('IT','CSE','ECE'));
QUERY OK, 0 ROWS AFFECTED (0.19 SEC)

MYSQL> DESC STUD5;

+--------+-----------------------+------+-----+---------+-------+
| Field  | Type                  | Null | Key | Default | Extra |
+--------+-----------------------+------+-----+---------+-------+
| rollno | int(6)                | NO   | PRI | NULL    |       |
```

```
| name   | varchar(10)          | YES  |     | NULL    |       |
| branch | enum('IT','CSE','ECE') | YES  |     | NULL    |       |
+--------+----------------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
MYSQL> INSERT INTO STUD5 VALUES(5001, 'ABILASH', 'IT');
QUERY OK, 1 ROW AFFECTED (0.04 SEC)

MYSQL> INSERT INTO STUD5 VALUES(5002, 'AKSHITHA', 'CSE');
QUERY OK, 1 ROW AFFECTED (0.02 SEC)

MYSQL> SELECT * FROM STUD5;
+--------+----------+--------+
| rollno | name     | branch |
+--------+----------+--------+
|   5001 | ABILASH  | IT     |
|   5002 | AKSHITHA | CSE    |
+--------+----------+--------+
2 rows in set (0.00 sec)
MYSQL> INSERT INTO STUD5 VALUES(5003, 'AKKI', 'MECH');
QUERY OK, 1 ROW AFFECTED, 1 WARNING (0.03 SEC)

MYSQL> SHOW WARNINGS;

+---------+------+-----------------------------------------+
| Level   | Code | Message                                 |
+---------+------+-----------------------------------------+
| Warning | 1265 | Data truncated for column 'branch' at row 1 |
+---------+------+-----------------------------------------+
1 row in set (0.00 sec)


MYSQL> SELECT * FROM STUD5;
+--------+----------+--------+
| rollno | name     | branch |
+--------+----------+--------+
|   5001 | ABILASH  | IT     |
|   5002 | AKSHITHA | CSE    |
|   5003 | AKKI     |        |
+--------+----------+--------+
3 rows in set (0.00 sec)
```

5. SET:

```
MYSQL> CREATE TABLE STUD7(ROLLNO INT(6) PRIMARY KEY ,
                          NAME VARCHAR(10),
                          GRADES SET ('A1','A2','A3','B1','B2','C1'));
QUERY OK, 0 ROWS AFFECTED (0.40 SEC)
```

```
MYSQL> DESC STUD7;
+--------+-----------------------------------+------+-----+---------+-------+
| Field  | Type                              | Null | Key | Default | Extra |
+--------+-----------------------------------+------+-----+---------+-------+
| rollno | int(6)                            | NO   | PRI | NULL    |       |
| name   | varchar(10)                       | YES  |     | NULL    |       |
| grades | set('A1','A2','A3','B1','B2','C1') | YES  |     | NULL    |       |
+--------+-----------------------------------+------+-----+---------+-------+
3 rows in set (0.00 sec)


MYSQL> INSERT INTO STUD7 VALUES(5001, 'ABILASH', 'A1,A2');
QUERY OK, 1 ROW AFFECTED (0.03 SEC)

MYSQL> INSERT INTO STUD7 VALUES(5002, 'AKSHITHA', 'A1,B1,C1');
QUERY OK, 1 ROW AFFECTED (0.03 SEC)

MYSQL> INSERT INTO STUD7 VALUES(5003, 'AKKI', 'A1,B3,C2') ;
QUERY OK, 1 ROW AFFECTED, 1 WARNING (0.04 SEC)

MYSQL> INSERT INTO STUD7 VALUES(5004, 'ABI', 'C2,D1') ;
QUERY OK, 1 ROW AFFECTED, 1 WARNING (0.05 SEC)

MYSQL> SHOW WARNINGS;
+---------+------+-------------------------------------------+
| Level   | Code | Message                                   |
+---------+------+-------------------------------------------+
| Warning | 1265 | Data truncated for column 'grades' at row 1 |
+---------+------+-------------------------------------------+
1 row in set (0.00 sec)

MYSQL> SELECT  * FROM STUD7;
+--------+----------+----------+
| rollno | name     | grades   |
+--------+----------+----------+
|   5001 | ABILASH  | A1,A2    |
|   5002 | AKSHITHA | A1,B1,C1 |
|   5003 | AKKI     | A1       |
|   5004 | ABI      |          |
+--------+----------+----------+
4 rows in set (0.00 sec)
```

6. DEFAULT:

```
MYSQL> CREATE TABLE STUD8(ROLLNO INT(6) PRIMARY KEY AUTO_INCREMENT, NAME
VARCHAR(10),BRANCH VARCHAR(6) DEFAULT 'IT');
```

```
QUERY OK, 0 ROWS AFFECTED (0.18 SEC)

MYSQL> DESC STUD8;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| rollno | int(6)      | NO   | PRI | NULL    | auto_increment |
| name   | varchar(10) | YES  |     | NULL    |                |
| branch | varchar(6)  | YES  |     | IT      |                |
+--------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

MYSQL> INSERT INTO STUD8(NAME) VALUES('AKSHITH');
QUERY OK, 1 ROW AFFECTED (0.04 SEC)

MYSQL> INSERT INTO STUD8(NAME) VALUES('AMARAVATHI');
QUERY OK, 1 ROW AFFECTED (0.03 SEC)

MYSQL> SELECT * FROM STUD8;
+--------+------------+--------+
| rollno | name       | branch |
+--------+------------+--------+
|      1 | AKSHITH    | IT     |
|      2 | AMARAVATHI | IT     |
+--------+------------+--------+
2 rows in set (0.00 sec)
```

7. CHECK:

```
MYSQL> CREATE TABLE STUDENT1(ROLLNO INT(6) PRIMARY KEY ,
                             NAME VARCHAR(10) UNIQUE,
                             BRANCH VARCHAR(6) DEFAULT 'IT',
                             PERCENTAGE INT(3) CHECK(PERCENTAGE <= 100));
QUERY OK, 0 ROWS AFFECTED (0.30 SEC)

MYSQL> DESC STUDENT1;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| rollno     | int(6)      | NO   | PRI | NULL    |       |
| name       | varchar(10) | YES  | UNI | NULL    |       |
| branch     | varchar(6)  | YES  |     | IT      |       |
| percentage | int(3)      | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

```
MYSQL> INSERT INTO STUDENT1 VALUES(5001, 'ABI', 'CSE',80);
QUERY OK, 1 ROW AFFECTED (0.03 SEC)

MYSQL> INSERT INTO STUDENT1 VALUES(5002, 'ANU',NULL, 90);
QUERY OK, 1 ROW AFFECTED (0.03 SEC)

MYSQL> SELECT * FROM STUDENT1;
+--------+------+--------+------------+
| rollno | name | branch | percentage |
+--------+------+--------+------------+
|   5001 | ABI  | CSE    |         80 |
|   5002 | ANU  | IT     |         90 |
+--------+------+--------+------------+
2 rows in set (0.00 sec)

MYSQL> INSERT INTO STUDENT111 VALUES(5003, 'ANUSHA','IT', 90);
QUERY OK, 1 ROW AFFECTED (0.03 SEC)

MYSQL> SELECT * FROM STUDENT111;
+--------+--------+--------+------------+
| rollno | name   | branch | percentage |
+--------+--------+--------+------------+
|   5001 | ABI    | CSE    |         80 |
|   5002 | ANU    | IT     |         90 |
|   5003 | ANUSHA | IT     |         90 |
+--------+--------+--------+------------+
3 rows in set (0.00 sec)

MYSQL> INSERT INTO STUDENT111 VALUES(5004, 'ABI','CSE', 200);
ERROR: CHECK CONSTRAINT VOILATED.
```

8. FORIGEN KEY:

```
MYSQL> CREATE TABLE STUDENT121(ROLLNO INT(6) PRIMARY KEY ,
                               NAME VARCHAR(10) UNIQUE,
                               BRANCH VARCHAR(6))ENGINE=INNODB;
QUERY OK, 0 ROWS AFFECTED (0.30 SEC)

MYSQL> CREATE TABLE GRADES121(ROLLNO INT(6) REFERENCES STUDENT121(ROLLNO) ,
                              GRADE ENUM('A','B','C','D'))ENGINE=INNODB;
QUERY OK, 0 ROWS AFFECTED (0.19 SEC)

MYSQL> INSERT INTO STUDENT121 VALUES(5001,'ANUSHA','IT');
QUERY OK, 1 ROW AFFECTED (0.03 SEC)

MYSQL> INSERT INTO STUDENT121 VALUES(5002,'AKSHITHA','IT');
QUERY OK, 1 ROW AFFECTED (0.03 SEC)
```

```
MYSQL> INSERT INTO STUDENT121 VALUES(5003,'ANUPAMA','IT');
QUERY OK, 1 ROW AFFECTED (0.03 SEC)

MYSQL> INSERT INTO GRADES121 VALUES(5001,'A');
QUERY OK, 1 ROW AFFECTED (0.03 SEC)

MYSQL> INSERT INTO GRADES121 VALUES(5002,'B');
QUERY OK, 1 ROW AFFECTED (0.04 SEC)

MYSQL> SELECT * FROM STUDENT121;
+--------+----------+--------+
| rollno | name     | branch |
+--------+----------+--------+
|   5001 | ANUSHA   | IT     |
|   5002 | AKSHITHA | IT     |
|   5003 | ANUPAMA  | IT     |
+--------+----------+--------+
3 rows in set (0.00 sec)

MYSQL> SELECT * FROM GRADES121;
+--------+-------+
| rollno | grade |
+--------+-------+
|   5001 | A     |
|   5002 | B     |
+--------+-------+
2 rows in set (0.00 sec)

MYSQL> INSERT INTO GRADES121 VALUES(5005,'C');
ERROR (23000): INTEGRITY CONSTRAINT VIOLATED - PARENT KEY NOT FOUND

MYSQL> DELETE FROM STUDENT121 WHERE ROLLNO=5002;
ERROR (23000): INTEGRITY CONSTRAINT VIOLATED - CHILD RECORD FOUND
```

Practical-6

Aim: Complex SQL Queries using Joins and Nested sub-queries

Theory:

COLLATING INFORMATION:

- Joins are used to combine columns from different tables
- The connection between tables is established through 'where' clause

Types Of Joins:

- Equi Joins (NATURAL JOIN)
- Cartesian Joins
- Outer Joins
- Self Joins

The Syntax for Select statement where we join 2 tables:

SELECT <Select-List>
FROM <table 1>, <table 2>, …….. <table N>
WHERE <table 1.column 1>=<table 2.column 2> and ……..

Set Operators: (Sub Queries)

SET operators are used to combine information of similar type from one or more than one table
Data type of corresponding columns must be the same
The types of SET Operators are:
UNION: Merges output of 2 or more queries into single set of rows & columns

Syntax:
Select<statement 1>
UNION
Select< statement 2>
[ORDER-BY-Clause]

INTERSECT: Returns the rows that are common between 2 sets of rows
Note: Implemented using IN operator in MYSQL.

Syntax:
Select<statement 1>
INTERSECT
Select< statement 2>
[ORDER-BY-Clause]

MINUS: Returns the rows unique to first query
Note: Implemented using NOT IN operator in MYSQL.

Syntax:
Select<statement 1>
MINUS
Select< statement 2>
[ORDER-BY-Clause]

Nested Queries:

1. The result of inner query is dynamically substituted in the condition of outer query.
2. There is no practical limitation to the level of nesting of queries in oracle 8.
3. When using relational operations, ensure the sub query returns a single column output.
4. In some cases, the distinct clause can be used to ensure single valued output.

Correlated sub query:

A correlated sub query is a nested sub query which is executed once for each 'candidate row' considered by the main query and which on execution uses a value from a column in the object query.

In a correlated sub query, the column value used in inner sub query refers to the column value present in the outer query forming a correlated sub query. The sub query is executed repeatedly once for each row of the main (outer) query table.

Using special operators in sub queries:

EXISTS, ANY, SOME, ALL

1. EXISTS: Used to check for the existence of values.
2. ANY, SOME: Compare the lowest value from the set
3. ALL: The predicate is true if every value selected by the sub query satisfies the condition in the predicate of the outer query.

Program:

1) **List the employee numbers, name, dept numbers and the dept name**.

SQL>select empno,ename,emp.deptno,dname from emp,dept where emp.deptno=dept.deptno;

OUTPUT:

EMPNO	ENAME	DEPTNO	DNAME
7369	SMITH	20	RESEARCH
7499	ALLEN	30	SALES
7521	WARD	30	SALES
7566	JONES	20	RESEARCH
7654	MARTIN	30	SALES
7698	BLAKE	30	SALES
7782	CLARK	10	ACCOUNTING
7788	SCOTT	20	RESEARCH
7839	KING	10	ACCOUNTING
7844	TURNER	30	SALES
7876	ADAMS	20	RESEARCH

```
    7900 JAMES          30              SALES
    7902 FORD           20              RESEARCH
    7934 MILLER         10              ACCOUNTING
14 rows selected.
```

2) Display the list of employees working in each dept and display the dept information even if no employee belongs to that department.

mysql> select empno, ename, emp.deptno, sal, dname,loc from emp left outer join dept using (deptno);

```
+-------+--------+--------+---------+------------+---------+
| empno | ename  | deptno | sal     | dname      | loc     |
+-------+--------+--------+---------+------------+---------+
|  7782 | CLARK  |     10 | 2450.00 | ACCOUNTING | NEWYORK |
|  7839 | KING   |     10 | 5000.00 | ACCOUNTING | NEWYORK |
|  7934 | MILLER |     10 | 1300.00 | ACCOUNTING | NEWYORK |
|  7369 | SMITH  |     20 |  800.00 | RESEARCH   | DALLAS  |
|  7566 | JONES  |     20 | 2975.00 | RESEARCH   | DALLAS  |
|  7788 | SCOTT  |     20 | 3000.00 | RESEARCH   | DALLAS  |
|  7876 | ADAMS  |     20 | 1100.00 | RESEARCH   | DALLAS  |
|  7902 | FORD   |     20 | 3000.00 | RESEARCH   | DALLAS  |
|  7499 | ALLEN  |     30 | 1600.00 | SALES      | CHICAGO |
|  7521 | WARD   |     30 | 1250.00 | SALES      | CHICAGO |
|  7698 | BLAKE  |     30 | 2850.00 | SALES      | CHICAGO |
|  7844 | TURNER |     30 | 1500.00 | SALES      | CHICAGO |
|  7900 | JAMES  |     30 |  950.00 | SALES      | CHICAGO |
|  7654 | MARTIN |   NULL | 1250.00 | NULL       | NULL    |
+-------+--------+--------+---------+------------+---------+
14 rows in set (0.04 sec)
```

mysql> select empno, ename, emp.deptno, sal, dname,loc from emp left outer join dept on emp.deptno= dept.deptno;

```
+-------+--------+--------+---------+------------+---------+
| empno | ename  | deptno | sal     | dname      | loc     |
+-------+--------+--------+---------+------------+---------+
|  7782 | CLARK  |     10 | 2450.00 | ACCOUNTING | NEWYORK |
|  7839 | KING   |     10 | 5000.00 | ACCOUNTING | NEWYORK |
|  7934 | MILLER |     10 | 1300.00 | ACCOUNTING | NEWYORK |
|  7369 | SMITH  |     20 |  800.00 | RESEARCH   | DALLAS  |
|  7566 | JONES  |     20 | 2975.00 | RESEARCH   | DALLAS  |
|  7788 | SCOTT  |     20 | 3000.00 | RESEARCH   | DALLAS  |
|  7876 | ADAMS  |     20 | 1100.00 | RESEARCH   | DALLAS  |
|  7902 | FORD   |     20 | 3000.00 | RESEARCH   | DALLAS  |
|  7499 | ALLEN  |     30 | 1600.00 | SALES      | CHICAGO |
```

```
| 7521 | WARD   |    30 | 1250.00 | SALES      | CHICAGO |
| 7698 | BLAKE  |    30 | 2850.00 | SALES      | CHICAGO |
| 7844 | TURNER |    30 | 1500.00 | SALES      | CHICAGO |
| 7900 | JAMES  |    30 |  950.00 | SALES      | CHICAGO |
| 7654 | MARTIN |  NULL | 1250.00 | NULL       | NULL    |
+-------+--------+--------+---------+------------+---------+
14 rows in set (0.00 sec)
```

mysql> select empno, ename,sal, dept.deptno, dname,loc from dept right outer join emp on emp.deptno= dept.deptno;

```
+-------+--------+---------+--------+------------+---------+
| empno | ename  | sal     | deptno | dname      | loc     |
+-------+--------+---------+--------+------------+---------+
| 7782 | CLARK  | 2450.00 |    10 | ACCOUNTING | NEWYORK |
| 7839 | KING   | 5000.00 |    10 | ACCOUNTING | NEWYORK |
| 7934 | MILLER | 1300.00 |    10 | ACCOUNTING | NEWYORK |
| 7369 | SMITH  |  800.00 |    20 | RESEARCH   | DALLAS  |
| 7566 | JONES  | 2975.00 |    20 | RESEARCH   | DALLAS  |
| 7788 | SCOTT  | 3000.00 |    20 | RESEARCH   | DALLAS  |
| 7876 | ADAMS  | 1100.00 |    20 | RESEARCH   | DALLAS  |
| 7902 | FORD   | 3000.00 |    20 | RESEARCH   | DALLAS  |
| 7499 | ALLEN  | 1600.00 |    30 | SALES      | CHICAGO |
| 7521 | WARD   | 1250.00 |    30 | SALES      | CHICAGO |
| 7698 | BLAKE  | 2850.00 |    30 | SALES      | CHICAGO |
| 7844 | TURNER | 1500.00 |    30 | SALES      | CHICAGO |
| 7900 | JAMES  |  950.00 |    30 | SALES      | CHICAGO |
| 7654 | MARTIN | 1250.00 |  NULL | NULL       | NULL    |
+-------+--------+---------+--------+------------+---------+
14 rows in set (0.00 sec)
```

3) **List the employee names and their corresponding manager names**.

SQL> select e.ename,m.ename from emp e,emp m where e.mgr=m.empno;

OUTPUT:

```
ENAME      ENAME
---------- ----------
FORD       JONES
SCOTT      JONES
JAMES      BLAKE
TURNER     BLAKE
MARTIN     BLAKE
WARD       BLAKE
ALLEN      BLAKE
MILLER     CLARK
```

ADAMS SCOTT
CLARK KING
BLAKE KING
JONES KING
SMITH FORD
13 rows selected.

4) **List all employees who joined the company before their manager**.

SQL> select e.ename,m.ename from emp e,emp m where e.mgr=m.empno and e.hiredate< m.hiredate;

OUTPUT:

ENAME ENAME
---------- ----------
WARD BLAKE
ALLEN BLAKE
CLARK KING
BLAKE KING
JONES KING
SMITH FORD
6 rows selected.

5) List all the different designations in dept 20 & 30.

SQL> select distinct job from emp where deptno=20

Union

select distinct job from emp where deptno=30;

OUTPUT:

JOB

ANALYST
CLERK
MANAGER
SALESMAN

6) **List jobs common to dept 20 & 30**.

mysql> select distinct job from emp where deptno=20 and job in (select job from emp where deptno=30);

+---------+

```
| job     |
+---------+
| CLERK   |
| MANAGER |
+---------+
2 rows in set (0.00 sec)
```

7) **List jobs unique to deptno=20**.

```
mysql> select distinct job from emp where deptno=20 and job not in ( select job from emp where deptno=30);

+---------+
| job     |
+---------+
| ANALYST |
+---------+
1 row in set (0.01 sec)
```

8) **Display all empno's and ename who work for dept 10,30**.

```
SQL> select empno,ename from emp where deptno=10
                    Union
          select empno,ename from emp where deptno=30;
```

OUTPUT:

```
    EMPNO ENAME
---------- ----------
     7499 ALLEN
     7521 WARD
     7654 MARTIN
     7698 BLAKE
     7782 CLARK
     7839 KING
     7844 TURNER
     7900 JAMES
     7934 MILLER

9 rows selected.
```

9) **List the employees belonging to the department of 'miller'**.

SQL> select empno,ename from emp where deptno=(select deptno from emp where ename ='MILLER');

OUTPUT:

```
    EMPNO ENAME
---------- ----------
   7782      CLARK
   7839       KING
   7934     MILLER
```

3 rows selected.

10) **List the names of employees drawing the highest salary**.

SQL> select ename from emp where sal=(select max(sal) from emp);

OUTPUT:

```
ENAME
----------
KING
```

1 row selected.

11) **Find all the emp details whose salary is > avg sal of emp whose hiredate is before '01-04-81'**.

SQL> select ename,sal from emp where sal>(select avg(sal) from emp where hiredate<'01-Apr-81');

OUTPUT:

```
ENAME           SAL
---------- ----------
ALLEN          1600
WARD           1250
JONES          2975
MARTIN         1250
BLAKE          2850
CLARK          2450
SCOTT          3000
KING           5000
TURNER         1500
FORD           3000
MILLER         1300
```

11 rows selected.

12) List the job with highest avg sal.

```
mysql> create temporary table avgsal as select avg(sal) as avgsalvalue from emp group by job;
Query OK, 5 rows affected (0.21 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> create temporary table highavgsal as select max(avgsalvalue) from avgsal;
Query OK, 1 row affected (0.14 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> select job, sal from emp where sal = ( select * from highavgsal1);
+-----------+---------+
| job       | sal     |
+-----------+---------+
| PRESIDENT | 5000.00 |
+-----------+---------+
1 row in set (0.00 sec)
```

13) **Find the details of dept whose managers emp code is '7698'**.

```
SQL> select * from dept where deptno=(select distinct deptno from emp where mgr= '7698');
```

OUTPUT:

```
   DEPTNO DNAME        LOC

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

      30 SALES        CHICAGO
```

14) **List the names of the employees who earns lowest salary in each dept**.

```
SQL> select ename,sal,deptno from emp where sal IN(select min(sal) from emp group by deptno);
```

OUTPUT:

```
ENAME           SAL     DEPTNO
---------- ---------- ----------
SMITH           800        20
JAMES           950        30
MILLER         1300         10
```

15) **List the emp details who earn sal>avg sal for their dept**.

SQL> select empno,ename,sal,deptno from emp e where sal>(select avg(sal) from emp where deptno=e.deptno);

OUTPUT:

```
    EMPNO ENAME          SAL     DEPTNO
---------- ---------- ---------- ----------
     7499 ALLEN          1600         30
     7566 JONES          2975         20
     7698 BLAKE          2850         30
     7788 SCOTT          3000         20
     7839 KING           5000         10
     7902 FORD           3000         20
6 rows selected.
```

16) **List emp details with dept name and whose salary is greater than 2000**.

SQL> select empno, ename, e.deptno, dname from emp e, dept where e.deptno=dept.deptno and sal>2000;

OUTPUT:

```
    EMPNO ENAME      DEPTNO  DNAME
    ---------- ---------- ---------  ----------
     7566 JONES      20          RESEARCH
     7698 BLAKE      30          SALES
     7782 CLARK      10          ACCOUNTING
     7788 SCOTT      20          RESEARCH
     7839 KING       10          ACCOUNTING
     7902 FORD       20          RESEARCH
6 rows selected.
```

17) **List all the employees who have atleast 1 person reporting to them**.

SQL> select empno,ename,job,deptno from emp e where EXISTS(select empno from emp where emp.mgr=e.empno) order by empno;

OUTPUT:

```
    EMPNO ENAME      JOB         DEPTNO
---------- ---------- --------- ----------
```

7566 JONES MANAGER 20

7698 BLAKE MANAGER 30

7782 CLARK MANAGER 10

7788 SCOTT ANALYST 20

7839 KING PRESIDENT 10

7902 FORD ANALYST 20

6 rows selected

18) **List the emp details iff more than 5 employees are present in deptno 10**.

SQL> select * from emp where deptno=10 and EXISTS (select * from emp where deptno=10 group by deptno having count(*)>5);

OUTPUT:

no rows selected

19) **List all the emp details who do not manage any one**.

SQL> select ename,job from emp e where not exists(select mgr from emp where mgr=e.empno);

OUTPUT:

ENAME JOB

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

TURNER SALESMAN

WARD SALESMAN

MARTIN SALESMAN

ALLEN SALESMAN

MILLER CLERK

SMITH CLERK

ADAMS CLERK

JAMES CLERK

8 rows selected.

20) List emp names whose salary is greater than the lowest salary of the employee belonging to the dept no 20.

SQL> select ename from emp where sal>any(select sal from emp where deptno=20);

OUTPUT:

ENAME

KING
FORD
SCOTT
JONES
BLAKE
CLARK
ALLEN
TURNER
MILLER
WARD
MARTIN
ADAMS
JAMES
13 rows selected.

21) List the emp details of those emp whose salary is greater than any of the manager.

SQL> select empno,ename,sal from emp where sal>any(select sal from emp where job ='manager');

OUTPUT:

EMPNO	ENAME	SAL
7566	JONES	3175
7698	BLAKE	2850
7788	SCOTT	5000
7782	CLARK	2450
7839	KING	5000
7902	FORD	3000

22) List emp names whose salary is greater than the highest salary of the employee belonging to the dept no 20.

SQL> select ename from emp where sal>all(select sal from emp where deptno=20);

OUTPUT:

ENAME

KING

23) **List the details of the employees earning more than the highest paid manager**.

```
SQL> select empno,ename,sal from emp where sal>all(select sal from emp where  job ='manager');

     EMPNO ENAME          SAL
---------- ---------- ----------
      7788 SCOTT          3000
      7839 KING           5000
      7902 FORD           3000
    3 rows selected.
```

Practical-7

Aim: Table creation using tables, Views, Authorizations

Theory:

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

The WITH CHECK OPTION:

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

Updating a View:

A view can be updated under certain conditions:

- ✓ The SELECT clause may not contain the keyword DISTINCT.
- ✓ The SELECT clause may not contain summary functions.
- ✓ The SELECT clause may not contain set functions.
- ✓ The SELECT clause may not contain set operators.
- ✓ The SELECT clause may not contain an ORDER BY clause.
- ✓ The FROM clause may not contain multiple tables.
- ✓ The WHERE clause may not contain subqueries.
- ✓ The query may not contain GROUP BY or HAVING.

Program:

1) **Create a table emp2 from emp table having empno, name , job , mgr, sal, comm of emp of dept 10 and 20 only**.

SQL> create table emp2 as select empno,ename,job,mgr,sal from emp where deptno in(10,20);

OUTPUT: Table created.

2) **Create a table salary from emp having ename, sal**.

SQL> create table salary(emp_name,emp_sal) as select ename,sal from emp;

OUTPUT: Table created.

3) **Clone the table salary**

Mysql> create table salary1 like salary;

OUTPUT: Table created.

4) Insert a row into emp2 table

SQL> insert into emp2 values(1254,'abc','manager',5869,2356);

5) Insert data into salary1 table from salary table;

Mysql> insert into salary1 select * from salary;

6) Using update , Increase salary by 10%.

SQL> update emp2 set sal=sal*1.10

OUTPUT:

9 rows updated.

7) Give everybody a commission of 500/-.

SQL> update emp set comm=comm+500;

OUTPUT: 14 ROW updated.

8) Change the department of king to 40.

SQL> update emp2 set deptno=40 where ename='KING' ;

OUTPUT: 1 row updated.

9) Delete all records of clerks from emp2.

SQL> delete from emp2 where job='CLERK';

OUTPUT: 1 rows deleted.

10) Delete all records of emp2.

SQL> delete from emp2;

OUTPUT: 6 rows deleted.

11) Alter the table emp2 to add a column to emp2, which will hold the grade for each employee.

SQL> alter table emp2 add grade varchar(2);

OUTPUT: Table altered.

12) Alter the emp2 table to delete the grade column.

SQL> alter table emp2 drop column grade;

OUTPUT: Table altered.

13) Delete the table emp2.

SQL> drop table emp2;

OUTPUT: Table dropped.

14) Create a view of emp belonging to dept 20.

SQL> create view emp20 as select empno,ename,sal,deptno from emp where deptno=20;

OUTPUT: View created.

15) Create a view empview, which will contain the empno, ename, sal, deptno and dept name.

SQL> create view empview as select empno,ename,sal,emp.deptno,dname from emp,dept where emp.deptno=dept.deptno;

OUTPUT: View created.

16) Create a view empsal, which contains empno, ename, net salary.

SQL> create or replace view empsal(empno,ename,net_sal) as select empno,ename,sal+comm from emp;

OUTPUT: View created.

17) Create a view dept-tot (deptno, no-of-emp) which contains deptno and number of employees in that dept. (from emp table only).

SQL> create or replace view dept_tot(deptno,no_of_emp) as select deptno,count(*) from emp group by deptno;

OUTPUT: View created.

18) Create a view empdept30 which contains empno,ename, deptno of employees belonging to dept 30 with check option.

SQL> create or replace view empdept30 as select empno,ename,deptno from emp where deptno=30 with check option;

OUTPUT: View created.

19) Create emp_mgr view with empname and his manager name with check option and try to delete any record from it.

SQL> create or replace view emp_mgr(empname,mgrname) as select e.ename,m.ename from emp e, emp m where e.mgr=m.empno with check option;

OUTPUT: View created.

20) Insert values into emp_dept view.

SQL> insert into emp_dept(ename,empno,deptno) values ('KURODA',9010,40);

OUTPUT: Values inserted

21) Delete the records of smith from emp_mgr view.

SQL>delete from emp_mgr where ename='smith';

OUTPUT: Table deleted;

22) Delete the emp_dept view.

SQL>drop view emp_dept;

OUTPUT: View deleted

Practical -8:

Aim: Authorizations and TCL commands

Program:

1) Grant all permissions on the emp table to everybody (public).

```
mysql> grant all on emp to public;
Query OK, 0 rows affected (0.01 sec)
```

2) Grant select permission on the emp table to all user with further grant option.

```
mysql> grant select on emp to public with grant option;
Query OK, 0 rows affected (0.00 sec)
```

3) Grant only select to everybody on column empno, ename and sal of emp table.

Step:1 create view emp12 as select empno, ename, sal from emp;

Step:2 grant select on emp12 to public

```
mysql> create view emp12 as select empno,ename,sal from emp;
Query OK, 0 rows affected (0.03 sec)

mysql> grant select on emp12 to public;
Query OK, 0 rows affected (0.00 sec)
```

4) Grant all privileges to everybody on column empno, ename and sal of dept no 20 only of emp table.

```
mysql> create view emp123 as select empno,ename,sal from emp where deptno=20;
Query OK, 0 rows affected (0.03 sec)

mysql> grant all on emp123 to public;
```

Query OK, 0 rows affected (0.00 sec)

5) Revoke update privilege from everybody on the emp table.

mysql> revoke update on emp from public;

Query OK, 0 rows affected (0.00 sec)

6) Create a unique index for the emp table on empno.

mysql> create unique index empindex on emp(empno);

Query OK, 0 rows affected (0.38 sec)

Records: 0 Duplicates: 0 Warnings: 0

7) Create a index on deptno and dname columns of the dept table.

mysql> create unique index deptindex on dept(deptno,dname);

Query OK, 0 rows affected (0.19 sec)

Records: 0 Duplicates: 0 Warnings: 0

8) Delete the index deptno.

mysql> drop index deptindex on dept;

Query OK, 0 rows affected (0.12 sec)

Records: 0 Duplicates: 0 Warnings: 0

9) Example for Commit, Rollback and Save points.

mysql> set autocommit=false;

Query OK, 0 rows affected (0.00 sec)

mysql> desc studreg;

```
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| sid    | int(6)      | YES  |     | NULL    |       |
| sname  | varchar(10) | YES  |     | NULL    |       |
| branch | varchar(5)  | YES  |     | NULL    |       |
| age    | int(2)      | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> insert into studreg values (121,'AKSHITHA','IT',20);
Query OK, 1 row affected (0.02 sec)
mysql> select * from studreg;
+------+----------+--------+------+
| sid  | sname    | branch | age  |
+------+----------+--------+------+
|    1 | sowjanya | cse    |   30 |
|  121 | AKSHITHA | IT     |   20 |
+------+----------+--------+------+
2 rows in set (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.04 sec)
mysql> select * from studreg;
+------+----------+--------+------+
| sid  | sname    | branch | age  |
+------+----------+--------+------+
|    1 | sowjanya | cse    |   30 |
```

```
+------+----------+--------+------+

1 row in set (0.00 sec)


mysql> insert into studreg values (121,'AKSHITHA','IT',20);

Query OK, 1 row affected (0.00 sec)

mysql> select * from studreg;

+------+----------+--------+------+
| sid  | sname    | branch | age  |
+------+----------+--------+------+
|    1 | sowjanya | cse    |   30 |
|  121 | AKSHITHA | IT     |   20 |
+------+----------+--------+------+
2 rows in set (0.00 sec)

mysql> commit;

Query OK, 0 rows affected (0.04 sec)

mysql> rollback;

Query OK, 0 rows affected (0.00 sec)

mysql> select * from studreg;

+------+----------+--------+------+
| sid  | sname    | branch | age  |
+------+----------+--------+------+
|    1 | sowjanya | cse    |   30 |
|  121 | AKSHITHA | IT     |   20 |
+------+----------+--------+------+
2 rows in set (0.00 sec)


mysql> savepoint s1;

Query OK, 0 rows affected (0.00 sec)

mysql> insert into studreg values (123,'AKKI','IT',20);

Query OK, 1 row affected (0.00 sec)

mysql> insert into studreg values (124,'ANU','IT',20);

Query OK, 1 row affected (0.00 sec)

mysql> insert into studreg values (125,'PRATIMA','IT',20);
```

```
Query OK, 1 row affected (0.00 sec)

mysql> select * from studreg;

+------+----------+--------+------+
| sid  | sname    | branch | age  |
+------+----------+--------+------+
|    1 | sowjanya | cse    |   30 |
|  121 | AKSHITHA | IT     |   20 |
|  123 | AKKI     | IT     |   20 |
|  124 | ANU      | IT     |   20 |
|  125 | PRATIMA  | IT     |   20 |
+------+----------+--------+------+
5 rows in set (0.00 sec)
mysql> rollback ;

Query OK, 0 rows affected (0.03 sec)

mysql> select * from studreg;

+------+----------+--------+------+

| sid  | sname    | branch | age  |

+------+----------+--------+------+

|    1 | sowjanya | cse    |   30 |

|  121 | AKSHITHA | IT     |   20 |

+------+----------+--------+------+

2 rows in set (0.00 sec)


mysql> set autocommit=1;

Query OK, 0 rows affected (0.00 sec)
```

Practical-9

PL/SQL PROGRAMS

Aim: Usage of Stored Procedures and Functions

Theory:

Procedure: A procedure is a subprogram that performs a specific action, a procedure can be called from any Pl/Sql program procedures can also be invoked from the SQL prompt

A procedure has 2 parts

1)Specification

2)Body

Syntax for procedure:

Create

[DEFINER = { **user** | CURRENT_USER }]

Procedure <procedure name> ([mode] < arg1> <datatype>,….)

Begin

[local declarations]

Pl/Sql executable stmt

End [proc_name]

Note: Mode indicates the type of arguments it can be IN,OUT,INOUT
Calling a procedure:

A Procedure is called as PL/SQL stmt, it can be called from any PL/SQL program by giving their names followed by parameters.

Syntax: call procedure_name (param-list);

Function: A Function is a subprogram that returns a value, a function must have the return clause

Syntax for function:

CREATE

[DEFINER = { **user** | CURRENT_USER }]

FUNCTION <function name> (< arg1> <datatype>,….)

RETURNS returndatatype

BEGIN

[local declarations]

Pl/Sql executable stmt

End [function_name]

Return statement immediately completes the execution of a subprogram and returns control to the calling program, execution then resumes with the statement following the subprogram call.

Syntax for deleting the procedure /function:

DROP PROCEDURE IF EXISTS proc_name / function_name;

1. **Write a Stored Procedure to display Fibonacci series**

Fibonacci series

```
DELIMITER $$
CREATE  PROCEDURE `FIBONACCI`(IN n INT)
BEGIN
DECLARE a int default 0;
DECLARE b int default 1;
DECLARE c int;
DECLARE i int default 1;
DECLARE fs varchar(50) default ' ';
select 'Fibonacci Series:';
set fs= concat(fs, a,' , ');
set fs= concat(fs,b,' , ');
while i < n-2 do
set c=a+b;
set fs= concat(fs,c,' , ');
set a=b;
set b=c;
set i=i+1;
end while;
select fs;
END
```

OUTPUT:

```
SQL script successfully applied.
mysql > call FIBONACCI (10);
Fibonacci Series: 0,1,1,2,3,,8,13,21,34
```

2. **Write a Stored Procedure to check whether a given number is ARMSTRONG or not**

Armstrong Number

```
DELIMITER $$
```

```
CREATE  PROCEDURE `armstrong'(in a int)
BEGIN
declare t int default 0;
declare arm int default 0;
declare d int default 0;
set t=a;
while t>0 do
set d=mod(t,10);
set arm=arm+power(d,3);
set t=truncate(t/10,0);
end while;
if arm=a then
select a, 'is an armstrong number.';
else
select a,' is not an armstrong number.';
end if;
END
```

OUTPUT:

SQL script successfully applied.

mysql > call Armstrong(153);

153 is an armstrong number.

mysql > call Armstrong(125);

125 is not an armstrong number.

3. Write a Stored Procedure to check whether a given number is PRIME or not

```
DELIMITER $$
CREATE  PROCEDURE `isprime`(in a int)
BEGIN
declare c int default 0;
declare i int default 1;
while i<a do
if mod(a,i)=0 then set c=c+1;
end if;
set i=i+1;
end while;
if c<2 then
select a, '- is a prime number. ';
else
select a, ' - is not a prime number.';
```

```
end if;
END
```

OUTPUT:

SQL script successfully applied.

```
mysql > call isprime(11);
11 - is a prime number.
mysql > call isprime(10);
10 - is not a prime number.
```

4. Write a Function to find Factorial of a given number

```
DELIMITER $$
CREATE  FUNCTION `factorial`(a int) RETURNS int(11)
BEGIN
declare i int default 1;
declare f int default 1;
while i<=a do
set f=f*i;
set i=i+1;
end while;
RETURN f;
END
```

OUTPUT:

SQL script successfully applied.

```
mysql > select  'Factorial of 5 is : ', factorial(5) ;

Factorial of 5 is :  120.
```

5. Write a Function to find Reverse of a given number

Reverse of a given number

```
DELIMITER $$
CREATE FUNCTION `reverseofnum` (a int)
RETURNS INTEGER
BEGIN
declare rev int default 0;
declare d int;
while a > 0 do
set d = mod(a,10);
set rev= (rev*10) + d;
set a=trucate(a/10,0);
end while;
RETURN  rev;
END
```

OUTPUT:
SQL script successfully applied.
mysql > select 'Reverse of number 123 is : ', reverseofnum(123) ;
Reverse of number 123 is : 321

6. **Write a Stored Procedure to insert a row into a given table**

Inserting row into Account table

```
Create table A ccount(acc_no int, acc_name varchar(20), acc_bal float(7,2));
DELIMITER $$
CREATE PROCEDURE `rowinsert`(a int, b varchar(20), c float(7,2))
BEGIN
insert into student values(a,b,c) ;
END
```

OUTPUT:
SQL script successfully applied.
mysql > call rowinsert (121,'ANUSHA', 2000.00);
Query OK, 1 row affected (0.00 sec)

7. **Write a Function to return the count of number of employees for given job.**

Count of employees

```
DELIMITER $$
CREATE FUNCTION `countof`(a varchar(20)) RETURNS int(11)
BEGIN
declare c int default 0;
select count(*) into c from EMP where job=a;
RETURN c;
END
```

OUTPUT:
SQL script successfully applied.
mysql > select 'Number of clerks in employee table are : ', countof('CLERK') ;
Number of clerks in employee table are : 4

8. **Write a procedure to return the number of rows in employee table.**

Count of employees

```
DELIMITER $$
CREATE PROCEDURE `count1`(OUT a int)
BEGIN
select count(*) into a from EMP;
END
```

OUTPUT:
SQL script successfully applied.
mysql > call count1(@a);

```
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @a;
+------+
| @a   |
+------+
| 14   |
+------+
```

9. **Write a procedure to update a row of salary table.**

Update Salary

```
Create table salary select empno,ename,sal,comm.,deptno from EMP;
DELIMITER $$
CREATE  PROCEDURE `salupdate`(a int(6))
BEGIN
update salary set sal=3000 where empno=a and sal<3000;
END
```

OUTPUT:

```
SQL script successfully applied.
mysql > call salupdate(7798);
Query OK, 1 rows affected (0.00 sec)
```

10. Update the commission of employees using salary table

Update Commission

```
DELIMITER $$

CREATE  PROCEDURE `updatecomm`(in a int(4))
BEGIN
declare vcom float(7,2);
Select comm into vcom from salary where empno=a;
If vcom is NULL then
Update salary set comm=300 where empno=a;
Else
Update salary set comm=comm*1.25 where empno=a;
End if;
END
```

OUTPUT:

```
SQL script successfully applied.
mysql > call updatecomm(7798);
Query OK, 1 rows affected (0.00 sec)
```

Practical-10

PL/SQL PROGRAMS

Aim: Usage of Cursors

Theory:

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

- Implicit cursors
- Explicit cursors

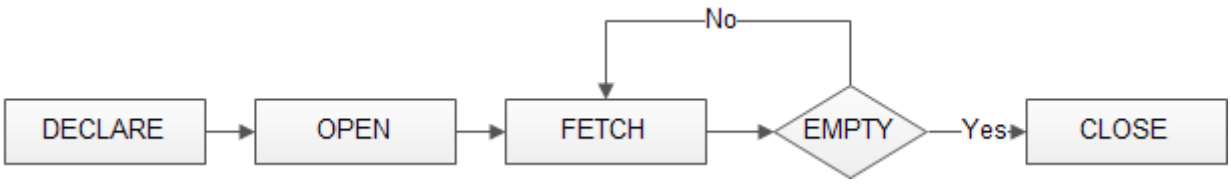
Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

Explicit Cursors

Explicit cursors are programmer defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.



Program:

1. **Display the employee details for a given job using cursors.**

Displaying Employee details using cursor

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `empdetails`(a varchar(20))
BEGIN
declare eno int;
declare nam varchar(20);
declare sal int;
declare done bool default false;
declare n int default 1;
declare c1 cursor for select empno,ename,sal from EMP where job=a ;
declare continue handler for not found set done = true;
open c1;
read_loop: loop
fetch c1 into eno,nam,sal;
if done then leave read_loop;
end if;
select eno,'   ',nam,'   ',sal;
end loop;
close c1;
END
```

OUTPUT:

```
SQL script successfully applied.
mysql> call empdetails('clerk');
+------+---+-------+---+------+
| eno |   | nam   |   | sal  |
```

```
+------+---+-------+---+------+
| 7369 |   | SMITH |   | 800 |
+------+---+-------+---+------+
1 row in set (0.00 sec)

+------+---+-------+---+------+
| eno  |   | nam   |   | sal  |
+------+---+-------+---+------+
| 7876 |   | ADAMS |   | 1100 |
+------+---+-------+---+------+
1 row in set (0.00 sec)

+------+---+-------+---+------+
| eno  |   | nam   |   | sal  |
+------+---+-------+---+------+
| 7900 |   | JAMES |   |  950 |
+------+---+-------+---+------+
1 row in set (0.00 sec)

+------+---+--------+---+------+
| eno  |   | nam    |   | sal  |
+------+---+--------+---+------+
| 7934 |   | MILLER |   | 1300 |
+------+---+--------+---+------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

2. **Write a procedure using cursors that will pay annual interest of 10% to accounts with balance less than 10000 and 15% otherwise. Create the necessary tables.**

Paying Interest:

```
DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `payinterest`()

BEGIN

declare bal float(7,2);

declare done bool default false;

declare c1 cursor for select acc_bal from account ;

declare continue handler for not found set done =true;

open c1;
```

```
read_loop : loop

fetch c1 into bal;

if done then leave read_loop;

end if;

if bal < 10000 then

update account set acc_bal=acc_bal*1.1;

else update accout set acc_bal=acc_bal*1.15;

end if;

end loop;

close c1;

END
```

Output:

```
mysql> create table account(accno int(6), name varchar(10), acc_bal float(7,2));
Query OK, 0 rows affected (0.25 sec)

mysql> insert into account values (101,'AKSHITH',5000);
Query OK, 1 row affected (0.03 sec)

mysql> insert into account values (102,'ANUSHA',15000);
Query OK, 1 row affected (0.02 sec)

mysql> insert into account values (103,'SRIKANTH',12000);
Query OK, 1 row affected (0.11 sec)

mysql> insert into account values (104,'SHYAM',8000);
Query OK, 1 row affected (0.04 sec)

mysql> select * from account;


+-------+----------+----------+
| accno | name     | acc_bal  |
+-------+----------+----------+
|   101 | AKSHITH  |  5000.00 |
|   102 | ANUSHA   | 15000.00 |
|   103 | SRIKANTH | 12000.00 |
|   104 | SHYAM    |  8000.00 |
+-------+----------+----------+
```

```
4 rows in set (0.00 sec)

mysql> call payinterest();
Query OK, 0 rows affected (0.10 sec)

mysql> select * from account;
+-------+----------+----------+
| accno | name     | acc_bal  |
+-------+----------+----------+
|   101 | AKSHITH  |  8801.23 |
|   102 | ANUSHA   | 26403.71 |
|   103 | SRIKANTH | 21122.97 |
|   104 | SHYAM    | 14081.98 |
+-------+----------+----------+
4 rows in set (0.00 sec)
```

3. List the names of account holders in account table

Name List:

```
DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `accountlist`()

BEGIN

declare accname varchar(10) default ' ' ;

declare acclist varchar(1000) default ' ' ;

declare done bool default false;

declare c1 cursor for select acc_name from account ;

declare continue handler for not found set done =true;

open c1;

read_loop : loop

fetch c1 into accname;

set acclist = concat(accname,' , ',acclist);

if done then leave read_loop;

end if;

end loop;

close c1;
```

```
select 'Names of account holders : ',acclist;

END
```

Output:

```
mysql> call accountlist();
+---------------------------+-------------------------------------------+
|Names of account holders :  |       acclist                            |
+---------------------------+-------------------------------------------+
| Names of account holders :  | SHYAM , SRIKANTH , ANUSHA , AKSHITH        |
+---------------------------+-------------------------------------------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
```

Practical-11

PL/SQL PROGRAMS

Aim: Usage of Triggers

Theory:

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events:

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).

Triggers could be defined on the table, view, schema, or database with which the event is associated

Advantages of triggers:

- Preventing invalid transactions
- Imposing security authorizations
- Enforcing referential integrity

Syntax for creating a trigger is:

```
CREATE  TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[ OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
```

```
WHEN (condition)
BEGIN
  … routine statements
END;
```

1. **Create a trigger for the Student table, which makes the entry in sname column in uppercase**.

```
mysql> create trigger t6 before insert on student for each row set new.sname=UPPER(new.sname);
Query OK, 0 rows affected (0.06 sec)

mysql> select * from student;
+-------+----------+
| sid   | sname    |
+-------+----------+
|    0 | anusha   |
|    1 | bhavana  |
|    2 | divya    |
|    3 | akshitha |
+-------+----------+
4 rows in set (0.00 sec)

mysql> insert into student values (5,'anu');
Query OK, 1 row affected (0.04 sec)

mysql> select * from student;
+-------+----------+
| sid   | sname    |
+-------+----------+
|    0 | anusha   |
|    1 | bhavana  |
|    2 | divya    |
|    3 | akshitha |
|    5 | ANU      |
+-------+----------+
11 rows in set (0.00 sec)
```

2. **Create a trigger on Salary table, which checks the salary amount before updating**

Update salary trigger

```
delimiter $$
```

```
create trigger updatecheck before update on salary

for each row

begin

if new.sal < 0 then

set new.sal=0;

else if new.sal >10000 then

set new.sal = 10000;

end if;

end;
```

OUTPUT:

```
Query OK, 0 rows affected (0.06 sec)


mysql> update salary set sal = 30000 where empno=7369;
Query OK, 1 row affected (0.04 sec)


mysql> update salary set sal = -2000 where empno=7499;
Query OK, 1 row affected (0.04 sec)
mysql> select * from salary;

+-------+--------+----------+---------+--------+
| empno | ename  | sal      | comm    | deptno |
+-------+--------+----------+---------+--------+
|  7369 | SMITH  |  10000   |    NULL |     20 |
|  7499 | ALLEN  |  0       |  300.00 |     30 |
|  7521 | WARD   |  1250.00 |  500.00 |     30 |
  ...
+-------+--------+----------+---------+--------+
15 rows in set (0.00 sec)
```

3. **Create a trigger on Empsalary table, which checks the salary amount before inserting/updating for president and manger**

Update empsalary trigger:

```
mysql> CREATE TABLE empsalary SELECT empno,ename,job,sal,deptno from EMP;

Query OK, 0 rows affected (0.25 sec)



delimiter //
create trigger updatecheck before update on empsalary
for each row
begin
if new.job in ( 'manager' , 'president') and new.sal < 10000 then
signal SQLSTATE '45000' set message_text = 'Cannot perform any DML for this job and salary
values';
end if;
end;


mysql> insert into empsalary values(7777,'abc','manager',3000,10);
ERROR 1644 (45000): Cannot perform any DML for this job and salary values


mysql> insert into empsalary values(7777,'abc','manager',11000,10);
Query OK, 1 rows affected (0.25 sec)
```

Practical-12

PL/SQL PROGRAMS

Aim: Data Validation using the Stored Procedure (Usage of Exceptional Handling)

Theory:

In PL/SQL, a warning or error condition is called an exception.

Exceptions are of 2 types

1) predefined and
2) user defined.

Predefined exceptions are internally defined by run time system and are raised implicitly whenever a pl/sql program violates an oracle rule or exceeds a system dependent limit. Each error has a number but handled by name. they are automatically raised and no need to give an command

When an error occurs inside a stored procedure, it is important to handle it appropriately, such as continuing or exiting the current code block's execution, and issuing a meaningful error message.

MySQL provides an easy way to define handlers that handle from general conditions such as warnings or exceptions to specific conditions e.g., specific error codes.

Declaring a handler:

To declare a handler, you use the DECLARE HANDLER statement as follows:

DECLARE action HANDLER FOR condition_value statement;

If a condition whose value matches the condition_value , MySQL will execute the statement and continue or exit the current code block based on the action .

The action accepts one of the following values:

- CONTINUE : the execution of the enclosing code block (BEGIN … END) continues.
- EXIT : the execution of the enclosing code block, where the handler is declared, terminates.

Program-1 : Write a procedure demonstrating handling of different exceptional conditions (exit handler)

```
DELIMITER $$

CREATE PROCEDURE `insert_aduit1` (in sid1 int, in aid1 int)

BEGIN

declare exit handler for 1062

select ' Error, 1062 EXCEPTION DUPLICATE VALUES NOT ALLOWED';

declare exit handler for sqlexception

select 'SQLEXCEPTION INVOKED, TABLE DOES NOT EXIST';

declare exit handler for SQLSTATE '23000'

select 'SQLSTATE 23000 INTEGRITY CONSTRAINTS VOILATED EXCEPTION INVOKED';

insert into studaudit values(sid1,aid1);

select count(*) from studaudit;

END
```

Output:

SQL script successfully applied.

```
mysql> create table  studaudit (sid int(4) primary key, aid int(4) not null);
Query OK, 0 rows affected (0.25 sec)

mysql> call insert_aduit1(1,2);
1 row in set (0.00 sec)

mysql> call insert_aduit1(1,2);
+-------------------------------------------------+
| Error, 1062 EXCEPTION DUPLICATE VALUES NOT ALLOWED  |
+-------------------------------------------------+
|  Error, 1062 EXCEPTION DUPLICATE VALUES NOT ALLOWED |
+-------------------------------------------------+

mysql> call insert_aduit1(2,null);
+-------------------------------------------------------------+
| SQLSTATE 23000 INTEGRITY CONSTRAINTS VOILATED EXCEPTION INVOKED'|
+-------------------------------------------------------------+
|  SQLSTATE 23000 INTEGRITY CONSTRAINTS VOILATED EXCEPTION INVOKED'|
+-------------------------------------------------------------+
```

mysql> call insert_aduit1(2,3); (after changing the name of table in procedure)

```
+-----------------------------------------+
| SQLEXCEPTION INVOKED, TABLE DOES NOT EXIST |
+-----------------------------------------+
| SQLEXCEPTION INVOKED, TABLE DOES NOT EXIST |
+-----------------------------------------+
```

1 row in set (0.04 sec)

Program-2 : Write a procedure demonstrating handling exceptions using continue handler.

```
DELIMITER $$
CREATE  PROCEDURE `exception2`(sid int , nam varchar(15))
BEGIN
declare continue handler for 1062
begin
select 'EXCEPTION INVOKED, duplicate key occured';
select concat ('duplicate key ( ', sid , nam, ') ...') as msg;
end;
declare exit handler for sqlexception
begin
rollback;
select ' SQL EXCEPTION INVOKED';
end;
insert into stud values(sid,nam);
select 'number of tuples in student table are ' ,count(*) from stud;
END
```

Output:

SQL script successfully applied.

```
mysql> create table  stud (sid int(4) primary key, sname varchar(20) not null);
Query OK, 0 rows affected (0.25 sec)

mysql> call exception2(1,'ANU');
1 row in set (0.00 sec)

mysql> call exception2(1,'x');
+---------------------------------+----------------------------+
| EXCEPTION: duplicate key occured | msg                        |
+---------------------------------+----------------------------+
| EXCEPTION: duplicate key occured | duplicate key ( 1  ,  x) ... |
+---------------------------------+----------------------------+
1 row in set (0.00 sec)

+---------------------------------------+----------+
| number of tuples in student table are  | count(*) |
+---------------------------------------+----------+
| number of tuples in student table are  |      1 |
+---------------------------------------+----------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
mysql> call exception2(2,'abc'); (after changing the name of table in procedure)
+-----------------------+
| SQL EXCEPTION INVOKED  |
+-----------------------+
|  SQL EXCEPTION INVOKED |
+-----------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

3. Write a program to signal an minimum bal exception during insertion of a row if the balance is less than 1000

Minimum balance exception:

```
DELIMITER $$

CREATE  PROCEDURE `min_bal_exception`(accno int(6),aname varchar(10),abal float(7,2))

BEGIN

if abal < 1000 then

signal SQLSTATE '45000'
```

```
set MESSAGE_TEXT = 'Minimum bal exception : Balance should be more than 1000';

else insert into account values (accno,aname,abal);

end if;

END
```

OUTPUT:

```
mysql> call min_bal_exception(106,'AMRITHA',800);
ERROR 1644 (45000): Minimum bal exception : Balance should be more than 1000
mysql> call min_bal_exception(106,'AMRITHA',1800);
Query OK, 1 row affected (0.02 sec)
```

4. **Write a program to signal an Divide by zero exception.**

Divide_by_zero Exception:

```
DELIMITER $$

CREATE  PROCEDURE `divide`(in num int(5), in denum int(5), out result float(5,2))

BEGIN

declare divide_by_zero condition for SQLSTATE '22012';

declare continue handler for divide_by_zero resignal

set message_text= 'Divide by zero exception, Denominator cannot be zero';

if denum = 0 then signal divide_by_zero;

else set result = num/denum;

end if;

END
```

OUTPUT:

```
mysql> call divide(1500,5,@a);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select @a;
+------+
| @a   |
+------+
|  300 |
+------+
1 row in set (0.00 sec)

mysql> call divide(1500,0,@a);
ERROR 1644 (22012): Divide by zero exception, Denominator cannot be zero
```

Practical-14

Aim: Illustrate the concept of table locking.

Theory:

LOCK TABLE Statement Manually lock one or more tables.

Syntax:

For locking table: LOCK TABLES tablename READ [local]/[LOW_PRORITY] WRITE;

For unlocking table: UNLOCK TABLES;

lockmodes:

read – for selecting the data. (A pure SELECT will not lock any rows.)

write – for insertion , updation and deletion of data.

Programs:

1. **Write a program to lock table stud in read and write mode**.

```
mysql> lock tables stud read;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from stud;
+------+----------+
| sid  | sname    |
+------+----------+
|    1 | sunny    |
|    2 | Bunny    |
+------+----------+
2 rows in set (0.00 sec)
```

```
mysql> insert into stud values(3,'ANU');
ERROR 1099 (HY000): Table 'stud' was locked with a READ lock and can't be updated
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)

mysql> lock tables stud write;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from stud;
+------+----------+
| sid  | sname    |
+------+----------+
|    1 | sunny    |
|    2 | Bunny    |
+------+----------+
2 rows in set (0.00 sec)

mysql> insert into stud values(12,'abc');
Query OK, 1 row affected (0.02 sec)

mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)
```

Practical # 15

Aim: MYSQL- JDBC connectivity

Java Program: (example.java)

```
import java.sql.*;

class example

{

public static void main(String args[]) throws ClassNotFoundException, SQLException

{

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

Connection con=DriverManager.getConnection("jdbc:mysql://localhost/test","","");

Statement stmt=con.createStatement();


        // for insertion
```

```
PreparedStatement pstmt = con.prepareStatement("insert into account values(?,?,?)");
pstmt.setInt(1,150);
pstmt.setString(2,"smith");
pstmt.setInt(3,1000);
pstmt.executeUpdate();
                    // for display
ResultSet rs=stmt.executeQuery("select * from account");
while(rs.next())
{
for(int i=1;i<=3;i++)
{

System.out.print(rs.getString(i)+"\t");
}
System.out.println();
}
}
}
//javac example.java
//java -cp  .:/usr/share/java/mysql.jar example
```

Practical- 16

Aim: Creation of forms and reports.

Theory:

Introduction

Use Form Builder to simplify for the creation of data-entry screens, also known as Forms.

Forms are the applications that connect to a database, retrieve information requested by the user, present it in a layout specified by Form designer, and allow the user to modify or add information. Form Builder allows you to build forms quickly and easily.

Program:

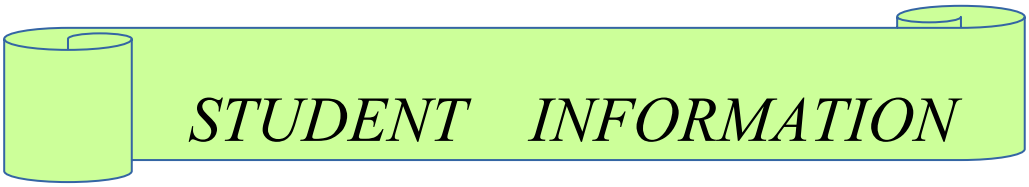
Create a form for student database (Note: create a table with name student with required fields)

Procedure:

1. Start server
2. Open libre office 4.4
3. Set the path for DB connectivity
   Tools -> options -> advanced -> class path -> add archive ->
   file system ->usr ->share->java-> mysql-connector-java.jar ->ok ->ok
4. Come to libre office 4.4 (home)
5. Select base database TO OPEN DATABASE WIZARD
6. Select Connect to an existing database (mysql) ->NEXT
7. Select Connect using jdbc (JAVA DATABASE CONNECTIVITY) ->NEXT
8. Input database name (test)
9. Input server name (localhost)
10. Input PORT NO =3306 ->next
11. GIVE USERNAME, PASSWAORD IF REQUIRED
12. Click Finish and save the database
13. DATA BASE WIZARD IS OPENED.
14 . Select tables and USE CREATE TABLE IN DESIGN VIEW OR USE WIZARD TO CREATE TABLE
15. Create student table by giving appropriate fields and save.
16. Now, select forms
17. Select use wizard to create forms
18. Select a student table from database
19. Select the columns to be included in the form
20. Specify the alignments and other features for the form
21. Select all the options available to modify the form
22. Select modify the form
23. Design the form to include headings and other options

set design mode off and use the form to insert and view the data

OUTPUT: FORM CREATED



Student id

Student name

Branch

Address

Phone

Email

REPORT GENERATION:

REPORT GENERATION AFTER CREATION OF FORM:
1. SELECT USE WIZARD TO CREATE REPORT
2. SELECT THE TABLE AND FIELDS
3. SET THE LABELS FOR FILDS
4. GIVE GROUPING OF COLUMNS IF REQUIRED
5. SET THE ORDER FOR DISPLAY (ASCE/DESC)
6. CHOOSE THE LAYOUT
7. SELECT THE TYPE OF REPORT(STATIC/DYNAMIC) and how you want to proceed (modify or create report)
8. Report is generated, modify it to add header and footer.

OUTPUT:

STUDENT ADDRESS LABELS:

STUDENT ID	1
STUDENT NAME	ANUSHA
BRANCH	IT
Address	KOTI,HYDERABAD
Phone	9888882222
EMAIL	ANUSHA@MVSR.COM
STUDENT ID	2
STUDENT NAME	SAI
BRANCH	IT
Address	KUKATPALLY,HYDERABAD
Phone	8888899999
EMAIL	SAI@MVSR.COM
STUDENT ID	3
STUDENT NAME	KARTHIK
BRANCH	IT
Address	RAMNAGER,HYDERABAD
Phone	7070707070
EMAIL	KARTHIK@MVSR.COM
STUDENT ID	4
STUDENT NAME	AKSHITH
BRANCH	IT
Address	DILSUKNAGER,HYDERABAD
Phone	8083457658
EMAIL	AKSHITH@MVSR.COM
STUDENT ID	5
STUDENT NAME	SWETHA
BRANCH	IT
Address	TARNAKA,SECUNDRABAD
Phone	9701234545
EMAIL	SWETHA@MVSR.COM