

Relational Database Management Systems (4330702)

Laboratory Manual

[Computer Engineering, Semester III]

Enrolment No	
Name	
Branch	
Academic Term	
Institute	



**Directorate of Technical Education Gandhinagar
- Gujarat**

DTE's Vision:

- To provide globally competitive technical education;
- Remove geographical imbalances and inconsistencies;
- Develop student friendly resources with a special focus on girls' education and support to weaker sections;
- Develop programs relevant to industry and create a vibrant pool of technical professionals.

Institute's Vision:

To cater skilled engineers having potential to convert global challenges into opportunities through embedded values and quality technical education.

Institute's Mission:

M1: Impart quality technical education and prepare diploma engineering professionals to meet the need of industries and society.

Adopt latest tools and technologies for promoting systematic

M2: problem solving skills to promote innovation and entrepreneurship

M3: Emphasize individual development of students by inculcating moral, ethical and life skills.

Department's Vision:

Develop globally competent Computer Engineering Professionals to achieve excellence in an environment conducive for technical knowledge, skills, moral values and ethical values with a focus to serve the society.

Department's Mission:

M1: To provide state of the art infrastructure and facilities for imparting quality education and computer engineering skills for societal benefit.

M2: Adopt industry oriented curriculum with an exposure to technologies for building systems & application in computer engineering.

To provide quality technical professional as per the industry and societal

M3: needs, encourage entrepreneurship, nurture innovation and life skills in consonance with latest interdisciplinary trends.



A. V. Parekh Technical Institute, Rajkot

CERTIFICATE

This is to certify that Mr./Ms
Enrolment No. of 3rd Semester of *Diploma in Computer Engineering* of
Computer Engineering Department, A. V. Parekh Technical Institute, Rajkot (602) has
satisfactorily completed the term work in course.....
..... for the academic year: Term: Odd prescribed in the GTU
curriculum.

Place:

Date:

Signature of Course Faculty

Preface

The primary aim of any laboratory/Practical/field work is enhancement of required skills as well as creative ability amongst students to solve real time problems by developing relevant competencies in psychomotor domain. Keeping in view, GTU has designed competency focused outcome-based curriculum - 2021 (COGC-2021) for Diploma engineering programmes. In this more time is allotted to practical work than theory. It shows importance of enhancement of skills amongst students and it pays attention to utilize every second of time allotted for practical amongst Students, Instructors and Lecturers to achieve relevant outcomes by performing rather than writing practice in study type. It is essential for effective implementation of competency focused outcome- based green curriculum-2021. Every practical has been keenly designed to serve as a tool to develop & enhance relevant industry needed competency in each and every student. These psychomotor skills are very difficult to develop through traditional chalk and board content delivery method in the classroom. Accordingly, this lab manual has been designed to focus on the industry defined relevant outcomes, rather than old practice of conducting practical to prove concept and theory.

By using this lab manual, students can read procedure one day in advance to actual performance day of practical experiment which generates interest and also, they can have idea of judgement of magnitude prior to performance. This in turn enhances predetermined outcomes amongst students. Each and every Experiment /Practical in this manual begins by competency, industry relevant skills, course outcomes as well as practical outcomes which serve as a key role for doing the practical. The students will also have a clear idea of safety and necessary precautions to be taken while performing experiment.

This manual also provides guidelines to lecturers to facilitate student-centred lab activities for each practical/experiment by arranging and managing necessary resources in order that the students follow the procedures with required safety and necessary precautions to achieve outcomes. It also gives an idea that how students will be assessed by providing Rubrics.

RDBMS is the cornerstone of modern data management, providing a structured and organized approach to storing, retrieving, and manipulating data. By leveraging the power of tables, relationships, and queries, RDBMS allows us to harness the potential of data, enabling businesses to make informed decisions, optimize processes, and gain valuable insights. In this subject, we delve into the fundamental concepts, principles, and practices of RDBMS, exploring its architecture, data modelling techniques, query optimization, and transaction management.

Although we try our level best to design this lab manual, but always there are chances of improvement. We welcome any suggestions for improvement.

Programme Outcomes (POs):

PO1:Basic and Discipline specific knowledge: Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.

PO2:Problem analysis: Identify and analyse well-defined *engineering* problems using codified standard methods.

PO3: Design/ development of solutions: Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.

PO4:Engineering Tools, Experimentation and Testing: Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.

PO5:Engineering practices for society, sustainability and environment: Apply appropriate technology in context of society, sustainability, environment and ethical practices.

PO6:Project Management: Use engineering management principles individually, as a team member or a leader to manage projects and effectively communicate about well-defined engineering activities.

PO7:Life-long learning: Ability to analyse individual needs and engage in updating in the context of technological changes *in field of engineering*.

Practical Outcome - Course Outcome matrix**Course Outcomes (COs):**

CO1: Perform queries on datasets using SQL*Plus

CO2: Perform joins, sub-queries and nested queries on multiple tables using SQL*plus

CO3: Apply rules on datasets using SQL*Plus constraints

CO4: Write PL/SQL block using concept of Cursor Management, Error Handling, Package and Triggers

CO5: Apply various Normalization techniques.

S. No.	Practical Outcome	CO1	CO2	CO3	CO4	CO5
1.	Implement SQL queries to perform various DDL Commands.	✓	-	-	-	-
2.	a. Implement SQL queries to perform various DML Commands. b. Retrieve data using SELECT command and various SQL operators.	✓		-	-	-
3.	Perform queries for TCL and DCL Commands	✓		-	-	-
4.	Implement SQL queries using Date functions like add-months, months-between, round, nextday, truncate etc	-	✓	-	-	-
5.	Implement SQL queries using Numeric functions like abs, ceil, power, mod, round, trunc, sqrt etc. and Character Functions like initcap, lower, upper, ltrim, rtrim, replace, substring, instr etc.	-	✓	-	-	-
6.	Implement SQL queries using Conversion Functions like to- char, to-date, to-number and Group functions like Avg, Min, Max, Sum, Count, Decode etc.	-	✓	-	-	-
7.	Implement SQL queries using Group by, Having and Order by Clause	-	✓	-	-	-
8.	Implement SQL queries using simple Case Operations and using Group functions and Case operations for getting summary data	-	✓	-	-	-
9.	Implement SQL queries using Set operators like Union, union all, Intersect, Minus etc.	-	✓	-	-	-
10.	Retrieve data spread across various tables or same table using various Joins.	-	✓	-	-	-
11.	Retrieve data from multiple tables using Sub-queries (Multiple, Correlated) (write minimum 3 level subquery)	-	✓	-	-	-
12.	Perform queries to Create, alter and update views	-	-	✓	-	-
13.	Implement Practical-1 again with Domain Integrity, Entity Integrity and Referential Integrity constraints.	-	-	✓	-	-
14.	Perform queries to Create synonyms, sequence and index	-	-	✓	-	-

15.	Implement PL/SQL programs using control structures	-	-	-	✓	-
16.	Implement PL/SQL programs using Cursors	-	-	-	✓	-
17.	Implement PL/SQL programs using exception handling.	-	-	-	✓	-
18.	Implement user defined procedures and functions using PL/SQL blocks	-	-	-	✓	-
19.	Perform various operations on packages.	-	-	-	✓	-
20.	Implement various triggers	-	-	-	✓	-
21.	Micro-Project(E-R Diagrams and Normalization)	-	-	-	-	✓

Industry Relevant Skills

The following industry relevant skills of the competency “**Design database tables, writing optimized queries for integration and other application, creating program views, functions and stored procedure.**” are expected to be developed in the student by undertaking the practical of this laboratory manual.

Guidelines to Course Faculty

1. Course faculty should demonstrate experiment with all necessary implementation strategies described in curriculum.
2. Course faculty should explain industrial relevance before starting of each experiment.
3. Course faculty should involve & give opportunity to all students for hands on experience.
4. Course faculty should ensure mentioned skills are developed in the students by asking.
5. Utilise 2 hrs of lab hours effectively and ensure completion of write up with quiz also.
6. Encourage peer to peer learning by doing same experiment through fast learners.

Instructions for Students

1. Organize the work in the group and make record of all observations.
2. Students shall develop maintenance skill as expected by industries.
3. Student shall attempt to develop related hand-on skills and build confidence.
4. Student shall develop the habits of evolving more ideas, innovations, skills etc.
5. Student shall refer technical magazines and data books.
6. Student should develop habit to submit the practical on date and time.
7. Student should well prepare while submitting write-up of exercise.

References:

- <https://www.geeksforgeeks.org/ddl-commands-syntax/>
- <https://www.w3schools.com/sql/>
- <https://www.tutorialspoint.com/sql/index.htm>

■ **CONTINUOUS ASSESSMENT (25 Marks):**

• **Laboratory Work and Questionnaire Component (25 Marks):**

Component	Criteria	Percentage	Marks	Assessment
Laboratory Work and Questionnaire	Excellent	91%-100%	22-25	Demonstrates exceptional proficiency in laboratory work and questionnaire assessments, consistently applying skills and understanding effectively.
	Proficient	71%-90%	18-22	Shows a strong command of both laboratory work and questionnaire assessments, with minor areas for improvement.
	Satisfactory	51%-70%	13-17	Achieves a satisfactory level of performance in laboratory work and questionnaire assessments, with room for improvement in some areas.
	Needs Improvement	31%-50%	8-12	Demonstrates limited proficiency in both laboratory work and questionnaire assessments, with significant areas for improvement.
	Inadequate	0%-30%	0-7	Fails to meet acceptable standards in both laboratory work and questionnaire assessments; significant improvement is required.

■ **END SEMESTER EXAMINATION (25 Marks):**

• **Viva Examination(25 Marks):**

Component	Criteria	Percentage	Marks	Assessment
Viva Examination	Excellent	91%-100%	22-25	Demonstrates exceptional proficiency in the viva exam, displaying an in-depth understanding and providing comprehensive and insightful answers.
	Proficient	71%-90%	18-22	Displays a strong grasp of the viva exam topics, providing clear and well-reasoned answers, with minor areas for improvement.
	Satisfactory	51%-70%	13-17	Provides satisfactory responses during the viva exam, covering the essential topics, with room for improvement in some areas.
	Needs Improvement	31%-50%	8-12	Demonstrates limited understanding of the viva exam topics, providing answers that may lack clarity or depth, with significant areas for improvement.
	Inadequate	0%-30%	0-7	Fails to meet acceptable standards in the viva exam, providing answers that are unclear, incorrect, or lacking substance; significant improvement is required.

***Note:** This Rubric is applied to the ESE Components of the Courses where End Semester Examination is conducted by Institute Faculty Internally. For the Final Year Courses, ESE Exam is conducted by an External Faculty appointed by university. So, for those courses the marks are converted from the GTU Grade and equally divided into all the COs.

Continuous Assessment Sheet**Enrolment No:****Name****Name:****Term:**

Sr	Practical Outcome/Title of experiment	Page	Date	*Marks (25)	Sign
1.	Implement SQL queries to perform various DDL Commands.				
2.	a. Implement SQL queries to perform various DML Commands. b. Retrieve data using SELECT command and various SQL operators.				
3.	Perform queries for TCL and DCL Commands				
4.	Implement SQL queries using Date functions like add-months, months-between, round, nextday, truncate etc				
5.	Implement SQL queries using Numeric functions like abs, ceil, power, mod, round, trunc, sqrt etc. and Character Functions like initcap, lower, upper, ltrim, rtrim, replace, substring, instr etc.				
6.	Implement SQL queries using Conversion Functions like to- char, to-date, to-number and Group functions like Avg, Min, Max, Sum, Count, Decode etc.				
7.	Implement SQL queries using Group by, Having and Order by Clause				
8.	Implement SQL queries using simple Case Operations and using Group functions and Case operations for getting summary data				
9.	Implement SQL queries using Set operators like Union, union all, Intersect, Minus etc.				
10.	Retrieve data spread across various tables or same table using various Joins.				
11.	Retrieve data from multiple tables using Sub-queries				
12.	Perform queries to Create, alter and update views				
13.	Implement Domain Integrity, Entity Integrity and Referential Integrity constraints.				
14.	Perform queries to Create synonyms, sequence and index				
15.	Implement PL/SQL programs using control structures				
16.	Implement PL/SQL programs using Cursors				
17.	Implement PL/SQL programs using exception handling.				
18.	Implement user defined procedures and functions using PL/SQL blocks				
19.	Perform various operations on packages.				
20.	Implement various triggers				
21.	E-R Diagrams and Normalization				

*Lab Work and Q & A

Practical No.1: Implement SQL queries to perform various DDL(Data Definition Language) Commands.

1. Create an Account table having three columns for account number, balance, branch name as described below.

Column Name	Data Type	Size
Ano	char	3
Balance	number	9
Bname	varchar2	10

2. Create an Employee table having structure as given below.

Column Name	Data Type	Size
Eid	Char	4
Ename	varchar2	10
Birthdate	Date	-
Salary	Number	7
City	Varchar2	10

3. Create a table named "**products**" with the following columns:
id (integer),name (varchar2(100)),price (decimal(10,2))
description (varchar2(500))
4. Create a table named "**orders**" with the following columns:
id (integer),user_id (integer), product_id (integer)
quantity (integer), price(number(5,2))
5. Change Bname column with width of 30 characters in Account table.
6. Add a new column birthdate in Account table specifying birthdate of account holder.
7. Remove column Birthdate from Account table
8. Change the name of column bname to branch_name in account table
9. Rename the table Employee to Emp
10. Create a table Acc2 which contains same structure as that of Account table, but doesn't contain any records from Account table.
11. Create a new table Emp_Manager from table Employee having all employees who are working as managers.
12. Insert all rows of an Account table into a table named Acc2.
13. Destroy an Emp_manager table along with data held.
14. Delete all data of Acc2 table without destroying structure of the table.

A. Objective:

To perform various Data Definition Language (DDL) commands to manage the structure of databases, tables, and indexes within a database management system (DBMS) using SQL commands

B. Expected Program Outcomes (POs): PO1, PO5, PO7**C. Expected Skills to be developed based on competency:**

Database design, SQL syntax and queries related to table creation, alteration and deletion.

D. Expected Course Outcomes(Cos)

Perform queries on datasets using SQL*Plus

E. Practical Outcome(PRo)

Write data definition queries, compilation, debugging, executing using oracle software

F. Expected Affective Domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

❖ **SQL DATA TYPES: -**

- | | |
|----------------------------------|-----------------|
| • CHAR (size) | • NUMBER (P, S) |
| • VARCHAR (size)/VARCHAR2 (size) | • LONG |
| • DATE | • RAW/LONGRAW |

➤ **CHAR (SIZE)**

- This data type is used to store alphanumeric data of fixed length.
- The size represents length of character to be stored.
- This data type can store maximum 2000 characters.
- This is faster to access but requires more storage.

➤ **VARCHAR (size)/VARCHAR2 (size)**

- This data type is used to store variable length alphanumeric data.
- The maximum size of this data type is 4000 characters.
- This is slower to access in comparison to CHAR but is efficient for storage.

➤ **DATE**

- This data type is used to store date and time.
- The standard format is DD-MON-YY for example 21-JUN-09.
- Date time stores time in the 24-hours format.
- Range of this data type is January 1, 4712 B.C. to December 31, 4712A.D.

➤ **NUMBER (P,S)**

- The NUMBER data type is used to store numbers (both Integer and floating-point numbers).
- We can store number up to 38 characters long.
- The precision (P), determines the maximum length of number, and scale (s), determines the decimal point sign(.) on the right of a number (P).

➤ **LONG**

- This data type is used to store variable length character strings upto 2GB.
- LONG data can be used to store binary data in ASCII format.

➤ **RAW/LONG RAW**

- The RAW/LONG RAW data type is used to store binary data, such as digitized picture or image.
- RAW data type can have a maximum length of 255 bytes.
- LONG RAW data type can contain upto 2GB.

❖ DESCRIBE COMMAND

- DESCRIBE or DESC (both are same) command to describe the structure of a table.
- DESCRIBE or DESC command shows the structure of table which include name of the column, data-type of column and the nullability which means, that column can contain null values or not.
- All of these features of table are described at the time of Creation of table.

Syntax:

SQL>DESCRIBE TABLE NAME;

OR

SQL> DESC TABLE NAME;

➤ **SQL DDL (DATA DEFINITION LANGUAGE) COMMANDS:** - CREATE, ALTER, RENAME, TRUNCATE, DROP

➤ **CREATE**

- Create command is used to create a table in database.

Syntax:

SQL>Create table table name

(columnname_1 datatype(datasize), columnname_2 datatype(datasize),

.....

columnname_ndatatype(datasize));

Example:

SQL>Create table client_master (client_no varchar2(6), name varchar2(20), city varchar2(15), pincode number(8), state varchar2(15));

Show the Structure of a Table:

Syntax:

SQL>DESCRIBE CLIENT_MASTER;

NAME	NULL?	TYPE
CLIENT_NO		VARCHAR2(6)
NAME		VARCHAR2(20)
CITY		VARCHAR2(15)
PINCODE		NUMBER(8)
STATE		VARCHAR2(15)

➤ **ALTER**

ALTER command is used to ALTER (Modify structure of) a table in database.

ALTER command is used in five ways.

1. To add new column in table.
2. To modify data type or data size of existing column of table.
3. To drop (remove) the existing column of table.
4. To rename the name of existing table.
5. To rename the name of existing column of table.

Syntax: (1st way- to add new column)

SQL>Alter table tablename add (

new_columnname_1 datatype(datasize), new_columnname_2 datatype(datasize),

.....

new_columnname_ndatatype(datasize));

Example:

SQL>Alter Table Client_Master Add(Mobile_No Number(8), Email Char(15));

Syntax: (2nd way-modify data type or data size)

SQL>Alter table tablename **modify** (existing_columnname_1 new_datatype (new_datasize),.....existing_columnname_n new_datatype(new_datasize));

Example:

SQL>alter table client_master modify(mobile_no number(10), email varchar2(15));

Syntax: (3rd way-remove the column)

SQL>ALTER TABLE TABLENAME DROP COLUMN COLUMNNAME;

Example:

SQL>ALTER TABLE CLIENT_MASTER DROP COLUMN MOBILE_NO;

Syntax: (4th way-rename table name)

SQL>ALTER TABLE OLD_TABLENAME RENAME TO NEW_TABLENAME;

Example:

SQL>ALTER TABLE CLIENT_MASTER RENAME TO CLIENT_MASTER_COPY;

Show the Structure of a Table:

Syntax:

Syntax: (5th way-rename the column name)

SQL>ALTER TABLE TABLENAME RENAME COLUMN OLD_COLUMNNAME TO NEW_COLUMNNAME;

Example:

SQL>ALTER TABLE CLIENT_MASTER RENAME COLUMNEMAIL TO
EMAIL_ID;

➤ **RENAME**

RENAME command is used to give new name of table.

Syntax:

SQL>RENAME OLD_TABLENAME TO NEW_TABLENAME;

Example:

SQL>RENAME CLIENT_MASTER_COPY RENAME TO CLIENT_MASTER;

➤ **TRUNCATE**

TRUNCATE command is used to remove all records from a table, including all memory allocated for the records.

Syntax:

SQL>TRUNCATE TABLE TABLE_NAME;

Example:

SQL>TRUNCATE TABLE CLIENT_MASTER;

➤ **DROP**

DROP command is used to delete all records of a table along with its structure in database.

Syntax:

SQL>DROP TABLE TABLE_NAME;

Example:

SQL>DROP TABLE CLIENT_MASTER;

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code / Solutions to queries:

Signature of Faculty

Practical No.2: a) Implement SQL queries to perform various DML Commands.**b) Retrieve data using SELECT command and various SQL operators.**

1. Insert Below records in Account table

Ano	balance	bname
A01	5000	vvv
A02	6000	ksad
A03	7000	anand
A04	8000	ksad
A05	6000	vvv

2. Insert the following records in Employee table

eid	ename	Birthdate	salary	city	MngID
E01	Tulsi	26-JAN-82	12000	Ahmedabad	E02
E02	Gopi	15-AUG-83	15000	Anand	Null
E03	Rajshree	31-OCT-84	20000	Vadodara	Null
E04	Vaishali	23-MAR-85	25000	Surat	E03
E05	Laxmi	14-FEB-83	18000	Anand	E03
E06	Shivali	05-SEP-84	20000	Surat	E02

3. Delete all accounts which belong to 'ksad' branch.
4. Update accounts to add 10% interest to balance in each account of an Account table.
5. Display only distinct branch names for an Account table
6. Show employee name, salary of employees whose salary is <8000.
7. Find out the employee who belongs to 'Anand' city and has salary less than 17000.
8. Find out the employee who belong to either 'Anand' city or 'Surat' city.
9. Display employees not coming from city 'Ahmedabad', 'Vadodara' or 'Surat'.
10. Show employee name and salary whose salary between 5000 and 15000.
11. Display employees name and city in the format < employee_name> lives in < city_name>
12. Show employees whose name starts with 'S' character.
13. Show employees whose name have minimum two 'A' characters.
14. Display employees whose names consist of five characters.

A. Objective:

To perform various Data Manipulation Language (DML) commands is to enable users to interact with a database and modify its contents in a structured and controlled manner.

B. Expected Program Outcomes (POs): PO1,PO5,PO7**C. Expected Skills to be developed based on competency:**

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Perform queries on datasets using SQL*Plus

E. Practical Outcome(Pro)

Write data definition queries , compilation, debugging, executing using oracle software

E. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.

- Maintain tools and equipment
- Follow ethical practices.

F. Prerequisite Theory:

❖ SQL DML (DATA MANIPULATION LANGUAGE) COMMANDS: -

➤ INSERT

➤ UPDATE

➤ DELETE

➤ INSERT

- Insert command is Used to insert data into a table or create a new row in table:
- We have two methods to insert data in a table
 1. By value method
 2. By address method

Syntax: (1st way-BY value method)

1st Way

SQL>INSERT INTO TABLENAME VALUES (VALUE_1,VALUE_2,... VALUE_N);

OR

2nd Way

SQL>INSERT INTO TABLENAME (COLUMN_1, COLUMN_2, COLUMN_3COLUMN_N) VALUES (VALUE_1, VALUE_2, VALUE_3 VALUE_N);

Example:

1st Way

SQL> INSERT INTO CLIENT_MASTER VALUES ('C00001', 'RAHUL', 'MUMBAI', 400054,'MAHARASHTRA');

OR

2nd Way

SQL>INSERT INTO CLIENT_MASTER (CLIENT_NO, NAME,CITY, PINCODE, STATE) VALUES ('C00001', 'RAHUL', 'MUMBAI', 400054, 'MAHARASHTRA');

To insert a new record again you have to type entire insert command, if there are lot of records this will be difficult. This will be avoided by using address method.

Syntax: (2nd way-BY address method)

SQL>INSERT INTO TABLENAME VALUES (&COLUMN_1, &COLUMN_2,... & COLUMN_N);

OR

SQL> INSERT INTO TABLENAME(COLUMN_1, COLUMN_2, COLUMN_3COLUMN_N) VALUES (&COLUMN_1, &COLUMN_2, &COLUMN_3 &COLUMN_N);

This will prompt you for the values but for every insert you have to use forward slash .

Example:

SQL>INSERT INTO CLIENT_MASTER VALUES ('&CLIENT_NO', '&NAME', '&CITY', &PINCODE, '&STATE');

Enter value for client_no: C0001

Enter value for name: Rahul

Enter value for city: Mumbai

Enter value for pincode: 400054

Enter value for state: Maharashtra

➤ UPDATE

- The UPDATE command is used to change or modify data of a table
- The Update command is used to update selected rows from table or all the rows from table

Syntax:(update selected rows)

```
SQL>UPDATE TABLENAMESET COLUMNNAME_1 = EXPRESSION_1,COLUMNNAME_2 = EXPRESSION_2
WHERE CONDITION;
```

Example:

```
SQL>UPDATE CLIENT_MASTER SET CITY='BOMBAY' WHERE CLIENT_NO='C00002';
```

Syntax:(update all rows)

```
SQL>UPDATE TABLENAME SET COLUMNNAME_1 = EXPRESSION_1;
```

Example:

```
SQL>UPDATE CLIENT_MASTER SET NAME = 'OHM';
```

➤ **DELETE**

- The DELETE command is used to delete data or rows from a table
- The delete command is used to delete selected rows from table or All the rows from table

Syntax: (delete selected rows)

SQL>DELETE FROM TABLENAME WHERE CONDITION;

Example:

```
SQL>DELETE FROM CLIENT_MASTER WHERE CLIENT_NO='C00001';
```

Syntax: (delete all rows)

```
SQL>DELETE FROM TABLE NAME;
```

Example:

```
SQL>DELETE FROM CLIENT MASTER;
```


- The **SELECT** command is used to retrieve selected rows from one or more tables.

Syntax: (retrieve all table data)

```
SQL>SELECT * FROM TABLE NAME;
```

Example:

```
SQL>SELECT * FROM CLIENT MASTER;
```

Syntax: (retrieve particular columns of a table)

```
SQL>SELECT COLUMNNAME 1,COLUMNNAME 2 FROM TABLE NAME;
```

```
SQL> SELECT NAME, CITY FROM CLIENT MASTER;
```

Syntax: (retrieve particular column of particular row of a table)

```
SQL>SELECT COLUMNNAME N FROM TABLE NAME WHERE CONDITION;
```

```
SQL>SELECT NAME FROM CLIENT MASTER WHERE CITY='BOMBAY';
```

Syntax: (retrieve particular rows of a table)

```
SQL> SELECT * FROM TABLE NAME WHERE CONDITION;
```

```
SQL>SELECT * FROM CLIENT MASTER WHERE CITY='BOMBAY';
```

❖ SQL Operators

➤ Arithmetic Operators

- Oracle allows arithmetic operators to use while viewing records from table or while performing data manipulation operations such as Insert, Update and Delete.
- These are:

+ Addition	* Multiplication
** Exponentiation	() Enclosed operation
- Subtraction	/ Division

Example:

```
SQL>Select product no,description, sell price * 0.05 from product master;
```

➤ Logical Operators

- The AND Operator
- The OR Operator
- The NOT Operator

➤ The AND Operator

- The AND Operator allows creating an SQL statement based on two or more conditions being met.

- It can be used in any valid SQL statement such as select, insert, or delete.

Example:

```
SQL>SELECT PRODUCT_NO, DESCRIPTION, SELL_PRICE * 0.05 FROM PRODUCT_MASTER WHERE  
SELL_PRICE BETWEEN 500 AND 1100;
```

➤ **The OR Operator**

- The OR condition allows creating an SQL statement where records are returned when any one of the conditions are met
- It can be used in any valid SQL statement such as select, insert, or delete.

Example:

```
SQL>SELECT CLIENT_NO, NAME, CITY, PINCODE FROM CLIENT_MASTER  
WHERE PINCODE=4000054 OR PINCODE=4000057;
```

➤ **The NOT Operator**

- The Oracle engine will process all rows in a table and display only those Records that do not satisfy the condition specified

Example:

```
SQL>SELECT * FROM CLIENT_MASTER WHERE NOT (CITY='BOMBAY' OR CITY='DELHI');
```

➤ **Comparison Operators**

- Comparison operators are used in condition to compare one expression with other.
- The comparison operators are =, <, >, >=, <=, !=, between, like, is null and in operators

➤ **Between operator**

- Between operator is used to check between two values or specific range

Example:

```
SQL>SELECT * FROM SALESMAN_MASTER WHERE SALARY BETWEEN 5000 AND 8000;
```

The above select statement will display only those rows where salary of salesman is between 5000 and 8000.

➤ **In operator:**

- The IN operator can be used to select rows that match one of the values in a list.

```
SQL>SELECT * FROM CLIENT_MASTER WHERE CLIENT_NO IN(C00001, C00003);
```

The above query will retrieve only those rows where client_no is either in C00001 or C00003

➤ **Like operator:**

- The Like operator is used to search character pattern.
- The like operator is used with special character % and _(underscore)

```
SQL>SELECT * FROM CLIENT_MASTER WHERE CITY LIKE 'B%';
```

The above select statement will display only those rows where city is startwith 'B' followed by any number of any characters

- % sign is used to refer number of character (it similar to * asterisk wildcard in DOS).
- While _(underscore) is used to refer single character.

```
SQL>SELECT * FROM CLIENT_MASTER WHERE NAME LIKE '_AHUL';
```

G. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

H. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

I. Source code/ Solutions to queries:

Signature of Faculty

Practical No.3: Perform SQL queries for TCL (Transaction Control Language) and DCL (Data Control Language) Commands.

Note: You can perform TCL and DCL commands tables created in Practical 1 or tables in HR Schema tables.

1. Perform TCL Commands:
 - a. Update the salary of the employee having id E01 to 5000 and permanently save record.
 - b. Delete all employees from the employees table whose city is Anand . View updated data and then bring back the old data using TCL command.
2. Perform DCL Commands:
 - a. Create a user account with your enrolment number.
 - b. Grant select, update of employee to your new user account
 - c. Revoke the update privilege from employee table
 - d. Grant all privileges to all users on account table;

A. Objective:

- TCL commands are used to manage transactions in SQL databases, to commit a transaction and permanently save the changes made during the transaction and to roll back a transaction and undo any changes made during the transaction
- DCL commands are used to manage user permissions and access to the database, to grant or revoke privileges to users or roles for specific database objects, to create or drop user accounts and manage user authentication

B. Expected Program Outcomes (POs): PO1, PO5, PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Perform queries on datasets using SQL*Plus

E. Practical Outcome(Pro)

Write data definition queries, compilation, debugging, executing using SQL

F. Expected Affective Domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

TCL Commands in SQL- Transaction Control Language Examples:

Transaction Control Language can be defined as the portion of a database language used for maintaining consistency of the database and managing transactions in database. A set of SQL statements that are co-related logically and executed on the data stored in the table is known as transaction. In this tutorial, you will learn different TCL Commands in SQL with examples and difference between them.

1. Commit Command

2. Rollback Command

3. Savepoint Command

Commit Commands:

The main use of Commit command is to make the changes done by the transaction permanent in the database.

Syntax: COMMIT;

Example:

```
UPDATE STUDENT SET STUDENT_NAME = 'Maria' WHERE STUDENT_NAME = 'Meena';  
COMMIT;
```

By using the above set of instructions, you can update the wrong student name by the correct one and save it permanently in the database. The update transaction gets completed when commit is used. If commit is not used, then there will be lock on 'Meena' record till the rollback or commit is issued.

Now have a look at the below table where 'Meena' is updated and there is a lock on her record. The updated value is permanently saved in the database after the use of commit and lock is released.

Student ID	Student Name
78	Jane
79	Meena

After Update :

Student ID	Student Name
78	Jane
79	Maria

After Commit :

Student ID	Student Name
78	Jane
79	Maria

Rollback:

Using this command, the database can be restored to the last committed state. Additionally, it is also used with savepoint command for jumping to a savepoint in a transaction.

Syntax:

ROLLBACK;

OR

ROLLBACK TO SAVEPOINT_NAME;

Example:

UPDATE STUDENT SET STUDENT_NAME = 'Manish' WHERE STUDENT_NAME = 'Meena';

ROLLBACK;

This command is used when the user realizes that he/she has updated the wrong information and wants to undo this update. The users can issue ROLLBACK command and then undo the update. Have a look at the below tables to know better about the implementation of this command.

Student ID	Student Name
78	Jane
79	Meena

After Update :

Student ID	Student Name
78	Jane
79	Manish

After Rollback:

Student ID	Student Name
78	Jane
79	Meena

Savepoint:

The main use of the Savepoint command is to save a transaction temporarily. This way users can rollback to the point whenever it is needed.

Syntax:SAVEPOINT SAVEPOINT_NAME;

Example:

SAVEPOINT SAVEPOINT_NAME;

Following is the table of a school class

ID	Name
98	Anita
99	Maria
100	Katilyn

Use some SQL queries on the above table and then watch the results

INSERT into CLASS VALUES (101, 'Rahul');

Commit;

UPDATE CLASS SET NAME= 'Tyler' where id= 101;

SAVEPOINT A;

INSERT INTO CLASS VALUES (102, 'Zack');

Savepoint B;

INSERT INTO CLASS VALUES (103, 'Bruno')

Savepoint C;

SELECT * FROM CLASS;

The result will look like

ID	Name
98	Anita
99	Maria
100	Katilyn
101	Tyler
102	Zack
103	Bruno

Now rollback to savepoint B

ROLLBACK TO B;

SELECT * FROM CLASS;

ID	Name
98	Anita
99	Maria
100	Katilyn
101	Tyler
102	Zack

Now rollback to savepoint A

ROLLBACK TO A;

SELECT * FROM CLASS;

ID	Name
98	Anita
99	Maria
100	Katilyn
101	Tyler

DCL Commands

Create user –

If a login ID does not exist, it has to be created manually. The following steps indicate how a DBA login ID is created and appropriate privileges given to the DBA login ID with a special focus on creating table space.

Creating a User:-

SYNTAX: -CREATE USER USERNAME IDENTIFIED BY PASSWORD;

Example:-

Run SQL Command Line

```
SQL>create user abc identified by abc123;
```

User Created.

When we create a user in SQL, it is not even allowed to login and create a session until and unless proper permissions/privileges are granted to the user.

Following command can be used to grant the session creating privileges.

SYNTAX: -

GRANT CREATE SESSION, connect, resource TO username;

Unlocking a user account: -

SYNTAX: -ALTER USER account IDENTIFIED BY password ACCOUNT UNLOCK;

DCL Commands

Oracle provides two commands - GRANT and REVOKE - to control the access of various database objects.

GRANT COMMAND

Grants a privilege to a user. It means that giving authority to other user by administrator. If you are an administrator, then only you have the authority for granting the privilege to other users. User can grant privilege only if user has been granted that privilege

Syntax:GRANT < OBJECT PRIVILEGES > ON <OBJECTNAME> TO <USERNAME> [WITH GRANT OPTION];

OBJECT PRIVILEGES

Each object privilege that is granted authorizes the grantee to perform some operation on the object. A user can grant all the privileges or grant only specific object privileges.

The list of object privileges is as follows:

ALTER: Allows the grantee to change the table definition with the ALTER TABLE command

DELETE: Allows the grantee to remove the records from the table with the DELETE command

INDEX: Allows the grantee to create an index on the table with the CREATE INDEX command

INSERT: Allows the grantee to add records to the table with the INSERT command

SELECT: Allows the grantee to query the table with the SELECT command

UPDATE: Allows the grantee to modify the records in the tables with the UPDATE command

Example:

Give the user rahul permission to only view and modify records in the table client_master.

Run SQL Command Line

```
SQL>GRANT SELECT, UPDATE ON client_master TO rahul;
```

Grant succeeded.

REVOKE COMMAND

The REVOKE statement is used to deny the grant given on an object. Revokes a privilege from a user. It is used in getting back authority from user.

Syntax:REVOKE < OBJECT PRIVILEGES > ON <OBJECT NAME> FROM <USERNAME>;

Example:

All privileges on the table salesman_master have been granted to rahul. Take back the Delete privilege on the table.

Run SQL Command Line

```
SQL>REVOKE DELETE ON salesman_master FROM Rahul;
```

Revoke succeeded.

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code / Solutions to queries:

Signature of Faculty

Practical No. 4:

Implement SQL queries using Date functions like add-months, months-between, round, nextday, truncate etc.

Write down the queries using date function: -

1. Find a date after adding 4 months to '01-06-19'.
2. Find total months between '01-01-19' to '01-05-19'.
3. Find a last date of '01-02-19'.
4. Find out date of next Tuesday after '18-06-19'.
5. Find the nearest date for '17-06-19 12:35pm'.
6. Find the lower nearest date for '17-06-19 12:35pm'.

A. Objective:

SQL provides various date functions to perform common operations on date and time values, such as extracting date and time components, calculating differences between dates, and formatting dates in different ways

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO4,PO5,PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Perform joins, sub-queries and nested queries on multiple tables using SQL*plus

E. Practical Outcome(Pro)

Write data definition queries , compilation, debugging, executing using SQL

F. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

❖ **SQL DATE FUNCTIONS:-**

- | | | |
|------------------|------------|---------|
| ➤ ADD_MONTHS | ➤ LAST_DAY | ➤ ROUND |
| ➤ MONTHS_BETWEEN | ➤ NEXT_DAY | ➤ TRUNC |

NOTE:- CONSIDER '31-DEC-2012' AS A SYSDATE.

➤ **ADD_MONTHS**

- ADD_MONTHS function returns new date after adding the number of months specified in the function.

Syntax:

SQL>ADD_MONTHS (D, N);

Example:

SQL>SELECT ADD_MONTHS (SYSDATE,3) "ADD MONTHS" FROM DUAL;

SQL>ADD MONTHS

31-MAR-13

- IF n IS NEGATIVE, Then n MONTH will be subtracted.

SQL>SELECT ADD_MONTHS (SYSDATE,-3) "ADD MONTHS" FROM DUAL;

SQL>ADD MONTHS

30-SEP-12

➤ **MONTHS_BETWEEN**

- MONTHS_BETWEEN function returns number of months between date1 and date2.

Syntax:

SQL>MONTHS_BETWEEN (DATE1, DATE2);

Example:

SQL>SELECT MONTHS_BETWEEN ('31-MAR-13','31-DEC-12') "MONTHS BETWEEN" FROM DUAL;

SQL>MONTHS BETWEEN

3

➤ **LAST_DAY**

- LAST_DAY function returns the last date of the month specified with the function.

Syntax:

SQL>LAST_DAY (DATE);

Example:

SQL>SELECT LAST_DAY ('1-MAR-13') "LASTDAY" FROM DUAL;

SQL>LASTDAY

31-MAR-13

➤ **NEXT_DAY**

- NEXT_DAY function returns the date of the next named weekday specified by day relative to date.

Syntax:

SQL>NEXT_DAY (DATE, DAY);

Example:

SQL>SELECT NEXT_DAY ('3-JUNE-19','SUNDAY') "NEXTDAY" FROM DUAL;

SQL> NEXTDAY

09-JUNE-19

➤ **ROUND**

- ROUND function returns the rounded date according to format.
- A format can be any valid format.
- If format is omitted, date is rounded to the next day if time is 12.00 pm or later otherwise return today's date.

Syntax:

SQL>ROUND (DATE, FORMAT);

Example:

SQL>SELECT ROUND(TO_DATE('31-DEC-12 03:30:45 PM','DD-MM-YY HH:MI:SS PM')) FROM DUAL;

SQL>ROUND(TO_

01-JAN-13

SQL>SELECT ROUND(TO_DATE('31-DEC-12 12:00:00 PM','DD-MM-YY HH:MI:SS PM')) FROM DUAL;

SQL>ROUND(TO_

01-JAN-13

```
SQL>SELECT ROUND(TO_DATE('31-DEC-12 03:30:45 AM','DD-MM-YY HH:MI:SS PM')) FROM  
DUAL;  
SQL>ROUND(TO_
```

```
-----
```

```
31-DEC-12
```

➤ **TRUNC**

- TRUNC function returns the truncated date according to format.
- A format can be any valid format.
- If format is omitted, date is truncated to 12.00 am.

Syntax:

```
SQL>TRUNC (DATE, FORMAT);
```

Example:

```
SQL>SELECT TRUNC(TO_DATE('31-DEC-12 03:30:45 PM','DD-MM-YY HH:MI:SS PM')) FROM  
DUAL;
```

```
SQL>TRUNC(TO_
```

```
-----
```

```
31-DEC-12
```

```
SQL> SELECT TRUNC(TO_DATE('31-DEC-12 12:00:00 PM','DD-MM-YY HH:MI:SS PM')) FROM  
DUAL;
```

```
SQL>TRUNC(TO_
```

```
-----
```

```
31-DEC-12
```

```
SQL>SELECT TRUNC(TO_DATE('31-DEC-12 03:30:45 AM','DD-MM-YY HH:MI:SS PM')) FROM  
DUAL;
```

```
SQL>TRUNC(TO_
```

```
-----
```

```
31-DEC-12
```

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code /Solutions to queries:

Signature of Faculty

Practical No.5: Implement SQL queries using Numeric functions like abs, ceil, power, mod, round, trunc, sqrt etc. and Character Functions like initcap, lower, upper, ltrim, rtrim, replace, substring, instr etc.

Write down the queries using numeric function:-

1. Find the absolute value of -15.
2. Find the square root of 81.
3. Find the value for 3 raised to power 4.
4. Find the remainder for 16 divided by 5.
5. Find the largest integer value which is greater than or equal to -27.2.
6. Find the value for 182.284 which is rounded to -2.
7. Find the value for 182.284 which is truncated to 1.

Write down the queries using character function:-

Note : Consider Employees table to solve queries.

1. Count number of characters in a string "Computer Engineering".
2. Display name of all employees in uppercase.
3. Display first letter of each word in a string "character function" in uppercase.
4. Extract 11 characters from the string "Computer Engineering", starting from 12th character.
5. Display last name of all employees' right justified with total length of 20 characters, fill up the blank characters with "#".
6. Remove characters "gbrsea" from string "greatest" from left side.
7. Change the word "govern" by "suppli" in a string "government".
8. Display ascii values of 's', 'A' and 'a'.
9. Find the first occurrence of "base" in string 'Database'.

A. Objective:

Numeric functions are used to perform mathematical operations, while character functions are used to manipulate and perform operations on character data

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO4,PO5,PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Perform joins, sub-queries and nested queries on multiple tables using SQL*plus.

E. Practical Outcome(Pro)

Write data definition queries , compilation, debugging, executing using SQL

F. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

❖ **SQL NUMERIC FUNCTIONS:-**

- 1) ABS 2) CEIL 3) EXP 4) FLOOR 5) POWER 6) MOD
7) ROUND 8) TRUNC 9) SQRT

➤ **ABS()**

- **ABS()** function returns absolute value.

Syntax:ABS (N);

Example:

```
SQL>SELECT ABS (3), ABS(-3) "ABSOLUTEVALUE" FROM DUAL;
      3      3
-----
ABSOLUTEVALUE ABSOLUTEVALUE
```

➤ **CEIL ()**

- **CEIL()** function returns the smallest integer that is greater than or equal to a specific value.

Syntax:CEIL (N);

Example:

```
SQL>SELECT CEIL(-25.4)"CEIL1",CEIL(25.4)"CEIL2" FROM DUAL;
      CEIL1    CEIL2
-----
      -25      26
```

➤ **EXP()**

- **EXP()** function returns **e** raised to the **nth** power, where e=2.71828183.

Syntax:EXP (N);

Example:

```
SQL> SELECT EXP ('1') "EXP" FROM DUAL;
SQL> EXP
      2.71828183
```

➤ **FLOOR()**

- **FLOOR()** function returns the greatest integer that is less than or equal to a specific value.

Syntax: FLOOR (N);

Example:

```
SQL>SELECT FLOOR ('25.4') "FLOOR" ,FLOOR ('25.5') "FLOOR",FLOOR ('25.5') "FLOOR" FROM
DUAL;
SQL>FLOOR  FLOOR  FLOOR
      25    25    25
SQL> SELECT FLOOR ('-25.4') "FLOOR" ,FLOOR ('-25.5') "FLOOR",FLOOR ('-25.5') "FLOOR"
FROM DUAL;
SQL> FLOOR  FLOOR  FLOOR
      - 26   -26   -26
```

➤ **POWER()**

- **POWER()** function returns the value raised to a given positive exponent.

Syntax:POWER(VALUE,EXPONENT);

Example:

```
SQL> SELECT POWER (3,2) "POWER" FROM DUAL;
SQL> POWER
      9
```

➤ **MOD()**

- **MOD()** function divides a value by a divisor and returns the remainder.

Syntax:MOD(VALUE,DIVISER);

Example:

```
SQL>SELECT MOD (9,2) "REMAINDER" FROM DUAL;
```


SQL> REMAINDER

1

➤ **ROUND()**

▪ **ROUND()** function rounds a number to given number of digits of precision.

Syntax:ROUND(VALUE,PRECISION);

Example:

SQL>SELECT ROUND (255.555,1) "ROUND",ROUND (255.555,2) "ROUND",ROUND (255.555,3) "ROUND" FROM DUAL;

SQL>ROUND ROUND ROUND
255.6 255.56 255.555

➤ **TRUNC()**

▪ **TRUNC()** function truncates digits of precision from a number.

Syntax:TRUNC(VALUE,PRECISION);

Example:

SQL>SELECT TRUNC (255.555,1) "TRUNC",TRUNC (255.555,2) "TRUNC",TRUNC (255.555,3) "TRUNC" FROM DUAL;

SQL>TRUNC TRUNC TRUNC
255.5 255.55 255.555

➤ **SQRT()**

▪ **SQRT()** function returns the square root of a given number.

Syntax:SQRT(N);

Example:

SQL>SELECT SQRT (9) "SQRT" FROM DUAL;

SQL>SQRT
3

❖ **SQL CHARACTER FUNCTIONS:-**

- | | | | |
|------------|--------------|------------|--------------|
| 1) INITCAP | 2) LOWER | 3) UPPER | 4)LTRIM |
| 5)RTRIM | 6) TRANSLATE | 7) REPLACE | 8) SUBSTRING |
| 9)LENGTH | 10) LPAD | 12) RPAD | 13) ASCII |

➤ **INITCAP()**

▪ **INITCAP()** function converts any string or (column of table)with the first character of each word in UPPER CASE.

Syntax:INITCAP(STRING);

Example:

SQL> select initcap('database management system') "1stcharacterinuppercase" from dual;

SQL>1stcharacterinuppercase
Database Management System

➤ **LOWER()**

▪ **LOWER()** function converts any string or (column of table)into lowercase.

Syntax:lower(string);

Example:

SQL> select lower ('DATABASE') "lower" from dual;

SQL>lower
database

➤ **UPPER()**

▪ **UPPER()** function converts any string or (column of table)into uppercase.
converts any string or (column of table) into uppercase.

Syntax: upper(string);

Example:

SQL> select upper ('database') "upper" from dual;

SQL>upper

DATABASE

➤ **LTRIM()**

- LTRIM() function returns a string with all characters contained in the set are removed from the left end of the string.

Syntax:

SQL>ltrim(mainstring,set);

Example:

SQL> select ltrim('000123', '0')"ltrim1",ltrim('123123Tech', '123') "ltrim2" from dual;

SQL>ltrim1 ltrim2

123 Tech

➤ **RTRIM()**

- RTRIM() function returns a string with all characters contained in the set are removed from the right end of the string.

Syntax:

SQL>rtrim(mainstring,set);

Example:

SQL> select rtrim('123000', '0')"rtrim1",rtrim('Tech123123', '123')

"rtrim2" from dual;

SQL>rtrim1rtrim2

123 Tech

➤ **TRANSLATE()**

- **TRANSLATE ()** function Replace a sequence of character in a string with another set of characters.

Syntax:TRANSLATE(string1,string to replace, replacement string);

Example:

SQL>SELECT TRANSLATE ('COMPUTER','COMPU','HEA') "TRANSLATESTRING"

FROM DUAL;

SQL>TRANSLATESTRING

HEATER

➤ **REPLACE()**

- REPLACE () function Replace a sequence of characters in a string with another set of characters.

Syntax:REPLACE(string1,character to be replaced,characters);

Example:

SQL>SELECT REPLACE ('COMPUTER','UE','IL') "REPLACE" ,REPLACE ('COMPUTER','UT','IL')

"REPLACE" FROM DUAL;

SQL>REPLACE REPLACE

COMPUTER COMPILER

➤ **SUBSTRING()**

- SUBSTRING () functions allows you to extract a substring from a string.

Syntax:SUBSTR (string, start position, length);

Example:

```
SQL>SELECT SUBSTR ('COMPUTER','1','4') "SUBSTRING" ,SUBSTR('COMPUTER','4','5')
"SUBSTRING" FROM DUAL;
SQL> SUBSTRING SUBSTRING
```

```
-----
COMP PUTER
```

➤ Instr()

- SQL INSTR() function detects the first occurrence of a string or a character in the other string. Thus, the INSTR() function witnesses the position of the initial/first occurrence of any string/sub-string in another string data value.

Syntax:INSTR(string1, string2);

Example:

```
SQL>SELECT INSTR('JYPython', 'P');
SQL> instr
```

```
-----
3
```

➤ LENGTH()

- **LENGTH()** function returns the length of string(count number of characters).

Syntax:

```
SQL>length(string);
```

Example:

```
SQL> select length ('DATABASE') "length" from dual;
```

```
SQL>length
```

```
8
```

➤ LPAD()

- **LPAD()** function returns string which is left padded with string2 up to length n.

Syntax:lpad(string , n ,string2);

Example:

```
SQL>select lpad('database',15,'3330703') "lpad" from dual;
```

```
SQL> lpad
```

```
-----
3330703database
```

➤ RPAD()

- **RPAD()** function returns string which is right padded with string2 up to length n.

Syntax:rpad (string , n ,string2);

Example:

```
SQL>select rpad('database',15,'3330703') "rpad" from dual;
```

```
SQL> rpad
```

```
-----
database3330703
```

➤ ASCII()

- **ASCII()** function returns the ascii code of a charcter.

Syntax:ascii (charcter);

Example:

```
SQL>select ascii('d')"ascii",ascii('D')"ascii" from dual;
```

```
SQL>ascii ascii
```

```
-----
100 68
```

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code/ Solutions to queries:

Signature of Faculty

Practical No.6: Implement SQL queries using Conversion Functions like to- char, to-date, to-number and Group functions like Avg, Min, Max, Sum, Count, Decode etc.

Note: Consider HR.EMPLOYEES table to solve all queries.

Write down the queries using conversion and miscellaneous function: -

1. Convert '+01234.78' to number.
2. Convert a value 123789 to character using '9,99,999' format.
3. List out Birth date in form of 'Saturday,14th feb,2009' for employee.
4. If input value is 'MAX' then display message 'this is maximum', if input value is 'MIN' then display message 'this is minimum' otherwise display message 'this is equal'.

Write down the queries using GROUP function:-

Consider Employee table to solve all queries.

1. Find total employees, maximum, minimum and average salary.
2. Find the total employees and total salary of the employees living in "Surat" City.

A. Objective:

Implementing SQL conversion functions in SQL queries is to convert data from one data type to another data type and manipulate data that is stored in different formats and group functions is to perform calculations on a set of values that are grouped together based on a specific column or columns. These group functions are also known as aggregate functions because they aggregate multiple rows into a single result.

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO4,PO5,PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Perform joins, sub-queries and nested queries on multiple tables using SQL*plus.

E. Practical Outcome(PRo)

Write data definition queries , compilation, debugging, executing using SQL

F. Expected Affective Domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

❖ **SQL CONVERSION FUNCTIONS: -**

- TO_NUMBER
- TO_CHAR
- TO_DATE

➤ **TO_NUMBER (CONVERTING CHARACTER TO NUMBER)**

- **TO_NUMBER ()** function converts a value of a character data type to number data type.
- Returns equivalent numeric value to string.
- String must consist of 0-9, decimal point and '+' or '-' sign.

Syntax:

SQL>TO_NUMBER(STRING);

Example:

```
SQL>SELECT TO_NUMBER('-123.3') "TO_NUMBER", TO_NUMBER('+123.3') "TO_NUMBER",
TO_NUMBER('123') "TO_NUMBER" FROM DUAL;
```

```
SQL> TO_NUMBER TO_NUMBER TO_NUMBER
```

```
-----
-123.3      123.3      123
```

➤ **TO_CHAR (CONVERTING DATE TO CHARACTER)**

- **TO_CHAR ()** function converts a value of DATE data type to CHAR value.
- It can also extract a part of the date, month or the year from the date value and use it for sorting or grouping of data according to the date, month or year.

Syntax:

```
SQL>TO_CHAR (DATE,FORMAT);
```

Example:

```
SQL>SELECT TO_CHAR (SYSDATE,'YYYY') "EXTRACTYEAR", TO_CHAR (SYSDATE,'MONTH')
"EXTRACTMONTH" FROM DUAL;
```

```
SQL>EXTRACTYEAR EXTRACTMONTH
```

```
2019      JUNE
```

➤ **TO_CHAR (CONVERTING NUMBER TO CHARACTER)**

- **TO_CHAR ()** function converts a value of NUMBER data type to CHAR value using optional format.
- A format consist '0','9' and ','.

Syntax:

```
SQL>TO_CHAR (N, FORMAT);
```

Example:

```
SQL>SELECT TO_CHAR(123456,'09,0999'),TO_CHAR(123456,'99,09999') FROM DUAL;
```

```
SQL> TO_CHAR( TO_CHAR(1
```

```
-----
12,3456  1,23456
```

➤ **TO_DATE (CONVERTING CHARACTER TO DATE)**

- **TO_DATE ()** function converts a value of CHAR datatype to DATE data type value.

Syntax:

```
SQL>TO_DATE (CHAR, FORMAT);
```

Example:

```
SQL>SELECT TO_DATE ('06/06/19','DD/MM/YYYY') FROM DUAL;
```

```
SQL> TO_DATE('
```

```
-----
06-JUN-19
```

➤ **USEFUL DATE FORMATS FOR TO_CHAR AND TO_DATE FUNCTIONS:**

FORMAT	SPECIFIES	FORMAT	SPECIFIES
MM	Number of months (1-12)	YYY	Last three digit of year: 986
MON	3-Letter month: JAN	YY	Last two digits of year: 86
MONTH	Full month: JUNE	Y	Last one digit of year: 6
DDD	No. of day of year	YEAR	Year spelled out
DD	No. of day of month	WW	No. of week in year

D	No. of day of week	W	No. of week in month
DY	3-letter day: MON	HH	Hour
DAY	Full day: SUNDAY	MI	Minute
YYYY	Four digit year: 1986	SS	Second

➤ **USEFUL FORMATS THAT CAN BE USED ONLY WITH TO_CHAR:**

FORMAT	SPECIFIES
TH	Produces 6 TH , 6 Th or 6 th based on DDTH, DdTH, ddTH
SP	Forces number to spell out, like -five
SPTH/THSP	Forces number to spell out, like -fifth

❖ **SQL MISCELLANEOUS FUNCTIONS:-**

➤ **DECODE**

- **DECODE** function has the functionality of an IF-THEN-ELSE statement.

Syntax:

SQL>DECODE(EXP, EXP1 THEN RESULT1,EXP2 THEN RESULT2,DEFAULT);

Example:

SQL>SELECT DECODE (1,1,1000,2,2000,3000) "DECODE", DECODE (2,1,1000,2,2000,3000) "DECODE", DECODE (3,1,1000,2,2000,3000)"DECODE" FROM DUAL;

```
SQL> DECODE          DECODE          DECODE
      -----          -----          -----
          1000          2000          3000
```

❖ **SQL GROUP(AGGREGATE)FUNCTIONS:-**

- AVG
- MIN
- COUNT
- SUM
- MAX

NOTE: consider following COURSE table for group function example.

TABLE NAME: COURSE

C_ID	C_NAME	C_FEE	C_SEATS
C01	COMPUTER	10000	20
C02	IT	8000	15
C03	MECHANICAL	5000	30
C04	PRINTING	5000	15
C05	CIVIL	10000	30
C06	ELECTRICAL	NULL	NULL

➤ **AVG**

- **AVG ()** function returns average value for a given column.
- AVG is allowed only on numeric data types.

Syntax:

SQL>AVG ([DISTINCT | ALL] *columnname*)

- The DISTINCT qualifier eliminates duplicates. The ALL qualifier retains duplicates.

Example:

```
SQL>SELECT      AVG      (C_FEE)"avg",AVG      (DISTINCT      C_FEE)"avg",
AVG(C_SEATS)"avg",AVG(DISTINCT C_SEATS)"avg"FROM course;
SQL>avg      avg      avg      avg
-----
7600      7666.66667      22      21.6666667
```

➤ SUM

- **SUM ()** function returns total sum for a given column.
- SUM is allowed only on numeric data types.

Syntax:

SQL>**SUM ([DISTINCT | ALL] columnname)**

- The DISTINCT qualifier eliminates duplicates. The ALL qualifier retains duplicates.

Example:

```
SQL>SELECT SUM (C_FEE) "sum", SUM (DISTINCT C_FEE)"sum", SUM(C_SEATS)"sum",
SUM(DISTINCT C_SEATS)"sum" FROM course;
SQL>sum      sum      sum      sum
-----
38000      23000      110      65
```

➤ MIN

- **MIN ()** function returns minimum value of a given column.
- MIN is allowed only on numeric data types.

Syntax: MIN(COLUMNNAME);

Example:

```
SQL>SELECT MIN (C_FEE) "MIN", MIN (C_SEATS) "MIN" FROM COURSE;
SQL>MIN      MIN
-----
5000      15
```

➤ MAX

- **MAX ()** function returns maximum value of a given column.
- MAX is allowed only on numeric data types.

Syntax: MAX (COLUMNNAME);

Example:

```
SQL>SELECT MAX (C_FEE) "MAX", MAX (C_SEATS) "MAX" FROM COURSE;
SQL>MAX      MAX
-----
10000      30
```

➤ COUNT(*)

- **COUNT (*)** function returns total no of rows in a table including duplicates and having null values.
- COUNT is allowed only on numeric data types.

Syntax: COUNT(*);

Example:

```
SQL>SELECT COUNT(*) "NO. OF ROWS "FROM COURSE;
SQL>NO. OF ROWS
-----
6
```

➤ COUNT

- **COUNT** function returns total no of rows in a table where column does not contain null values.
- COUNT is allowed only on numeric data types.

Syntax: COUNT ([DISTINCT| ALL] COLUMNNAME);

Example:

```
SQL>SELECT COUNT (*) "NO. OF ROWS ", COUNT (C_FEE) "NO.OF ROWS", COUNT(DISTINCT  
C_FEE) "NO.OF ROWS", COUNT (C_SEATS) "NO.OF ROWS", COUNT (DISTINCT C_SEATS)  
"NO.OF ROWS"FROM COURSE;
```

```
SQL>NO. OF ROWS  NO.OF ROWS  NO.OF ROWS  NO.OF ROWS  NO.OF ROWS  
-----  
        6        5          3          5          3
```

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code/ Solutions to queries:

Signature of Faculty

Practical No.7: Implement SQL queries using Group by, Having and Order by Clause.

Consider Employee and Account table to solve the queries.

1. Display total balance of different accounts for each branch.
2. Display total balance of different accounts for 'vvn' branch.
3. Display branch names and total balance for each branch having total balance greater than 12000.
4. Write a query to find the total salary paid to employees for each city. Display the city name and total salary in descending order of total salary

A. Objective:

Implementing SQL queries using GROUP BY, HAVING, and ORDER BY clauses is to retrieve and analyse data from a database by grouping and aggregating similar data together, filtering the results based on specific conditions, and sorting the data in a specified order.

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO4,PO5,PO7

C. Expected Skills to be developed based on competency:

Develop skills of Data analysis and Problem solving by Compilation, debugging,executing SQL queries

D. Expected Course Outcomes(Cos)

Perform joins, sub-queries and nested queries on multiple tables using SQL*plus.

E. Practical Outcome(Pro)

Write data definition queries , compilation, debugging, executing using SQL

F. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

➤ **GROUP BY**

- GROUP BY clause is used in a SELECT statement to group rows into a set of summary rows by values of columns or expressions. The GROUP BY clause returns one row per group.
- The GROUP BY clause is often used with AGGREGATE FUNCTION such as AVG (),COUNT (),MAX (),MIN () and SUM (). In this case, the aggregate function returns the summary information per group.

Syntax:

```
SQL>SELECT COLUMN_1, COLUMN_2, COLUMN_3 .....COLUMN_N, AGGREGATE  
FUNCTION(COLUMN) FROM TABLENAME GROUP BY COLUMN_1, COLUMN_2, COLUMN_3  
.....COLUMN_N;
```

Example:

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

➤ **HAVING**

- HAVING clause is used in combination with the GROUP BY clause to restrict the groups of returned rows to only those who's the condition is TRUE.

Syntax:

```
SQL>SELECT COLUMN_1, COLUMN_2, COLUMN_3 .....COLUMN_N, AGGREGATE  
FUNCTION(COLUMN) FROM TABLENAME GROUP BY COLUMN_1, COLUMN_2, COLUMN_3  
.....COLUMN_N HAVING HAVING_CONDITION;
```

Example:

```
SQL>select job,sum(sal) from emp group by job having job='SALESMAN';
```

SQL>

JOB	SUM(SAL)
-----	----------

SALESMAN	5600
----------	------

(Above query calculate the total salary paid to all SALESMEN from employee table)

➤ **ORDER BY**

- The Oracle ORDER BY clause is used to sort the records in your result set. The ORDER BY clause can only be used in SELECT STATEMENT.

Syntax:

```
SQL>SELECT COLUMN_1, COLUMN_2, COLUMN_3 .....COLUMN_N OR * FROM  
TABLENAME ORDER BY COLUMN ASC OR DESC;
```

Example:

```
SQL>select * from emp order by ename asc;
```

SQL>Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

H. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

I. Source code /Solutions to queries:

Signature of Faculty

Practical No.8: Implement SQL queries using simple Case Operations and using Group functions and Case operations for getting summary data.

1. Write a query to categorise employee salary into Low, Medium and High. If salary <15000 then Low, between 15000 to 20000 then Medium and > 20000 then High.
2. Find out the total of the salaries in Low, Medium and High category. Refer above query for category limits.

A. Objective:

Write and execute SQL queries using simple Case Operations and using Group functions and Case operations for getting summary data.

B. Expected Program Outcomes (POs): PO1, PO2, PO3, PO4, PO5, PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Perform joins, sub-queries and nested queries on multiple tables using SQL*plus.

E. Practical Outcome(Pro)

Write data definition queries, compilation, debugging, executing using oracle software

F. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

- The case statement in SQL returns a value on a specified condition. We can use a Case statement in select queries along with Where, Order By, and Group By clause. It can be used in the Insert statement as well. In this article, we would explore the CASE statement and its various use cases.

- **Simple CASE expression**

simple CASE expression matches an expression to a list of simple expressions to determine the result.

The following illustrates the syntax of the simple CASE expression:

```
CASE e
  WHEN e1 THEN
    r1
  WHEN e2 THEN
    r2
  WHEN en THEN
    [ ELSE r_else ]
END
```

In this syntax, Oracle compares the input expression (e) to each comparison expression e1, e2, ..., en.

The following query uses the CASE expression to calculate the discount for each product category i.e., CPU 5%, video card 10%, and other product categories 8%

```
SELECT product_name,list_price,  
CASE category_id  
  WHEN 1  
    THEN ROUND(list_price * 0.05,2) -- CPU  
  WHEN 2  
    THEN ROUND(List_price * 0.1,2) -- Video Card  
    ELSE ROUND(list_price * 0.08,2) -- other categories  
END discount  
FROM products ORDER BY product_name;
```

Example:

Suppose we have two salary groups, above 2000 and below 2000. We want to see in which group which employee falls then we can write,

```
select ename,deptno, sal,case when sal<=2000 then 1 else 0 end below2000, case  
when sal>2000 then 1 else 0 end above2000 from emp;
```

Also when we want to get summary of same data, then we can write,

```
select deptno,sum(case when sal<=2000 then 1 else 0 end) below2000, sum(case  
when sal>2000 then 1 else 0 end) above2000 from emp group by deptno;
```

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code / Solutions to queries:

Signature of Faculty

Practical No.9: Implement SQL queries using Set operators like Union, union all, Intersect, Minus etc..

Create following tables :

Customer:		Account_holder:	
cid	cname	cid	ano
C01	Palak	C01	A01
C02	Tulsi	C02	A02
C03	Krishna	C03	A03
C04	Vaishali	C04	A04
C05	Shivali	C05	A05

Branch:	
Bname	baddress
vvn	Mota Bazar, VV Nagar
ksad	Chhota bazar, Karamsad
anand	Nana Bazar, Anand

Consider Customer and Employee table to perform below queries:

1. List out name of persons who are either customer or employee.
2. List out name of persons who are customer as well as employee.
3. List out name of persons who are customer but not employee.

A. Objective:

Write and execute SQL queries on various Set Operators

B. Expected Program Outcomes (POs): PO1, PO2, PO3, PO4, PO5, PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Perform joins, sub-queries and nested queries on multiple tables using SQL*plus

E. Practical Outcome(PRO)

Write data definition queries, compilation, debugging, executing using oracle software

F. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

- ORACLE installation

❖ SQL SET OPERATORS:-

- | | |
|-------------|-------------|
| ➤ UNION | ➤ INTERSECT |
| ➤ UNION ALL | ➤ MINUS |

UNION

- The UNION set operator returns all distinct rows selected by either query, without ignoring the NULL values. That means any duplicate rows will be removed. (Ascending by default)

Syntax:

```
SQL>SELECT COLUMNNAME FROM TABLENAME1
      UNION
      SELECT COLUMNNAME FROM TABLENAME2;
```

UNION ALL

- The UNION ALL set operator returns all rows selected by either query, without ignoring the NULL values. That means any duplicates will remain in the final result set.

Syntax:

```
SQL>SELECT COLUMNNAME FROM TABLENAME
      UNION ALL
      SELECT COLUMNNAME FROM TABLENAME;
```

INTERSECT

- The INTERSECT set operator returns all distinct rows selected by both queries. That means only those rows common to both queries will be present in the final result set.

Syntax:

```
SQL>SELECT COLUMNNAME FROM TABLENAME
      INTERSECT
      SELECT COLUMNNAME FROM TABLENAME;
```

MINUS

- The MINUS set operator returns all distinct rows selected by the first query but not the second.

Syntax:

```
SQL>SELECT COLUMNNAME FROM TABLENAME
      MINUS
      SELECT COLUMNNAME FROM TABLENAME;
```

Example: Suppose we have two different tables Fdetail3 and Fdetail5 mentioning 3rd and 5th semester faculty details as following.

Fdetail3	
Faculty Name	Department
A	Computer
B	Mechanical
C	Computer
D	Mechanical

Fdetail5	
Faculty Name	Department
A	Computer
Q	Electrical
R	Printing
C	Computer

Now we want to get list of all 3rd and 5th sem faculties without duplicates then,

SQL>select fname from fdetail3 union select fname from fdetail5;

Output Gives: A,B,C,D,Q,R,C

Example: In above mentioned example, suppose we want to keep duplicate values as it is while collecting all faculty details then,

SQL> select fname from fdetail3 union all select fname from fdetail5;

Output Gives: A,B,C,D,A,Q,R,C

Example: Suppose we want only those faculties which teach in both 3rd and 5th sem then,
SQL> select fname from fdetail3 Intersect select fname from fdetail5;

Output Gives: A

Example:

SQL> Suppose we want to get only those faculty which teach in sem 3 not in sem 5 then,
select fname from fdetail3 Minus select fname from fdetail5;

Output Gives: B,D

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code/Solutions to queries:

Signature of Faculty

Practical No.10: Retrieve data spread across various tables or same table using various Joins.

1. Combine consistent information of Account and branch table
2. Find branch name and address for account having account number 'A01'.
3. Display employee id and name along with name of their manager.
4. Perform cross product on account and branch table
5. Perform Left outer Join of Employee and Customer table on names.
6. Perform Right outer Join of Employee and Customer table on names.
7. Perform Full outer Join of Employee and Customer table on names.

A. Objective:

Write and execute SQL queries on various Joins.

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO4,PO5,PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Perform joins, sub-queries and nested queries on multiple tables using SQL*plus.

B. Practical Outcome(Pro)

Write data definition queries, compilation, debugging, executing using oracle software

C. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

D. Prerequisite Theory:

- Join operation is used to combine rows from two or more tables based on a related column between them. It allows you to retrieve data from multiple tables in a single query by specifying the relationship between them. Here's a brief overview of the theory behind performing queries with joins in a DBMS:

Types of Joins:

- Inner Join: Returns only the matching rows from both tables based on the join condition.
- Left Join: Returns all rows from the left (or first) table and the matching rows from the right (or second) table. If there is no match, NULL values are included for the columns from the right table.
- Right Join: Returns all rows from the right table and the matching rows from the left table. If there is no match, NULL values are included for the columns from the left table.
- Full Join: Returns all rows from both tables. If there is no match, NULL values are included for the columns from the respective table.
- Cross Join: Returns the Cartesian product of the two tables, resulting in all possible combinations of rows.
- Join Conditions:
- Equality Join: The most common type of join, where rows are matched based on the equality of values in the specified columns.

- Non-Equality Join: Rows are matched based on conditions other than equality, such as greater than, less than, etc.
- Self-Join: Joining a table with itself based on a relationship between different columns within the same table.
- Syntax: The syntax for performing joins in a DBMS varies slightly depending on the specific database system, but the general structure is as follows:

```
SELECT columns FROM table1 JOIN table2 ON join_condition;
```

E. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

F. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

G. Source code /Solutions to queries:

Signature of Faculty

Practical No.11: Retrieve data from multiple tables using Sub-queries

1. Find out the balance of an account which belongs to customer 'C01'.
2. Find out the balance of accounts which belongs to customer 'Tulsi'.

A. Objective:

- To retrieve data from multiple tables using sub-queries is to extract data from multiple tables based on some conditions or criteria. Sub-queries are used to break down complex queries into smaller, more manageable pieces, and help to retrieve data that is not easily obtainable from a single table.
- Multiple level subqueries are a powerful tool in SQL that allows us to extract data from nested subqueries. This means that the result of one subquery can be used as the input for another subquery, allowing for complex data extraction and manipulation.

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO4,PO5,PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Perform joins, sub-queries and nested queries on multiple tables using SQL*plus.

E. Practical Outcome(Pro)

Write data definition queries , compilation, debugging, executing using oracle software

F. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

- A sub-query is a query that is nested inside another query, and it can be used to retrieve data that will be used in the outer query.
- For example, if we want to get all employees data from Sales department then we can write,

```
Select * from emp where deptno = (select deptno from dept where deptname='SALES');
```

If the inner query returns more than one row, then we will get error of "inner query returns multiple rows" error in oracle. To resolve it we can use IN, ANY,ALL etc operators.

For example,

```
Select * from emp where deptno in (select deptno from dept where deptname='SALES' or deptname='MARKETING');
```

- Here's an example of how to use a co-related sub-query to retrieve data from multiple tables in Oracle:
Suppose we have two tables, "Customers" and "Orders". The "Customers" table contains customer information such as customer ID, name, and address, while the

"Orders" table contains information about the orders placed by each customer, such as order ID, date, and total cost.

- To retrieve the name and total number of orders for each customer, we can use a sub-query as follows:

```
SELECT c.name,
```

```
       (SELECT COUNT(*) FROM Orders o WHERE o.customer_id = c.customer_id) as  
num_orders FROM Customers c;
```

In this query, the sub-query is the SELECT statement inside the parentheses. It counts the number of orders for each customer by selecting all orders from the "Orders" table where the customer ID matches the customer ID in the outer query. The outer query selects the customer name from the "Customers" table and displays the number of orders returned by the sub-query as "num_orders".

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code /Solutions to queries:

Signature of Faculty

Practical No.12: Perform queries to Create, alter and update views

1. Create a view called "employee_info" that shows the employee's Name, Birthdate and salary.
2. Update the "employee_info" view to only show employees who have a salary greater than 10,000.
3. Drop the " employee_info " view from the database

A. Objective:

- To perform queries to create, alter, and update views is to allow for efficient data manipulation and retrieval in a database management system. Views are essentially virtual tables that are based on the results of a query, and they provide a simplified, customized representation of the data stored in the underlying tables.
- Creating views can be useful for simplifying complex queries, reducing the amount of code needed to retrieve certain information, and providing controlled access to specific data

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO5,PO6,PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Apply rules on datasets using SQL*Plus constraints

E. Practical Outcome(Pro)

Write Viewqueries, compilation, debugging, executing using oracle software

F. Expected Affective Domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

- A view is a virtual table that is derived from one or more base tables. Views can be used to simplify complex queries, to restrict access to sensitive data, or to present data in a particular format
1. Creating a View: To create a view, you need to define a query that specifies the data you want to include in the view. The query can involve one or more tables and can include various filtering conditions, aggregations, and joins. Here's the general syntax for creating a view:
 - Syntax:
CREATE VIEW view_name AS SELECT_Query...

For example, create view emp_dept_20 as select * from emp where deptno=20;
 2. Altering a View: To alter a view, you can modify its definition by changing the underlying query. The alteration can include adding or removing columns, modifying the filtering conditions, or changing the join conditions. However, some DBMS may have limitations on altering views, such as not allowing modifications to the underlying query structure. The specific syntax for altering a view depends on the DBMS you are using.
For oracle, you can write, Create or Replace view view_name as select Query....;
For example,

Create or replace view emp_dept_20 as select ename, salary from emp where deptno=20;

3. Removing the view: To remove the view, you can use drop command.

Drop view viewname;

For example, Drop view emp_dept_20;

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code /Solutions to queries:

Signature of Faculty

Practical No.13 Implement with Domain Integrity, Entity Integrity and Referential Integrity constraints.

1. Create a table named "**users**" with the following columns/ attributes:
id (integer, primary key), name (varchar2(50)),email (varchar2(100))
password (varchar2(100))
2. Create a table named "**products**" with the following columns:
id (integer, primary key), name (varchar2(100)), price (decimal(10,2) check price>100),
description (text)
3. Create a table named "**orders**" with the following columns:
id (integer, primary key),user_id (integer, foreign key references users(id)), product_id
(integer, foreign key references products(id)) quantity (integer) not null
4. Alter table Employee and add primary key constraint on employee number column and
not null constraint on employee name column.

A. Objective:

- To perform queries to domain integrity, entity integrity, and referential integrity constraints is to ensure that data in a database is consistent, accurate, and reliable.
- To ensure the quality and accuracy of data stored in a database

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO5,PO6,PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Apply rules on datasets using SQL*Plus constraints.

E. Practical Outcome(PRo)

Write data definition queries using integrity constraints, compilation, debugging, executing using oracle software

F. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

- Domain Integrity, Entity Integrity, and Referential Integrity are fundamental concepts in database management systems (DBMS) like Oracle. These constraints ensure that the data stored in the database is accurate, consistent, and valid.
 1. Domain Integrity: Domain Integrity ensures that the values entered in a column or attribute of a table conform to a specified data type or domain. For example, if a column is defined as storing only positive integers, any attempt to enter a negative number or a non-integer value will be rejected.
 2. Entity Integrity: Entity Integrity ensures that each row or record in a table has a unique identifier, typically a primary key. This constraint guarantees that there are no duplicate records in the table, ensuring that each record can be identified uniquely.

3. **Referential Integrity:** Referential Integrity ensures that the relationships between tables are maintained by enforcing the consistency of foreign key values. For example, if a table has a foreign key constraint that references a primary key in another table, the referential integrity constraint ensures that any changes to the referenced primary key value are reflected in the foreign key value in the referencing table.

- In Oracle, these constraints can be defined when creating a table or added later using the ALTER TABLE statement. We can use “constraint constraint_name” ahead of the all constraints to give our own constraint name, otherwise oracle will give system defined constraint name.

For example,

- To add a domain integrity constraint to a table, the following syntax can be used:
- Create table tablename(columnname1 datatype1 constraint constraint_name NotNull/Check condition, Columnname2 datatype2 NotNull/Check condition,ColumnnameN datatypeN NotNull/Check condition)

```
create table xyz (a number(10) constraint xyz_a_nn not null,b number(5));
```

- Alter table table_name add constraint constraint_name check (column_name <value>);
Alter table xyz add constraint xyz_b_ck check (b>50);
- To add a primary key constraint to a table, the following syntax can be used:
Create table tablename(columnname1 datatype1 primary key, Columnname2 datatype2,ColumnnameN datatypeN)

```
alter table table_name  
add constraint constraint_name  
primary key (column_name)
```

- To add a foreign key constraint to a table, the following syntax can be used:

```
Create table tablename1(columnname1 datatype1 references table2(columnx),  
Columnname2 datatype2, .....ColumnnameN datatypeN);
```

```
Alter table table_name add constraint constraint_name foreign key  
(column_name) references referenced_table (referenced_column)
```

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code /Solutions to queries:

Signature of Faculty

Practical No.14: Perform queries to Create synonyms, sequence and index.

1. Create a synonym for the employees table.
2. Create a sequence named seq_emp_id that starts at 1 and increments by 1
3. Create an index named idx_emp_name on the ename column of the employees table.

A. Objective:

- The objective of creating synonyms is to provide an alternative name or alias for an existing database object, such as a table, view, or procedure. Synonyms can be useful in simplifying the syntax of SQL statements and in providing an additional layer of security by hiding the underlying object names.
- The objective of creating sequences is to generate unique sequential numbers or values that can be used as primary keys or other identifiers in tables. Sequences can be used to simplify the process of inserting new records into a table, as they can automatically generate the next available value.
- The objective of creating indexes is to improve the performance of database queries by creating a data structure that allows for faster searching and retrieval of data. Indexes can be created on one or more columns in a table, and they can greatly improve the performance of queries that use those columns in a WHERE clause or JOIN condition.

B. Expected Program Outcomes (POs): PO1, PO2, PO3, PO5, PO6, PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries related to database objects.

D. Expected Course Outcomes(Cos)

Use Database Objects like Synonym, Views, Sequence and Index for efficient database handling.

E. Practical Outcome(Pro)

Write data objects, compilation, debugging, executing using oracle software

F. Expected Affective Domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

• **Synonym –**

A synonym is an alternative name for objects such as tables, views, sequences, stored procedures and other database objects.

Creating a Synonym for an object: -

SYNTAX: -

CREATE [PUBLIC] SYNONYM [SCHEMA.] synonym_name FOR [SCHEMA.] object_name;

If the word PUBLIC is written in query, then the synonym can be accessible by all the users.

But user must have appropriate privileges to the object.

If the SCHEMA phrase is not used, then reference is made to user's own schema.

Dropping Synonym:-

A synonym can be dropped by using DROP SYNONYM. When the synonym is dropped, the changes made in table through synonym remains unchanged.

SYNTAX:-

DROP [PUBLIC] SYNONYM [SCHEMA.] synonym_name [FORCE];

While dropping synonym, if the word PUBLIC is written, then there is no need to write a schema name.

The FORCE word will force oracle to drop synonym, even if it has dependencies.

Use of Synonym:-

- To hide the actual identity of the object.
- To abbreviated the longer name.

Sequence –

A sequence is an automatic counter which generates sequential numbers whenever required.

Creating a Sequence:-

SYNTAX:-

```
CREATE SEQUENCE sequence_name
INCREMENT BY n
START WITH n
MAXVALUE n / NOMAXVALUE
MINVALUE n / NOMINVALUE
CACHE n / NOCACHE
CYCLE / NOCYCLE
ORDER / NOORDER;
```

Here, in syntax, **START WITH** specifies the first sequence number, **INCREMENT BY** specifies the interval between sequence numbers, **MAXVALUE** specifies the sequence maximum value, **CYCLE** specifies to repeat cycle of generating values after reaching maximum value, **and CACHE** specifies how many values to generate in advance and to keep in memory for faster access.

The value generated by a sequence can be used in insert and update operations. The minimum information required for generating numbers using a sequence is

- The starting number
- The maximum number that can be generated by a sequence
- The increment value for generating the next number

Altering Sequence:-

An already created sequence can be altered by following command. The start value of the sequence cannot be altered.

SYNTAX:-

```
ALTER SEQUENCE sequence_name
INCREMENT BY n
MAXVALUE n / NOMAXVALUE
MINVALUE n / NOMINVALUE
CACHE n / NOCACHE
CYCLE / NOCYCLE
ORDER / NOORDER;
```

Dropping Sequence:-

A sequence can be dropped by using DROP SEQUENCE command.

SYNTAX:-

```
DROP SEQUENCE sequence_name;
```

Referencing Sequence:-

Oracle provides two pseudo columns NEXTVAL and CURRVAL to access the values generated by the sequence.

To view sequence value, SELECT statement is used as below:

```
SELECT sequence_name.CURRVAL FROM DUAL;
```

This will display the current value of the sequence. To display the next value held in the cache, following SELECT statement is used:

```
SELECT sequence_name.NEXTVAL FROM DUAL;
```

Index –

An index is an ordered list of content of a column or group of columns in a table.

An index created on the single column of the table is called simple index. When multiple table columns are included in the index, it is called composite index.

Creating an Index for a table:-**SYNTAX: -****Simple Index:-**

```
CREATE INDEX index_name ON tablename (column_name);
```

Composite Index:-

```
CREATE INDEX index_name ON tablename (column_name, column_name);
```

Above indexes do not enforce uniqueness so the columns included in the index can hold duplicate values. Indexes that deny duplicate values for the indexed columns are called Unique Index.

Creating a unique Index for a table: -

To create unique index, the keyword UNIQUE should be included in the CREATE INDEX command.

SYNTAX:-**Simple Unique Index:-**

```
CREATE UNIQUE INDEX index_name ON tablename (column_name);
```

Composite Unique Index: -

```
CREATE UNIQUE INDEX index_name ON tablename (column_name, column_name);
```

Dropping Indexes:-

An index associated with the table can be removed by using DROP INDEX command.

```
SYNTAX: -DROP INDEX index_name;
```

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code /Solutions to queries:

Signature of Faculty

Practical No.15: Implement PL/SQL programs using control structures.

1. Write a PL/SQL block to print the values 1 to 10 in reverse order using for loop.
2. Write a PL/SQL block to accept the value of A,B&C display maximum number from them

A. Objective:

- To implement PL/SQL programs using control structures is to develop programs that can make decisions and perform different actions based on those decisions. Control structures in PL/SQL include conditional statements, loops, and exception handling.
- Using these control structures, PL/SQL programmers can create more sophisticated and robust programs that can handle a wide range of scenarios and user inputs. Additionally, control structures help to improve the efficiency and readability of PL/SQL code by reducing the amount of repetitive code that needs to be written.

B. Expected Program Outcomes (POs): PO1, PO2, PO3, PO4, PO5, PO6, PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing PL-SQL Blocks

D. Expected Course Outcomes(Cos)

Write PL/SQL block using concept of Cursor Management, Error Handling, Package and Triggers

E. Practical Outcome(Pro)

Write PL SQL Blocks, compilation, debugging, executing using oracle software

F. Expected Affective Domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

Basics of PL/SQL

Programming language/structured query language. It provides facility of procedural and function. It provides programming techniques like branching, looping, and condition check. It is fully structured programming language. It decreases the network traffic because entire block of code is passed to the DBA at one time for execution.

Basic Structure of PL/SQL

PL/SQL stands for Procedural Language/SQL. PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a block. A block has the following structure:

Syntax:

DECLARE

/* Declarative section: variables, types, and local subprograms. */

BEGIN

/* Executable section: procedural and SQL statements go here. */

/* This is the only section of the block that is required. */

EXCEPTION

/* Exception handling section: error handling statements go here. */

END;**Control Structures**

PL/SQL Control Structures are used to control flow of execution. PL/SQL provides different kinds of statements to provide such type of procedural capabilities. These statements are almost same as that of provided by other languages.

The flow of control statements can be classified into the following categories:

- Conditional Control
- Iterative Control
- Sequential Control

Conditional Control:

PL/SQL allows the use of an IF statement to control the execution of a block of code. In PL/SQL, the IF -THEN - ELSIF - ELSE - END IF construct in code blocks allow specifying certain conditions under which a specific block of code should be executed.

Syntax:

```
IF < Condition > THEN
    < Statement >
ELSIF <Condition> THEN
    < Statement >
ELSE < Statement >
END IF;
```

Example:

Create file named "condi.sql"

```
DECLARE
    a Number := 30;
    b Number;
BEGIN
    IF a > 40 THEN
        b := a - 40;
        DBMS_OUTPUT.PUT_LINE('b=' || b);
    elsif a = 30 then
        b := a + 40;
        DBMS_OUTPUT.PUT_LINE('b=' || b);
    ELSE
        b := 0;
        DBMS_OUTPUT.PUT_LINE('b=' || b);
    END IF;
END;
/
```

NOTE: Remember to use SET SERVEROUTPUT ON at start of PL-SQL Session to see the output.

Iterative Control:

Iterative control indicates the ability to repeat or skip sections of a code block. A **loop** marks a sequence of statements that has to be repeated. The keyword loop has to be placed before the first

statement in the sequence of statements to be repeated, while the keyword end loop is placed immediately after the last statement in the sequence.

Once a loop begins to execute, it will go on forever. Hence a conditional statement that controls the number of times a loop is executed always accompanies loops.

PL/SQL supports the following structures for iterative control:

Simple loop:

In simple loop, the key word loop should be placed before the first statement in the sequence and the keyword end loop should be written at the end of the sequence to end the loop.

Syntax:

```
Loop
  < Sequence of statements >
End loop;
```

Example:

Create file named it.sql

```
DECLARE
  i number := 0;
BEGIN
  LOOP
    dbms_output.put_line ('i = ' || i);
    i:=i+1;
    EXIT WHEN i>=11;
  END LOOP;
END;
/
```

WHILE loop

The **while** loop executes commands in its body as long as the condition remains true

Syntax:

```
while< condition >
loop
  < action >
end loop
```

Example:

Find reverse of given number using while loop

```
declare
  num number(3) :=123;
  ans number(3) :=0;
  i number(3) :=0;
begin
  while num != 0
  loop
    i:=mod(num,10);
    ans:=(ans * 10 ) + i;
    num:=floor(num/10);
  end loop;
```

```
dbms_output.put_line('reverse of given number is: ' || ans);  
end;  
/
```

The FOR Loop

The **FOR** loop can be used when the number of iterations to be executed are known.

Syntax:

The variable in the For Loop need not be declared. Also the increment value cannot be specified. The For Loop variable is always incremented by 1.

```
FOR variable IN [REVERSE] start..end  
LOOP  
    < Action >  
END LOOP;
```

Example:

```
declare  
    i number ;  
begin  
    for i in 1 .. 10  
    loop  
        dbms_output.put_line ('i = ' || i);  
    end loop;  
end;  
/
```

Sequential Control:

The GOTO Statement

The GOTO statement changes the flow of control within a PL/SQL block. This statement allows execution of a section of code, which is not in the normal flow of control. The entry point into such a block of code is marked using the tags «userdefined name». The GOTO statement can then make use of this user-defined name to jump into that block of code for execution.

Syntax :

```
GOTO jump;  
....  
<<jump>>
```

Example :

```
declare
begin
  dbms_output.put_line ('code starts');
  dbms_output.put_line ('before gotostatements');
  goto down;
  dbms_output.put_line ('statement will not get executed..');
<<down>>
  dbms_output.put_line ('flow of execution jumped here.');
```

end;
/

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code/Solutions to queries:

Signature of Faculty

Practical No.16: Implement PL/SQL programs using Cursors.

1. Write a PL/SQL block that will accept an account number from the user. Check if the user's balance is less than the minimum balance, only then deduct Rs.100/- from the balance. (minimum balance is 5000)
2. In account table , branch names are stored in lowercase. Convert into uppercase for branch specified by user and display how many accounts are affected.

A. Objective:

- To provide a way for the PL/SQL programmer to manipulate and process data stored in a relational database. Cursors in PL/SQL allow programmers to retrieve data from one or more database tables and process it one row at a time, which can be very useful in scenarios where it's not feasible or efficient to retrieve all the data at once.
- Cursors can be used in a wide range of PL/SQL applications, from simple database queries to complex data processing tasks, and can help improve the performance and efficiency of database operations

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO4,PO5,PO6,PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Write PL/SQL block using concept of Cursor Management, Error Handling, Package and Triggers.

E. Practical Outcome(Pro)

Write data definition queries, compilation, debugging, executing using oracle software

F. Expected Affective Domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

Cursors

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the active set. There are two types of cursors in PL/SQL:

1. Implicit cursors.
2. Explicit cursors.

Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed.

Cursor Attributes	
Name	Description
%FOUND	Returns TRUE if record was fetched successfully, FALSE otherwise.

%NOTFOUND	Returns TRUE if record was not fetched successfully, FALSE otherwise.
%ROWCOUNT	Returns number of records fetched from cursor at that point in time.
%ISOPEN	Returns TRUE if cursor is open, FALSE otherwise.

Implicit cursor:

These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed. When you execute DML statements like DELETE, INSERT, UPDATE and SELECT statements, implicit statements are created to process these statements. Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations.

The cursor attributes available are

- %FOUND
- %NOTFOUND
- %ROWCOUNT
- %ISOPEN

For example, when you execute INSERT, UPDATE, or DELETE statements the cursor attributes tell us whether any rows are affected and how many have been affected. When a SELECT... INTO statement is executed in a PL/SQL Block, implicit cursor attributes can be used to find out whether any row has been returned by the SELECT statement. PL/SQL returns an error when no data is selected.

Consider the PL/SQL Block that uses implicit cursor attributes as shown below:

Example:

```

DECLARE
  Eid number(3);
BEGIN
  UPDATE emp set eid=&eid where salary=&salary;
  eid:=sql%rowcount;
  IF SQL%found then
    dbms_output.put_line('success');
  ELSE
    dbms_output.put_line( ' not' );
  END IF;
  dbms_output.put_line( 'rowcount' || eid);
END;
/

```

Explicit Cursors:

They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row. An explicit cursor is defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row. We can provide a suitable name for the cursor.

Syntax:

```
CURSOR cursor_name IS select_statement;
```

Here,

cursor_name -A suitable name for the cursor.

Select_statement - A select query which returns multiple rows.

How to use Explicit Cursor?

There are four steps in using an Explicit Cursor.

1. **DECLARE** the cursor in the declaration section
2. **OPEN** the cursor in the Execution Section.
3. **FETCH** the data from cursor into PL/SQL variables or records in the Execution Section.
4. **CLOSE** the cursor in the Execution Section before you end the PL/SQL Block.

Declaring a Cursor in the Declaration Section:

Syntax:

```
CURSOR CURSORNAME ISSELECT.....;
```

How to access an Explicit Cursor?

These are the three steps in accessing the cursor.

OPEN THE CURSOR:

Syntax:

```
OPEN cursor_name;
```

FETCH THE RECORDS IN THE CURSOR ONE AT A TIME:

Syntax:

```
FETCH cursor_name INTO record_name;  
OR  
FETCH cursor_name INTO variable_list;
```

CLOSE THE CURSOR:

Syntax:

```
CLOSE cursor__name;
```

When a cursor is opened, the first row becomes the current row. When the data is fetched it is copied to the record or variables and the logical pointer moves to the next row and it becomes the current row. On every fetch statement, the pointer moves to the next row. If you want to fetch after the last row, the program will throw an error. When there is more than one row in a cursor we can use loops along with explicit cursor attributes to fetch all the records.

Points to remember while fetching a row:

1. We can fetch the rows in a cursor to a PL/SQL Record or a list of variables created in the PL/SQL Block.
2. If you are fetching a cursor to a PL/SQL Record, the record should have the same structure as the cursor.
3. If you are fetching a cursor to a list of variables, the variables should be listed in the same order in the fetch statement as the columns are present in the cursor.

General Form of using an explicit cursor is:

Syntax:

```
DECLARE  
variables;
```

```

records;
create a cursor;
BEGIN
  OPEN cursor;
  FETCH cursor;
  process the records;
  CLOSE cursor;
END;
/

```

Example:

```

DECLARE
  CURSOR er IS select eid,name from emp order by name ;
  id emp.eid%type;
  ename emp.name%type;
BEGIN
  OPEN er;
  Loop
    FETCH er into id,ename;
    Exit when er%notfound;
    dbms_output.put_line (id || ename);
  end loop;
  close er;
END;
/

```

CURSOR FOR LOOP:

Oracle provides other loop-statements to control loops specifically for cursors. This statement is a variation of the basic FOR loop, and it is known as cursor for loops.

Syntax:

```

FOR VARIABLE IN CURSORNAME
LOOP
<EXECUTE COMMANDS>
END LOOP

```

PARAMETERIZED CURSORS:

Oracle allows passing parameters to cursors that can be used to provide condition with WHERE clause. If parameters are passed to cursor, that cursor is called **parameterized cursors**.

Syntax:

```

CURSOR CURSORNAME (VARIABLENAME DATATYPES ) IS SELECT.....

```

And, parameters can be passed to cursor while opening it using syntax-

Syntax:

```

OPEN CURSORNAME (VALUE / VARIABLE / EXPRESSION );

```

Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

H. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

I. Source code/Solutions to queries:

Signature of Faculty

Practical No.17: Implement PL/SQL programs using exception handling.

1. Write a program queries the employees table for a non-existing employee, which will raise a NO_DATA_FOUND exception.
2. Create a PL/SQL program that inserts data into the employee table. The program should handle exceptions raised on encountering a NULL value for the primary key column or, if a record already exists with the same employee ID, the program should raise a named exception called "DUP_VAL_ON_INDEX". Display proper message

A. Objective:

To improve the reliability and robustness of the code by handling unexpected or exceptional situations that may occur during program execution.

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO4,PO5,PO6,PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Write PL/SQL block using concept of Cursor Management, Error Handling, Package and Triggers.

E. Practical Outcome(PRo)

Write PL-SQL programs with Error Handling.

F. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

EXCEPTIONS

An Exception is an error situation, which arises during program execution. When an error occurs exception is raised, normal execution is stopped and control transfers to exception handling part. Exception handlers are routines written to handle the exception. The exceptions can be internally defined (system-defined or pre-defined) or User-defined exception.

EXCEPTION

WHEN <ExceptionName> THEN

<User Defined Action To Be Carried Out>

Syntax:

PREDEFINED EXCEPTION:

Predefined exception is raised automatically whenever there is a violation of Oracle coding rules. Predefined exceptions are those like ZERO_DIVIDE, which is raised automatically when we try to divide a number by zero. Other built-in exceptions are given below. You can handle unexpected Oracle errors using OTHERS handler. It can handle all raised exceptions that are not handled by any other handler. It must always be written as the last handler in exception block.

Exception	Raised when....
DUP_VAL_ON_INDEX	When you try to insert a duplicate value into a unique column.
INVALID_CURSOR	It occurs when we try accessing an invalid cursor.
INVALID_NUMBER	On usage of something other than number in place of number value.
LOGIN_DENIED	At the time when user login is denied.

TOO_MANY_ROWS	When a select query returns more than one row and the destination variable can take only single value.
VALUE_ERROR	When an arithmetic, value conversion, truncation, or constraint error occurs.
CURSOR_ALREADY_OPEN	Raised when we try to open an already open cursor.

Predefined exception handlers are declared globally in package STANDARD. Hence we need not have to define them rather just use them. The biggest advantage of exception handling is it improves readability and reliability of the code. Errors from many statements of code can be handles with a single handler. Instead of checking for an error at every point we can just add an exception handler and if any exception is raised it is handled by that. For checking errors at a specific spot it is always better to have those statements in a separate begin – end block.

Example:

```
declare
  n number;
begin
  n:=10/0;
exception when zero_divide then
  dbms_output.put_line ('divide by zero error occurs...');
end;
/
```

USER-DEFINED EXCEPTIONS:

The technique that is used is to bind a numbered exception handler to a name using Pragma Exception_init (). This binding of a numbered exception handler, to a name (i.e. a String), is done in the Declare section of a PL/SQL block. The Pragma action word is a call to a pre-compiler, which immediately binds the numbered exception handler to a name when encountered.

The function Exception_init() takes two parameters the first is the user defined exception name the second is the Oracle engine's exception number. These lines will be included in the Declare section of the PL/SQL block.

The user defined exception name must be the statement that immediately precedes the Pragma Exception_init() statement.

Syntax:

```
DECLARE
< ExceptionName > EXCEPTION ;
  PRAGMA EXCEPTION_INIT (< ExceptionName >, <ErrorCodeNo>);
BEGIN
```

Using this technique it is possible to bind appropriate numbered exception handlers to names and use these names in the Exception section of a PL/SQL block. When this is done the default exception handling code of the exception handler is overridden and the user-defined exception handling code is executed.

Syntax:

```
DECLARE
```

```
< ExceptionName > EXCEPTION ;
    PRAGMA EXCEPTION_INIT (< ExceptionName >, <ErrorCodeNo>);
BEGIN
    . . . . .
EXCEPTION
    WHEN < ExceptionName > THEN
< Action >
END;
```

Example:

Create table named emp have column like id with notnull constraint, name and etc.

```
DECLARE
    e_MissingNull EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_MissingNull, -1400);
BEGIN
    INSERT INTO emp(id) VALUES (NULL);
EXCEPTION
    WHEN e_MissingNull then
        DBMS_OUTPUT.put_line('ORA-1400 occurred');
END;
/
```

USER DEFINED EXCEPTION HANDLING:

To trap business rules being violated the technique of raising user-defined exceptions and then handling them, is used.

User-defined error conditions must be declared in the declarative part of any PL/SQL block. In the executable part, a check for the condition that needs special attention is made. If that condition exists, the call to the user-defined exception is made using a RAISE statement. The exception once raised is then handled in the Exception handling section of the PL/SQL code block.

Syntax:

```
DECLARE
< EXCEPTIONNAME > EXCEPTION;
BEGIN
<SQL SENTENCE >;
    IF < CONDITION > THEN
        RAISE <EXCEPTIONNAME>;
    END IF;
EXCEPTION
    WHEN <EXCEPTIONNAME> THEN {USER DEFINED ACTION TO BE TAKEN};
END;
```

EXAMPLE:

```
DECLARE
    EX EXCEPTION;
BEGIN
    IF TO_CHAR(SYSDATE, 'DY') == 'SUN' THEN
```

```
RAISE EX;  
END IF;  
EXCEPTION  
    WHEN EX THEN  
        DBMS_OUTPUT.PUT_LINE('NO TRANSCATIONS TODAY');  
END;  
/
```

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code /Solutions to queries :

Signature of Faculty

Practical No.18: Implement user defined procedures and functions using PL/SQL blocks.

1. Create a PL/SQL block that defines a function to calculate the area of a rectangle. The user should be prompted to enter the length and width of the rectangle, and the results should be displayed
2. Create a procedure that will take Employee number as input and employee name as output parameter. print the name of the employee for given employee number.

A. Objective:

To implement user-defined procedures and functions using PL/SQL blocks is to provide a way to encapsulate business logic and perform specific tasks on the database. Procedures and functions allow you to group a set of SQL statements together and give them a name, so that they can be executed as a single unit.

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO4,PO5,PO6,PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Write PL/SQL block using concept of Cursor Management, Error Handling, Package and Triggers.

E. Practical Outcome(Pro)

Write PL-SQL Procedures and Functions.

F. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

PROCEDURES & FUNCTIONS

"A **procedures** or **function** is a group or set of SQL and PL/SQL statements that perform a specific task."A function and procedure is a named PL/SQL Block which are similar. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

PROCEDURES:

A procedure is named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages. A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure.

The body consists of declaration section, execution section and exception section similar to a general PL/SQL Block. A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

We can pass parameters to procedures in three ways :

Parameters	Description
IN type	These types of parameters are used to send values to stored procedures.
OUT type	These types of parameters are used to get values from stored procedures. This is similar to a return type in functions.
IN OUT type	These types of parameters are used to send values and get values from stored procedures.

A procedure may or may not return any value.*Syntax:*

```
CREATE [ORREPLACE] PROCEDURE procedure_name (<Argument> {IN, OUT, INOUT}
<Datatype>,...)
IS
  Declaration section<variable, constant>;
BEGIN
  Execution section
EXCEPTION
Exceptionsection
END;
/
```

IS - marks the beginning of the body of the procedure and is similar to DECLARE in anonymous PL/SQL Blocks. The code between IS and BEGIN forms the Declaration section. The syntax within the brackets [] indicate they are optional. By using CREATE OR REPLACE together the procedure is created if no other procedure with the same name exists or the existing procedure is replaced with the current code.

How to execute a Procedure?

There are two ways to execute a procedure :

- From the SQL prompt : EXECUTE [or EXEC] procedure_name;
- Within another procedure – simply use the procedure name : procedure_name;

Example:

Create table named emp have two column id and salary with number datatype.

```
create or replace procedure p1(id in number, sal in number)
as
begin
insert into emp values(id, sal);
  dbmd_output.put_line('value inserted.');
```

```
end;
/
```

Run SQL Command

```
SQL>set serveroutput on
SQL>start D://pr.sql
Procedure created.

SQL>exec p1(5,4);
VALUE INSERTED.
PL/SQL procedure successfully completed.

SQL>select * from emp;
   ID    SALARY
----  -
     2      5000
```

FUNCTIONS:

A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

Syntax:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
RETURN return_datatype {IS, AS}
Declaration_section <variable,constant> ;
BEGIN
  Execution_section
  Return return_variable;
EXCEPTION
```

```

exception section
  Return return_variable;
END;
/

```

RETURN TYPE: The header section defines the return type of the function. The return datatype can be any of the oracle datatype like varchar, number etc.

The execution and exception section both should return a value which is of the datatype defined in the header section.

How to execute a Function?

A function can be executed in the following ways.

- As a part of a SELECT statement : SELECT emp_details_func FROM dual;
- In a PL/SQL Statements like, : dbms_output.put_line(emp_details_func);

This line displays the value returned by the function .

Example:

```

Create or replace function getsal (noINnumber)
  return number
is
  sal number(5);
begin
  select salary into sal from emp where id=no;
  return sal;
end;
/

```

Output:

Run SQL Command Line

```

SQL>select * from emp;
   ID   SALARY
-----
    2     5000

SQL>start D://fun.sql
Function created.

SQL>select getsal(2) from dual;
GETSAL(2)
-----
      5000

```

In the example we are retrieving the 'salary' of employee with id 2 to variable 'sal'. The return type of the function is number.

Destroying procedure and function:

Syntax:

DROP PROCEDURE/FUNCTION PROCEDURE/FUNCTION_NAME;

Procedures VS Functions:

- A function MUST return a value
- A procedure cannot return a value
- Procedures and functions can both return data in OUT and IN OUT parameters
- The return statement in a function returns control to the calling program and returns the results of the function
- The return statement of a procedure returns control to the calling program and cannot return a value
- Functions can be called from SQL, procedure cannot
- Functions are considered expressions, procedure are not

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code /Solutions to queries:

Signature of Faculty

Practical No.19: Perform various operations on packages.

1. Create a package called "Employee_Info" that contains two functions, "get_salary" and "get_city". The "get_salary" function takes an employee ID as input and returns the employee's salary. The "get_city" function takes an employee ID as input and returns the employee's job title

A. Objective:

To perform various operations on packages in Oracle is to organize and manage code more efficiently, improve performance, and simplify application maintenance.

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO4,PO5,PO6,PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL packages

D. Expected Course Outcomes(Cos)

Write PL/SQL block using concept of Cursor Management, Error Handling, Package and Triggers.

E. Practical Outcome(PRO)

Write code to create, compile, debug, execute packages using oracle software

F. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

PACKAGES

"A **package** is a container for other database objects."

A package can hold other database objects such as variables, constants, cursors, exceptions, procedures, functions and sub-programs.

A package has usually two components, a **specification** and a **body**.

A package's specification declares the types (variables of the record type), memory variables, constants, exceptions, cursors, and subprograms that are available for use.

A package's body fully defines cursors, functions, and procedures and thus implements the specification.

PACKAGE SPECIFICATION:

The package specification contains:

- Name of the package
- Names of the data types of any arguments
- This declaration is local to the database and global to the package

Syntax:

```
Create [or replace] package package_name
{is | as}
  Package_specification
End package_name;
```

Example:

Create package operation that contains procedure 'add' and function 'sub'.

```
Create or replace package operation
Is
  procedure addition(a in number,b in number);
  function sub(a in number,b in number)return number;
End operation;
/
```

PACKAGE BODY:

The body of a package contains the definition of public objects that are declared in the specification.

The body can also contain other object declarations that are private to the package. The objects declared privately in the package body are not accessible to other objects outside the package.

Unlike package specification, the package body can contain subprogram bodies. After the package is written, debugged, compiled and stored in the database applications can reference the package's types, call its subprograms, use its cursors, or raise its exceptions.

Syntax:

```
Create [or replace] package body package_name
{is | as}
  Package_body
End package_name;
```

Example:

```
Create or replace package body operation
is
  procedure addition(a in number,b in number)
  is
  begin
    dbms_output.put_line('addtion of two number :'|add(a,b));
  end;
  function sub(a in number,b in number)return number
  is
    ans number(3);
  begin
    ans:=a-b;
    return ans;
  end;
end operation;
/
```


H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code/Solutions to queries :

Signature of Faculty

Practical No.20: Implement various triggers.

1. Create a trigger that prevents any changes to the salary column of the employees table for employees whose city is "Surat"
2. Create trigger on Account table before update, delete, insert and display the message which operation is performed on the table.

A. Objective:

To implement various triggers is to automate actions in response to specific events or changes in data. Triggers are used in software applications to monitor and respond to specific actions, such as user input or changes in data. By setting up triggers, you can automate certain tasks, reduce manual effort, and ensure that important actions are performed consistently and in a timely manner

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO4,PO5,PO6,PO7

C. Expected Skills to be developed based on competency:

Compilation, debugging, executing SQL queries

D. Expected Course Outcomes(Cos)

Write PL/SQL block using concept of Cursor Management, Error Handling, Package and Triggers.

E. Practical Outcome(Pro)

Write code to create, compile, debug, execute Triggers using oracle software

F. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

TRIGGERS

A trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

Syntax:

```
Create [or replace ] trigger trigger_name
{before | after | instead of }
{insert [or] | update [or] | delete}
[of col_name]
on table_name
[referencing old as o new as n]
[for each row]
when (condition)
begin
--- sql statements
end;
```

Each statements are described below :

Statement	Description
CREATE [OR REPLACE] TRIGGER trigger_name	This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
{BEFORE AFTER INSTEAD OF }	This clause indicates at what time the trigger should get fired. i.e for example: before or after updating a table. INSTEAD OF is used to create a trigger on a view. before and after cannot be used to create a trigger on a view.
{INSERT [OR] UPDATE [OR] DELETE}	This clause determines the triggering event. More than one triggering events can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.
[OF col_name]	This clause is used with update triggers. This clause is used when you want to trigger an event only when a specific column is updated.
CREATE [OR REPLACE] TRIGGER trigger_name	This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
[ON table_name]	This clause identifies the name of the table or view to which the trigger is associated.
[REFERENCING OLD AS o NEW AS n]	This clause is used to reference the old and new values of the data being changed. By default, you reference the values as :old.column_name or :new.column_name. The reference names can also be changed from old (or new) to any other user-defined name. You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.
[FOR EACH ROW]	This clause is used to determine whether a trigger must fire when each row gets affected (i.e. a Row Level Trigger) or just once when the entire sql statement is executed(i.e.statement level Trigger).
WHEN (condition)	This clause is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

Example:

Create or replace Trigger np before insert on account for each row

Begin

IF :NEW.bal < 0 THEN

DBMS_OUTPUT.PUT_LINE('BALANCE IS NAGATIVE..');

END IF;

End;

Output:

```

Run SQL Command Line
SQL>start D://t.sql
Trigger created.

SQL>insert into account values(1,-100);
BALANCE IS NAGATIVE..
1 row created.

```

TYPES OF TRIGGERS

A trigger's type is defined by the type of triggering transaction and by the level at which the trigger is executed. Oracle has the following types of triggers depending on the different applications.

- Row Level Triggers
- Statement Level Triggers
- Before Triggers
- After Triggers

Row Level Triggers :

Row level triggers execute once for each row in a transaction. The commands of row level triggers are executed on all rows that are affected by the command that enables the trigger.

For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement. If the triggering statement affects no rows, the trigger is not executed at all. Row level triggers are created using the FOR EACH ROW clause in the CREATE TRIGGER command.

Application:

Consider a case where our requirement is to prevent updation of empno 100 record cannot be updated, then whenever UPDATE statement update records, there must be PL/SQL block that will be fired automatically by UPDATE statement to check that it must not be 100, so we have to use Row level Triggers for that type of applications.

Statement Level Triggers:

Statement level triggers are triggered only once for each transaction. For example when an UPDATE command update 15 rows, the commands contained in the trigger are executed only once, and not with every processed row. Statement level trigger are the default types of trigger created via the CREATE TRIGGER command.

Application:

Consider a case where our requirement is to prevent the DELETE operation during Sunday. For this whenever DELETE statement deletes records, there must be PL/SQL block that will be fired only once by DELETE statement to check that day must not be Sunday by referencing system date, so we have to use Statement level Trigger for which fires only once for above application.

Before Trigger:

Since triggers are executed by events, they may be set to occur immediately before or after those events.

When a trigger is defined, you can specify whether the trigger must occur before or after the triggering event i.e. INSERT, UPDATE, or DELETE commands. BEFORE trigger execute the trigger action before the triggering statement. These types of triggers are commonly used in the following situation:

1. BEFORE triggers are used when the trigger action should determine whether or not the triggering statement should be allowed to complete.

2. By using a BEFORE trigger, you can eliminate unnecessary processing of the triggering statement.
3. For example: To prevent deletion on Sunday, for this we have to use Statement level before trigger on DELETE statement.
4. BEFORE triggers are used to derive specific column values before completing a triggering INSERT or UPDATE statement.

After Trigger:

AFTER trigger executes the trigger action after the triggering statement is executed. AFTER triggers are used when you want the triggering statement to complete before executing the trigger action.

For example:

To perform cascade delete operation, it means that user delete the record from one table, but the corresponding records in other tables are deleted automatically by a trigger which fired after the execution of delete statement issued by the user. When combining the different types of triggering actions, there are mainly 4 possible Valid trigger types available to us.

The possible configurations are:

- BEFORE statement Trigger
- BEFORE row Trigger
- AFTER statement Trigger
- AFTER row Trigger

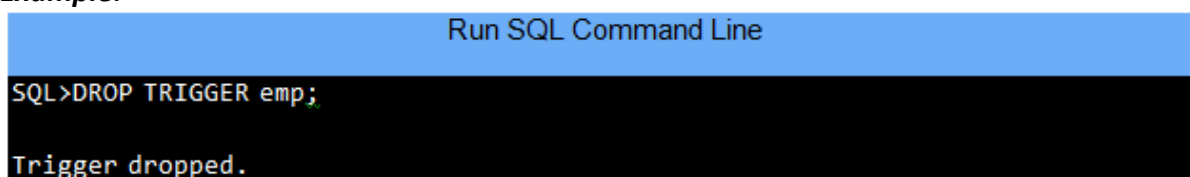
Dropping Triggers:

Triggers may be dropped via the drop trigger command. In order to drop a trigger, you must either own the trigger or have the DROP ANY TRIGGER system privilege.

Syntax:

```
DROP TRIGGER trigger_name;
```

Example:



The screenshot shows a terminal window titled "Run SQL Command Line". The prompt is "SQL>DROP TRIGGER emp;". The output is "Trigger dropped.".

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Source code /Solution to queries :

Signature of Faculty

Practical No.21: E-R Diagrams and Normalization.

Design an E-R diagram for a university database management system that stores information about students, courses, and faculty members. The database should allow the university to manage course schedules, student enrollment, and faculty assignments. The database should also track student grades and maintain academic records. Convert database schema of university database management system into suitable Normal form(upto third normal form)

A. Objective:

- E-R diagrams provide a high-level view of the database's purpose and requirements, ER diagrams provide a detailed view of the database's structure and relationships. Both are essential tools in database design and development
- To reduce data redundancy, minimize the possibility of data inconsistencies, and improve the performance of the database. By following the principles of normalization, you can create a database that is more organized, efficient, and easier to maintain over time.

B. Expected Program Outcomes (POs): PO1,PO2,PO3,PO4,PO5,PO6,PO7

C. Expected Skills to be developed based on competency:

- E-R diagramming skills can improve data modeling, problem-solving, communication, analytical, and technical skills, all of which are essential in the field of database design and management
- Ability to analyze data and identify relationships between different data points.
- Understanding of normalization rules and techniques for organizing data in a database.
- Proficiency in designing and implementing a normalized database schema to ensure data consistency and eliminate redundancies.

D. Expected Course Outcomes(Cos)

Apply various Normalization techniques

E. Practical Outcome(Pro)

- E-R diagram used in data modeling, database design, communication, maintenance, and optimization.
- Normalization can improve the efficiency, consistency, flexibility, and maintenance of a database.

F. Expected Affective domain Outcome (ADos)

- Follow safety practices.
- Practice good housekeeping.
- Demonstrate working as a leader/a team member.
- Maintain tools and equipment
- Follow ethical practices.

G. Prerequisite Theory:

- The Entity Relational Model is a model for identifying entities to be represented in the database and representation of how those entities are related. The ER data model specifies enterprise schema that represents the overall logical structure of a database graphically.
- The Entity Relationship Diagram explains the relationship among the entities present in the database. ER models are used to model real-world objects like a person, a car,

or a company and the relation between these real-world objects. In short, ER Diagram is the structural format of the database.

- ER Model is used to model the logical view of the system from a data perspective which consists of these symbols:

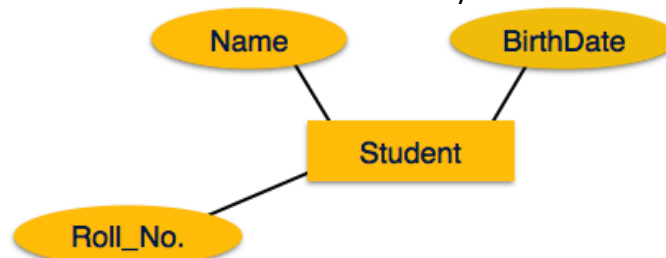
Entity

Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

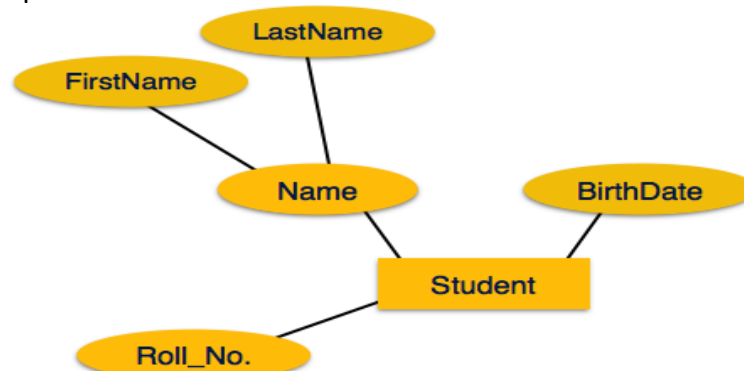


Attributes

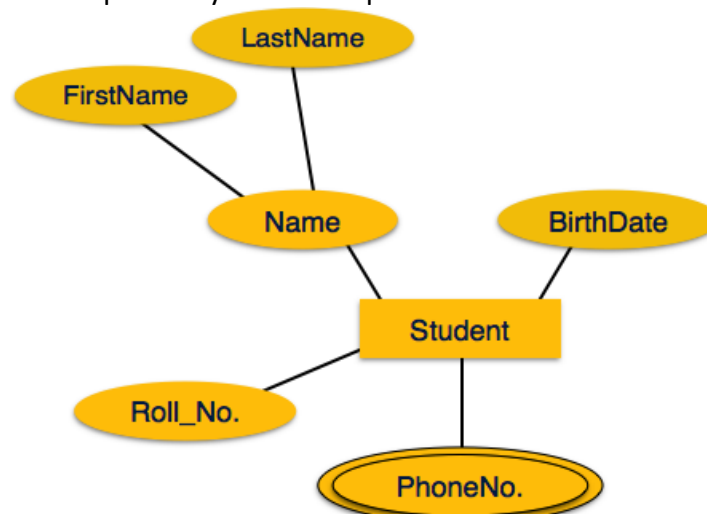
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



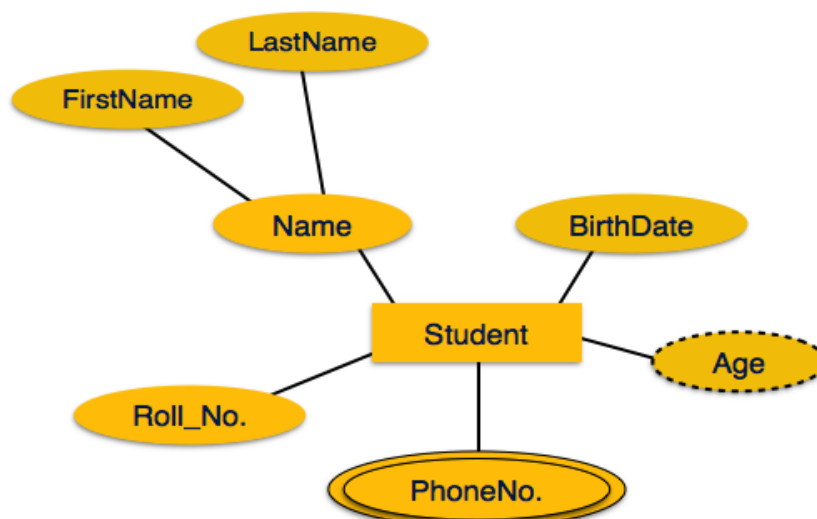
If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.



Multivalued attributes are depicted by double ellipse.



Derived attributes are depicted by dashed ellipse.



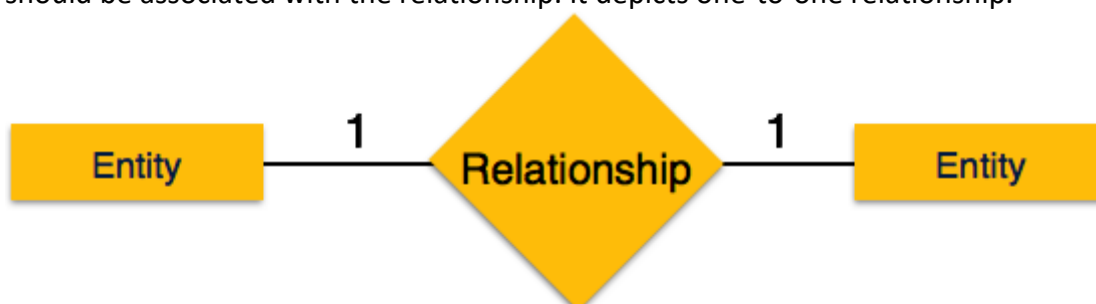
Relationship

Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.

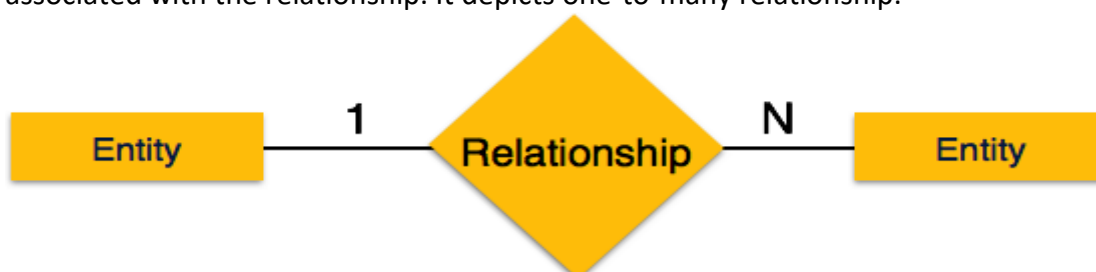
Binary Relationship and Cardinality

A relationship where two entities are participating is called a **binary relationship**. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.

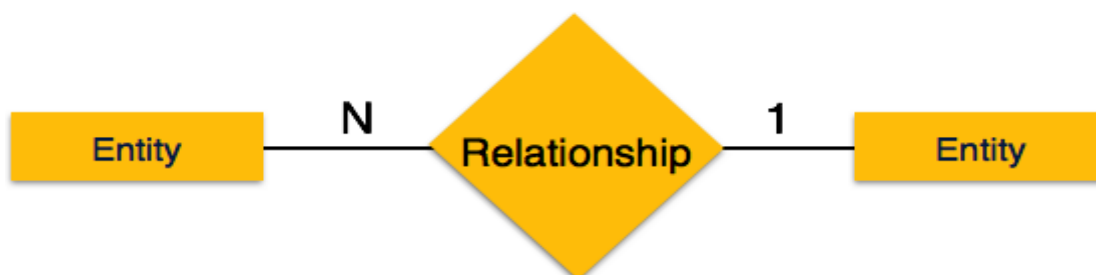
- **One-to-one** – When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.



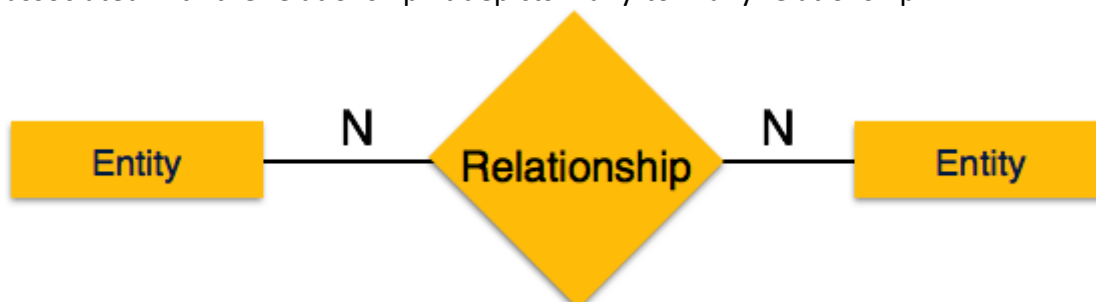
- **One-to-many** – When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many relationship.



- **Many-to-one** – When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship. It depicts many-to-one relationship.

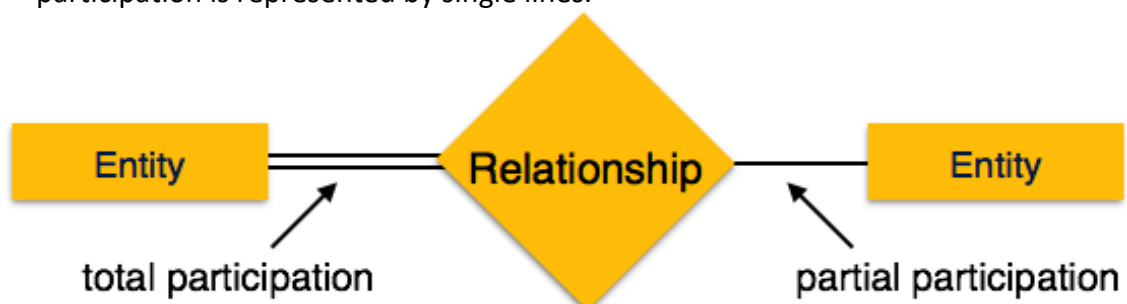


- **Many-to-many** – The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.



Participation Constraints

- **Total Participation** – Each entity is involved in the relationship. Total participation is represented by double lines.
- **Partial participation** – Not all entities are involved in the relationship. Partial participation is represented by single lines.



Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

Data modification anomalies can be categorized into three types:

- **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.

- **Updatation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Types of Normal Forms:

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.

H. Resources/Equipment Required

Computer or laptop with a database management system (DBMS) software installed, such as Oracle

I. Safety and necessary Precautions followed

Shutdown computer system properly once the Lab hours are finished

J. Solution

