Practical 1: Create tables and Insert records

Create following tables with given attributes and Insert ten records to each tables.

Student (No, Name, Address, Ph_no, DOB, Marks) Course (No, Name, Intake)

Subjects (Code, Name, Credit, Textbook_name)

- **Create command:** Create command defines each column of table uniquely. Each column has a minimum of three attributes: name, datatype, size. Each column definition is separated from other by a comma.

  To create new table:

  CREATE TABLE tablename (columnname1 datatype(size), column_name2 datatype(size), ...);

  To Create new table with Primary key:

  CREATE TABLE tablename (columnname1 datatype(size) PRIMARY KEY, column_name2 datatype(size), ...);

  To create new with foreign key:

  CREATE TABLE tablename (columnname1 REFERENCES table2name (columnname), column_name2 datatype(size), ...);

- **Insert command:** It is used for inserting data into tables.

  Insert data into all columns:

  INSERT INTO tablename VALUES (value1, value2, …);

  Insert data into specific columns:

  INSERT INTO tablename (columnname1, columnname2, …) VALUES (value1, value2, …);

➢ Student table:

Create table:

Insert records:

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

Display table:

➢ Course table:

Create table:

Insert records:

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

Display table:

➢ Subject table:

Create table:

Insert records:

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

Display table:

Practical 2: Implement integrity constraints with given tables and insert given values

Create following tables and insert given values:

1. SUPPLIER

SUPPLIER_NO	NAME	ADDRESS
1001	MICHAEL	BASILDON
1002	RINGWORLD	GERMANY
1003	BABYLON	LONDON
1004	JOHN	BASILDON
1005	SMITH	GERMANY
1006	PETER	LONDON

2. PRODUCT

PRODUCT_NO	DESCRIPTION	PRICE	SUPPLIER_NO	MARKETING_REP_NO	SUPPLY_DEPOT_NO
121	REDUCER	1200	1005	5	6
122	PLATE	1500	1004	3	1
123	HANDLE	700	1003	2	4
124	WIDGET REMOVER	900	1005	4	2
136	SIZE WIDGET	1000	1001	1	5
137	SIZE WIDGET	15000	1002	2	16

3. SALESREP

REP_NO	NAME
1	MIKE
2	FRED
3	ALI
4	SAM
5	BILL ADAMS
6	SAM

4. DEPOT

DEPOT_NO	LOCATION	ADDRESS	REP_NO
1	NORTH	UK	1
2	SOUTH	USA	2
3	LONDON WEST	USA	3
4	EAST	NZ	4
5	WALES	UK	5
6	NORTH	KENYA	6
16	SOUTH	UK	2

5. CUSTOMER

CUSTOMER_NO	NAME	ADDRESS	DEPOT_NO	CREDIT_LIMIT
10	GARRY SMITH	BRIXTON	6	1000
20	PATEL	GRANGE	1	4000
30	DRAKE	BRIXTON	4	7000
40	BOB SMITH	LONDON	2	10000
50	JAMES	GRANGE	3	5000
60	NORTON	SAN FRANSISCO	5	17000
70	JOHN MICHAEL	EUROPE	16	8000

6. CORDER

CORDER_NO	CUSTOMER_NO	DATE_PLACED	DATE_DELIVERED
200	20	01-JAN-1993	04-JAN-1993
201	40	17-JAN-1993	20-JAN-1993
202	20	01-JAN-1993	04-JAN-1993
203	30	02-FEB-1995	05-FEB-1995
204	10	13-MAR-1996	16-MAR-1996
205	70	31-JAN-1993	03-FEB-1993
206	40	01-JAN-1993	04-JAN-1993
207	20	02-AUG-1994	05-AUG-1994

7. OLINE

CORDER_NO	PRODUCT_NO	QUANTITY
200	120	5
201	121	10
202	120	5
203	122	20
204	136	30
205	124	15
206	136	30

8. STOCK

DEPOT_NO	PRODUCT_NO	QUANTITY	RACK	BIN_NO
1	120	50	1	1
2	137	100	10	2
3	136	40	2	3
4	120	60	7	1
5	121	90	5	4
6	124	120	4	7
16	122	80	10	8

➢ Supplier:

Create table:

Insert records:

1.

2.

3.

4.

5.

6.

Display table:

➢ SalesRep:

Create table:

Insert records:

1.

2.

3.

4.

5.

6.

Display table:

➢ Depot table:

Create table:

Insert records:

1.

2.

3.

4.

5.

6.

Display table:

➢ Product:

Create table:

Insert records:

1.

2.

3.

4.

5.

6.

Display table:

➢ Customer:

Create table:

Insert records:

1.

2.

3.

4.

5.

6.

7.

Display table:

➢ Corder:

Create table:

Insert records:

1.

2.

3.

4.

5.

6.

7.

8.

Display table:

➢ Oline:

Create table:

Insert records:

1.

2.

3.

4.

5.

6.

7.

Display table:

➢ Stock:

Create table:

Insert records:

1.

2.

3.

4.

5.

6.

7.

Display table:

Practical 3: Execute DDL and DML queries.

- **To create table from another table available in database:**

  CREATE TABLE new_tablename (columnname1, columnname2, …) AS SELECT (columnname1, columnname2, … from existing_tablename);

- **Desc command:** DESC command use for describe the list of column definitions for specified table.

  DESC tablename;

- **Alter command**: ALTER command is used to add, delete, or modify columns in an existing table.

  To ADD column in table:

  ALTER TABLE table_name ADD ColumnName datatype(size);

  To change datatype of column in table:

  ALTER TABLE tablename MODIFY ColumnName datatype;

  To Drop/Delete column from table:

  ALTER TABLE tablename DROP COLUMN ColumnName;

- **Truncate command:** The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

  To remove all records from table:

  TRUNCATE TABLE tablename;

- **Drop command:** The DROP TABLE statement is used to drop an existing table in a database.

  To delete entire table:

  DROP TABLE tablename;

- **Rename command:** The rename command is used to change the name of an existing database object to a new name.

  RENAME TABLE current_tablename TO new_tablename;

- **Update command:** The UPDATE statement is used to modify the existing records in a table.

  Update/insert data into whole column:

  UPDATE tablename SET column1=value1;

  Update data into specific row:

  UPDATE tablename SET column1=value1,.. WHERE condition;

- **Delete command:** The DELETE statement is used to delete existing records in a table.

  Delete all records from table:

  DELETE FROM tablename;

  Delete specific records from table:

  DELETE FROM tablename WHERE condition;

1. Change the price of “Plate” from 1500 to 2000.

2. Modify credit limit to 8000 for those customers who lives in ‘Grange’.

3. Change the size of customer address to 30.

4. Create a table Cust1 with the attributes Customer_no, Name, Address.

5. Add new field email id in Cust1 table.

6. Display the structure of Cust1 table.

7. Display the content of Cust1 table.

8. Delete details of customer no 20 from Cust1 table.

9. Delete email id field from Cust1 table.

10. Delete all the data rows from Cust1 and look at the contents again.

11. Delete the table Cust1 and then try to look at its contents again.

Practical 4: Execute DQL Queries

Select command: It is used to fetch data from a database. It is used to extract only those records that fulfill a specified condition.

To display all data from table:

SELECT * FROM tablename;

To display specific column from table:

SELECT columnname1, columnname2,… FROM tablename;

To display specific rows from table:

SELECT columnname1, columnname2,… FROM tablename WHERE condition;

To display unique values from column:

SELECT DISTINCT(columnname) FROM tablename;

To display column name with Alias:

SELECT column_name AS alias_name FROM table_name;

To display table name with Alias:

SELECT column_name(s) FROM table_name AS alias_name;

1. Display current date and time.

2. Display all tables created by user.

3. List the customer numbers (customer_no) and names (name) of all customers.

4. List all details of the product with a product number (product_no) of 121 and 136.

5. List all details of depots with rep 5 as their rep (rep_no).

6. List the product number (product_no) and description only of all products from supplier number 1005.

7. List all details for all customers with names starting from 'sm' followed by 1 character followed by 't' followed by anything.

8. List all details for all orders with date_placed from 01-Jan-1993 to 31-Jan-1993. (Use between).

9. List the sales rep number, depot number and address for depots located at NORTH and address is UK.

10. List each product description and its price increased by 10%.

Practical 5: Queries for numerical functions, string functions and date conversion functions

Demonstrate the use of following functions:

a) Aggregate/ Group functions:

AVG(): It is used to find average of a given column's values.

```
select AVG(column_name) from tablename;
```

SUM(): It is used to perform summation operation on given column's values.

```
select SUM(column_name) from tablename;
```

MIN(): It is used to find minimum number from given column's values.

```
select MIN(column_name) from tablename;
```

MAX(): It is used to find maximum number from given column's values.

```
select MAX(column_name) from tablename;
```

COUNT(): It is used to count total number of rows in given column.

```
select COUNT(column_name) from tablename;
```

COUNT(*): It will count total number of rows in a given table.

```
select COUNT(*) from tablename;
```

AVG():

Output:

SUM():

Output:

MIN():

Output:

MAX():

Output:

COUNT():

Output:

COUNT(*):

Output:

1. Give the total number of items (quantity) in stock in all depots.

2. Give the total number of items (order line quantity) which have been ordered with corder_no 200.

b) Numeic functions:

ABS(): It will convert given value in positive number.

Select ABS(number) from dual;

POWER(): It returns the value of a number raised to the power of another number.

select POWER(x, y) from dual;

SQRT(): It returns the square root of a number. select SQRT(number) from number;

Select SQRT(number) from dual;

ROUND(): It rounds a number to a specified number of decimal places.

select ROUND(number, decimals) from dual;

MOD(): It returns the remainder of a number divided by another number.

select MOD(x, y) from dual;

GREATEST(): It is used to find greater value from given values.

select GREATEST(value1, value2, ...,valueN) from dual;

LEAST(): It is used to find lesser value from given values.

select LEAST(value1, value2, ...,valueN) from dual;

ABS():

Output:

POWER():

Output:

SQRT():

Output:

ROUND():

Output:

MOD():

Output:

GREATEST():

1.

Output:

2.

Output:

LEAST():

1.
Output:
2.
Output:

c) Character functions:

LENGTH(): It returns the length of a string.

Select LENGTH(string) from dual;

LOWER(): It converts a string to lower-case.

Select LOWER(string) from dual;

UPPER(): It converts a string to upper-case.

Select UPPER(string) from dual;

INITCAP(): It will make all the first letters of given words into capital letter.

Select INITCAP(string) from dual;

LPAD(): The LPAD() function left-pads a string with another string, to a certain length.

Select LPAD(*string*, *length*, *lpad_string*) from dual;

RPAD(): The LPAD() function right-pads string with another string, to a certain length.

Select RPAD(*string*, *length*, *rpad_string*) from dual;

TRIM(): It is basically used to remove extra spaces from a given string. But by using trim function we can also remove any character from starting or ending of a string.

Select TRIM(LEADING/TRAILING/BOTH 'character_to_trim' FROM 'string') from dual;

TRANSLATE(): It will take three arguments as a parameter. And replace 2nd arguments values with 3rd arguments values in a string given as 1st argument.

Select TRANSLATE(string, 'characters_to_replace', 'replacement_characters') from dual;

LENGTH():

Output:

LOWER():

Output:

UPPER():

Output:

INITCAP():

Output:

LPAD():

Output:

RPAD():

Output:

TRANSLATE():

Output:

TRIM():

1.

Output:

2.

Output:

3.

Output:

d) Date conversion functions:

TO_CHAR(number) & TO_CHAR(date): TO_CHAR() function converts a DATE or number in a specified string format.

Select TO_CHAR(date/ number, string_format) from dual;

TO_DATE(): TO_DATE function converts a string to a date.

Select TO_DATE (string1 , format_mask)

LAST_DAY(): This function extracts the last day of the month for a given date.

Select LAST_DAY(date) from dual;

MONTHS_BETWEEN(): It is used to get the number of months between given dates.

Select MONTHS_BETWEEN(date1,date2) from dual;

TO_CHAR(number):

1.
Output:
2.
Output:

TO_CHAR(date):

1.
Output:
2.
Output:

TO_DATE():

1.

Output:

2.

Output:

LAST_DAY():

Output:

MONTHS_BETWEEN():

1.

Output:

2.

Output:

Practical 6: Execute join queries

Inner join: The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e., value of the common field will be same.

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

Outer join:

Left outer join: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain *null*.

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

Right outer join: This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain *null*.

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

Full outer join: FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain *NULL* values

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

1. Give a list of depot locations paired with the name of sales rep who covers that depot.

2. List the customer's name and the depot location for the depot delivering to that customer for all customers who receive deliveries from depots looked after by sales rep number (rep_no) 3.

3. List the sales rep number (rep_no) and depot location and address for depots looked after by the sales rep whose name is mike.

4. For all order lines (oline) for all orders (corder) for customers whose name is patel, list the customer address, the date_placed, the product_no and the quantity.

5. List the names of all customers who receive deliveries from depots which are looked after by the sales rep whose name is fred.

6. List the customer name, order date_placed, order line quantity and product description for each order line (with its linked, order, customer and product rows) for customers who receive deliveries from depot number 2.

7. List supplier names paired with the names of the sales reps who market products supplied by that supplier.

8. List supplier names paired with the names of sales reps who look after the depots where products from that supplier are delivered.

9. List the names of all customers who have ordered products which are marketed by the sales rep whose name is Ali.

10. List the names of all customers who are delivered to by the depot which delivers to the customer whose name is drake.

11. List all order lines for the customer with customer_no 20 giving the product description, the order line quantity and the value of the order line. (i.e. the order line quantity * the price from the linked product row)

12. List the locations and addresses of all depots which do not stock product number 122. (i.e. where there is no stock row for that product for the depot)

13. Set up a query which lists the names of all customers who have placed an order with the order number (corder_no) of the order merged with the names of all customers who have never placed an order (shown once, with the order number attribute null(i.e. an outer join.

Practical 7: Execute sub queries

Subqueries: Subqueries are most frequently used with the SELECT statement.

SELECT column1name, column2name FROM table1name WHERE column_name OPERATOR (SELECT column1name, column2name FROM table1name WHERE condition);

IN Operator: The IN operator allows you to specify multiple values in a WHERE clause. The IN operator is a shorthand for multiple OR conditions.

SELECT column_name(s) FROM tablename
WHERE column_name IN (value1, value2, ...);

or

SELECT column_name(s) FROM tablename
WHERE column_name IN (SELECT STATEMENT);

NOT IN Operator: The IN operator makes sure that the expression proceeded does not have any of the values present in the arguments.

SELECT column_name(s) FROM tablename
WHERE column_name NOT IN (value1, value2, ...);

or

SELECT column_name(s) FROM tablename
WHERE column_name NOT IN (SELECT STATEMENT);

ANY Operator: The ANY operator is used with a WHERE or HAVING clause. The ANY operator returns true if any of the subquery values meet the condition.

SELECT column_name(s) FROM tablename WHERE column_name operator
ANY
(SELECT column_name FROM tablename WHERE condition);

ALL Operator: The ANY operator are used with a WHERE or HAVING clause. The ALL operator returns true if all of the subquery values meet the condition.

SELECT column_name(s) FROM table_name WHERE column_name operator
ALL
(SELECT column_name FROM table_name WHERE condition);

EXISTS Operator: The EXISTS operator is used to test for the existence of any record in a subquery. The EXISTS operator returns true if the sub query returns one or more records.

```
SELECT column_name(s) FROM table_name WHERE
EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

NOT EXISTS Operator: The NOT EXISTS will check the Sub query for rows existence, and if there are no rows then it will return TRUE, otherwise FALSE.

```
SELECT column_name(s) FROM table_name WHERE
NOT EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

1. List the description of product which are supplied by supplier SMITH using IN.

2. List all product no which are not ordered by the customer having same CORDER_NO as the CUSTOMER_NO 20.

3. List the locations and addresses of all depots which stock any product which is supplied to the depot whose location is wales.

4. List the customer_no, date_placed and date_delivered for all orders which contain order lines for the product with product_no 137 using existential quantification (ie the where exists condition).

5. List the depots which do not stock any product supplied by the supplier whose name is ringworld.

6. List the locations and addresses of all depots which stock all products supplied by the supplier babylon.

7. Give product number which has maximum quantity stock at any depot.

8. Give customer address which has minimum credit limit.

Practical 8: Execute Group by and Having clause

GROUP BY CLAUSE: The GROUP BY statement groups rows that have the same values into summary rows. The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

SELECT column_name from tablename where condition GROUP BY column_name;

HAVING CLAUSE: The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

SELECT column_name FROM tablename WHERE condition GROUP BY column_name HAVING condition*;*

ORDER BY: The ORDER BY keyword is used to sort the result-set in ascending or descending order. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

To arrange in ascending order:

SELECT column1, column2, ... FROM tablename ORDER BY column1;

To arrange in descending order:

SELECT column1, column2, ... FROM tablename ORDER BY column1 DESC;

1. List the number of different products supplied by each supplier_no.

2. List the name of each supplier with the location of each depot and the number of products supplied by that supplier and stocked at that depot.

3. List the depot_no's of all depots where the average credit_limit for all the customers receiving deliveries from the depot is > 20,000.

4. List total no of quantity and product number ordered by customer.

5. List supplier no who has supplied products whose total price is < 1000.

6. Give total number of customers who has ordered product on same date.

7. List sum of quantity stocked at each rack.

8. Display total no of customers who has received product from same location.

9. Display customer name who has ordered on same date.

10. List product descriptions in reverse alphabetic order.

11. List all product descriptions with the product's supplie name, sorted by product description within supplier name(i.e. all products for a supplier listed together in alphabetic order).

Practical 9: Execute case statements, set operations and view queries

Case statement: The CASE statement goes through conditions and returns a value when the first condition is met. So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause. If there is no ELSE part and no conditions are true, it returns NULL.

- **Simple case:**

```
SELECT [<column_names>,]
        CASE <column_name> WHEN <value> THEN 'statement'
        END [AS <alias_name>]
FROM <table_name>;
```

- **Simple If-Else:**

```
SELECT [<column_names>,]
        CASE WHEN <expression> THEN 'statement'
        ELSE 'statement'
        END [AS <alias_name>]
FROM <table_name>;
```

- **If-Else Ladder:**

```
SELECT [<column_names>,]
        CASE WHEN <expression> THEN 'statement'
        WHEN <expression> THEN 'statement' ELSE 'statement'
        END [AS <alias_name>]
FROM <table_name>;
```

- **Nested If-Else:**

```
SELECT [<column_names>,]
        CASE WHEN <expression> THEN
                CASE WHEN <expression> THEN 'stmt'
                        ELSE 'stmt'
                END
            ELSE 'stmt'
        END [AS <alias_name>]
FROM <table_name>;
```

Set operations: Set operators combine the results of two component queries into a single result.

UNION: It will take values of selected column of first table and values of selected column from second table and merge values of both the tables and eliminate duplicate values.

```
select column1, column2, … , columnN from table1
UNION
select column1, column2, … , columnN from table2;
```

UNION ALL: All rows selected by either query, including all duplicates

```
select column1, column2, … , columnN from table1
UNION ALL
select column1, column2, … , columnN from table2;
```

INTERSECT: It will take common values of selected column of first table and selected column from second table and generate answer after eliminating duplicate values.

```
select column1, column2, … , columnN from table1
INTERSECT
select column1, column2, … , columnN from table2;
```

MINUS: It will fetch unique values from table1 which is not there in table2.

```
select column1, column2, … , columnN from table1
MINUS
select column1, column2, … , columnN from table2;
```

View: View is the logical table created from existing one or more tables. View is virtual table and it is not presented physically in memory.

Creating view:

CREATE VIEW <view_name> AS SELECT <column_names> FROM <table_name> WHERE <condition> GROUP BY <criteria> HAVING <predicates> ;

Retrieving data from view:

SELECT <column_name> FROM <view_name> WHERE <condition>;

Destroying view:

DROP VIEW <view_name>;

1. Implement “IF” Condition in Query
    a. Put if condition on “price” attribute (IF Else)

    b. Try nested IF on “price” attribute.

    c. Display Price and quantity and there rating with “High”, “Medium” and “Low”.

2. Create a view VProduct of product‟s id, description and price.

3. Create a view of Vorder to get orders (order_id, product_id, description, customer_name, quantity) placed by customer who belongs to “BRIXTON”.

4. Insert a new row in VProduct 135, “Sofa” and 35000.

5. Update product’s quantity to 25 which is brought by customer “DRAKE‟ in Vorder.

6. Delete details of product id 121 from VProduct.

7. Delete view VProduct.

8. Display name of all customers and all suppliers with their id by using union operator.

9. List product which are not bought by any customer using minus operator.

10. Give the name of suppliers who is also customer.

Practical 10: Execute PL/SQL blocks

PL/SQL block: PL/SQL is a block-structured language whose code is organized into blocks. A PL/SQL block consists of three sections: declaration, executable, and exception-handling sections. In a block, the executable section is mandatory while the declaration and exception-handling sections are optional.

PL/SQL block syntax:

```
DECLARE --optional
        <declarations>
BEGIN  --mandatory
        <executable statements.  At least one executable statement is
        mandatory>
EXCEPTION --optional
        <exception handles>
END;  --mandatory
```

Control Structure:

- **Conditional Control:**

```
IF <condition> THEN <ACTION>
ELSIF <condition> THEN <ACTION>
ELSE <ACTION>
END IF;
```

- **Iterative Control:**

  Simple Loop

```
LOOP
   <sequence of statements>
   EXIT WHEN <condition>
END LOOP;
```

WHILE Loop:

```
WHILE <condition>
LOOP
        <ACTION>
END LOOP;
```

FOR Loop

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

- **Sequential Control:**

GOTO Statement

```
GOTO <<code_block name>>
```

1. Write a PL/SQL block to swap two numbers.

2. Write a PL/SQL block to take personal details (Enrolment Number, Name, Phone Number and Qualification) from user and display it.

3. Write a PL/SQL block to find given year is leap year or not.

4. Write a PL/SQL block to find factorial of a given number using for loop.

5. Write a PL/SQL block to reverse a given number using while loop.

6. Write a PL/SQL block to check whether given number is prime or not.

Practical 11: Execute cursor

Cursor: Cursor is a location in the memory where the SQL statement is processed. That is whenever we send an SQL statement; the system allocates some memory by giving a name. After processing the statement, it automatically cleans that memory.

There are four steps, which are processed by the system whenever we send an SQL statement they are

1. Creating cursor: This step allocates memory
2. Opening cursor: In this step process the SQL statement
3. Fetching cursor: The values satisfied by the SQL statement are fetched from the table into cursor row by row
4. Closing cursor: This statement closes the memory allocated for the cursor There are two types of cursors –

- Implicit cursors
- Explicit cursors

Implicit Cursors: Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it. Whenever a DML statement INSERT, UPDATE and DELETE is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

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

The syntax for creating an explicit cursor is:

```
CURSOR cursor_name IS select_statement;
```

Working with an explicit cursor involves four steps:

a. Declaring the cursor for initializing in the memory
b. Opening the cursor for allocating memory
c. Fetching the cursor for retrieving data
d. Closing the cursor to release allocated memory

a. Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement.

Syntax:

CURSOR CursorName IS SELECT statement;

b. Opening the Cursor

Opening the cursor allocates memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it.

Syntax:

OPEN CustomerName;

c. Fetching the Cursor

Fetching the cursor involves accessing one row at a time.

Syntax:

FETCH CursorName INTO Variable1, Variable2, …;

d. Closing the Cursor

Closing the cursor means releasing the allocated memory.

Syntax:

CLOSE CursorName;

1. Retrieving the records from any table and displaying them one by one using cursors.

2. Create a cursor to increase salary. The salary increase is 20% for the employees making less than $90,000 and 12% for the employees making $90,000 or more.

3. Display Customer name and Deport location using Implicit cursor.

Practical 12: Execute procedures

Procedure: A Procedure or Function is a logically grouped set of SQL and PL/SQL statements that performs a specific task. A stored procedure or stored function is a named PL/SQL code block that has been compiled and stored in one of the Oracle engine's system tables.

While creating a procedure or function, Oracle engine performs following steps:

- Compiles the procedure or function.
- Stores procedure or function in database.

Syntax:

```
CREATE OR REPLACE PROCEDURE [Schema.] <ProcedureName>
                    (<Argument> {IN, OUT, IN OUT} <data_type>,…)
  {IS,AS}
   <variable> <datatype>;
       BEGIN
            <PL/SQL body>
       EXCEPTION
            <Exception PL/SQL block>
       END;
```

1. Create one Stored Procedure to Insert, Delete and Update into one table.

Insert:

Output:

Update:

Output:

Delete:

Output:

2. Create one Stored Procedure to increment salary with salary range limit.

Update:

3. Create one Stored Procedure to find largest price with that product name.

Practical 13: Execute Triggers

Trigger: In a DBMS, a *trigger* is a SQL procedure that initiates an action (i.e., *fires* an action) when an event (INSERT, DELETE or UPDATE) occurs. A trigger cannot be called or executed; the DBMS automatically fires the trigger as a result of a data modification to the associated table. Triggers are used to maintain the referential integrity of data by changing the data in a systematic fashion.

PL/SQL Triggers Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
   BEFORE | AFTER
   [INSERT, UPDATE, DELETE [COLUMN NAME..]
   ON table_name
   Referencing [ OLD AS OLD | NEW AS NEW ]
   FOR EACH ROW | FOR EACH STATEMENT [ WHEN Condition ]
DECLARE
     [declaration_section
     variable declarations;
     constant declarations;]
BEGIN
  [executable_section ]
EXCEPTION
  [exception_section
     PL/SQL Exception block]
END;
```

1. Trigger will display the salary difference between the old values and new values.

2. Stop the transaction if the quantity entered (insert) by the user exceeds 1000.

3. Maintain Log Table with use of trigger.

Checking Trigger: