Experiment No: 1

Title: Implementation of DDL commands of SQL with suitable examples

- Create table
- Alter table
- Drop Table

Objective:

To understand the different issues involved in the design and implementation of a database system

To understand and use data definition language to write query for a database ***Theory:***

Oracle has many tools such as SQL * PLUS, Oracle Forms, Oracle Report Writer, Oracle Graphics etc.

❖ **SQL * PLUS**: The SQL * PLUS tool is made up of two distinct parts. These are

- **Interactive SQL:** Interactive SQL is designed for create, access and manipulate data structures like tables and indexes.
- **PL/SQL:** PL/SQL can be used to developed programs for different applications.

❖ **Oracle Forms:** This tool allows you to create a data entry screen along with the suitable menu objects. Thus it is the oracle forms tool that handles data gathering and data validation in a commercial application.

❖ **Report Writer:** Report writer allows programmers to prepare innovative reports using data from the oracle structures like tables, views etc. It is the report writer tool that handles the reporting section of commercial application.

❖ **Oracle Graphics:** Some of the data can be better represented in the form of pictures. The oracle graphics tool allows programmers to prepare graphs using data from oracle structures like tables, views etc.

SQL (Structured Query Language):

Structured Query Language is a database computer language designed for managing data in relational database management systems (RDBMS), and originally based upon Relational Algebra. Its scope includes data query and update, schema creation and modification, and data access control. SQL was one of the first languages for Edgar F. Codd's relational model and became the most widely used language for relational databases.

- IBM developed SQL in mid of 1970's.
- Oracle incorporated in the year 1979.
- SQL used by IBM/DB2 and DS Database Systems. SQL adopted as standard language for RDBS by ASNI in 1989.

DATA TYPES:

1. **CHAR (Size):** This data type is used to store character strings values of fixed length. The size in brackets determines the number of characters the cell can hold. The maximum number of character is 255 characters.

2. **VARCHAR (Size) / VARCHAR2 (Size)**: This data type is used to store variable length alphanumeric data. The maximum character can hold is 2000 character.

3. **NUMBER (P, S):** The NUMBER data type is used to store number (fixed or floating point). Number of virtually any magnitude may be stored up to 38 digits of precision.

   Number as large as 9.99 * 10 124. The precision (p) determines the number of places to the right of the decimal. If scale is omitted then the default is zero. If precision is omitted, values are stored with their original precision up to the maximum of 38 digits.

4. **DATE:** This data type is used to represent date and time. The standard format is DD- MM-YY as in 17-SEP-2009. To enter dates other than the standard format, use the appropriate functions. Date time stores date in the 24-Hours format. By default the time in a date field is 12:00:00 am, if no time portion is specified. The default date for a date field is the first day the current month.

5. **LONG:** This data type is used to store variable length character strings containing up to 2GB. Long data can be used to store arrays of binary data in ASCII format. LONG values cannot be indexed, and the normal character functions such as SUBSTR cannot be applied.
6. **RAW:** The RAW data type is used to store binary data, such as digitized picture or image. Data loaded into columns of these data types are stored without any further conversion. RAW data type can have a maximum length of 255 bytes. LONG RAW data type can contain up to 2GB.

SQL language is sub-divided into several language elements, including:

- *Clauses*, which are in some cases optional, constituent components of statements and queries.
- *Expressions,* which can produce either scalar values or tables consisting of columns and rows of data.
- *Predicates* which specify conditions that can be evaluated to SQL three-valued logic

  (3VL) Boolean truth values and which are used to limit the effects of statements and queries, or to change program flow.
- *Queries* which retrieve data based on specific criteria.

- *Statements* which may have a persistent effect on schemas and data, or which may control transactions, program flow, connections, sessions, or diagnostics.
- SQL statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.

- *Insignificant white space* is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

There are five types of SQL statements. They are:

1. DATA DEFINITION LANGUAGE (DDL)
2. DATA MANIPULATION LANGUAGE (DML)
3. DATA RETRIEVAL LANGUAGE (DRL)
4. TRANSATIONAL CONTROL LANGUAGE (TCL)
5. DATA CONTROL LANGUAGE (DCL)

1. DATA DEFINITION LANGUAGE (DDL)

The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project.

Let's take a look at the structure and usage of four basic DDL commands:

I. CREATE II. ALTER III. DROP IV. RENAME

I. CREATE TABLE

CREATE TABLE: This is used to create a new relation (table)

Syntax: CREATE TABLE <table_name> (field_1 data_type(size), field_2 data_type(size), .. .);

Example:

SQL> CREATE TABLE Student (sno NUMBER (3), sname CHAR (10), class CHAR (5));

II. ALTER:

a. **ALTER TABLE ...ADD...:** This is used to add some extra fields into existing relation.

Syntax: ALTER TABLE relation_name ADD (new field_1 data_type(size), new field_2 data_type(size),..);

Example: SQL>ALTER TABLE std ADD (Address CHAR(10));

b. **ALTER TABLE...MODIFY...:** This is used to change the width as well as data type of fields of existing relations.

Syntax: ALTER TABLE relation_name MODIFY (field_1newdata_type(Size), field_2 newdata_type(Size), ... field_newdata_type(Size));

Example: SQL>ALTER TABLE student MODIFY (sname varchar(10), class varchar(5));

c. **ALTER TABLE..DROP ....** This is used to remove any field of existing relations.

Syntax: ALTER TABLE relation_name DROP COLUMN (field_name);

Example:SQL>ALTER TABLE student DROP column (sname);

d. ALTER TABLE..RENAME...: This is used to change the name of fields in existing relations.

Syntax: ALTER TABLE relation_name RENAME COLUMN (OLD field_name) to (NEW field_name);

Example: SQL>ALTER TABLE student CHANGE sname stu_name VARCHAR(20);

III. DROP TABLE: This is used to delete the structure of a relation. It permanently deletes the records in the table.

Syntax: DROP TABLE relation_name;

Example: SQL>DROP TABLE std;

1. RENAME: It is used to modify the name of the existing database object.

Syntax: RENAME TABLE old_relation_name TO new_relation_name;

Example: SQL>RENAME TABLE std TO std1;

LAB PRACTICE ASSIGNMENT:

1. Create a table EMPLOYEE with following schema:
(Emp_no, E_name, E_address, E_ph_no, Dept_no, Dept_name,Job_id , Salary)

```
CREATE TABLE EMPLOYEE (
    Emp_no INT,
    E_name VARCHAR(100),
    E_address VARCHAR(255),
    E_ph_no VARCHAR(20),
    Dept_no INT,
    Dept_name VARCHAR(100),
    Job_id CHAR(5),
    Salary INT
);
```

Field	Type	Null	Key	Default	Extra
Emp_no	int	YES		NULL	
E_name	varchar(100)	YES		NULL	
E_address	varchar(255)	YES		NULL	
E_ph_no	varchar(20)	YES		NULL	
Dept_no	int	YES		NULL	
Dept_name	varchar(100)	YES		NULL	
Job_id	char(5)	YES		NULL	
Salary	int	YES		NULL	

2. Add a new column; HIREDATE to the existing relation.

```
ALTER TABLE EMPLOYEE
ADD HIREDATE DATE;
```

Salary	int	YES
HIREDATE	date	YES

3. Change the datatype of JOB_ID from char to varchar.

```
ALTER TABLE EMPLOYEE
MODIFY COLUMN Job_id VARCHAR(20);
```

Dept_name	varchar(100)	YES
Job_id	varchar(20)	YES
Salary	int	YES

4. Change the name of column/field Emp_no to E_no.

```
ALTER TABLE EMPLOYEE
RENAME COLUMN Emp_no TO E_no;
```

After Table:

E_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Salary	HIREDATE

5. Modify the column width of the job field of emp table

```
ALTER TABLE EMPLOYEE
MODIFY COLUMN Job_id VARCHAR(30);
```

Before:

Dept_name	varchar(100)	YES
Job_id	varchar(20)	YES
Salary	int	YES

After:

Dept_name	varchar(100)	YES
Job_id	varchar(30)	YES
Salary	int	YES

Experiment No:2

Title : Implementation of DML commands of SQL with suitable examples

- Insert table
- Update table
- Delete Table

Objective :

To understand the different issues involved in the design and implementation of a database system

To understand and use data manipulation language to query, update and manage a database

Theory :

DATA MANIPULATION LANGUAGE (DML): The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

1. INSERT 2. UPDATE 3. DELETE

1. INSERT INTO: This is used to add records into a relation. These are three type of INSERT INTO queries which are as

a) Inserting a single record

Syntax: INSERT INTO < relation/table name> (field_1,field_2……field_n)VALUES (data_1,data_2, ... data_n);

Example: SQL>INSERT INTO student(sno,sname,class,address)VALUES (1,'Ravi','M.Tech','Palakol');

```
INSERT INTO student (sno, sname, class, address)
VALUES (1, 'Ravi', 'M.Tech', 'Palakol');
```

Table:

sno	sname	class	address
1	Ravi	M.Tech	Palakol

b) Inserting a single record

Syntax: INSERT INTO < relation/table name>VALUES (data_1,data_2, ... data_n);

Example: SQL>INSERT INTO student VALUES (1,'Ravi','M.Tech','Palakol');

c) Inserting all records from another relation

Syntax: INSERT INTO relation_name_1 SELECT Field_1,field_2,field_n
FROM relation_name_2 WHERE field_x=data;

Example: SQL>INSERT INTO std SELECT sno,sname FROM student
WHERE name = ‘Ramu‘;

std table:

sno	sname	class	address
2	Ramu	M.Tech	India

After:

```
INSERT INTO stu (sno, sname)
SELECT sno, sname
FROM student
WHERE sname = 'Ravi';
```

Table student:

sno	sname	class	address
2	Ramu	M.Tech	India
1	Ravi	NULL	NULL

d) Inserting multiple records

Syntax: INSERT INTO relation_name field_1,field_2, ... field_n) VALUES
(&data_1,&data_2, ... &data_n);

Example: SQL>INSERT INTO student (sno, sname, class,address)
VALUES (&sno,’&sname’,’&class’,’&address’);

Enter value for sno: 101
Enter value for name: Ravi
Enter value for class: M.Tech
Enter value for name: Palakol

```
INSERT INTO student (sno, sname, class, address)
VALUES
    (101, 'Ravi', 'M.Tech', 'Palakol'),
    (102, 'Ramu', 'B.Tech', 'Hyderabad'),
    (103, 'Sita', 'M.Sc', 'Chennai'),
    (104, 'Gita', 'B.Sc', 'Mumbai'),
    (105, 'Hari', 'MCA', 'Delhi');
```

After insertion of multiple records:

sno	sname	class	address
101	Ravi	M.Tech	Palakol
102	Ramu	B.Tech	Hyderabad
103	Sita	M.Sc	Chennai
104	Gita	B.Sc	Mumbai
105	Hari	MCA	Delhi

2. UPDATE-SET-WHERE: This is used to update the content of a record in a relation.

Syntax: SQL>UPDATE relation name SET Field_name1=data,field_name2=data, WHERE field_name=data;

Example: SQL>UPDATE student SET sname = 'kumar' WHERE sno=1;

Before:

sno	sname	class	address
1	Ravi	M.Tech	Palakol

After:

```
UPDATE student
SET sname = 'kumar'
WHERE sno = 1;
```

sno	sname	class	address
1	kumar	M.Tech	Palakol

3. DELETE-FROM: This is used to delete all the records of a relation but it will retain the structure of that relation.

a) DELETE-FROM: This is used to delete all the records of relation.

Syntax: SQL>DELETE FROM relation_name;

Example: SQL>DELETE FROM stu;

Before:

sno	sname	class	address
2	Ramu	M.Tech	India
1	Ravi	NULL	NULL

After:

sno	sname	class	address

b) DELETE -FROM-WHERE: This is used to delete a selected record from a relation. ***Syntax:*** SQL>DELETE FROM relation_name WHERE condition;

Example: SQL>DELETE FROM student WHERE sno = 2;

Before:

sno	sname	class	address
1	kumar	M.Tech	Palakol
2	Ramu	B.Tech	Hyderabad
3	Sita	M.Sc	Chennai
4	Gita	B.Sc	Mumbai
5	Hari	MCA	Delhi

After:

sno	sname	class	address
1	kumar	M.Tech	Palakol
3	Sita	M.Sc	Chennai
4	Gita	B.Sc	Mumbai
5	Hari	MCA	Delhi

2. TRUNCATE: This command will remove the data permanently. But structure will not be removed.

Before:

sno	sname	class	address
1	kumar	M.Tech	Palakol
2	Ramu	B.Tech	Hyderabad
3	Sita	M.Sc	Chennai
4	Gita	B.Sc	Mumbai
5	Hari	MCA	Delhi

After:

sno	sname	class	address

Difference between Truncate & Delete:-

- ✓ By using truncate command data will be removed permanently & will not get back where as by using delete command data will be removed temporally & get back by using roll back command.
- ✓ By using delete command data will be removed based on the condition where as by using truncate command there is no condition.
- ✓ Truncate is a DDL command & delete is a DML command.

Syntax: TRUNCATE TABLE <Table name>

Example TRUNCATE TABLE student;

To Retrieve data from one or more tables.

1. SELECT FROM: To display all fields for all records.

sno	sname	class	address
101	Ravi	M.Tech	Palakol
102	Ramu	B.Tech	Hyderabad
103	Sita	M.Sc	Chennai
104	Gita	B.Sc	Mumbai
105	Hari	MCA	Delhi

2. SELECT FROM: To display a set of fields for all records of relation.

Syntax: SELECT a set of fields FROM relation_name;

Example: SQL> select sno, sname from student;

sno	sname
101	Ravi
102	Ramu
103	Sita
104	Gita
105	Hari

3. SELECT - FROM -WHERE: This query is used to display a selected set of fields for a selected set of records of a relation.

Syntax: SELECT a set of fields FROM relation_name WHERE condition;

Example: SQL> select * FROM student WHERE sno<=103;

sno	sname	class	address
101	Ravi	M.Tech	Palakol
102	Ramu	B.Tech	Hyderabad
103	Sita	M.Sc	Chennai

LAB PRACTICE ASSIGNMENT:

Create a table EMPLOYEE with following schema:

(Emp_no, E_name, E_address, E_ph_no, Dept_no, Dept_name,Job_id , Salary)

```
CREATE TABLE EMPLOYEE (
    Emp_no INT,
    E_name VARCHAR(100),
    E_address VARCHAR(255),
    E_ph_no VARCHAR(20),
    Dept_no INT,
    Dept_name VARCHAR(100),
    Job_id CHAR(5),
    Salary INT
);
```

Table:

Emp_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Salary

Write SQL queries for following question:

1. Insert atleast 5 rows in the table.

```
INSERT INTO EMPLOYEE
(E_no, E_name, E_address, E_ph_no, Dept_no, Dept_name, Job_id, Salary, Email, HIREDATE)
VALUES
(1, 'Binamra', 'Kathmandu', '9876543210', 10, 'HR', '101', 60000, 'binamra@gmail.com', '2023-01-15'),
(2, 'Rohan', 'Lalitpur', '9876543211', 20, 'IT', '102', 60000, 'rohan@gmail.com', '2023-02-20'),
(3, 'Ajit', 'Bhaktapur', '9876543212', 10, 'HR', '103', 55000, 'ajit@yahoo.com', '2023-03-25'),
(4, 'Suprim', 'Chitwan', '9876543213', 30, 'SALES', '104', 50000, 'suprim@gmail.com', '2023-04-10'),
(5, 'Bibas', 'Kathmandu', '9876543214', 20, 'MECH', '105', 62000, 'bibas@yahoo.com', '2023-05-15');
```

2. Display all the information of EMP table.

E_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Salary	HIREDATE	Email
1	Binamra	Kathmandu	9876543210	10	HR	101	60000	2023-01-15	binamra@gmail.com
2	Rohan	Lalitpur	9876543211	20	IT	102	60000	2023-02-20	rohan@gmail.com
3	Ajit	Bhaktapur	9876543212	10	HR	103	55000	2023-03-25	ajit@yahoo.com
4	Suprim	Chitwan	9876543213	30	SALES	104	50000	2023-04-10	suprim@gmail.com
5	Bibas	Kathmandu	9876543214	20	MECH	105	62000	2023-05-15	bibas@yahoo.com

3. Display the record of each employee who works in department D10.

```
SELECT * FROM EMPLOYEE
WHERE Dept_no = '10';
```

Display Table:

E_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Salary	HIREDATE	Email
1	Binamra	Kathmandu	9876543210	10	HR	101	60000	2023-01-15	binamra@gmail.com
3	Ajit	Bhaktapur	9876543212	10	HR	103	55000	2023-03-25	ajit@yahoo.com

4. Update the city of E_no-4 with current city as Nagpur.
   Before:

E_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Salary	HIREDATE	Email
4	Suprim	Chitwan	9876543213	30	SALES	104	50000	2023-04-10	suprim@gmail.com

   After:

E_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Salary	HIREDATE	Email
4	Suprim	Nagpur	9876543213	30	SALES	104	50000	2023-04-10	suprim@gmail.com

5. Display the details of Employee who works in department MECH.

```
SELECT * FROM EMPLOYEE
WHERE Dept_name = 'MECH';
```

E_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Salary	HIREDATE	Email
5	Bibas	Kathmandu	9876543214	20	MECH	105	62000	2023-05-15	bibas@yahoo.com

6. Delete the email_id of employee James.

E_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Salary	HIREDATE	Email
1	Binamra	Kathmandu	9876543210	10	HR	101	60000	2023-01-15	binamra@gmail.com
2	Rohan	Lalitpur	9876543211	20	IT	102	60000	2023-02-20	rohan@gmail.com
3	Ajit	Bhaktapur	9876543212	10	HR	103	55000	2023-03-25	ajit@yahoo.com
4	Suprim	Nagpur	9876543213	30	SALES	104	50000	2023-04-10	suprim@gmail.com
5	Bibas	Kathmandu	9876543214	20	MECH	105	62000	2023-05-15	bibas@yahoo.com
6	James	Chitwan	9876543215	30	SALES	106	55000	2023-06-01	NULL

7. Display the complete record of employees working in SALES Department.

```
SELECT * FROM EMPLOYEE
WHERE Dept_name = 'SALES';
```

E_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Salary	HIREDATE	Email
4	Suprim	Chitwan	9876543213	30	SALES	104	50000	2023-04-10	suprim@gmail.com
6	James	Chitwan	9876543215	30	SALES	106	55000	2023-06-01	james@example.com

Experiment No: 3

Title: Implementation of different types of functions with suitable examples.

- Number Function
- Aggregate Function
- Character Function
- Conversion Function § Date Function

Objective:
NUMBER FUNCTION:

Abs(n) :Select abs(-15) from dual;

Exp(n): Select exp(4) from dual;

Power(m,n): Select power(4,2) from dual;

Mod(m,n): Select mod(10,3) from dual;

Round(m,n): Select round(100.256,2) from dual;

Trunc(m,n): ;Select trunc(100.256,2) from dual;

Sqrt(m,n);Select sqrt(16) from dual;

Develop aggregate plan strategies to assist with summarization of several data entries.

Aggregative operators: In addition to simply retrieving data, we often want to perform some computation or summarization. SQL allows the use of arithmetic expressions. We now consider a powerful class of constructs for computing aggregate values such as MIN and SUM.

1. **Count:** COUNT following by a column name returns the count of tuple in that column. If DISTINCT keyword is used then it will return only the count of unique tuple in the column. Otherwise, it will return count of all the tuples (including duplicates) count (*) indicates all the tuples of the column.
   Syntax: COUNT (Column name)
   Example: SELECT COUNT (Salary) FROM EMPLOYEE;

E_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Salary	HIREDATE	Email
1	Binamra	Kathmandu	9876543210	10	HR	101	60000	2023-01-15	binamra@gmail.com
2	Rohan	Lalitpur	9876543211	20	IT	102	60000	2023-02-20	rohan@gmail.com
3	Ajit	Bhaktapur	9876543212	10	HR	103	55000	2023-03-25	ajit@yahoo.com
4	Suprim	Chitwan	9876543213	30	SALES	104	50000	2023-04-10	suprim@gmail.com
5	Bibas	Kathmandu	9876543214	20	MECH	105	62000	2023-05-15	bibas@yahoo.com
6	James	Chitwan	9876543215	30	SALES	106	55000	2023-06-01	james@example.com

COUNT:

COUNT(Salary)
6

2. SUM: SUM followed by a column name returns the sum of all the values in that column.

Syntax: SUM (Column name)

Example: SELECT SUM (Salary) FROM EMPLOYEE;

SUM(Salary)
342000

3. AVG: AVG followed by a column name returns the average value of that column values.

Syntax: AVG (n1, n2...)

Example: SELECT AVG(Salary) FROM EMPLOYEE;

AVG(Salary)
57000.0000

4. MAX: MAX followed by a column name returns the maximum value of that column.

Syntax: MAX (Column name)

Example: SELECT MAX (Salary) FROM EMPLOYEE;

MAX(Salary)
62000

5. MIN: MIN followed by column name returns the minimum value of that column.

Syntax: MIN (Column name)

Example: SELECT MIN (Salary) FROM EMPLOYEE;

MIN(Salary)
50000

SQL>SELECT Dept_no, MIN(Salary) AS Min_salary FROM EMPLOYEE GROUP BY Dept_no HAVING MIN(Salary) > 1000;

```
SELECT Dept_no, MIN(Salary) AS Min_salary
FROM EMPLOYEE
GROUP BY Dept_no
HAVING MIN(Salary) > 1000;
```

Dept_no	Min_salary
10	55000
20	60000
30	50000

CHARACTER FUNCTION:

initcap(char) : select initcap(“hello”) from dual; lower (char):

select lower (‘HELLO’) from dual; upper (char) :select

upper (‘hello’) from dual; ltrim (char,[set]): select ltrim

(‘cseit’, ‘cse’) from dual; rtrim (char,[set]): select rtrim

(‘cseit’, ‘it’) from dual;

replace (char,search): select replace(‘jack and jue’,‘j’,‘bl’) from dual;

CONVERSION FUNCTIONS:

To_char: TO_CHAR (number) converts n to a value of VARCHAR2 data type, using the optional number format fmt. The value n can be of type NUMBER, BINARY_FLOAT, or BINARY_DOUBLE.

SQL>select to_char(65,'RN')from dual;

LXV

To_number : TO_NUMBER converts expr to a value of NUMBER data type.

SQL>Select to_number ('1234.64') from Dual; 1234.64

To_date:TO_DATE converts char of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of DATE data type.

SQL>SELECT TO_DATE('January 15, 1989, 11:00 A.M.')FROM DUAL;

TO_DATE

15-JAN-89

STRING FUNCTIONS:

Concat: CONCAT returns char1 concatenated with char2. Both char1 and char2 can be any of the datatypes

SQL>SELECT CONCAT(‘ORACLE’,’CORPORATION’)FROM DUAL;

ORACLECORPORATION

Lpad: LPAD returns expr1, left-padded to length n characters with the sequence of characters in expr2.

SQL>SELECT LPAD(‘ORACLE’,15,’*’)FROM DUAL;

*********ORACLE

Rpad: RPAD returns expr1, right-padded to length n characters with expr2, replicated as many times as necessary.

SQL>SELECT RPAD (‘ORACLE’,15,’*’)FROM DUAL; ORACLE*********

Ltrim: Returns a character expression after removing leading blanks.

SQL>SELECT LTRIM(‘SSMITHSS’,’S’)FROM DUAL; MITHSS

Rtrim: Returns a character string after truncating all trailing blanks

SQL>SELECT RTRIM(‘SSMITHSS’,’S’)FROM DUAL; SSMITH

Lower: Returns a character expression after converting uppercase character data to lowercase.

SQL>SELECT LOWER(‘DBMS’)FROM DUAL;
dbms

Upper: Returns a character expression with lowercase character data converted to uppercase

SQL>SELECT UPPER(‘dbms’)FROM DUAL;
DBMS

Length: Returns the number of characters, rather than the number of bytes, of the given string expression, excluding trailing blanks.

SQL>SELECT LENGTH(‘DATABASE’)FROM DUAL;
8

Substr: Returns part of a character, binary, text, or image expression.

SQL>SELECT SUBSTR(‘ABCDEFGHIJ’3,4)FROM DUAL;
CDEF

Instr: The INSTR functions search string for substring. The function returns an integer indicating the position of the character in string that is the first character of this occurrence.

SQL>SELECT INSTR('CORPORATE FLOOR','OR',3,2)FROM DUAL;
14

DATE FUNCTIONS:

Sysdate:

SQL>SELECT SYSDATE FROM DUAL;

29-DEC-08 **Next_day:**

SQL>SELECT NEXT_DAY(SYSDATE,’WED’)FROM DUAL;

05-JAN-09 **Add_months:**

SQL>SELECT ADD_MONTHS(SYSDATE,2)FROM DUAL;

28-FEB-09 **Last_day:**

SQL>SELECT LAST_DAY(SYSDATE)FROM DUAL;

31-DEC-08

Months_between:

SQL>SELECT MONTHS_BETWEEN(SYSDATE,HIREDATE)FROM EMP;

4

Least:

SQL>SELECT LEAST('10-JAN-07','12-OCT-07')FROM DUAL; 10-JAN-07 **Greatest:**

SQL>SELECT GREATEST('10-JAN-07','12-OCT-07')FROM DUAL; 10-JAN-07 **Trunc:**

SQL>SELECT TRUNC(SYSDATE,'DAY')FROM DUAL;

28-DEC-08 **Round:**

SQL>SELECT ROUND(SYSDATE,'DAY')FROM DUAL; 28-DEC-08 **to_char:**

SQL> select to_char(sysdate, "dd\mm\yy") from dual; 24-mar-05.

to_date:

SQL> select to date (sysdate, "dd\mm\yy") from dual; 24-mar-o5.

LAB PRACTICE ASSIGNMENT:

Create a table EMPLOYEE with following schema:

(Emp_no, E_name, E_address, E_ph_no, Dept_no, Dept_name,Job_id, Designation , Salary)

```
CREATE TABLE EMPLOYEE1(
    Emp_no INT PRIMARY KEY,
    E_name VARCHAR(100),
    E_address VARCHAR(100),
    E_ph_no VARCHAR(15),
    Dept_no INT,
    Dept_name VARCHAR(100),
    Job_id VARCHAR(10),
    Designation VARCHAR(100),
    Salary DECIMAL(10, 2),
    Hiredate DATE
);
```

Write SQL statements for the following query.

1. List the E_no, E_name, Salary of all employees working for MANAGER.

```
SELECT Emp_no, E_name, Salary
FROM employee1
WHERE Designation = 'MANAGER';
```

Table:

Emp_no	E_name	Salary
1	Prasis	60000.00
5	Rochak	62000.00

2. Display all the details of the employee whose salary is more than the Sal of any IT PROFF..

```
SELECT *
FROM employee1
WHERE Salary > (SELECT MAX(Salary) FROM employee WHERE Designation = 'IT PROFF.');
```

Emp_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Designation	Salary	Hiredate
2	Prayag	UK	985555678	20	IT	JOB002	IT PROFF.	65000.00	1995-04-20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

3. List the employees in the ascending order of Designations of those joined after 1981.

```
SELECT *
FROM employee1
WHERE Hiredate > '1981-01-01'
ORDER BY Designation ASC
LIMIT 1000;
```

Emp_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Designation	Salary	Hiredate
7	Ronish	Portugal	985551357	20	IT	JOB007	ANALYST	70000.00	1993-10-12
9	Kushal	Lalitpur	985553690	40	MARKETING	JOB009	ANALYST	68000.00	1996-08-14
6	Bishwas	Pokhara	985552468	10	HR	JOB006	CLERK	45000.00	1985-07-01
8	Sujan	Argentina	985558024	30	SALES	JOB008	CLERK	48000.00	1997-03-28
2	Prayag	UK	985555678	20	IT	JOB002	IT PROFF.	65000.00	1995-04-20
5	Rochak	Lalitpur	985558765	50	OPERATIONS	JOB005	MANAGER	62000.00	1998-02-15
4	Subesh	Bhaktapur	985557321	40	MARKETING	JOB004	MARKETING MANAGER	57000.00	1990-11-25
3	Sawin	Bhaktapur	985559876	30	SALES	JOB003	SALES MANAGER	58000.00	1995-04-20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

4. List the employees along with their Experience and Daily Salary.

```
SELECT Emp_no,
       E_name,
       DATEDIFF(NOW(), Hiredate) AS Experience,
       Salary / 30 AS Daily_Salary
FROM EMPLOYEE1;
```

Table:

Emp_no	E_name	Experience	Daily_Salary
1	Prasis	16078	2000.000000
2	Prayag	10656	2166.666667
3	Sawin	10656	1933.333333
4	Subesh	12263	1900.000000
5	Rochak	9624	2066.666667
6	Bishwas	14236	1500.000000
7	Ronish	11211	2333.333333
8	Sujan	9948	1600.000000
9	Kushal	10174	2266.666667

5. List the employees who are either 'CLERK' or 'ANALYST' .

```
SELECT *
FROM EMPLOYEE1
WHERE Designation IN ('CLERK', 'ANALYST');
```

Emp_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Designation	Salary	Hiredate
6	Bishwas	Pokhara	985552468	10	HR	JOB006	CLERK	45000.00	1985-07-01
7	Ronish	Portugal	985551357	20	IT	JOB007	ANALYST	70000.00	1993-10-12
8	Sujan	Argentina	985558024	30	SALES	JOB008	CLERK	48000.00	1997-03-28
9	Kushal	Lalitpur	985553690	40	MARKETING	JOB009	ANALYST	68000.00	1996-08-14
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

6. List the employees who joined on 1-MAY-81, 3-DEC-81, 17-DEC-81,19-JAN-80 .

```
SELECT *
FROM EMPLOYEE1
WHERE hiredate IN ('1981-05-01', '1981-12-03', '1981-12-17', '1980-01-19');
```

Emp_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Designation	Salary	Hiredate
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

7. List the employees who are working for the Deptno 10 or 20.

```
SELECT *
FROM EMPLOYEE1
WHERE Dept_no IN (10, 20);
```

Emp_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Designation	Salary	Hiredate
1	Prasis	Australia	985551234	10	HR	JOB001	MANAGER	60000.00	1980-06-15
2	Prayag	UK	985555678	20	IT	JOB002	IT PROFF.	65000.00	1995-04-20
6	Bishwas	Pokhara	985552468	10	HR	JOB006	CLERK	45000.00	1985-07-01
7	Ronish	Portugal	985551357	20	IT	JOB007	ANALYST	70000.00	1993-10-12
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

8. List the Enames those are starting with ‘S’ .

```
SELECT E_name
FROM EMPLOYEE1
WHERE E_name LIKE 'S%';
```

Table:

E_name
Sawin
Subesh
Sujan

9. Dislay the name as well as the first five characters of name(s) starting with ‘H’

```
SELECT E_name, LEFT(E_name, 5) AS First_Five_Chars
FROM EMPLOYEE1
WHERE E_name LIKE 'H%';
```

Table: It’s null because there no name starting with letter H.

E_name	First_Five_Chars

10. List all the emps except ‘PRESIDENT’ & ‘MGR” in asc order of Salaries.

```
SELECT *
FROM EMPLOYEE1
WHERE Designation NOT IN ('PRESIDENT', 'MANAGER')
ORDER BY Salary ASC;
```

Emp_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Designation	Salary	Hiredate
6	Bishwas	Pokhara	985552468	10	HR	JOB006	CLERK	45000.00	1985-07-01
8	Sujan	Argentina	985558024	30	SALES	JOB008	CLERK	48000.00	1997-03-28
4	Subesh	Bhaktapur	985557321	40	MARKETING	JOB004	MARKETING MANAGER	57000.00	1990-11-25
3	Sawin	Bhaktapur	985559876	30	SALES	JOB003	SALES MANAGER	58000.00	1995-04-20
2	Prayag	UK	985555678	20	IT	JOB002	IT PROFF.	65000.00	1995-04-20
9	Kushal	Lalitpur	985553690	40	MARKETING	JOB009	ANALYST	68000.00	1996-08-14
7	Ronish	Portugal	985551357	20	IT	JOB007	ANALYST	70000.00	1993-10-12
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Experiment No: 4

Title : Implementation of different types of operators in SQL.

- Arithmetic Operator
- Logical Operator
- Comparision Operator
- Special Operator
- Set Operator

Objective:

To learn different types of operator.

Theory:

ARIHMETIC OPERATORS:

SN	op1	op2
1	10	5
2	15	8
3	20	12
4	7	3

(+) : Addition - Adds values on either side of the operator.

```
SELECT SN, op1, op2, (op1 + op2) AS Addition_result
FROM Experiment_4;
```

SN	op1	op2	Addition_result
1	10	5	15
2	15	8	23
3	20	12	32
4	7	3	10

(-): Subtraction - Subtracts right hand operand from left hand operand.

```
SELECT SN, op1, op2, (op1 - op2) AS Subtraction_result
FROM Experiment_4;
```

SN	op1	op2	Subtraction_result
1	10	5	5
2	15	8	7
3	20	12	8
4	7	3	4

(*):Multiplication - Multiplies values on either side of the operator.

```
SELECT SN, op1, op2, (op1 * op2) AS Multiplication
FROM Experiment_4;
```

SN	op1	op2	Multiplication
1	10	5	50
2	15	8	120
3	20	12	240
4	7	3	21

(/):Division - Divides left hand operand by right hand operand.

```
SELECT SN, op1, op2, (op1 / op2) AS Division
FROM Experiment_4;
```

SN	op1	op2	Division
1	10	5	2.0000
2	15	8	1.8750
3	20	12	1.6667
4	7	3	2.3333

(%):Modulus - Divides left hand operand by right hand operand and returns remainder.

```
SELECT SN, op1, op2, (op1 % op2) AS Modulus
FROM Experiment_4;
```

SN	op1	op2	Modulus
1	10	5	0
2	15	8	7
3	20	12	8
4	7	3	1

LOGICAL OPERATORS:

AND: The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

```
SELECT *
FROM Experiment_4
WHERE op1 = 15 AND op2 = 8;
```

SN	op1	op2
2	15	8

OR: The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

```
SELECT *
FROM Experiment_4
WHERE op1 > 15 OR op2 < 3;
```

SN	op1	op2
3	20	12

NOT: The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**

```
SELECT *
FROM Experiment_4
WHERE NOT (op1 > 15 OR op2 < 5);
```

SN	op1	op2
1	10	5
2	15	8

COMPARISION OPERATORS:

(=): Checks if the values of two operands are equal or not, if yes then condition becomes true.

```
SELECT *
FROM Experiment_4
WHERE op1 = 15;
```

SN	op1	op2
2	15	8

(!=): Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

```
SELECT *
FROM Experiment_4
WHERE op1 != 15;
```

SN	op1	op2
1	10	5
3	20	12
4	7	3

(< >): Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

```
SELECT *
FROM Experiment_4
WHERE op1 <> 15;
```

SN	op1	op2
1	10	5
3	20	12
4	7	3

(>): Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true

```
SELECT *
FROM Experiment_4
WHERE op1 > 15;
```

SN	op1	op2
3	20	12

(<): Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.

```
SELECT *
FROM Experiment_4
WHERE op1 < 15;
```

SN	op1	op2
1	10	5
4	7	3

(>=): Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.

```
SELECT *
FROM Experiment_4
WHERE op1 >= 15;
```

SN	op1	op2
2	15	8
3	20	12

(<=): Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

```
SELECT *
FROM Experiment_4
WHERE op1 <= 15;
```

SN	op1	op2
1	10	5
2	15	8
4	7	3

SPECIAL OPERATOR:

BETWEEN: The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.

IS NULL: The NULL operator is used to compare a value with a NULL attribute value.

ALL: The ALL operator is used to compare a value to all values in another value set

ANY: The ANY operator is used to compare a value to any applicable value in the list according to the condition.

LIKE: The LIKE operator is used to compare a value to similar values using wildcard operators.It allows to use percent sign(%) and underscore (_) to match a given string pattern.

IN: The IN operator is used to compare a value to a list of literal values that have been specified.

EXIST: The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.

SET OPERATORS:

The Set operator combines the result of 2 queries into a single result. The following are the operators:

- Union
- Union all
- Intersect
- Minus

Union: Returns all distinct rows selected by both the queries

Union all: Returns all rows selected by either query including the duplicates.

Intersect: Returns rows selected that are common to both queries. **Minus:** Returns all distinct rows selected by the first query and are not by the second

LAB PRACTICE ASSIGNMENT:

1. Display all the dept numbers available with the dept and emp tables avoiding duplicates.

```
SELECT Dept_no
FROM EMPLOYEE1
UNION
SELECT Dept_no
FROM EMPLOYEE1;
```

Dept_no
10
20
30
40
50

2. Display all the dept numbers available with the dept and emp tables.

```
SELECT Dept_no
FROM EMPLOYEE1
UNION ALL
SELECT Dept_no
FROM EMPLOYEE1;
```

Dept_no
10
20
30
40
50
10
20
30
40
10
20
30
40
50
10
20
30
40

3. Display all the dept numbers available in emp and not in dept tables and vice versa.

```
SELECT DISTINCT Employees.department_id
FROM Employees
LEFT JOIN Dept ON Employees.department_id = Dept.dept_id
WHERE Dept.dept_id IS NULL;
```

department_id

```
SELECT DISTINCT Dept.dept_id
FROM Dept
LEFT JOIN Employees ON Dept.dept_id = Employees.department_id
WHERE Employees.department_id IS NULL;
```

dept_id
4

Experiment No: 5

Title : Implementation of different types of Joins

- Inner Join
- Outer Join Natural Join..etc

Objective :

To implement different types of joins

Theory :

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.The join is actually performed by the 'where' clause which combines specified rows of tables.

Syntax:

```
SELECT column 1, column 2, column 3...
FROM table_name1, table_name2
WHERE table_name1.column name  = table_name2.columnname;
```

Types of Joins :

1. Simple Join
2. Self Join
3. Outer Join

Simple Join:

It is the most common type of join. It retrieves the rows from 2 tables having a common column and is further classified into

Equi-join :

A join, which is based on equalities, is called equi-join.

Example:

```
Select * from item, cust where item.id=cust.id;
```

In the above statement, item-id = cust-id performs the join statement. It retrieves rows from both the tables provided they both have the same id as specified by the where clause. Since the where clause uses the comparison operator (=) to perform a join, it is said to be equijoin. It combines the matched rows of tables. It can be used as follows:

- ✓ To insert records in the target table.
- ✓ To create tables and insert records in this table.
- ✓ To update records in the target table.
- ✓ To create views.

Non Equi-join:

It specifies the relationship between columns belonging to different tables by making use of relational operators other than'='.

Example:

Select * from item, cust where item.id<cust.id;

Table Aliases
Table aliases are used to make multiple table queries shorted and more readable. We give an alias name to the table in the 'from' clause and use it instead of the name throughout the query.

Self join:

Joining of a table to itself is known as self-join. It joins one row in a table to another. It can compare each row of the table to itself and also with other rows of the same table.
Example:

select * from emp x ,emp y where x.salary >= (select avg(salary) from x.emp where x. deptno =y.deptno);

Outer Join:
It extends the result of a simple join. An outer join returns all the rows returned by simple join as well as those rows from one table that do not match any row from the table. The symbol(+) represents outer join.

-Left outer join -Right outer join -Full outer join

LAB PRACTICE ASSIGNMENT:

Consider the following schema:

Sailors (sid, sname, rating, age)

Boats (bid, bname, color)

Reserves (sid, bid, day(date))

Sailors:

Field	Type	Null	Key	Default	Extra
sid	int	YES		NULL	
sname	varchar(255)	YES		NULL	
rating	int	YES		NULL	
age	int	YES		NULL	

Sailors Details:

sid	sname	rating	age
101	Ajit	8	25
102	Rohan	7	28
103	James	6	22
104	Suprim	9	24
105	Trevor	7	26
106	Kushal	8	23
107	Sujan	6	27
108	Bibas	9	25
109	Oggy	7	29
110	Emma	8	24

Boats:

Field	Type	Null	Key	Default	Extra
bid	int	YES		NULL	
bname	varchar(255)	YES		NULL	
color	varchar(50)	YES		NULL	

Boat Details:

bid	bname	color
101	B1	Blue
102	B2	Red
103	B3	Green
104	B4	Yellow
105	B5	White

Reserves:

Field	Type	Null	Key	Default	Extra
sid	int	YES		NULL	
bid	int	YES		NULL	
day	varchar(50)	YES		NULL	

Reserves Details:

sid	bid	day
101	101	2023-06-01
102	102	2023-06-02
103	101	2023-06-03
104	103	2023-06-04
105	102	2023-06-05
106	101	2023-06-06
107	104	2023-06-07
108	103	2023-06-08
109	105	2023-06-09
110	101	2023-06-10

1. Find all information of sailors who have reserved boat number 101.

```
SELECT s.*
FROM Sailors s
JOIN Reserves r ON s.sid = r.sid
WHERE r.bid = 101;
```

sid	sname	rating	age
101	Ajit	8	25
103	James	6	22
106	Kushal	8	23
110	Emma	8	24

2. Find the name of boat reserved by Ajit.

```
SELECT b.bname
FROM Boats b
JOIN Reserves r ON b.bid = r.bid
JOIN Sailors s ON s.sid = r.sid
WHERE s.sname = 'Ajit';
```

bname
B1

3. Find the names of sailors who have reserved a red boat, and list in the order of age.

```
SELECT s.sname
FROM Sailors s
JOIN Reserves r ON s.sid = r.sid
JOIN Boats b ON b.bid = r.bid
WHERE b.color = 'Red'
ORDER BY s.age;
```

sname
Trevor
Rohan

4. Find the names of sailors who have reserved at least one boat.

```
SELECT DISTINCT s.sname
FROM Sailors s
JOIN Reserves r ON s.sid = r.sid;
```

sname
Ajit
Rohan
James
Suprim
Trevor
Kushal
Sujan
Bibas
Oggy
Emma

5. Find the ids and names of sailors who have reserved two different boats on the same day.

```
SELECT s.sid, s.sname
FROM Sailors s
JOIN Reserves r1 ON s.sid = r1.sid
JOIN Reserves r2 ON s.sid = r2.sid AND r1.day = r2.day AND r1.bid <> r2.bid;
```

sid	sname

6. Find the ids of sailors who have reserved a red boat or a green boat.

```
SELECT DISTINCT s.sid
FROM Sailors s
JOIN Reserves r ON s.sid = r.sid
JOIN Boats b ON b.bid = r.bid
WHERE b.color IN ('Red', 'Green');
```

sid
102
104
105
108

7. Find the name and the age of the youngest sailor.

```
SELECT s.sname, s.age
FROM Sailors s
WHERE s.age = (SELECT MIN(age)
FROM Sailors);
```

sname	age
James	22

8. Count the number of different sailor names.

```
SELECT COUNT(DISTINCT s.sname)
AS Count
FROM Sailors s;
```

Count
10

9. Find the average age of sailors for each rating level.

```
SELECT s.rating, AVG(s.age)
AS avg_age
FROM Sailors s
GROUP BY s.rating;
```

rating	avg_age
8	24.0000
7	27.6667
6	24.5000
9	24.5000

10. Find the average age of sailors for each rating level that has at least two sailors.

```
SELECT s.rating, AVG(s.age)
AS avg_age
FROM Sailors s
GROUP BY s.rating
HAVING COUNT(s.sid) >= 2;
```

rating	avg_age
8	24.0000
7	27.6667
6	24.5000
9	24.5000

Experiment No: 6

Title : Study & Implementation of

- Group by & Having Clause
- Order by Clause
- Indexing

***Objective*:**

To learn the concept of group functions

***Theory*:**

1. **GROUP BY:** This query is used to group to all the records in a relation together for each and every value of a specific key(s) and then display them for a selected set of fields the relation.

Syntax: SELECT <set of fields> FROM <relation_name> GROUP BY <field_name>;

Example: SQL> SELECT Emp_no, SUM(Salary) FROM EMPLOYEE1 GROUP BY Emp_no;

SELECT * FROM EMPLOYEE1;

Emp_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Designation	Salary	Hiredate
1	Prasis	Australia	985551234	10	HR	JOB001	MANAGER	60000.00	1980-06-15
2	Prayag	UK	985555678	20	IT	JOB002	IT PROFF.	65000.00	1995-04-20
3	Sawin	Bhaktapur	985559876	30	SALES	JOB003	SALES MANAGER	58000.00	1995-04-20
4	Subesh	Bhaktapur	985557321	40	MARKETING	JOB004	MARKETING MANAGER	57000.00	1990-11-25
5	Rochak	Lalitpur	985558765	50	OPERATIONS	JOB005	MANAGER	62000.00	1998-02-15
6	Bishwas	Pokhara	985552468	10	HR	JOB006	CLERK	45000.00	1985-07-01
7	Ronish	Portugal	985551357	20	IT	JOB007	ANALYST	70000.00	1993-10-12
8	Sujan	Argentina	985558024	30	SALES	JOB008	CLERK	48000.00	1997-03-28
9	Kushal	Lalitpur	985553690	40	MARKETING	JOB009	ANALYST	68000.00	1996-08-14

SELECT Emp_no, SUM(Salary) FROM EMPLOYEE1 GROUP BY Emp_no;

Emp_no	SUM(Salary)
1	60000.00
2	65000.00
3	58000.00
4	57000.00
5	62000.00
6	45000.00
7	70000.00
8	48000.00
9	68000.00

2. **GROUP BY-HAVING :** The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions. The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

Syntax: SELECT column_name, aggregate_function(column_name) FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;

***Example* :** SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID) GROUP BY LastName HAVING COUNT (Orders.OrderID) > 10;

3. **JOIN using GROUP BY:** This query is used to display a set of fields from two relations by matching a common field in them and also group the corresponding records for each and every value of a specified key(s) while displaying.

Syntax: SELECT <set of fields (from both relations)> FROM relation_1,relation_2 WHERE relation_1.field_x=relation_2.field_y GROUP BY field_z; ***Example:***

SQL> SELECT emp_id,SUM(salary) FROM Employees,dept
WHERE Employees.department_id =1 GROUP BY emp_id;

```
SELECT emp_id,SUM(salary)
FROM Employees,dept
WHERE Employees.department_id =1
GROUP BY emp_id;
```

emp_id	SUM(salary)
8	64000.00
4	120000.00
3	60000.00
2	80000.00
1	48000.00

4. **ORDER BY:** This query is used to display a selected set of fields from a relation in an ordered manner base on some field.

Syntax: SELECT <set of fields> FROM <relation_name>
ORDER BY <field_name>;

Example: SQL>SELECT emp_id, emp_name, job FROM Employees ORDER BY job;

Result Grid:

```
SELECT emp_id, emp_name, job
FROM Employees
ORDER BY job;
```

emp_id	emp_na...	job
10	Emma	Accou...
4	Rohan	CTO
1	Bibas	Engineer
3	Prayag	Engineer
8	Ronaldo	Engineer
5	Binamra	HR
7	Alex	HR
2	Ajit	Manager
6	James	Manager
9	David	Manager

5. **JOIN using ORDER BY:** This query is used to display a set of fields from two relations by matching a common field in them in an ordered manner based on some fields. ***Syntax:*** SELECT <set of fields (from both relations)> FROM relation_1, relation_2
WHERE relation_1.field_x = relation_2.field_y ORDER BY field_z;

Example: SQL> SELECT emp_id, emp_name, job, department_name FROM Employees INNER JOIN Dept ON Employees.department_id = Dept.dept_id WHERE Employees.department_id = 1 ORDER BY job;

```sql
SELECT emp_id, emp_name, job, department_name
FROM Employees
INNER JOIN Dept ON Employees.department_id = Dept.dept_id
WHERE Employees.department_id = 1
ORDER BY job;
```

Result Grid:

emp_id	emp_na...	job	depart...
4	Rohan	CTO	IT
1	Bibas	Engineer	IT
3	Prayag	Engineer	IT
8	Ronaldo	Engineer	IT
2	Ajit	Manager	IT

6. INDEXING: An *index* is an ordered set of pointers to the data in a table. It is based on the data values in one or more columns of the table. SQL Base stores indexes separately from tables.

An index provides two benefits:

- It improves performance because it makes data access faster.
- It ensures uniqueness. A table with a unique index cannot have two rows with the same values in the column or columns that form the index key.

Syntax:

CREATE INDEX <index_name> on <table_name> (attrib1,attrib 2….attrib n);

Example:

CREATE INDEX id1 on emp(empno,dept_no);

```sql
CREATE INDEX id1
ON Employees(emp_id,dept_id);
SELECT * FROM employees
```

Result Grid:

emp_id	emp_na...	job	manager_id	salary	depa...	depart...
1	Bibas	Engineer	4	12000.00	1	IT
2	Ajit	Manager	NULL	20000.00	1	IT
3	Prayag	Engineer	4	15000.00	1	IT
4	Rohan	CTO	NULL	30000.00	1	IT
5	Binamra	HR	6	14000.00	2	HR
6	James	Manager	NULL	22000.00	2	HR
7	Alex	HR	6	13000.00	2	HR
8	Ronaldo	Engineer	4	16000.00	1	IT
9	David	Manager	NULL	25000.00	3	Finance
10	Emma	Accou...	9	18000.00	3	Finance
NULL	NULL	NULL	NULL	NULL	NULL	NULL

LAB PRACTICE ASSIGNMENT:

Create a relation and implement the following queries.

1. Display total salary spent for each job category. Result:

```
SELECT job, SUM(salary) AS total_salary
FROM Employees
GROUP BY job;
```

job	total_salary
Engineer	43000.00
Manager	67000.00
CTO	30000.00
HR	27000.00
Accountant	18000.00

2. Display lowest paid employee details under each manager.

```
SELECT e1.manager_id, e1.emp_id, e1.emp_name, e1.salary
FROM Employees e1
JOIN (
    SELECT manager_id, MIN(salary) AS min_salary
    FROM Employees
    WHERE manager_id IS NOT NULL
    GROUP BY manager_id
) e2 ON e1.manager_id = e2.manager_id AND e1.salary = e2.min_salary;
```

Result:

manager_id	emp_id	emp_name	salary
4	1	Bibas	12000.00
6	7	Alex	13000.00
9	10	Emma	18000.00

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

Result:

```
SELECT department_id, department_name,
COUNT(emp_id)
AS num_employees
FROM Employees
GROUP BY department_id, department_name;
```

department...	department_na...	num_employe
1	IT	5
2	HR	3
3	Finance	2

4. Display the details of employees sorting the salary in increasing order.

```
SELECT *
FROM Employees
ORDER BY salary ASC;
```

Result:

emp_id	emp_name	job	manager_id	salary	department...	department_na...
1	Bibas	Engineer	4	12000.00	1	IT
7	Alex	HR	6	13000.00	2	HR
5	Binamra	HR	6	14000.00	2	HR
3	Prayag	Engineer	4	15000.00	1	IT
8	Ronaldo	Engineer	4	16000.00	1	IT
10	Emma	Accountant	9	18000.00	3	Finance
2	Ajit	Manager	NULL	20000.00	1	IT
6	James	Manager	NULL	22000.00	2	HR
9	David	Manager	NULL	25000.00	3	Finance
4	Rohan	CTO	NULL	30000.00	1	IT

5. Show the record of employee earning salary greater than 16000 in each department.

Result Grid:

```
SELECT *
FROM Employees
WHERE salary > 16000
ORDER BY department_id;
```

emp_id	emp_na...	job	manager_id	salary	depa...	depart...
2	Ajit	Manager	NULL	20000.00	1	IT
4	Rohan	CTO	NULL	30000.00	1	IT
6	James	Manager	NULL	22000.00	2	HR
9	David	Manager	NULL	25000.00	3	Finance
10	Emma	Accou...	9	18000.00	3	Finance

6. Write queries to implement and practice the above clause.

```
-- Query 1: Total salary spent for each job category
SELECT job, SUM(salary) AS total_salary
FROM Employees
GROUP BY job;

-- Query 2: Lowest paid employee details under each manager
SELECT e1.emp_id, e1.emp_name, e1.manager_id, e1.salary
FROM Employees e1
JOIN (
    SELECT manager_id, MIN(salary) AS min_salary
    FROM Employees
    WHERE manager_id IS NOT NULL
    GROUP BY manager_id
) e2 ON e1.manager_id = e2.manager_id AND e1.salary = e2.min_salary;

-- Query 3: Number of employees working in each department and their department name
SELECT department_id, department_name, COUNT(emp_id) AS num_employees
FROM Employees
GROUP BY department_id, department_name;

-- Query 4: Details of employees sorted by salary in increasing order
SELECT *
FROM Employees
ORDER BY salary ASC;

-- Query 5: Employees earning salary greater than 16000 in each department
SELECT *
FROM Employees
WHERE salary > 16000
ORDER BY department_id;
```

Experiment No: 7

Title : Study & Implementation of

- Sub queries
- Views

***Objective*:**

- To perform nested Queries and joining Queries using DML command
- To understand the implementation of views.

***Theory*:**

SUBQUERIES: The query within another is known as a sub query. A statement containing sub query is called parent statement. The rows returned by sub query are used by the parent statement or in other words A subquery is a SELECT statement that is embedded in a clause of another SELECT statement You can place the subquery in a number of SQL clauses:

- WHERE clause
- HAVING clause
- FROM clause
- OPERATORS(IN.ANY,ALL,<,>,>=,<= etc..)

Types

1. Sub queries that return several values

Sub queries can also return more than one value. Such results should be made use along with the operators in and any.

2. Multiple queries

Here more than one sub query is used. These multiple sub queries are combined by means of 'and' & 'or' keywords.

3. Correlated sub query

A sub query is evaluated once for the entire parent statement whereas a correlated Sub query is evaluated once per row processed by the parent statement.

VIEW: In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

A view is a virtual table, which consists of a set of columns from one or more tables. It is similar to a table but it does not store in the database. View is a query stored as an object.

Syntax: CREATE VIEW <view_name> AS SELECT <set of fields>

FROM relation_name WHERE (Condition) ***Example:***

SQL> CREATE VIEW employee AS SELECT empno,ename,job FROM EMP

WHERE job = 'clerk'; SQL>

View created.

Example:

CREATE VIEW [Current Product List] AS

SELECT ProductID, ProductName

FROM Products

WHERE Discontinued=No;

UPDATING A VIEW : A view can updated by using the following syntax :

Syntax : CREATE OR REPLACE VIEW view_name AS

SELECT column_name(s)

FROM table_name

WHERE condition

DROPPING A VIEW: A view can deleted with the DROP VIEW command.
Syntax: DROP VIEW <view_name> ;

LAB PRACTICE ASSIGNMENT:

Consider the following schema:

Sailors (sid, sname, rating, age)

Boats (bid, bname, color)

Reserves (sid, bid, day(date))

Write subquery statement for the following queries.

1. Find all information of sailors who have reserved boat number 101.
2. Find the name of boat reserved by Bob.
3. Find the names of sailors who have reserved a red boat, and list in the order of age.
4. Find the names of sailors who have reserved at least one boat.
5. Find the ids and names of sailors who have reserved two different boats on the same day.
6. Find the ids of sailors who have reserved a red boat or a green boat.
7. Find the name and the age of the youngest sailor.
8. Count the number of different sailor names.
9. Find the average age of sailors for each rating level.
10. Find the average age of sailors for each rating level that has at least two sailors.

Already Done at Experiment No 5.

Experiment No: 8

Title : • Study & Implementation of different types of constraints

***Objective*:**

To practice and implement constraints

***Theory*:**

CONSTRAINTS:

Constraints are used to specify rules for the data in a table. If there is any violation between the constraint and the data action, the action is aborted by the constraint. It can be specified when the table is created (using CREATE TABLE statement) or after the table is created (using ALTER TABLE statement).

1. **NOT NULL:** When a column is defined as NOTNULL, then that column becomes a mandatory column. It implies that a value must be entered into the column if the record is to be accepted for storage in the table.

   Syntax:

   CREATE TABLE Table_Name (column_name data_type (*size*) **NOT NULL,**); ***Example:***

   CREATE TABLE student (sno **NUMBER(3)NOT NULL,** name **CHAR**(**10**));

2. **UNIQUE:** The purpose of a unique key is to ensure that information in the column(s) is unique i.e. a value entered in column(s) defined in the unique constraint must not be repeated across the column(s). A table may have many unique keys.

   Syntax:

   CREATE TABLE Table_Name(column_name data_type(*size*) **UNIQUE, ….**); ***Example:***

   CREATE TABLE student (sno **NUMBER(3) UNIQUE,** name **CHAR**(**10**));

3. **CHECK:** Specifies a condition that each row in the table must satisfy. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null).

   Syntax:

   CREATE TABLE Table_Name(column_name data_type(*size*) **CHECK(*logical expression*), ….**);

   Example:

   CREATE TABLE student (sno **NUMBER (3),** name **CHAR**(**10**),class **CHAR(5),CHECK**(class **IN**('CSE','CAD','VLSI'));

4. **PRIMARY KEY:** A field which is used to identify a record uniquely. A column or combination of columns can be created as primary key, which can be used as a reference from other tables. A table contains primary key is known as Master Table.

- ✓ It must uniquely identify each record in a table.
- ✓ It must contain unique values.
- ✓ It cannot be a null field.
- ✓ It cannot be multi port field.
- ✓ It should contain a minimum no. of fields necessary to be called unique.

Syntax:

CREATE TABLE Table_Name(column_name data_type(*size*) **PRIMARY KEY,** **….**);

Example:

CREATE TABLE faculty (fcode **NUMBER(3) PRIMARY KEY,** fname **CHAR**(**10**));

5. **FOREIGN KEY:** It is a table level constraint. We cannot add this at column level. To reference any primary key column from other table this constraint can be used. The table in which the foreign key is defined is called a **detail table**. The table that defines the primary key and is referenced by the foreign key is called the **master table**.

Syntax: **CREATE TABLE** Table_Name(column_name data_type(*size*) **FOREIGN KEY**(column_name) **REFERENCES** table_name);

Example:

CREATE TABLE subject (scode **NUMBER (3) PRIMARY KEY,** subname **CHAR**(**10**),fcode **NUMBER(3), FOREIGN KEY**(fcode) **REFERENCE** faculty);

Defining integrity constraints in the alter table command:

Syntax: **ALTER TABLE** Table_Name **ADD PRIMARY KEY** (column_name);
Example: **ALTER TABLE** student **ADD PRIMARY KEY** (sno);

(Or)

Syntax: **ALTER TABLE** table_name **ADD CONSTRAINT** constraint_name **PRIMARY KEY**(colname)

Example: **ALTER TABLE** student **ADD CONSTRAINT** SN **PRIMARY KEY(**SNO**)**

Dropping integrity constraints in the alter table command:

Syntax: **ALTER TABLE** Table_Name **DROP** constraint_name;

Example: **ALTER TABLE** student **DROP PRIMARY KEY**;

(or)

Syntax: **ALTER TABLE** student **DROP CONSTRAINT** constraint_name**;**

Example: **ALTER TABLE** student **DROP CONSTRAINT** SN**;**

6. **DEFAULT** : The DEFAULT constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified. **Syntax:**

CREATE TABLE Table_Name(col_name1,col_name2,col_name3 DEFAULT '<value>');

Example:

CREATE TABLE student (sno **NUMBER(3) UNIQUE,** name **CHAR(10)**,address **VARCHAR(20) DEFAULT** 'Aurangabad');

LAB PRACTICE ASSIGNMENT:

1. Create a table called EMP with the following structure.

Name:	**Type:**
EMPNO	NUMBER(6)
ENAME	VARCHAR(20)
JOB	VARCHAR(10)
DEPTNO	NUMBER(3)
SAL	NUMBER(7,2)

Allow NULL for all columns except ename and job.

```
CREATE TABLE EMP (
    EMPNO INT,
    ENAME VARCHAR(20) NOT NULL,
    JOB VARCHAR(10) NOT NULL,
    DEPTNO INT,
    SAL DECIMAL(7,2)
);
INSERT INTO EMP (EMPNO, ENAME, JOB, DEPTNO, SAL) VALUES
(1, 'Emma', 'Manager', 10, 75000.00),
(2, 'Kylie', 'Clerk', 20, 25000.00),
(3, 'Kendrick', 'Analyst', 30, 55000.00),
(4, 'Drake', 'Sales', 40, 45000.00),
(5, 'Sabrina', 'Engineer', 50, 60000.00);
```

Result:

EMPNO	ENAME	JOB	DEPTNO	SAL
1	Emma	Manager	10	75000.00
2	Kylie	Clerk	20	25000.00
3	Kendrick	Analyst	30	55000.00
4	Drake	Sales	40	45000.00
5	Sabrina	Engineer	50	60000.00

2. Add constraints to check, while entering the empno value (i.e) empno > 100.

```
CREATE TABLE EMP1 (
    EMPNO INT CHECK (EMPNO > 100),
    ENAME VARCHAR(20) NOT NULL,
    JOB VARCHAR(10) NOT NULL,
    DEPTNO INT,
    SAL DECIMAL(7,2)
);
```

3. Define the field DEPTNO as unique.

```
CREATE TABLE EMP1 (
    EMPNO INT CHECK (EMPNO > 100),
    ENAME VARCHAR(20) NOT NULL,
    JOB VARCHAR(10) NOT NULL,
    DEPTNO INT UNIQUE,
    SAL DECIMAL(7,2)
);
```

Result:

Field	Type	Null	Key	Default
EMPNO	int	YES		NULL
ENAME	varchar(20)	NO		NULL
JOB	varchar(10)	NO		NULL
DEPTNO	int	YES	UNI	NULL
SAL	decimal(7,2)	YES		NULL

4. Create a primary key constraint for the table(EMPNO).

```
ALTER TABLE EMP1
ADD CONSTRAINT pk_empno
PRIMARY KEY (EMPNO);
```

Result:

Field	Type	Null	Key	Default
EMPNO	int	NO	PRI	NULL
ENAME	varchar(20)	NO		NULL
JOB	varchar(10)	NO		NULL
DEPTNO	int	YES	UNI	NULL
SAL	decimal(7,2)	YES		NULL

5. Write queries to implement and practice constraints.

I. Inserting data to test the check constraint (EMPNO > 100):

-- Inserting a record with EMPNO > 100 (should succeed)

INSERT INTO EMP (EMPNO, ENAME, JOB, DEPTNO, SAL) VALUES (101, 'Kylie', 'Manager', 10, 5000);

-- Inserting a record with EMPNO <= 100 (should fail due to check constraint)

INSERT INTO EMP (EMPNO, ENAME, JOB, DEPTNO, SAL) VALUES (99, 'Ajit', 'Clerk', 20, 3000);

Experiment No: 9

Title :

Study and Implementation of Database Backup & Recovery Commands.

Study and Implementation of Rollback, Commit, Save point.

***Objective*:**

To understand the concept of administrative commands

***Theory*:**

A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be made permanent to the database only if they are committed a transaction begins with an executable SQL statement & ends explicitly with either rollback or commit statement.

1. COMMIT: This command is used to end a transaction only with the help of the commit command transaction changes can be made permanent to the database.

Syntax: SQL> COMMIT;

Example: SQL> COMMIT;

2. SAVE POINT: Save points are like marks to divide a very lengthy transaction to smaller once. They are used to identify a point in a transaction to which we can latter role back. Thus, save point is used in conjunction with role back.

Syntax: SQL> SAVE POINT ID;

Example: SQL> SAVE POINT xyz;

3. ROLLBACK: A role back command is used to undo the current transactions. We can role back the entire transaction so that all changes made by SQL statements are undo (or) role back a transaction to a save point so that the SQL statements after the save point are role back.

Syntax: ROLLBACK (current transaction can be role back)

ROLLBACK to save point ID;

Example: SQL> ROLLBACK;

SQL> ROLLBACK TO SAVE POINT xyz;

LAB PRACTICE ASSIGNMENT:

1. Write a query to implement the save point.

Emp_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Designation	Salary	Hiredate
1	Prasis	Australia	985551234	10	HR	JOB001	MANAGER	60000.00	1980-06-15
2	Prayag	UK	985555678	20	IT	JOB002	IT PROFF.	65000.00	1995-04-20
3	Sawin	Bhaktapur	985559876	30	SALES	JOB003	SALES MANAGER	58000.00	1995-04-20
4	Subesh	Bhaktapur	985557321	40	MARKETING	JOB004	MARKETING MANAGER	57000.00	1990-11-25
5	Rochak	Lalitpur	985558765	50	OPERATIONS	JOB005	MANAGER	62000.00	1998-02-15
6	Bishwas	Pokhara	985552468	10	HR	JOB006	CLERK	45000.00	1985-07-01
7	Ronish	Portugal	985551357	20	IT	JOB007	ANALYST	70000.00	1993-10-12
8	Sujan	Argentina	985558024	30	SALES	JOB008	CLERK	48000.00	1997-03-28
9	Kushal	Lalitpur	985553690	40	MARKETING	JOB009	ANALYST	68000.00	1996-08-14

```
START TRANSACTION;
-- Perform some operations
UPDATE EMPLOYEE1 SET Designation = 'CLERK' WHERE Emp_no = 1;
-- Create a savepoint
SAVEPOINT my_savepoint;
-- Perform additional operations
UPDATE EMPLOYEE1 SET Designation = 'ANALYST' WHERE emp_id = 2;
```

Emp_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Designation	Salary	Hiredate
1	Prasis	Australia	985551234	10	HR	JOB001	CLERK	60000.00	1980-06-15
2	Prayag	UK	985555678	20	IT	JOB002	ANALYST	65000.00	1995-04-20
3	Sawin	Bhaktapur	985559876	30	SALES	JOB003	SALES MANAGER	58000.00	1995-04-20

2. Write a query to implement the rollback.

```
ROLLBACK TO my_savepoint;
```

When it is rollbacked to “my_savepoint” it went back to the last save point and the additional operations performed in Emp_no 2 after the savepoint is rollbacked as:

Emp_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Designation	Salary	Hiredate
1	Prasis	Australia	985551234	10	HR	JOB001	CLERK	60000.00	1980-06-15
2	Prayag	UK	985555678	20	IT	JOB002	IT PROFF.	65000.00	1995-04-20
3	Sawin	Bhaktapur	985559876	30	SALES	JOB003	SALES MANAGER	58000.00	1995-04-20

3. Write a query to implement the commit.

```
-- Commit the transaction
COMMIT;
```

Experiment No: 10

Title: Creating Database/ Table Space

- Managing Users: - Create User, Delete User
- Managing Passwords
- Managing roles: - Grant , Revoke

***Objective*:**

To understand the concept of administrative commands

***Theory*:**

DATABASE is collection of coherent data.

To create database we have :

Syntax: CREATE DATABASE <database_name>

Example : CREATE DATABASE my_db;

TABLESPACE:

The oracle database consists of one or more logical storage units called *tablespaces.* Each tablespace in an Oracle database consists of one or more files called *datafiles,* which are physical structures that confirm to the operating system in which Oracle is running.

Syntax:

CREATE<tablespace name> DATAFILE'C:\oraclexe\app\oracle\product\10.2.0\ server \<file name.dbf 'SIZE 50M; Example:

Create tablespace te_cs DATAFILE 'C:\oraclexe\app\oracle\product\10.2.0\ server\usr.dbf 'SIZE 50M;

CREATE USER:

The DBA creates user by executing CREATE USER statement.

The user is someone who connects to the database if enough privilege is granted.

Syntax:

SQL> CREATE USER < *username*> -- (name of user to be created) IDENTIFIED BY <*password*> -- (specifies that the user must login with this password)

SQL> user created

Eg: create user *James* identified by *bob*;

(The user does not have privilege at this time, it has to be granted.These privileges determine what user can do at database level.)

PRIVILEGES:

A privilege is a right to execute an SQL statement or to access another user's object. In Oracle, there are two types of privileges

- System Privileges
- Object Privileges

- ***System Privileges :*** are those through which the user can manage the performance of database actions. It is normally granted by DBA to users.

Eg: Create Session,Create Table,Create user etc..

- ***Object Privileges :*** allow access to objects or privileges on object, i.e. tables, table columns. tables,views etc..It includes alter,delete,insert,select update etc.

(After creating the user, DBA grant specific system privileges to user)

GRANT:

The DBA uses the GRANT statement to allocate system privileges to other user.

Syntax:

SQL> GRANT privilege [privilege…. …]

TO USER ;

SQL> Grant succeeded

Eg: Grant create session, create table, create view to James;

Object privileges vary from object to object.An owner has all privilege or specific privileges on object.

SQL> GRANT object_priv [(column)]

ON object

TO user;

SQL>GRANT *select, insert* ON emp TO James;

SQL>GRANT *select ,update (e_name,e_address)*

ON emp TO James;

CHANGE PASSWORD:

The DBA creates an account and initializes a password for every user.You can change password by using ALTER USER statement.

Syntax:

Alter USER <*some user name*>

IDENTIFIED BY<*New password*>

Eg: ALTER USER James

IDENTIFIED BY sam **REVOKE:**

REVOKE statement is used to remove privileges granted to other users.The privileges you specify are revoked from the users.

Syntax:

REVOKE [*privilege..* …]

ON *object*

FROM *user* **Eg:**

- REVOKE create session,create table from James;
- REVOKE select ,insert

ON emp

FROM James **ROLE:**

A role is a named group of related privileges that can be granted to user.In other words, role is a predefined collection of previleges that are grouped together,thus privileges are easier to assign user.

SQL> Create role *custom*;

SQL> Grant create table, create view TO *custom*;

SQL> Grant select, insert ON emp TO *custom*;

Eg: Grant *custom* to James, Steve;

LAB PRACTICE ASSIGNMENT:

1. Create user and implement the following commands on relation (Employees and Dept).

```
5   12:42:05   CREATE USER 'bikaf'@'localhost' IDENTIFIED BY 'password1'
```

2. Develop a query to grant all privileges of employees table into departments table.

```
9   12:43:14   GRANT ALL PRIVILEGES ON Employees TO 'bikaf'@'localhost'
10  12:43:24   GRANT ALL PRIVILEGES ON Dept TO 'bikaf'@'localhost'
```

3. Develop a query to grant some privileges of employees table into departments table.

```
11  12:44:03   GRANT SELECT, INSERT ON Employees TO 'bikaf'@'localhost'
12  12:44:05   GRANT SELECT, INSERT ON Dept TO 'bikaf'@'localhost'
```

4. Develop a query to revoke all privileges of employees table from departments table.

```
13  12:47:46   REVOKE DELETE ON Employees FROM 'bikaf'@'localhost'
14  12:47:50   REVOKE DELETE ON Dept FROM 'bikaf'@'localhost'
```