LAB MANUAL
FOR
DATABASE MANAGEMENT SYSTEM

Fourth Semester

Bachelor in Computer Engineering

Prepared by

Er. Ravi Khadka (Lecturer)

Department of Computer Engineering

National Academy of Science and Technology (NAST)

Uttar Behadi-04, Dhangadhi

INSTRUCTIONS TO STUDENTS FOR PREPARING DATABASE MANAGEMENT SYSTEM LAB REPORT

This Lab Manual is prepared to help the students with their practical understanding and develop knowledge in database SQL concepts, technology and practice to groom students into well-informed database application developers and may be used as a base reference during the lab/practical classes.

Students have to submit Lab Exercise report of previous lab into corresponding next lab, and can be collected back after the instructor/course coordinator after it has been checked and signed. At the end of the semester, students should compile all the Lab Exercise reports into a single report and submit during the end semester sessional examination.

"Sample of Lab report" is shown for LAB Sheet #1 in this manual. For the rest of the labs, the reporting style as provided is to be followed. The lab report to be submitted during the end semester Sessional Examination should include at least the following topics: -

1. Top Cover page (to be used while compiling all the Lab Exercise reports into a single report)
2. Index (to be used while compiling all the Lab Exercise reports into single report)
3. Cover page (to be attached with every Lab Exercise)
4. Title
5. Objective
6. Theory (Syntax and Example)
7. Problem
8. Query
9. Test Output

For additional lab exercises given in each module, students have to show query and output to the instructor or course coordinator in the lab.

Note: The lab exercises may not be completed in a single specific lab. Students are encouraged to complete the questions given in the exercise prior to come to the lab hour and do the lab for the given problems.

Pokhara University

National Academy of Science and Technology (NAST)

DATABASE MANAGEMENT SYSTEM LAB REPORT

Submitted By:

Name: ___________________________

Roll No. ____________________________

Submitted To:

Department of Computer Engineering

Submission Date Signature

National Academy of Science and Technology (NAST)

Uttar Behadi-04, Dhangadhi

List of Lab Exercise

S.N.	Lab Exercise	Sheet	Remarks
1.	DDL Commands	Lab Sheet #1	
2.	DML Commands	Lab Sheet #2	
3.	Constraints	Lab Sheet #3	
4.	Aggregate Functions	Lab Sheet #4	
5.	Group By, Order By and Having Clause	Lab Sheet #5	
6.	Types of Operators	Lab Sheet #6	
7.	Sub-queries	Lab Sheet #7	
8.	Joins	Lab Sheet #8	
9.	Views	Lab Sheet #9	
10.	Index	Lab Sheet #10	
11.	TCL Commands	Lab Sheet #11	
12.	DCL Commands	Lab Sheet #12	

DATABASE MANAGEMENT SYSTEM

Lab Report

4th Semester, BE Computer

LAB Sheet #1

Submitted By	**Submitted To**
Name:	Instructor/Lecturer Name
Roll No:	Signature:

Submission Date

…………………

National Academy of Science and Technology (NAST)

Uttar Behadi-04, Dhangadhi

Lab Exercise 1

Title: Implementation of DDL Commands of SQL

- Create
- Alter
- Drop
- Rename
- Truncate

Objective:

- To understand and use data definition language to write query for a database.

Theory:

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:

1. CREATE 2. ALTER 3. DROP 4. RENAME

1**. CREATE:**

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

Syntax:

```
CREATE TABLE <relation_name/table_name >
(
        field_1 data_type(size),
        field_2 data_type(size), .. .
);
```

Example:

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

2. 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:

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_1 newdata_type(Size), field_2 newdata_type(Size),....field_newdata_type(Size));

Example:

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:

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:

ALTER TABLE student RENAME COLUMN sname to stu_name;

3. 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:

DROP TABLE std;

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

Syntax:

RENAME TABLE old_relation_name TO new_relation_name;

Example:

```
RENAME TABLE std TO std1;
```

5. TRUNCATE: It is used to remove data permanently from table.

Syntax:

```
TRUNCATE TABLE <Table name>;
```

Example:

```
TRUNCATE TABLE student;
```

Desc command: Describe command is external command of Oracle. The describe command is used to view the structure of a table as follows.

Syntax: desc <table name>

Problem 1:

Create a table BUS with following Schema:

(BUS_NO , SOURCE , DESTINATION ,COUCHTYPE,FAIR)

Query:

```
CREATE TABLE BUS
(
        BUS_NO              varchar(5)      primary key,
        SOURCE               varchar(20),
        DESTINATION         varchar(20),
        COUCHTYPE           varchar2(10),
        FAIR                number
);
```

Test Output:

```
desc bus;

Name                              Null?             Type
---------------------------       -----------       -------------------
BUS_NO                            NOT NULL          INTEGER2(5)
SOURCE                                              VARCHAR2(20)
DESTINATION                                         VARCHAR2(20)
COUCH TYPE                                          VARCHAR2(10)
FAIR                                                NUMBER
```

Problem 2:

Query:

Test Output:

Lab exercises (please code yourself and show the output to instructor):

1. Create a table EMPLOYEE with following schema:

   (Emp_no, E_name, E_address, E_ph_no, Dept_no, Job_id , Salary)
2. Add a new column; HIREDATE to the existing relation.
3. Change the datatype of JOB_ID from char to varchar2.
4. Change the name of column/field Emp_no to E_no.
5. Modify the column width of the job field of emp table
6. Delete all the data from table permanently.

Lab Exercise 2

Title: Implementation of DML Commands of SQL

- Insert
- Update
- Select
- Delete

Objective:

• 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. SELECT 4. DELETE

1**. INSERT INTO:** This is used to add records into a relation.

(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:

INSERT INTO student(sno,sname,class,address)VALUES (1,’Ram’,’M.E’,’Dhn’);

b) Inserting a single record

Syntax:

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

Example:

INSERT INTO student VALUES (1,’Ram’,’M.E’,’Dhn’);

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:

INSERT INTO std SELECT sno,sname FROM student WHERE name = ‘Ram‘;

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

(a)UDATE.. SET : This is used to update the content of all records.

Syntax:

UPDATE relation name SET Field_name1=data,field_name2=data;

Example:

UPDATE student SET sname = ‘kamal’, address = ’dhn’;

(b)UPDATE.. SET...WHERE.: This is used to change content of selected record.

Syntax:

UPDATE relation name SET Field_name1=data,field_name2=data,
WHERE field_name=data;

Example:

UPDATE student SET sname = ‘komal’ WHERE sno=1;

3. SELECT: To Retrieve data from one or more tables.

(a)SELECT FROM : To display all fields for all records.

Syntax:

SELECT * FROM relation_name;

Example :

select * from dept;

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

(b)SELECT FROM : To display a set of fields for all records of relation.

Syntax:

SELECT a set of fields FROM relation_name;

Example :

select deptno, dname from dept;

```
DEPTNO              DNAME
-------             ----------
10                  ACCOUNTING
20                  RESEARCH
30                  SALES
```

(c) 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:

select * FROM dept WHERE deptno<=20;

```
DEPTNO     DNAME           LOC
------     -----------     ------------
10         ACCOUNTING      NEW YORK
20         RESEARCH        DALLAS
```

4. DELETE: 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:

DELETE FROM relation_name;

Example:

DELETE FROM std;

(b) DELETE -FROM-WHERE: This is used to delete a selected record from a relation.

Syntax:

```
DELETE FROM relation_name WHERE condition;
```

Example:

```
DELETE FROM student WHERE sno = 2;
```

Lab exercises (please code yourself and show the output to instructor):

Create a table EMPLOYEE with following schema:

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

1. Insert aleast 5 rows in the table.
2. Display all the information of EMPLOYEE table.
3. Display name, address, salary of each employee who works in department D10.
4. Update the city of Emp_no-12 with current city as Dhangadhi.
5. Display the details of Employee who works in department COMP.
6. Delete the email_id of employee James.
7. Display the complete record of employees working in SALES Department.

Lab Exercise 3

Title: 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 'Dhangadhi');

Lab exercises (please code yourself and show the output to instructor):

Create a table called EMP with the following structure.

```
Name            Type
----------      ----------------------
EMPNO            NUMBER (6)
ENAME           VARCHAR2 (20)
JOB             VARCHAR2 (10)
DEPTNO          NUMBER (3)
SAL             NUMBER (7,2)
```

Allow NULL for all columns except ename and job.

2. Add constraints to check, while entering the empno value (i.e) empno > 100.
3. Define the field DEPTNO as unique.
4. Create a primary key constraint for the table(EMPNO).
5. Write queries to implement and practice constraints.

Lab Exercise 4

Title: Implementation of Aggregate Function of SQL

- Count
- Sum
- Avg
- Min
- Max

Objective:

- To understand and implement various types of aggregate function in SQL.

Theory:

AGGREGATE FUNCTION: 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 (Sal) FROM emp;
```

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 (Sal) From emp;
```

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

Syntax:

```
AVG (Column name);
```

Example:

```
Select AVG (sal) FROM emp;
```

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

Syntax:

```
MIN (Column name);
```

Example:

```
SELECT MIN (Sal) FROM emp;
```

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

Syntax:

```
MAX (Column name);
```

Example:

```
SELECT MAX (Sal) FROM emp;
```

<u>Lab exercises (please code yourself and show the output to instructor):</u>

Create a table EMPLOYEE with following schema:

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

1. Select employee name, deptno whose salary is maximum.
2. Display all the information whose salary is minimum.
3. Count number of employee whose department is SALES.
4. Count total number of records of employee table.
5. Find average of given numbers: 10,20, 30.

Lab Exercise 5

Title: Implementation of different clauses

- Group By
- Order By
- Having

Objective:

- To learn the concept of group by, order by and having clause.

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:

```
SELECT EMPNO, SUM (SALARY) FROM EMP GROUP BY EMPNO;
```

2. 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 1:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

Example 2:

```
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. 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:

```
SELECT empno, ename, job FROM emp ORDER BY job;
```

Lab exercises (please code yourself and show the output to instructor):

Create a table with following schema:

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

Job (Job_id, category)

1. Display total salary spent for each job category.
2. Display employee who have maximum salary of each department.
3. Display number of employees working in each department and their department name.
4. Display the details of employees sorting the salary in increasing order.
5. Show the record of employee earning salary greater than 16000 in each department.

Lab Exercise 6

Title: Implementation of different types of operator

- Logical Operator
- Set Operator
- Comparison Operator
- Special Operator

Objective:

- To learn the concept of different types of operator.

Theory:

1. LOGICAL OPERATORS:

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

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Example:

```
SELECT * FROM Customers
WHERE Country='Nepal' AND City='Dhangadhi';
```

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

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

Example:

```
SELECT * FROM Customers
WHERE Country='Nepal' OR City='Dhangadhi';
```

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

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition ;
```

Example:

```
SELECT * FROM Customers
WHERE NOT Country='Nepal';
```

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

a. UNION: Returns all distinct rows selected by both the queries.

Syntax:

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

Example 1:

```
SELECT sname FROM student
UNION
SELECT tname FROM teacher;
```

Example 2:

```
SELECT sname FROM student where address = ‘dhn’
UNION
SELECT tname FROM teacher where address = ‘dhn’;
```

Note: For selecting duplicate values, use UNION ALL instead of UNION.

b. INTERSECT: Returns rows selected that are common to both queries.

Syntax:

```
SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;
```

Example 1:

```
SELECT sname FROM student
INTERSECT
SELECT tname FROM teacher;
```

Example 2:

```
SELECT sname FROM student where address = ‘dhn’
INTERSECT
SELECT tname FROM teacher where address = ‘dhn’;
```

c. MINUS: Returns all distinct rows selected by the first query and are not by the second.

Syntax:

```
SELECT column_name FROM table1
MINUS
SELECT column_name FROM table2;
```

Example 1:

```
SELECT sname FROM student
MINUS
SELECT tname FROM teacher;
```

Example 2:

```
SELECT sname FROM student where address = ‘dhn’
MINUS
SELECT tname FROM teacher where address = ‘dhn’;
```

3. COMPARISION OPERATOR: Comparison operators are used in the WHERE clause to determine which records to select. Here is a list of the comparison operators that you can use in MySQL:

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

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

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

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

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

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

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

Example 1:

```
SELECT *
FROM contacts
WHERE last_name <> 'Johnson';
```

Example 2:

```
SELECT *
FROM contacts
WHERE last_name = 'Johnson';
```

4. SPECIAL OPERATOR: The special operators are used with a WHERE or HAVING clause.

a. 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.

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

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

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

e. 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.

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

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

Example 1:

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails WHERE
Quantity = 10);
```

Example 2:

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

Example 3:

```
SELECT *
FROM Customers
WHERE CustomerName LIKE '_r%';
```

<u>Lab exercises (please code yourself and show the output to instructor):</u>

a. Create a table with following schema:

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

1. Display all the dept numbers available with the dept and emp tables avoiding duplicates.
2. Display all the dept numbers available with the dept and emp tables.
3. Display all the dept numbers available with both the dept and emp tables.
4. Display all the dept numbers available in emp and not in dept tables and vice versa.

b. Create a table with following schema:

Customer (Id, Name, Address, Age, Salary)

1. Write a query to find the salary of a person where age is <= 26 and salary >= 25000.
2. Write a query to find the salary of a person where age is <= 26 or salary > =33000.
3. Write a query to find the customer details using “IN” and “Between” operator where age can be 25 or 27.
4. Write a query to find the name of customer whose name is like “Ku%”.

Lab Exercise 7

Title: Implementation of sub-queries

Objective:

- To understand and use sub-queries and nested queries.

Theory:

SUB-Queries: 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..)

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE expr operator (select select_list from table);
```

Example 1:

```
SELECT lastName, firstName
FROM employees
WHERE officeCode IN (SELECT officeCode
                        FROM offices WHERE country = 'USA');
```

Example 2:

```
SELECT customerNumber, checkNumber, amount
FROM payments
WHERE amount = (SELECT MAX(amount) FROM payments);
```

Example 3:

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY (SELECT ProductID
                          FROM OrderDetails WHERE Quantity = 10);
```

Lab exercises (please code yourself and show the output to instructor):

Create a table with 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 at least one boat.
4. Find the ids of sailors who have reserved a red boat or a green boat.
5. Find the name and the age of the youngest sailor.
6. Find the average age of sailors for each rating level.

Lab Exercise 8

Title: Implementation of different types of joins

- Inner Join
- Outer Join
- Natural join..etc.

Objective:

- To learn the concept of different types of joins.

Theory:

JOIN: 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.

Types of Joins:

1. Simple Join
2. Self-Join
3. Outer Join
4. Natural Join
5. Cross Join
6. Inner Join

1. 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

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

Syntax:

```
SELECT column_list
FROM table1, table2....
WHERE table1.column_name = table2.column_name;
```

Example 1:

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

b. Non Equi-join: It specifies the relationship between columns belonging to different tables by making use of relational operators other than’=’.

Syntax:

```
SELECT column_list
FROM table1, table2....
WHERE table1.column_name > table2.column_name;
```

Example 1:

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

2. 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.

Syntax:

```
SELECT a.column_name, b.column_name...
FROM table1 a, table1 b
WHERE a.common_filed = b.common_field;
```

Example:

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

3. Natural Join: The SQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that, columns with the same name of associated tables will appear once only.

Syntax:

```
SELECT *
FROM table1
NATURAL JOIN table2;
```

Example:

```
SELECT *
FROM foods
NATURAL JOIN company;
```

4. Cross Join: The SQL CROSS JOIN produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table if no WHERE clause is used along with CROSS JOIN. This kind of result is called as Cartesian Product.
If WHERE clause is used with CROSS JOIN, it functions like an INNER JOIN

Syntax:

```
SELECT *
FROM table1
CROSS JOIN table2;
```

Example:

```
SELECT foods.item_name,foods.item_unit,
company.company_name,company.company_city
FROM foods
CROSS JOIN company;
```

5. Inner Join: The INNER JOIN selects all rows from both participating tables as long as there is a match between the columns. An SQL INNER JOIN is same as JOIN clause, combining rows from two or more tables.

Syntax:

```
SELECT *
FROM table1 INNER JOIN table2
ON table1.column_name = table2.column_name;
```

Example:

```
SELECT foods.item_name,foods.item_unit,
company.company_name,company.company_city
FROM foods
INNER JOIN company
ON foods.company_id =company.company_id;
```

6. Outer Join: It extends the result of a simple join. An outer join returns all the rows returned by simplejoin as well as those rows from one table that do not match any row from the table.

a. Full outer join: The FULL OUTER JOIN combines the results of both left and right outer joins and returns all (matched or unmatched) rows from the tables on both sides of the join clause.

Syntax:

```
SELECT *
FROM table1
FULL [OUTER] JOIN table2
ON table1.column_name=table2.column_name;
```

Example 1:

```
SELECT company.company_id,company.company_name,
company.company_city,foods.company_id,foods.item_name
FROM   company
FULL JOIN foods
ON company.company_id = foods.company_id;
```

b. Left outer join: The SQL LEFT JOIN (specified with the keywords LEFT JOIN and ON) joins two tables and fetches all matching rows of two tables for which the SQL-expression is true, plus rows from the first table that do not match any row in the second table.

Syntax:

```
SELECT *
FROM table1
LEFT [ OUTER ] JOIN table2
ON table1.column_name=table2.column_name;
```

Example 1:

```
SELECT company.company_id,company.company_name,
company.company_city,foods.company_id,foods.item_name
FROM   company
LEFT JOIN foods
ON company.company_id = foods.company_id;
```

c. Right outer join: The SQL RIGHT JOIN, joins two tables and fetches rows based on a condition, which is matching in both the tables (before and after the JOIN clause mentioned in the syntax below), and the unmatched rows will also be available from the table written after the JOIN clause (mentioned in the syntax below).

Syntax:

```
SELECT *
FROM table1
RIGHT [ OUTER ] JOIN table2
ON table1.column_name=table2.column_name;
```

Example 1:

```
SELECT company.company_id,company.company_name,
company.company_city,foods.company_id,foods.item_name
FROM   company
RIGHT JOIN foods
ON company.company_id = foods.company_id;
```

Lab exercises (please code yourself and show the output to instructor):

Create a table with following schema:

Student (sid, sname, sex, dob,dno)
Department (dno, dname)
Faculty (F_id, fname, designation, salary,dno)
Course (cid, cname, credits,dno)
Register (sid,cid,sem)
Teaching (f_id,cid,sem)
Hostel(hid,hname,seats,)

Write query statement for the following queries.

1. List out the ID, Name and Date of Birth of students registered for a specific course.
2. List out the ID, Name and Date of Birth of students registered for a specific course, staying in a specific Hostel.
3. List the names of faculties who teach for a specific course.
4. Display the student details by implementing left inner join.
5. Display the student details by implementing a right outer join.

Lab Exercise 9

Title: Implementation of views

Objective:

- To understand the implementation of views.

Theory:

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 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.

<u>Creating view</u>

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example 1:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

Example 2:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

<u>Selecting a data set from a view</u>

Syntax:

```
SELECT column1, column2, ...
FROM view_name
WHERE condition;
```

Example:

```
SELECT * FROM [Brazil Customers];
```

<u>Replacing a view</u>

Syntax:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example:

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
```

```
WHERE Country = 'Brazil';
```

<u>Dropping a view</u>
Syntax:

```
DROP VIEW view_name;
```

Example:

```
DROP VIEW [Brazil Customers];
```

<u>Lab exercises (please code yourself and show the output to instructor):</u>

Create a table with following schema:

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

Write query statement for the following queries.

1. Create a view emp from employee such that it contains only emp_no and emp_name and department.
2. Create a view dept from department with only dept_no and location.
3. Create a view that contains the details of employees who lives in Dhangadhi.
4. Drop the views.

Lab Exercise 10

<u>Title:</u> Implementation of Index

<u>Objective:</u>

- To understand the implementation of index.

<u>Theory:</u>

INDEX: Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

<u>Creating Index</u>
Syntax:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

Syntax:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

Example 1:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

Example 2:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

<u>Dropping a index</u>

Syntax 1:

```
ALTER TABLE table_name
DROP INDEX index_name;
```

Syntax 2:

```
DROP INDEX index_name;
```

Example:

```
DROP INDEX idx_pname;
```

<u>Lab exercises (please code yourself and show the output to instructor):</u>

Create a table with following schema:

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

Write query statement for the following queries.

1. Create an index on employee table, Emp_no.
2. Create an index on employee table, Emp_no,Dept_no.
3. Create a unique index on employee table, Emp_no.
4. Drop the index.

Lab Exercise 11

<u>Title:</u> Implementation of TCL Commands

- Rollback
- Commit
- Save point

<u>Objective:</u>

- To understand the implementation of transaction control commands.

Theory:

Transaction Control Language(TCL): 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:

```
COMMIT;
```

Example:

```
COMMIT;
```

2. **SAVEPOINT:** 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 roll back.

Syntax:

```
SAVE POINT ID;
```

Example:

```
SAVE POINT xyz;
```

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

Syntax:

```
ROLLBACK (current transaction can be roll back)
ROLLBACK to save point ID;
```

Example:

```
ROLLBACK;
ROLLBACK TO SAVE POINT xyz;
```

Lab exercises (please code yourself and show the output to instructor):

1. Write a query to implement the save point.
2. Write a query to implement the rollback.
3. Write a query to implement the commit

Lab Exercise 12

Title: Implementation of DCL Commands

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

Objective:

- To understand the implementation of data control commands.

Theory:

Data Control Language (DCL): DCL is short name of Data Control Language which includes commands such as GRANT and mostly concerned with rights, permissions and other controls of the database system.

1. **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:

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

Example:

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.)

2. **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)

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

Syntax:

GRANT privilege [privilege…. …] TO USER ;

Example:

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.

Syntax:

GRANT object_priv [(column)] ON object TO user;

Example 1:

GRANT select, insert ON emp TO James;

Example 2:

GRANT select, update (e_name,e_address) ON emp TO James;

4. **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>;

Example:

ALTER USER James IDENTIFIED BY sam;

5. **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;

Example 1:

REVOKE create session,create table from James;

Example 2:

REVOKE select, insert ON emp FROM James;

<u>Lab exercises (please code yourself and show the output to instructor):</u>

Employee (emp_id, emp_name, emp_contact, salary, mgr_emp_Id, Dept_No)

Department (Dept_No, Dept_name, Dept_Location)

1. Create a user name “Bhaskar” and provide all privileges on employee table.
2. Create a user who can only view information of department table.
3. Change the password of user “Bhaskar” for employee table.
4. Revoke all the privileges for employee table user.