ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

LAB MANUAL

Subject: Database Management System **Class:** SE CMPN A & B

Subject Code: CSL402 **Academic Year:** 2023-2024

Lab: 431

CSL402: Database Management System Lab

Lab Outcomes:	
CSC403.1:	Recognize the need of database management system
CSC403.2:	Design ER and EER diagram for real life applications
CSC403.3:	Construct relational model and write relational algebra queries.
CSC403.4:	Formulate SQL queries and PL/SQL constructs.
CSC403.5:	Apply the concept of normalization to relational database design.
CSC403.6:	Demonstrate the concept of transaction, concurrency, and recovery along with frontend-backend connectivity

Lab Outcomes: At the end of the course, the students will be able to	
LO1	Design ER /EER diagram and convert to relational model for the real world application.
LO2	Apply DDL, DML, DCL and TCL commands.
LO3	Write simple and complex queries .
LO4	Use PL / SQL Constructs.
LO5	Demonstrate the concept of concurrent transactions execution and frontend-backend connectivity

Sr. No	List of Experiment	CO mapped
1	Identify the case study and detail statement of problem. Design an Entity-Relationship (ER) / Extended Entity-Relationship (EER)Model	LO1, CSC403.1, CSC403.2
2	Mapping ER/EER to Relational schema model. Create and populate database using Data Definition Language (DDL) and DML.	LO2 CSC403.3, CSC403.4, CSC403.5
3	Apply Integrity Constraints for the specified system. Use DCL and TCLcommands to provide security to data.	LO2, CSC403.3, CSC403.4
4	Perform Simple queries based on string manipulation, Numeric, Characterand Date SQL functions	LO3,CSC403.4
5	Implement Aggregate function, Group, Set operators, Simple queries	LO3,CSC403.4
6	Perform Join operations and Complex queries	LO3,CSC403.4
7	Implement Views and Triggers	LO3,CSC403.4

8	To implement PL/SQL Basic loops.	LO4,CSC403.4
9	To implement Procedures .	LO4,CSC403.4
10	To implement Function .	LO4,CSC403.4
11	To implement database connectivity.	LO5, CSC403.6

Faculty In-charge

Prachiti Pimple

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

EXPERIMENT NO. 1

Aim: Identify the case study and detail statement of problem. Design an Entity-Relationship (ER) / Extended Entity-Relationship (EER) Model.

Theory:

1. Features of Database system
2. Explain ER & EER- Entities, Attributes, Relationship, Specialization, generalization with notations.

Implementation:

1. Identify Case study and write detail problem statement.
2. Draw Entity Relationship diagram
3. If EER concept available, then extent ER to EER Diagram

Conclusion:

Your understanding about the experiments and lab Outcomes attained.

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

EXPERIMENT NO. 2

Aim: Mapping ER/EER to Relational schema model. Create and populate database using Data Definition Language (DDL) and DML.

Theory:

1. Explain steps of ER/ EER to relational Schema with example.
2. Explain DDL and DML with syntaxes and examples.

Implementation:

1. For the ER/ EER diagram designed in first experiment, convert ER/EER to relation schema design.
2. Use DDL and DML syntax to create and populate database for your Problem Statement.

Practical:

1. Create users

   To create user use the below syntax:

   Create User <username> identified by <Password> grant connect, resource to <username>

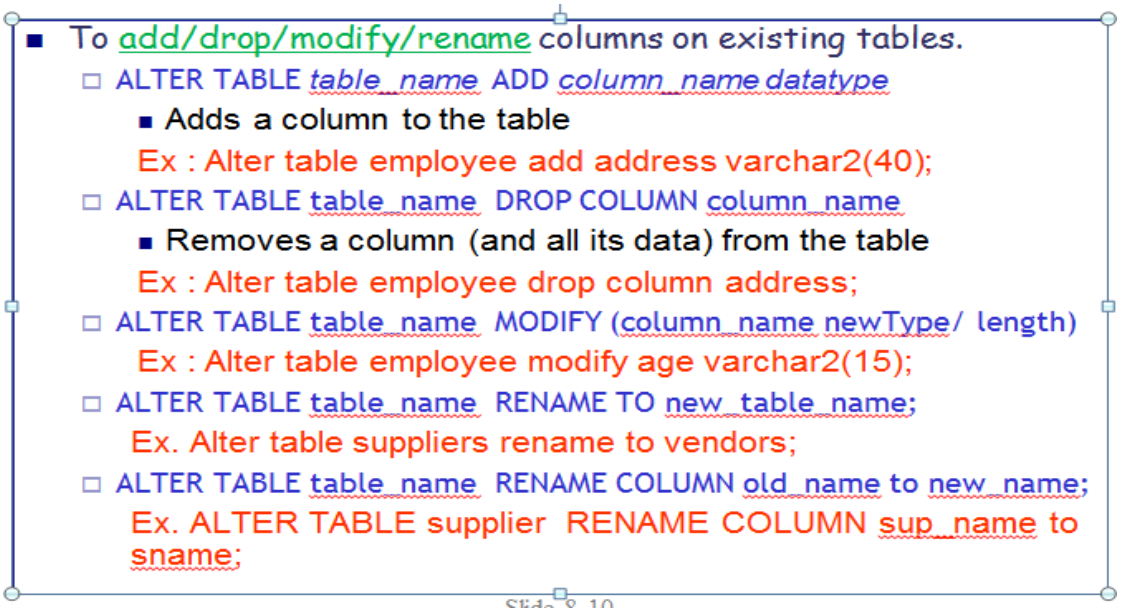
2. Create the following relational Schema using DDL and DML Commands under user login

DDL Commands:

a) Create

CREATE TABLE <table name> (field1 datatype (NOT NULL), field2 datatype);

b) Alter



c) **Truncate**

Truncate table table_name;

d) **Drop**

Drop table table_name;

DML Commands:

Insert into <tablename> values ('Rahul', 2000, '02-Jan-2018');

Update <tablename> set salary=3000 where name= 'Rahul';

Delete from <tablename> where name= 'Rahul';

Conclusion:
Your understanding about the experiments and lab Outcomes attained.

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

EXPERIMENT NO. 3

Aim: Apply Integrity Constraints for the specified system. Use DCL and TCL commands to provide security to data.

Theory:

1. Explain different SQL integrity Constraint with example
2. Explain DCL and TCL with Syntaxes and examples.

Implementation:

1. Apply SQL Integrity constraint for given exercise
2. Use DCL and TCL syntax for given exercise

Lab Manual:

1. **Null Constraint:** It means an Unknown Value.

   E.g. mobile number (10) null

2. **Not Null Constraint:** It means always a Known Value.

   E.g. Name varchar2 (20) not null

3. **Unique Constraint:** It ensures that no two rows have the same value in the specified column(s). I.e. Known Value (Distinct) or Unknown Value.

   E.g. ecode number (5) unique

4. **Primary Key Constraint:** It is similar to unique constraint except that the Primary Key cannot allow Null values so that this constraint must be applied to columns declared as Not Null. I.e. Always Known Value (Distinct).

   E.g. Empid char (5) primary key

5. **Default Constraint:** A default value can be specified for a column using default clause when a user does not enter a value for that column.

   E.g. Grade char (2) default 'E1'

6. **Check Constraint:** It limits values that can be inserted into a column.

   E.g. Sal number (10) check (Sal > 2000)

7. **Foreign Key Constraint:** References primary key of another table

   E.g. D_id references Employee

DCL Commands:

a) Grant

Grant <permissions> 'select, insert, delete, update on <object_name> to <username>;
E.g. Grant insert on emp to user1; //only user1 can insert
Grant all on emp to public; //assign all permissions to all users.

b) Revoke

Revoke <permission> on <object_name> from <username>;
E.g. revoke all on emp from user1; //get back all permissions from user1.
Revoke select on emp from public; get back select permission from all users

TCL Commands:

1. **Commit:**
   E.g. Commit;// end or start a transaction

2. **Rollback:**
   E.g. Rollback; //undo up to commit

3. **Savepoint:**
   E.g. savepoint P1;

4. **Rollback to:**
   E.g. rollback to P1;

Exercise:

1. **SQL Integrity Constraint:**
   a. Create table Employee (Eid, Ename, E_country, E_contact,E_Salary)
   b. EID: Eid has to be unique and no null accepted.
   c. Ename: Ename has to be provided by the user.
   d. E_country: If user doesn't provide E_country value then country should be India
   e. E_contact: If user doesn't't have contact details then null can be accepted
   f. E_Salary: Make Salary as check (salary inserted should not be less than 30000)

2. DCL and TCL Exercise:

a. create one more user (your user account already there and above employee table is also in your user account)
b. Grant inserts privilege from your user login to other user which you have just created.
c. Add 2 records using DML Syntax insert through other user login.
d. Then revoke the inert privilege from your account and check again trying to insert record from other user account.
e. login to your user account, retrieve the table records using select
f. Save your transaction using commit.
g. Insert 1 records using DML Syntax insert through your user login.
h. Create savepoint A
i. Insert 1 records using DML Syntax insert through your user login.
j. Create savepoint B
k. Roll back till save point B
l. Check the table records
m. perform Rollback
n. Check the table records again.

Conclusion:
Your understanding about the experiments and lab Outcomes attained.

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

EXPERIMENT NO. 4

Aim: Perform Simple queries based on Numeric, Character and Date SQL functions.

Theory:

Explain different Numeric, Character and Date SQL functions.

Lab Manual:

a) Numeric Functions: It accept numeric input and return numeric values as output.

i. Abs (n): This function returns the absolute value of the given number.

SQL> select ABS (-15) from dual;

Absolute

15

ii. Cos: COS returns the cosine of an angle expressed in radians.

SQL> select cos (0) from dual;

COS (0)

1

iii. Sin: SINreturns the sine of an angle expressed in radians.

SQL> select sin (30 * 3.14/180) "Sine" from dual;

Sine

.5

iv. Tan: TANreturns the tangent of an angle expressed in radians.

SQL> select tan (135 * 3.14/180) "Tan" from dual;

Tan

-1

v. Round: ROUND returns *a value* rounded to *integer* specified to be placed to the right of thedecimal point.

SQL> select round (15.193, 1) "Round" from dual;

Round

15.2

vi. Truncate: The TRUNC function returns *a number value (x)* truncated by a number (y) to its decimal places.

SQL> select Trunc(15.79, 1) "Truncate" from dual;

Truncate

15.7

vii. Power: POWERreturns *n2* raised to the *n1* power.**SQL> select Power (3, 2) "Power" from dual; Power**

9

b) Character Functions: It accept character input and return character values as output.

a) LOWER: Converts mixed case or uppercase character string to lowercase.
SQL> select lower (city) from emp;

LOWER (CITY)
----
mumbai
pune
nagpur

b) UPPER: Converts mixed case or lowercase character string to uppercase.
SQL> select upper (city) from emp;

UPPER (CITY)

MUMBAI
PUNE
NAGPUR

c) INITCAP: Converts first letter of each word to uppercase and remaining letters to lowercase.
SQL> select initcap (f_name) from emp;

INITCAP (F_NAME)

Anil
Sunil
Smita

d) CONCATE: Use to concatenate value of one or more columns and display it or concatenate string with the columns and display it.
SQL> select concat(ID, f_name) from emp;
CONCAT (ID, F_NAME)

E01 Anil

E02 Sunil
E03 Smita

e) **LTRIM:** Use to trim/cut characters contained in the set from the left side.
SQL> select Ltrim('Nagpur', 'Nag') as Leftm from emp where f_name= 'Anil';
LEF
------
pur

f) **RTRIM:** Use to trim/cut characters contained in the set from the right side.
SQL> select Rtrim ('Mumbai', 'bai') from emp where f_name= 'Sunil';

RTR

Mum

g) **SUBSTRING:** It returns a part of string, beginning at a given character position and ends where the substring_length specified ends. If the staring character of substring not specified then it takes first character as default.
SQL> Select substr ('Mumbai', 1, 3) "Substring" from dual;
Substring

Mum

h) **LENGTH:** It takes the character as input and returns the length of the string as an output. Output is measured in bits by default, for measuring in bytes we need to write Lengthb.
SQL> select Length ('Nagpur') as Len from emp where f_name= 'Smita';
LEN

6

i) **REPLACE:** It takes the string as input in which character needs to be replaced. Also takes old and new characters that are to be replaced with each other. So it searches the old character occurrence in the string and replaces it with the new character.
SQL> Select replace ('Jack and Jue','J', 'BL') "Changed_String" from dual;
Changed_String

Black and Blue

c) **Date Time Functions:** Oracle stores dates in default date format as DD-MON-YY. It basically comprises of: century, year, month, day, hours, minutes, and seconds.

a) **TO_CHAR:** Converts a date or number to a string
SQL> Select to_char (DOJ, 'Month, DD, Year') as "Date_in_string" from Month;
Date_in_string

JAN, 01, 2013
JAN, 01, 2013

b) **ADD MONTH (D, N):** It takes as input a date and adds one month to that date and displays the result.
SQL> Select add_months (DOL, 1) as "Added month" from Month;

Added month

01-MAR-13
01-MAY-13

c) **LAST DAY (D):** It takes as input a date and for that date returns the last date of its month.
SQL> Select Last_day (DOL, 1) as "Last day of month" from Month;
Last day of month

28-FEB-13
30-APRIL-13

d) **MONTHS_BETWEEN (D1, D2):** It takes as input 2 dates and returns number of months between these two dates.
SQL> Select months_between (DOL, DOJ) as "Bet_month" from Month;

Bet_month

1
3

e) **MONTH (DATE):** It takes as input a date and returns the month number of that date as a result.
SQL> Select month (DOL) as "Month no" from Month;

Month no

2
4

f) **MONTHNAME (DATE)** It takes as input a date and returns the month name of that date as a result.
SQL> Select monthname (DOL) as "Month name" from Month;
Month name

February
April

g) **DAYNAME (DATE):** It takes as input a date and returns the weekday name of that date as a result.

SQL> Select dayname (DOL) as "Week day" from Month;
Week day

Friday
Monday

h) **DAYOFMONTH (DATE)**: It takes as input a date and returns which day it is of that month, the result is always between 1 t0 31.
SQL> Select dayofmonth (DOL) as "Day no" from Month;
Day no

1
1

i) **DAYOFWEEK (DATE):** It takes as input a date and returns which day it is of the week, the result is always between 1 t0 7.
SQL> Select dayofweek (DOL) as "Week Day no" from Month;
Week Day no

6
2

j) **DAYOFYEAR (DATE):** It takes as input a date and returns which day it is of that year, the result is always between 1 t0 365.
SQL> Select dayofyear ('27-02-03') as "Year Day no" from dual;
Year Day no

34

k) **NOW ():** It is a date function used to display from dual the system current date and time together.
SQL> Select Now ( );
Now ()

'2014-06-17 23:50:26'

l) **CURDATE ():** It is a date function used to display from dual the system current date.
SQL> Select Curdate ( );
Curdate ()

'2014-06-17'

m) **CURTIME ():** It is a date function used to display from dual the system current time.
SQL> Select Curtime ( );

Curtime ()

'23:50:26'

Implementation:

5. Apply SQL function for given exercise

Exercise:

1. Create table Flight(F_id, DoT, DoI, Time, Pass_name, Source, Destination, Fare) where DOT is Date of travel and DOI is the date of issue
2. Insert 5 values
3. Display today's date and time in the prompt.
4. Display the absolute value of -184.
5. Select a value from the dual and for that value find its cube.
6. Display the date (doI) 2 months after date of Issue of Ticket.
7. Display the last day of month of date of Travel.
8. Display the month between date of travel and date of Issue.
9. Display the next occurrence of Monday from the day of Travel.
10. Display the First letter of Pass_name into capitals.
11. Display the Pass_name into upper case.
12. Display the Destination & source name into Lower case.
13. Display the first 3 characters of the Destination place name.
14. Display the last 3 characters of the Destination place name.
15. Display the pass name that begins with 'm' and replace with 'B'.
16. Display only 3 characters from the 3rd character with names of Source.
17. Display the rounded value of fare up to 2 characters.
18. Display 20th September 2008 in the date format.
19. Display the day truncated up to the year for the DOT in the Flight table.
20. Take DOT as input and display the output like "my travel Date is_____".

Conclusion:

Your understanding about the experiments and lab Outcomes attained.

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

EXPERIMENT NO. 5

Aim: - Implement SQL operators, Aggregate function, Group by and having clause.

Theory:-

1. Set Operator
2. Aggregate function
3. Group by clause with having

Lab Manual:

1. SQL Operators:

1. Arithmetic Operators:

Arithmetic operators perform mathematical operations on two expressions of one or more of the data types of the numeric data type category.
SELECT <Expression> [arithmetic operator]<expression>... FROM [table_name] WHERE [expression]; -

Operator	Description	Example
/	Division (numbers and dates)	SELECT SAL / 10 FROM EMP;
*	Multiplication	SELECT SAL * 5 FROM EMP;
+	Addition (numbers and dates)	SELECT SAL + 200 FROM EMP;
-	Subtraction (numbers and dates)	SELECT SAL - 100 FROM EMP;

2. Comparison/Relational Operator:

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.

<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

3. Logical/Boolean Operators: (AND, OR, NOT, Between, Exists, IN, NOT IN, Like)

IN	"Equivalent to any member of" test. Equivalent to "= ANY".	SELECT * FROM EMP WHERE ENAME IN ('SMITH', 'WARD');
ANY/ SOME	Compares a value to each value in a list or returned by a query. Must be preceded by =, !=, >, <, <=, or >=. Evaluates to FALSE if the query returns no rows.	SELECT * FROM DEPT WHERE LOC = SOME ('NEW YORK','DALLAS');
[NOT] IN	Equivalent to "!= ANY". Evaluates to FALSE if any member of the set is NULL.	SELECT * FROM DEPT WHERE LOC NOT IN ('NEW YORK', 'DALLAS');
ALL	Compares a value with every value in a list or returned by a query. Must be preceded by =, !=, >, <, <=, or >=. Evaluates to TRUE if the query returns no rows.	SELECT * FROM emp WHERE sal >= ALL (1400, 3000);
[NOT] BETWEEN *x* and *y*	[Not] greater than or equal to *x* and less than or equal to *y*.	SELECT ENAME, JOB FROM EMP WHERE SAL BETWEEN 3000 AND 5000;
EXISTS	TRUE if a sub-query returns at least one row.	SELECT * FROM EMP WHERE EXISTS (SELECT ENAME FROM EMP WHERE MGR IS NULL);
[NOT] LIKE	TRUE if *x* does [not] match the pattern *y*. Within *y*, the character "%" matches any string of zero or more characters except null. The character "_" matches any single character.	SELECT * FROM EMP WHERE ENAME LIKE '%E%';
IS [NOT] NULL	Tests for nulls. This is the only operator that should be used to test for nulls.	SELECT * FROM EMP WHERE COMM IS NOT NULL AND SAL > 1500;

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

Exercise 1 on SQL Operator:

1. **Create table employee with the following structure : (Emp_id, Ename, Salary, Age, DOJ, DEPT)**

2. **INSERT 5 RECORDS**

EMP_ID	ENAME	SALARY	AGE	DOJ	DEPT
1	Suraj	50000	27	05-April-2012	Finance
2	Bhaviya	70000	29	03-Dec-2013	Finance
3	Saijal	90000	30	23-June-2014	Marketing
4	Ronak	35000	42	22-Oct-2011	Marketing
5	Aniket	60000	52	20-Feb-2011	HR

❖ **Queries on SQL Operator:**

1. Find all employee names that have salary greater than 50000.
2. Give 10% raise in salary of each employee
3. Give the details of employee joined from 05-april-2012 to 23-june-2014.
4. Find all employees who are having salary greater than 70000 and have joined after 3 dec 2013.
5. Find all employees with name starting with s.
6. Find all employees who have at least one 'e' in their names.
7. Find all employees with age either 29 or 30.
8. Find all employees who have not joined on 05-april-2012.
9. Alter the table by adding new column as amount deducted from salary towards tax. Update the value of tax in the table as 20% of salary.
10. Calculate the net salary for each employee.
11. Find all employees whose age is greater than 25 and earns salary. (use exists clause)
12. Find all employee names whose age is from the list given '25, 30, 24, 29'.
13. Find all employee names who has not joined on these dates {22-oct-201, 20-feb-2011, 03-dec-2013}.
14. List all employees in descending order of their salary.
15. List all employees name in ascending order with joining date '05-april-2012' or after this date.
16. List the employees whose age is not null.
17. List the employees whose age is greater than the age of all the employees having salary greater than 5000.

****

2. Aggregate function, group by and having clause:

Aggregate functions take a collection of values i.e. a set or multiple set of values as input and return a single value as output. These are often referred to as Group functions as they operate on groups of rows and return one value for the entire group.

Different Aggregate Functions are:

1. **avg:** average value- Avg(value)
2. **min:** minimum value- Min(value)
3. **max:** maximum value- Max(value)
4. **Sum:** sum of values- Sum(value)
5. **Count:** number of values- Count(value)

SQL GROUP BY Syntax:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;
```

SQL HAVING Syntax:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING [conditions]
    ORDER BY column;
```

Exercise 2 on aggregate function, group by and having clause:

1. Find average salary of all employees for a particular department.
2. Find the details of that employee who has maximum salary in all departments.
3. Find the details of employee who has minimum salary and who has joined after 23-oct-2011.
4. What are the total numbers of rows in the Employee table?
5. For all of the employees in computer department, what is the joining date of an employee with lowest salary in that department?
6. Display the name of each employee department wise and order it in descending order.

7. Find the total salary of all employees for each department.
8. Find the names of all departments where the average salary of employee is more than 30,000.
9. Sort Employee name in ascending order, and if Employee name is same, then it is sorted Department wise in descending order.
10. Count the number of Employees in each department.
11. Display the name of Employees who have joined on the same date.
12. Display the name of employees who have less than 2 employees in a particular department.

Conclusion:
Your understanding about the experiments and lab Outcomes attained.

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

/

EXPERIMENT NO. 6

Aim: - Perform Join operations and Complex queries.

Theory:-

1. Different types of Joins.
2. Set Comparison & set membership operator

Lab Manual:

1. SQL JOINS:

Here are the different types of the JOINs in SQL:

1. **EQUI JOIN:** Most joins are "equi-joins" where the data from a column in one table exactly matches data in the column of another table. (P.K & F.K).

2.
```
SELECT * FROM
Employee e, department d WHERE
e.DepartmentID = d.DepartmentID;
```
```
SELECT * FROM employee
  NATURAL JOIN
```
NATURAL JOIN: Natural join is basically an equijoin followed by removal of the superfluous attributes.

3. **(INNER) JOIN**: Returns records that have matching values in both tables

4.
```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1

INNER JOIN table2

ON table1.matching_column = table2.matching_column;
```
LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table.

5.
```
SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching_column = table2.matching_column;
```
RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table.

6.
```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table.

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

7. **SELF JOIN:** A self-join is joining a table to itself

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

4. SET COMPARISON OPERATORS:

A sub query is a Select-From-Where expression that is nested within another query.
Sub query is to perform tests for

1. Set membership
    - **IN** – tests for presence of set membership
        - As Value
            - Select distinct customer-name from borrower where address in (‘Mumbai’, ’Pune’);
        - As Query
            - Select distinct customer-name from borrower where customer-name in (select customer-name from depositor);
    - **NOT IN** – tests for absence of set membership
        - As Value
            - Select distinct customer-name from borrower where customer-name not in (‘Smith’, ’Jones’)
        - As Query
            - Select distinct customer-name from borrower where customer-name not in (select customer-name from depositor)

2. Make Set comparisons
    - **Some or any Clause: greater than at least one**

- Select b_name from Branch Where assets >some (Select assets from branch where b_city='Brooklyn');

- **all Clause: greater than all**
  - **Select b_name from Branch Where assets >all (Select assets from branch where b_city='Brooklyn');**

Lab Exercise:-

1. For the following relational Schema, Solve the queries:

Table:-Employee

EmpId	EmpName	Department	ContactNo	EmailId	EmpHeadId
101	Isha	E-101	1234567890	abc@gmail.com	105
102	Priya	E-104	1234567890	abc@gmail.com	103
103	Neha	E-101	1234567890	abc@gmail.com	101
104	Rahul	E-102	1234567890	abc@gmail.com	105
105	abhishek	E-101	1234567890	abc@gmail.com	102

Queries:

1. **For the above given Relational Schema, Perform the following Join Operations**:

i. Use Natural Join, to join employee and EmpDept tables.

ii. Use Right outer Join, to join employee and EmpDept tables.

iii. Use Self Join, to display the names of Employee head along with each employee details.

Table:-EmpDept

DeptId	DeptName	DeptHead
E-101	HR	105
E-102	Development	101
E-103	House Keeping	
E-104	sales	104
E-105	purchase	104

Table:-EmpSalary

EmpId	Salary	IsPermanent
101	2000	Yes
102	10000	Yes
103	5000	No
104	1900	Yes
105	2300	yes

iv. Use Inner Join, to join employee and EmpDept tables for department id doesn't match.

v. Use Equi Join, to join employee and EmpDept tables.

2. **For the above given Relational Schema, Perform the following Nested Sub Query Operations:**
   i. Display the department details of a company which is assigned to the employee with employee id above 103.
   ii. Display the details of Employee who is working under 'Priya'.
   iii. Display the details of Employee who is the department head of HR.
   iv. Display the detail of employee who is working in 'development' department and they are permanent.
   v. Display the salary of the employee who is currently working in the HR Department and is a permanent employee.
   vi. Display the maximum salary of an employee with its details from each department.
3. **For the above given Relational Schema, Perform the following Set Operator Operations:**
   i. Display the department which is not yet being assigned to any employee up till now.
   ii. Find Id of employee for salary less than 5000 and greater than 2300.
   iii. Find Id of employee for salary less than 2000.
   iv. Find Employee Names starting from A, P and N.
   v. Find employee details other than employee having salary less than 2000.

Conclusion:

Your understanding about the experiments and lab Outcomes attained.

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

EXPERIMENT NO. 7

Aim: - Perform View and Triggers in SQL.

Theory:-

1. Views. Its advantages and dis advantages
2. Triggers.

Lab Manual:

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

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

Views, which are a type of virtual tables allow users to do the following −

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

SQL CREATE VIEW Statement:

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.

SQL CREATE VIEW Syntax:

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition;
```

Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

SQL CREATE VIEW Examples

The view "Current Product List" lists all active products (products that are not discontinued) from the "Products" table. The view is created with the following SQL:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID, ProductName
FROM Products
WHERE Discontinued=No;
```

We can query the

view above as follows:

```
SELECT * FROM [Current Product List];
```

SQL Updating a View:

You can update a view by using the following syntax:

SQL CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

SQL Dropping a View:

You can delete a view with the DROP VIEW command.

SQL DROP VIEW Syntax

```
DROP VIEW view_name;
```

Triggers:

Triggers are similar to stored procedures. A trigger stored in the database can include SQL and PL/SQL or Java statements to run as a unit and can invoke stored procedures. However, procedures and triggers differ in the way that they are invoked. A procedure is explicitly run by a user, application, or trigger. Triggers are implicitly fired by Oracle when a triggering event occurs, no matter which user is connected or which application is being used. A trigger can also call out to a C procedure, which is useful for computationally intensive operations. The events that fire a trigger include the following:

- DML statements that modify data in a table (INSERT, UPDATE, or DELETE)
- DDL statements
- System events such as startup, shutdown, and error messages
- User events such as logon and logoff

How Triggers Are Used

Triggers supplement the standard capabilities of Oracle to provide a highly customized database management system. For example, a trigger can restrict DML operations against a table to those issued during regular business hours. You can also use triggers to:

Automatically generate derived column values
Prevent invalid transactions
Enforce complex security authorizations
Enforce referential integrity across nodes in a distributed database
Enforce complex business rules
Provide transparent event logging
Provide auditing
Maintain synchronous table replicates
Gather statistics on table access
Modify table data when DML statements are issued against views
Publish information about database events, user events, and SQL statements to subscribing applications

There are three components in trigger

Event: When this event happens, the trigger is activated
Condition (optional): If the condition is true, the trigger executes, otherwise skipped
Action: The actions performed by the trigger
Semantics: When the Event occurs and Condition is true, execute the Action

Trigger syntax:

```
CREATE TRIGGER <triggerName>
BEFORE|AFTER  INSERT|DELETE|UPDATE
    [OF <columnList>] ON <tableName>|<viewName>
    [REFERENCING [OLD AS <oldName>] [NEW AS
    <newName>]]
[FOR EACH ROW] (default is “FOR EACH STATEMENT”)
[WHEN (<condition>)]
< trigger body>;
```

In SQL*Plus, you can also use the following shortcut to view compilation errors:

SQL> **SHOW ERRORS TRIGGER** MY_TRIGGER

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

Lab Exercise on Views:-

1. Create table Employee having eid varchar (5), ename varchar(15), emobile number(10), salary number(10), ecountry varchar(10), designation varchar(10).
2. Insert 5 values inside the Employee table.
3. Create view India_emp_view, list all from Employee table where ecountry='India'
4. Create view Manager_emp_view, list all details for employee where designation is manager.
5. Display India_emp_view
6. Display Manager_emp_view.
7. Insert 2 tuples in Employee where country of employee is India, and then check no. of records in India_emp_view.
8. Update Manager_emp_view, update its emobile, and then check the record in Employee table.

Lab Exercise on Triggers:-

1. For Relational Schema Employee (Eid, ename, emobile, salary, ecountry, designation),create following triggers:
    a. Write a trigger to avoid updating on Salary attribute for employee relation.
    b. Write a trigger to avoid insert on employee relation on Weekends.
    c. Write a trigger that displays the employee id for the record which gets deleted.
2. For Relational Schema Department (Did, Dname, Location,Dmgr),create following triggers:
    a. Write a trigger that displays the system date whenever there is an update on Location attribute for department relation.
    b. Write a trigger that outputs a statement stating old name which got updated by the new name whenever the Dmgr gets updated for department relation.

Conclusion:

Your understanding about the experiments and lab Outcomes attained.

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

EXPERIMENT NO. 8

Aim: To implement PL/SQL-Basic loops.

Theory:

1. **PL/SQL**

PL/SQL stands for Procedural Language extension of SQL. PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

Each PL/SQL program consists of SQL and PL/SQL statements which from a PL/SQL block.

PL/SQL Block consists of three sections:

- The Declaration section (optional).
- The Execution section (mandatory).
- The Exception (or Error) Handling section (optional).

Declaration Section:

The Declaration section of a PL/SQL Block starts with the reserved keyword DECLARE. This section is optional and is used to declare any placeholders like variables, constants, records and cursors, which are used to manipulate data in the execution section. Placeholders may be any of Variables, Constants and Records, which stores data temporarily. Cursors are also declared in this section.

Execution Section:

The Execution section of a PL/SQL Block starts with the reserved keyword BEGIN and ends with END. This is a mandatory section and is the section where the program logic is written to perform any task. The programmatic constructs like loops, conditional statement and SQL statements form the part of execution section.

Exception Section:

The Exception section of a PL/SQL Block starts with the reserved keyword EXCEPTION. This section is optional. Any errors in the program can be handled in this section, so that the PL/SQL Blocks terminates gracefully. If the PL/SQL Block contains exceptions that cannot be handled, the Block terminates abruptly with errors. Every statement in the above three sections must end with a semicolon **;** . PL/SQL blocks can be nested within other PL/SQL blocks. Comments can be used to document code.

How a Sample PL/SQL Block Looks

```
DECLARE
    Variable declaration
BEGIN
    Program Execution
EXCEPTION
    Exception handling
END;
```

2. Iterative Statements in PL/SQL

Iterative control Statements are used when we want to repeat the execution of one or more statements for specified number of times.

There are three types of loops in PL/SQL:

- Simple Loop
- While Loop
- IF Then Loop
- For Loop

A. Simple Loop

A Simple Loop is used when a set of statements is to be executed at least once before the loop terminates. An EXIT condition must be specified in the loop, otherwise the loop will get into an infinite number of iterations. When the EXIT condition is satisfied the process exits from the loop.

General Syntax to write a Simple Loop is

:

```
LOOP
  statements;
  EXIT;
  {or EXIT WHEN condition;}
END LOOP;
```

These are the important steps to be followed while using Simple Loop.

1) Initialize a variable before the loop body.

2) Increment the variable in the loop.

3) Use a EXIT WHEN statement to exit from the Loop. If you use a EXIT statement without WHEN condition, the statements in the loop is executed only once.

B. While Loop

A WHILE LOOP is used when a set of statements has to be executed as long as a condition is true. The condition is evaluated at the beginning of each iteration. The iteration continues until the condition becomes false.

The General Syntax to write a WHILE LOOP is:

WHILE <condition>

LOOP statements;

END LOOP;

i. Important steps to follow when executing a while loop: Initialize a variable before the loop body.
ii. Increment the variable in the loop.
iii. EXIT WHEN statement and EXIT statements can be used in while loops but it's not done often.

C. If THEN Loop

In this, if the condition is satisfied, then the statements are executed.

The general syntax for If then loop is:

IF condition THEN

  Statements

END IF;

IF ..THEN… ELSE Statement:

It is used to provide an alternate sequence of statements, if the condition is not TRUE.

IF condition THEN

  Statements A;

ELSE

Statements B;
END IF;

IF..THEN.. ELSEIF Statements
It is used as nested IF…THEN loop.
IF condition 1
THEN
Statements A;
ELSEIF condition 2;
THEN
Statements B;
ELSE Statements C;
END IF

D. FOR Loop

A FOR LOOP is used to execute a set of statements for a predetermined number of times. Iteration occurs between the start and end integer values given. The counter is always incremented by 1. The loop exits when the counter reaches the value of the end integer.
The General Syntax to write a FOR LOOP is:

FOR counter IN val1..val2
LOOP statements;
END LOOP;

- val1 - Start integer value.
- val2 - End integer value.

Important steps to follow when executing a while loop:

1) The counter variable is implicitly declared in the declaration section, so it's not necessary to declare it explicitly.
2) The counter variable is incremented by 1 and does not need to be incremented explicitly.
3) EXIT WHEN statement and EXIT statements can be used in FOR loops but it's not done often.

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

Exercise

1. Program to display a number.
2. Program to get salary of an employee with id E016 and display it on the screen.
3. Program to display grade based on marks obtained.
4. Program to find the largest value from two value use if then loop.
5. Program to add numbers and print the value using simple loop.
6. Program to find sum of first 10 numbers using for loop.
7. Program to print numbers from 1 to 10 using while loop.

Conclusion:

Your understanding about the experiments and lab Outcomes attained.

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

EXPERIMENT NO. 9

Aim: To implement Procedures.

Theory:

1. PL/SQL Procedures

A **subprogram** is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the calling program.

A subprogram can be created:

- At schema level
- Inside a package
- Inside a PL/SQL block

A schema level subprogram is a **standalone subprogram**. It is created with the CREATE PROCEDURE or CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.

A subprogram created inside a package is a **packaged subprogram**. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement. We will discuss packages in the chapter 'PL/SQL - Packages'.

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms:

- **Functions**: these subprograms return a single value, mainly used to compute and return a value.

- Procedures**:** these subprograms do not return a value directly, mainly used to perform an action.

Parts of a PL/SQL Subprogram

Each PL/SQL subprogram has a name, and may have a parameter list. Like anonymous PL/SQL blocks and, the named blocks a subprograms will also have following three parts:

S.N.	Parts & Description
1	**Declarative Part** It is an optional part. However, the declarative part for a subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local to the subprogram and cease to exist when the subprogram completes execution.
2	**Executable Part** This is a mandatory part and contains statements that perform the designated action.
3	**Exception-handling** This is again an optional part. It contains the code that handles run-time errors.

Creating a Procedure

A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END procedure_name;
```

Where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows modifying an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Exercise: Procedure

1. Calling procedure from SQL prompt.
2. Program to pass parameters to procedures for addition of n numbers.
3. Program to calculate area of a circle.
4. Procedure to update any table and reflect the changes.

Conclusion:

Your understanding about the experiments and lab Outcomes attained.

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

EXPERIMENT NO.10

Aim: To implement Functions.

Theory:

2. PL/SQL – Functions

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value**.**

Creating a Function

A standalone function is created using the CREATE FUNCTION statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
< function_body >
END [function_name];
```

Here:

- **Function_name:** specifies the name of the function.
- **[OR REPLACE]** option allows modifying an existing function.
- The **optional parameter list** contains name, mode and types of the parameters.
- **IN** represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.

Exercise : Functions

1. Program to implement factorial of a number using functions.
2. Program to write a function to concatenate the strings.
3. Program to write a function to implement multiplication of two numbers.
4. Program to retrieve the name of the employee having employee id “E02”.

Conclusion:

Your understanding about the experiments and lab Outcomes attained.

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

EXPERIMENT NO.11

Aim: To implement database connectivity.

Theory:

3. Features of **SQL** databases
4. Explain database connectivity in terms of insert, update and delete operation.

Implementation:

Code for database connectivity (Insert, delete and update operation) along with screenshots of outputs.

Conclusion:

Your understanding about the experiments and lab Outcomes attained.

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI