DBMS PROJECT REPORT.

ONLINE SHOPPING MANAGEMENT SYSTEM.

.>Declaration

We, the undersigned, solemnly declare that the project report is based on our own work carried out during the course of our study under the supervision of Dr. Mamta Juneja. We assert the statements made and conclusions drawn are an outcome of our research work. I further certify that:-

I. The work contained in the report is original and has been done by us under the general supervision of our supervisor.

II. The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or any other University of India or abroad.

III. We have followed the guidelines provided by the university in writing the report. IV. Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and given their details in the references.

.>TEAM MEMBERS

MAYANK SRIVASTAVA (69) **NAMIT GOYAL (76)** **SHREYAS MISHRA (104)**

ACKNOWLEDGEMENT

*We would like to express our special thanks of gratitude to our teacher **Dr. Mamta Juneja** for giving us the golden opportunity to do this wonderful project on the "**Online shopping Management System"**, which also helped us in doing a lot of Research and we came to know about so many new things. We are really thankful to them.*

Secondly we would also like to thank our friends who helped us a lot in finalizing this project within the limited time frame.

We are overwhelmed in all humbleness and gratefulness to acknowledge our depth to all those who have helped us to put these ideas, well above the level of simplicity and into something concrete.

We would like to express our special thanks of gratitude to our teacher who gave us the golden opportunity to do this wonderful project on the topic "**Online Shopping Management System**" which also helped us in doing a lot of Research and we came to know about so many new things, we got a chance to learn SQL and Microsoft SQL Server to make our project successful. We are really thankful to them.

Any attempt at any level can't be satisfactorily completed without the support and guidance of our families and friends.

Thanking you

The team,

Mayank Srvastava (UE193069)

Namit Goyal (UE193076)

Shreyas Mishra (UE193104)

TABLE OF CONTENTS

6. a> Topic 1 : ER Diagram

b> Topic 2 : Relational Model

c> Topic 3 : Normalisation.

Screenshots

Conclusion and Recommendations

References

List of Figures

a)Arithmetic

b)Logical

c)Relational

B. 2. Functions:

a)String functions

b)Numeric functions

c)Aggregate functions

d)Date functions

4. Order By, Group by and Having statements
5. Subqueries

A. Nested queries (Single row, Multi row)

B. Correlated queries

6. Joins:

A. Cross Join

B. Natural Join

C. Inner Join

D. Outer Join(left ,right and full)

7. Views

A. Introduction

B. Types: Materialised and View Resolution

C. Features/Advantages

D. CREATE, DROP VIEW(using single table, Multiple Tables

E. INSERT UPDATE DELETE IN VIEWS

8. Indexes

A. Introduction

B. Types: Unique, Duplicate, simple, Composite

C. CREATE, DROP INDEX (on single column, multiple columns)

9. Sequence

A. Introduction

B. CREATE, DROP SEQUENCE

C. USING SEQUENCE TO INSERT VALUES IN COLUMN OF A

TABLE(ex employee id, student rollno etc)

D. ALTER SEQUENCE

10. Introduction to PL/SQL

A. Syntax of PL/SQL

B. Control structures in PL/SQL and their syntax

(Sequential control, Conditional , Iterative)

P1: To check if a year is a leap or not .

P2: To find area and circumference of a circle using a case statement.

P3: To check whether a prime number or not using while loop and for loop

P4: To generate fibonacci series

P5: To generate factorial of a given number

P6: To Check Palindrome

P7: To create a dummy calculator taking two numbers as input as performing selected operation(Addition, subtraction, multiply, etc) and displaying output

11. Subprograms in PL/SQL:

A. Introduction to Procedures

Syntax of Procedures

Write procedure for P1 to P7

B. Introduction to Functions

Syntax of Functions

Write function for P1 to P7

12. Cursors in PL/SQL

A. Introduction

B. Advantages and Disadvantages of cursor

C. Types of cursors:Implicit and Explicit Cursors

Program:To count the number of records using implicit cursor.

Program:To display records of 5 least paid employees using explicit cursor.

Program:To display records of 5 highest paid employees using

explicit cursor.

13. Triggers in PL/SQL
    A. Introduction to Triggers
    B. Advantages and disadvantages of triggers
    C. Syntax of Triggers
    D. Types of Triggers
        Before event Triggers
        After event Triggers
        Row Trigger
        Statement trigger
        Hybrid Trigger
    E. Write programs to demonstrate D

14. Exception Handling
    A. System defined Exception handling
    B. User defined Exception handling
    C. Write Programs to demonstrate A and B

15. Package in PL/SQL
    A. Introduction to Package
    B. Package Specification
    C. Package Body
    D. WAP to Implement Package for your

project 16. Miscellaneous queries (REFER

ANNEXURE

1)https://discord.com/channels/78064727405703989

3/780647274057039896/800679101631299585

Introduction

1.Problem Statement

Develop the online database management system for a Shop.

2. Problem Objective

The purpose of the Shopping Management System is to act as an interface to manipulate the database records of various customers and suppliers related information such as product details, product availability,order details, shipment details and various other tasks performed by them and store them as any new record that has relation to the shop in any way.

Our goal is to create a database and query system which can handle manipulation and retrieval of data as per the requirement of the user.

3. Academic Objective

To learn about SQL, working on databases,oracle apex online server.

3. Technology Platform

· SQL

· Oracle Apex online workplace

Body of Report

1.INTRODUCTION TO SQL

SQL is a standard language for accessing and manipulating databases.

SQL is used to:

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database

- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

A. SQL Commands

These SQL commands are mainly categorized into four categories as:

1.Data Definition Language (DDL): DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

Examples of DDL commands:

CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).

Syntax:-

CREATE DATABASE database_name;

CREATE TABLE table_name(column1 datatype, column2 datatype,….);

Example:-

```
CREATE TABLE Order_Details (
   OrderID varchar(11) NOT NULL,
  ProductID varchar(11) default NULL,
  ItemQuantity int default NULL,
   PRIMARY KEY (OrderID));
```

DROP – is used to delete objects from the database.

Syntax:-

DROP DATABASE database_name;

DROP TABLE table_name;

Example:- DROP products

ALTER-is used to alter the structure of the database i.e. to add a column, delete a column, to modify datatype of database.

Syntax:-

```
ALTER TABLE table_name ADD column_name datatype;
ALTER TABLE table_name DROP COLUMN column_name;
ALTER TABLE table_name ALTER COLUMN column_name
datatype;
```

Example:- ALTER TABLE Products RENAME TO Products details;

```
ALTER TABLE Products
ADD FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID);
```

TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.

Syntax:-

```
TRUNCATE TABLE table_name;
```

Example:- TRUNCATE PRODUCTS

RENAME –is used to rename an object (table,column) existing in the database.

Syntax:-

```
RENAME TABLE old_table_name TO new_table_name;
```

Example:- RENAME Products details TO Products;

2.Data Manipulation Language(DML):- DML commands are used to modify the database. It is responsible for all forms of changes in the database. The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rolled back.

Example of DML commands:

INSERT – The INSERT statement is a SQL query. It is used to insert

data into the row of a table.

Syntax:-

INSERT INTO TABLE_NAME (col1, col2, col3,.... col N) VALUES

(value1, value2, value3, ... valueN);

Example:-

```
INSERT INTO Products (ProductID,ProductName,SupplierID,Category,UnitPrice,Color,QuantityOnHand) VALUES ('748130-5105','Lotion','12241497023','Hair','$6.83','yellow',109);
INSERT INTO Products (ProductID,ProductName,SupplierID,Category,UnitPrice,Color,QuantityOnHand) VALUES ('206908-3232','Remover','31704489846','Lip','$29.58','green',898);
INSERT INTO Products (ProductID,ProductName,SupplierID,Category,UnitPrice,Color,QuantityOnHand) VALUES ('332338-2345','Beauty','68659281779','Face','$56.17','red',271);
INSERT INTO Products (ProductID,ProductName,SupplierID,Category,UnitPrice,Color,QuantityOnHand) VALUES ('096419-4823','Beauty','30918186609','Cheek','$26.70','red',551);
INSERT INTO Products (ProductID,ProductName,SupplierID,Category,UnitPrice,Color,QuantityOnHand) VALUES ('585230-0077','Sets','39656653836','Cheek','$5.24','red',514);
INSERT INTO Products (ProductID,ProductName,SupplierID,Category,UnitPrice,Color,QuantityOnHand) VALUES ('197710-3876','Lotion','70268860023','Eye, Face','$26.98','green',794);
INSERT INTO Products (ProductID,ProductName,SupplierID,Category,UnitPrice,Color,QuantityOnHand) VALUES ('533169-3365','Shampoo','30751277561','Body, Hair','$84.35','yellow',916);
INSERT INTO Products (ProductID,ProductName,SupplierID,Category,UnitPrice,Color,QuantityOnHand) VALUES ('588681-0034','Moisturizer','39599259968','Lip','$73.53','blue',162);
INSERT INTO Products (ProductID,ProductName,SupplierID,Category,UnitPrice,Color,QuantityOnHand) VALUES ('998757-5991','Sets','43636501698','','$59.44','orange',827);
INSERT INTO Products (ProductID,ProductName,SupplierID,Category,UnitPrice,Color,QuantityOnHand) VALUES ('375163-6428','Primer','12716365607','','$2.72','red',464);
```

UPDATE – This command is used to update or modify the value of a column in the table.

Syntax:-

UPDATE table_name SET [column_name1=value1,...col_nameN=valueN] [WHERE CONDITION];

Example:-

UPDATE Products SET CARDTYPE = ’Discover’ WHERE PAYMENTID = 79922775587;

DELETE – It is used to remove one or more rows from a table.

Syntax:-

DELETE FROM table_name [WHERE condition];

EXAMPLE: DELETE FROM Products WHERE PAYMENTID =

79922775587;

3.Data Control Language(DCL):- DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

Examples of DCL:-

GRANT- gives user's access privileges to the database such as select,create,update, delete etc . This command grants a SELECT permission on Employee table to user1

Syntax:-

```
GRANT SELECT,UPDATE ON table_name TO
User1, User2,…UserN;
```

REVOKE- withdraw user's access privileges given by using the GRANT command.

Syntax:-

```
REVOKE SELECT,UPDATE ON table_name FROM
User1, User2,….UserN;
```

Example:- REVOKE SELECT ON Employee FROM user1;

This command will REVOKE a SELECT privilege on the Employee table from user1.

4.Transaction Control Language:- TCL commands can only be used with DML commands like INSERT, DELETE and UPDATE only.These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Example of TCL Commands:-

COMMIT– Commit command is used to save all the transactions to the database.

Syntax:-.
COMMIT; (After a transaction,if keyword COMMIT is written then all the transaction is saved permanently).

Example:- COMMIT;

· ROLLBACK– Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:- ROLLBACK; (If after a transaction, keyword rollback is executed, the database will be restored to its previous state).

Example:- ROLLBACK TO savepoint_name;

SAVEPOINT– It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:-

SAVEPOINT savepoint_name; (Whenever any savepoint is executed then transaction is saved till that point).

Example:- SAVEPOINT savepoint_name;

B.USE OF CREATE AS, USER_CONSTRAINTS, SPOOL ON

1.CREATE AS:- CREATE TABLE AS statement to create a table from an existing table by copying the existing table's columns.

Syntax:-

CREATE TABLE new_table AS (SELECT * FROM old_table);

EXAMPLE:

```
CREATE TABLE Products (
  ProductID varchar2(11) NOT NULL,
  ProductName varchar2(255) default NULL,
  SupplierID varchar2(11) default NULL,
  Category varchar2(255) default NULL,
  UnitPrice varchar2(100) default NULL,
  Color varchar2(255) default NULL,
  QuantityOnHand int default NULL,
   PRIMARY KEY (ProductID));
```

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
PRODUCTS	PRODUCTID	VARCHAR2	11	-	-	1	-	-	-
	PRODUCTNAME	VARCHAR2	255	-	-	-	✓	NULL	-
	SUPPLIERID	VARCHAR2	11	-	-	-	✓	NULL	-
	CATEGORY	VARCHAR2	255	-	-	-	✓	NULL	-
	UNITPRICE	VARCHAR2	100	-	-	-	✓	NULL	-
	COLOR	VARCHAR2	255	-	-	-	✓	NULL	-
	QUANTITYONHAND	NUMBER	22	-	0	-	✓	NULL	-

2.USER_Constraints:- USER_CONSTRAINTS describes all constraint definitions on tables owned by the current user.

Syntax:-

SELECT CONSTRAINT_NAME, SEARCH_CONDITION AS CONSTRAINT_TYPE FROM USER_CONSTRAINTS [WHERE TABLE_NAME=table_name];

Example:-

```
SELECT * FROM USER_CONSTRAINTS;
```

SELECT OWNER, CONSTRAINT_NAME,
CONSTRAINT_TYPE,SEARCH_CONSTRAINT, TABLE_NAME FROM
USER_CONSTRAINTS;

3.SPOOL ON:- Log the output of a MaxL Shell session to a file. Message logging begins with spool on and ends with spool off.

Example: spool on to '../../../output.txt';

Sends output of MaxL statements to a file called output.txt, located in the preexisting directory specified by a relative path.

2.Data Constraints and its types

A. Naming of Constraint

The SQL CONSTRAINTS are an integrity which defines some conditions that restrict the column to remain true while inserting or updating or deleting data in the column. Constraints can be specified when the table is created first with a CREATE TABLE statement or at the time of modification of the structure of an existing table with ALTER TABLE statement.

We can name the constraint by our own while creating or altering the table to give the constraint a unique name.

SYNTAX:-

CREATE TABLE table_name

(col1 datatype1,……colN datatype N CONSTRAINT constraint_name constraint_type);

ALTER TABLE table_name ADD CONSTRAINT constraint_name constraint_type;

B. Types of Constraints: Column level, Table level

A constraint name is formatted as below: <table name> <column name> <constraint abbreviation> Constraints are abbreviated as:-

1. foreign key - fk

2. primary key - pk

3. unique - un

4. check - ck

5. not null - nn

6. default -

Constraint names can be upto 30 characters long (in oracle). The primary key constraint on the table Employee can thus be named as employee employee id pk. B. Types of constraints:-

1. **Column Level Constraints** - Column level constraints refer to a single column in the table and do not specify a column name (except check constraints). They refer to the column that they follow.

2. **Table Level Constraints** - Table level constraints refer to one or more columns in the table. Table level constraints specify the names of the columns to which they apply. Table level CHECK constraints can refer to 0 or more columns in the table

Different types of Constraints Used are:-

(a) **Primary Key**-The PRIMARY KEY constraint uniquely identifies each record in a table. Primary keys must contain UNIQUE values, and cannot contain NULL values .A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

SYNTAX:-

CREATE TABLE table_name(col1 datatype PRIMARY KEY,col2 datatype,..);

OR

CREATE TABLE table_name (col1 datatype,col2 datatype,…,PRIMARY KEY(col1,col2));

EXAMPLE:-

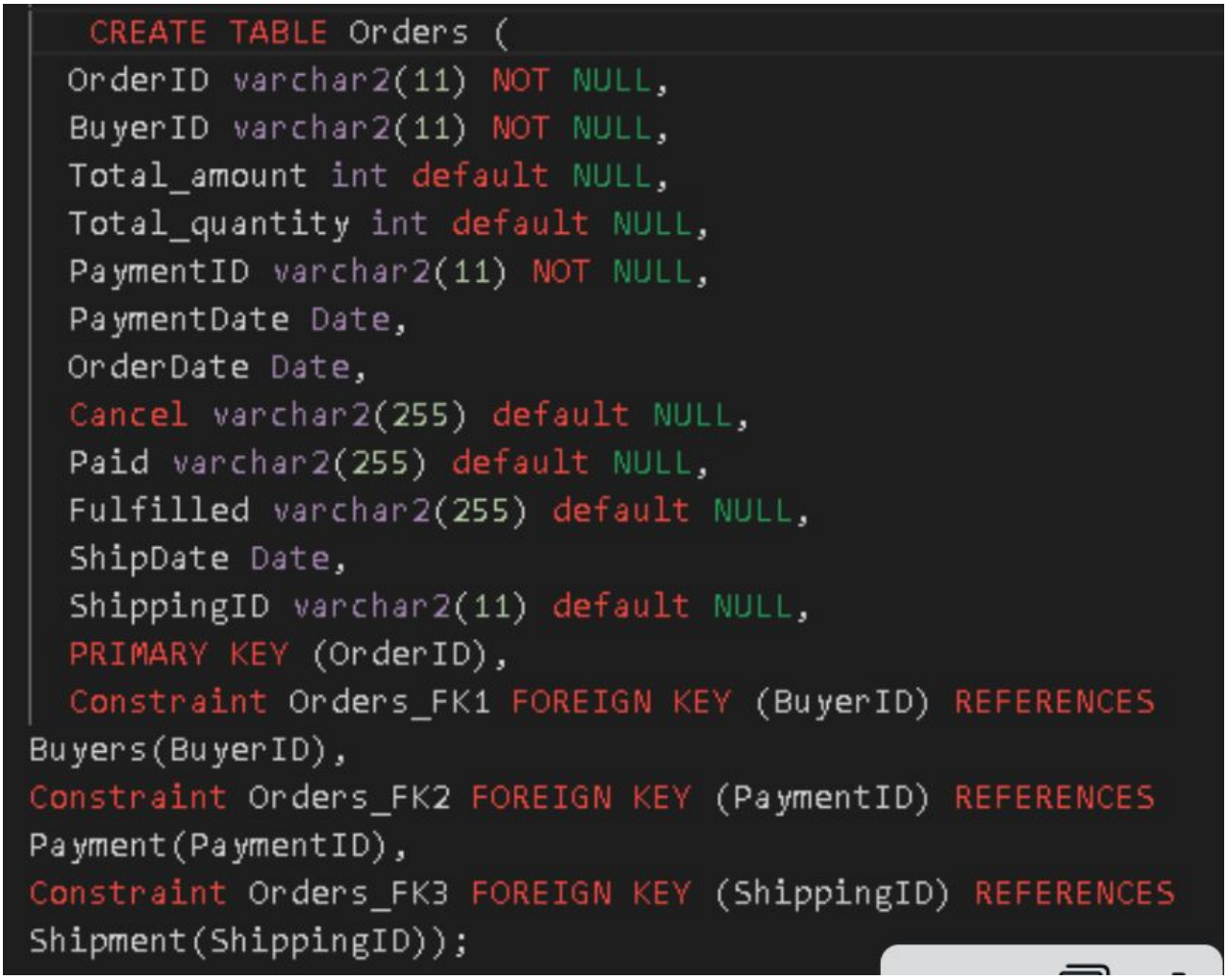
```
 CREATE TABLE Shipment (
ShippingID varchar2(11)  NOT NULL,
ShipTEL varchar2(100) default NULL,
ShipperName varchar2(255),
 PRIMARY KEY (ShippingID));
```

(b) **Foreign Key**-A FOREIGN KEY is a key used to link two tables together. A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table. The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

SYNTAX:-

CREATE TABLE table_name(col1 datatype,…,colN datatype FOREIGN KEY REFERENCES table2_name(colX));

EXAMPLE:-



```
CREATE TABLE Orders (
OrderID varchar2(11) NOT NULL,
BuyerID varchar2(11) NOT NULL,
Total_amount int default NULL,
Total_quantity int default NULL,
PaymentID varchar2(11) NOT NULL,
PaymentDate Date,
OrderDate Date,
Cancel varchar2(255) default NULL,
Paid varchar2(255) default NULL,
Fulfilled varchar2(255) default NULL,
ShipDate Date,
ShippingID varchar2(11) default NULL,
PRIMARY KEY (OrderID),
Constraint Orders_FK1 FOREIGN KEY (BuyerID) REFERENCES
Buyers(BuyerID),
Constraint Orders_FK2 FOREIGN KEY (PaymentID) REFERENCES
Payment(PaymentID),
Constraint Orders_FK3 FOREIGN KEY (ShippingID) REFERENCES
Shipment(ShippingID));
```

c.) Unique- The UNIQUE constraint ensures that all values in a column are different.Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.A PRIMARY KEY constraint automatically has a UNIQUE constraint.

SYNTAX:-

CREATE TABLE table_name(col1 datatype UNIQUE,..colN datatype UNIQUE);

EXAMPLE:-

```sql
CREATE TABLE suppliers (
    supplier_id INT AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    phone VARCHAR(15) NOT NULL UNIQUE,
    address VARCHAR(255) NOT NULL,
    PRIMARY KEY (supplier_id),
    CONSTRAINT uc_name_address UNIQUE (name , address)
);
```

(d)Not Null- By default, a column can hold NULL values.The NOT NULL constraint enforces a column to NOT accept NULL values.This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

SYNTAX:-

CREATE TABLE table_name(col1 datatype NOT NULL,..,colN datatype NOT

NULL); **EXAMPLE:-**

```
 CREATE TABLE Payment (
PaymentID varchar2(11) NOT NULL,
CardHolderName varchar2(255) default NULL,
CardType varchar2(255) default NULL,
CreditCard varchar2(255),
CredExpMo varchar2(255) default NULL,
CardExpYr varchar2(255) default NULL,
BillingAddress varchar2(255) default NULL,
BillingCity varchar2(255),
BillingState varchar2(50) default NULL,
BillingZipcode varchar2(10) default NULL,
BillCountry varchar2(100) default NULL,
OrderID varchar2(11) default NULL,
 PRIMARY KEY (PaymentID));
```

(e)Check-The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a single column it allows only certain values for this column. If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SYNTAX:-

CREATE TABLE table_name(col1 datatype CHECK (CONDITION));

EXAMPLE:-

```
CREATE TABLE Order_Details (
   OrderID varchar(11) NOT NULL CHECK(OrderID>0),
  ProductID varchar(11) default NULL,
  ItemQuantity int default NULL,
   PRIMARY KEY (OrderID));
```

(f)Default-The DEFAULT constraint is used to provide a default value for a column. The default value will be added to all new records IF no other value is specified.

SYNTAX:-

CREATE TABLE table_name(col1 datatype DEFAULT

default_value); **EXAMPLE:-**

```
CREATE TABLE Products (
  ProductID varchar2(11) NOT NULL,
  ProductName varchar2(255) default NULL,
  SupplierID varchar2(11) default NULL,
  Category varchar2(255) default NULL,
  UnitPrice varchar2(100) default NULL,
  Color varchar2(255) default NULL,
  QuantityOnHand int default NULL,
   PRIMARY KEY (ProductID));
```

C. Behaviours of foreign key table: Unlike primary keys, foreign keys can contain duplicate values. Also, it is OK for them to contain NULL values. Indexes aren't automatically created for foreign keys; however, as a DBA, you can define them. A table is allowed to contain more than one foreign key.

D. On Delete/Update Restrict

RESTRICT means that any attempt to delete and/or update the parent table will fail throwing an error. This is the default behaviour in the event that a referential action is not explicitly specified.

ON UPDATE RESTRICT : *the default* : if you try to update a FOREIGN KEY ATTRIBUTE in table, the database compiler will reject the operation if one of the value in the foreign key attribute at least links to one of the values in the main table where the attribute is PRIMARY KEY.
Example:- **DELETE FROM Supplier WHERE Supplier_id = '30918186609';**

ON DELETE RESTRICT : *the default* : if you try to delete a FOREIGN KEY ATTRIBUTE in table, the database compiler will reject the operation if one of the foreign key column value at least links in the main table where that particular column is PRIMARY KEY.

E. On delete/Update Cascade

When the Referenced Foreign key is deleted or updated, all rows referencing that key are deleted or updated respectively.

UPDATE CASCADE: When we create a foreign key using UPDATE CASCADE the referencing rows are updated in the child table when the referenced row is updated in the parent table which has a primary key.

SYNTAX:-

CREATE TABLE table_name(col1 datatype,..,

colN datatype FOREIGN KEY REFERENCES table2_name(colX) ON UPDATE CASCADE);

DELETE CASCADE: When we create a foreign key using this option, it deletes the referencing rows in the child table when the referenced row is deleted in the parent table which has a primary key.

SYNTAX:- CREATE TABLE table_name(col1 datatype,..,

colN datatype FOREIGN KEY REFERENCES table2_name(colX) ON DELETE CASCADE);

EXAMPLE:-

```
SELECT * FROM supplier;
DELETE FROM supplier WHERE supplier_id = '70268860023';
SELECT * From Manager;
```

Before and after deletion:- As you can see, suppliers with deleted supplier id have also been deleted.

F. On delete/Update Set Null

Delete or update the row from the parent table and set the foreign key column or columns in the child table to NULL. Both ON DELETE SET NULL and ON UPDATE SET NULL clauses are supported.

If you specify a SET NULL action, make sure that you have not declared the columns in the child table as NOT NULL.

SYNTAX:-

On Update SET NULL

CREATE TABLE table_name(col1 datatype,....,colN FOREIGN KEY REFERENCES table2_name(colX) ON UPDATE SET NULL);

On Delete SET NULL

CREATE TABLE table_name(col1 datatype,....,colN FOREIGN KEY REFERENCES table2_name(colX) ON DELETE SET NULL);

EXAMPLE:

```
SELECT (order_id) FROM Orders;
DELETE FROM Orders WHERE Order_id = 'NULL';
SELECT (order_id) FROM Orders;
```

G. Integrity Constraints Via Alter Table Command

We can also add or modify constraints using Alter table command on existing table.Data Constraint can be added by using Alter table command.When we require to define constraints on existing table then alter table command can be helpful.We can add any constraint (primary key, foreign key,unique and so on) And give it a unique name.

SYNTAX:-

ALTER TABLE table_name ADD CONSTRAINT <const_name> <const_type>;

EXAMPLE:-

```
ALTER TABLE orders;
ADD CONSTRAINT (bill_billID_pk) PRIMARY KEY (order_id);
```

3. SQL OPERATORS AND FUNCTIONS

A.OPERATORS

An operator manipulates individual data items and returns a result. The data items are called *operands* or *arguments*. Operators are represented by special characters or by keywords.

(i) Arithmetic Operators

An arithmetic operator in an expression is used to negate, add, subtract, multiply, and divide numeric values. The result of the operation is also a numeric value. Some of these operators are also used in date arithmetic.

Operators	Purpose	Syntax
+	Adds up the values given.It is a unary operator.	SELECT col+100 FROM table_name;
-	Subtracts the values given.It is a unary operator.	SELECT col-100 FROM table_name;
*	Multiplies the data items.It is a binary operator.	SELECT col*100 FROM table_name;
/	Divides the data items.It is a binary operator.	SELECt col/100 FROM table_name;
%	It is use to get remainder when one data is divided by another	SELECT col%100 FROM table_name;

(ii) Logical Operators

A logical operator combines the results of two component conditions to produce a single result based on them or to invert the result of a single condition.

Operators	Purpose	Syntax

NOT	Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE. If it is UNKNOWN, it remains UNKNOWN.	SELECT colN FROM table_name WHERE colN NOT (CONDITION);
AND	Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE. Otherwise returns UNKNOWN.	SELECT colN FROM table_name WHERE condition1 AND condition2;
OR	Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE. Otherwise returns UNKNOWN.	SELECT colN FROM table_name WHERE condition1 OR condition2;

(iii) Relational Operators

These are the Comparison operators which are used to compare one expression with another. The result of such a comparison can be TRUE, FALSE, or UNKNOWN.

Operators	Purpose	Syntax
=	Equals	SELECT colX FROM table_name WHERE val1=val2;
>	Greater Than	SELECT colX FROM table_name WHERE val1>val2;
<	Less Than	SELECT colX FROM table_name WHERE val1<val2;
<=	Less than Equal to	SELECT colX FROM table_name WHERE val1<=val2;

>=	Greater than Equal to	SELECT colX FROM table_name WHERE val1>=val2;
!= <>	Not Equals To	SELECT colX FROM table_name WHERE val1!=val2;

EXAMPLES:-

1.>query

```
SELECT * FROM orders WHERE TOTAL_AMOUNT > ANY (SELECT TOTAL_AMOUNT FROM orders WHERE TOTAL_AMOUNT>=7);
```

Output

ORDERID	BUYERID	TOTAL_AMOUNT	TOTAL_QUANTITY
35764031654	94908631679	928	2
47217024154	26350935579	662	5
70701609058	89264121725	238	13
07261862587	12714972662	177	6
89878325779	72330070747	134	11
99251300121	81921795835	68	8
96945305380	42711312958	56	4
90817183713	67711139299	54	9
21942566549	50345881058	48	1

2.>query

```
SELECT * FROM orders WHERE TOTAL_QUANTITY BETWEEN 5 AND 20;
```

Output

Results | Explain | Describe | Saved SQL | History

ORDERID	BUYERID	TOTAL_AMOUNT	TOTAL_QUANTITY
07261862587	12714972662	177	6
89878325779	72330070747	134	11
70701609058	89264121725	238	13
05553373345	75521050479	22	7
99251300121	81921795835	68	8
90817183713	67711139299	54	9
47217024154	26350935579	662	5

7 rows returned in 0.02 seconds Download

3.>
```
select * from buyers
```

BUYERID	TELNUMBER	EMAIL	USERNAME	PASSWORD	SHIPADRESS	SHIPCITY	SHIPSTATE	SHIPCOUNTRY	SHIPZIPCODE
94908631679	1-772-737-2701	molestie@idantedictum.ca	Clark	JFA79CQJ8XU	P.O. Box 504, 7630 Lacus Rd.	Viransehir	Sanliurfa	Liberia	65242
89264121725	1-905-188-8108	mi.pede@non.edu	Noble	AOF35ISM0QN	2510 Tellus Avenue	Joliet	IL	Christmas Island	86876
50345881058	1-850-899-9543	euismod.Etiam@etultrices.edu	Tashya	PDE87LWL3PY	8200 Quis, Street	Vezirköprü	Samsun	Guinea-Bissau	M5Y 1Z0
81921795835	1-583-615-6760	velit.dui@sapien.org	Aretha	ANP81YWC9CU	Ap #516-795 Nisi Road	Mildura	VIC	Singapore	4505
26350935579	1-865-217-7844	ac.mattis@felispurus.org	Abbot	V5R52BPA7OY	951-727 A Street	Laramie	WY	Morocco	007708
12714972662	1-102-293-5153	Quisque@elitdictumeu.net	Vance	RST79SIK9CB	1341 In St.	Jerez de la Frontera	AN	Singapore	53506
75521050479	1-959-804-3019	id.libero.Donec@gmail.com	Matthew	PWN64YUG3AJ	253-4847 Rutrum Ave	Callander	PE	Mongolia	ZN53 8TG
72330070747	1-184-195-5645	gravida.sagittis.Duis@magnaa.ca	Dominic	OFX66VPY2TS	Ap #976-802 Dignissim Street	Islahiye	Gaz	Finland	33067
42711312958	1-317-594-7754	Nunc.ac@molestieSedid.ca	Lynn	JEE58NSU1DO	7377 Cras Rd.	Gotzis	Vbg	Seychelles	C5E 9C8
67711139299	1-632-629-5645	gravida.non@urnasuscipitnonummy.net	Dustin	BSW81VNC4BY	Ap #972-7019 Nulla Street	Orangeville	ON	Marshall Islands	113921

B.FUNCTIONS

(i) String Functions

String Functions are used in SQL to do some particular type of operations with String data items. There are many types of String Functions given which are discussed in the table below.

Functions	Description	Syntax
CONCAT	Adds two or more Strings together.	SELECT CONCAT(string1, string2, ..., string_n)
LEN	Returns the length of the string.	SELECT LEN(string)

LTRIM	Remove left white spaces of the string.	SELECT LTRIM(string)
RTRIM	Remove right white spaces of the string.	SELECT RTRIM(string)
TRIM	Remove both left and right white spaces from the string.	SELECT TRIM([characters FROM]string)
UPPER	Changes all letters of string to upper case.	SELECT UPPER(text)
LOWER	Changes all letters of string to lower case.	SELECT LOWER(text)
REPLACE	Remove all occurrences of a substring within a string, with a new substring.	SELECT REPLACE(string, old_string, new_string)
REVERSE	Reverses a string and returns the result.	SELECT REVERSE(string);
SUBSTRING	Extracts some characters from a string.	SELECT SUBSTRING(string, start, length);

EXAMPLES:-

query:

```
/////string function
SELECT ProductName, LENGTH(ProductName) AS LENGTHOFNAME FROM Products;
```

Output

PRODUCTNAME	LENGTHOFNAME
Lotion	6
Remover	7
Beauty	6
Beauty	6
Sets	4
Lotion	6
Shampoo	7
Moisturizer	11
Sets	4
Primer	6

query:

```
SELECT UPPER(USERNAME) AS UpperFirstName FROM buyers;
```

Output:

UPPERFIRSTNAME
CLARK
NOBLE
TASHYA
ARETHA
ABBOT
VANCE
MATTHEW
DOMINIC
LYNN

Query:

```
SELECT LOWER(USERNAME) AS LowerLastName From buyers;
```

Output:

LOWERLASTNAME
clark
noble
tashya
aretha
abbot
vance
matthew
dominic
lynn
dustin

(ii) Numeric Functions

Numeric Functions are used to perform operations on numbers and return numbers.These can perform operations on the data items of a particular table as well.Many of the numeric functions are given in the table below.

Functions	Description	Syntax
ABS()	It returns the absolute value of a number.	SELECT ABS(num);
ACOS()	It returns the arc cosine of a number.	SELECT ACOS(num);
ASIN()	It returns the arc sine of a number.	SELECT ASIN(num);

ATAN()	It returns the arc tangent of a number.	SELECT ATAN(num);
CEIL()/CEILING()	It returns the smallest integer value that is greater than or equal to a number.	SELECT CEIL(num);
COS()	It returns the cosine of the integer.	SELECT COS(num);
SIN()	It returns the sine of the integer.	SELECT SIN(num);
TAN()	It returns the tangent of the integer.	SELECT TAN(num);
FLOOR()	It returns the largest integer value that is less than or equal to a number.	SELECT FLOOR(num);
GREATEST()	It returns the greatest value in a list of expressions.	SELECT GREATEST(list_of_numb ers);
LEAST()	It returns the smallest value in a list of expressions	SELECT LEAST(list_of_numbers);
SQRT()	It returns the square root of a number	SELECT SQRT(num);
ROUND()	It returns a number rounded to a certain number of decimal places.	SELECT ROUND(num);
TRUNCATE()	This doesn't work for SQL Server. It returns 7.53635 truncated to 2 places right of the decimal point.	SELECT TRUNCATE(num,trunc_pl aces);

EXAMPLES:-

Query:

```
select avg(Total_amount) from orders;
```

Output:

(iii)Aggregate Functions

An aggregate function performs a calculation on a set of values, and returns a single value. Aggregate functions are often used with the GROUP BY clause of the SELECT statement.Except for `COUNT(*)`, aggregate functions ignore null values.

Functions	Description	Syntax
MAX()	Returns the maximum value in the column.	SELECT MAX(col_name) FROM table_name;
MIN()	Returns the minimum value in the column.	SELECT MIN(col_name) FROM table_name;
SUM()	Returns the sum of all the values of a particular column.	SELECT SUM(col_name) FROM table_name;
AVG()	Return the average value by doing the sum of all values in the particular column.	SELECT AVG(col_name) FROM table_name;
COUNT()	Returns the total numbers of values in column after neglecting NULL values.	SELECT COUNT(col_name) FROM table_name;

We can also use group by and having statement with aggregate functions.Although we will see what are these in next section but here is the syntax of aggregate functions with them:-

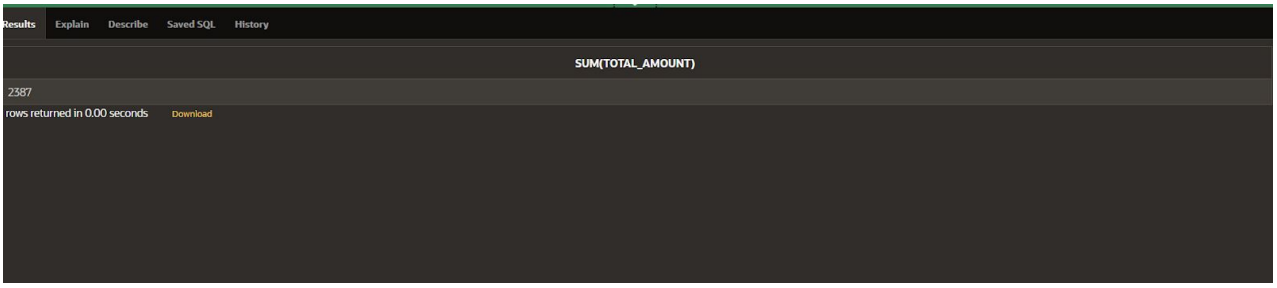
SELECT func_name() FROM table_name WHERE cond1 GROUP BY col_name HAVING cond2;

EXAMPLES:-

FOLLOWING ARE THE QUERIES FROM OUR DATABASE:

```
//////sum
select sum(Total_amount) from orders;
```

AND HERE IS THE OUTPUT;



(iv) Date Functions

Date Functions in SQL are very effective in helping to provide dates and time of the system or calculating from a particular date and many more. These are selected from DUAL table which already exists in the databases.

Functions	Description	Syntax
DATE_FORMAT()	Displays date/time data in different formats	SELECT DATE_FORMAT(date,format) ;
FROM_DAYS()	Given a day number N, returns a DATE value.	SELECT FROM_DAYS(number);
DATE_DIFF()	It gives difference between two dates given.	SELECT DATE_DIFF(date1,date2);
NOW()	It gives the current date and time.	SELECT NOW();

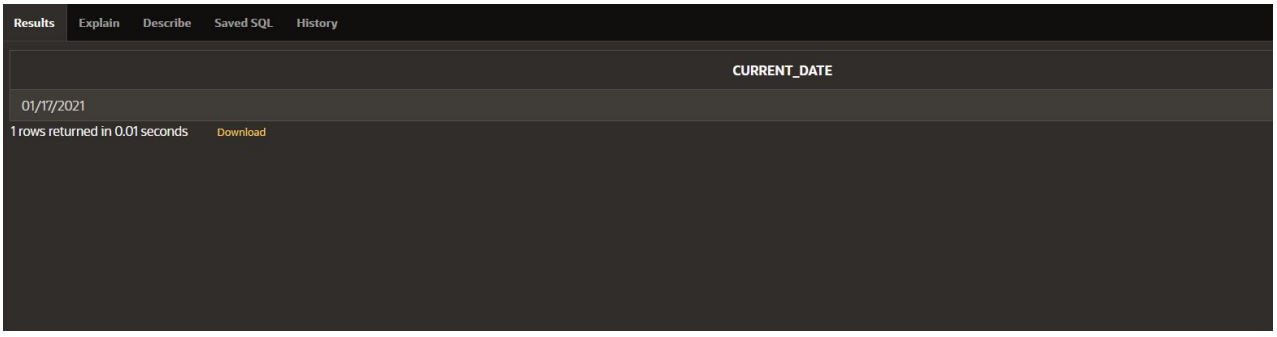
MONTH()	Returns the month for date, in the range 0 to 12.	SELECT MONTH(date);
CURDATE()	Return the current date of the system.	SELECT CURDATE();
SYSDATE()	Returns the current system date and time.	SELECT SYSDATE();
TIMESTAMP()	Returns the date as datetime value.	SELECT TIMESTAMP(date);
DAYOFWEEK()	Returns the number of the day of week.	SELECT DAYOFWEEK(date);

EXAMPLES:-

QUERY: CURDATE();

```
///////current date
SELECT CURRENT_DATE FROM DUAL;
```

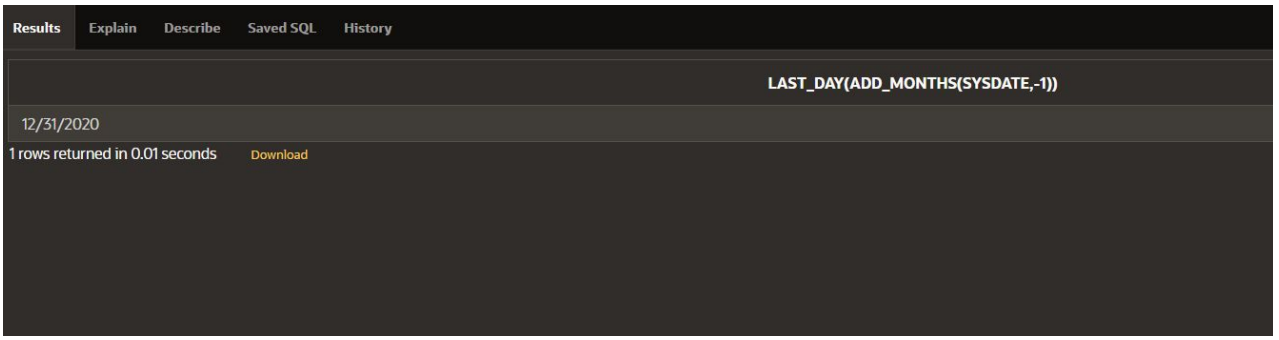
OUTPUT::



QUERY: SYSDATE()

```
SELECT LAST_DAY(ADD_MONTHS(SYSDATE,-1)) FROM DUAL;
```

OUTPUT:



4. Order By, Group by and Having statements

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.

SYNTAX:-

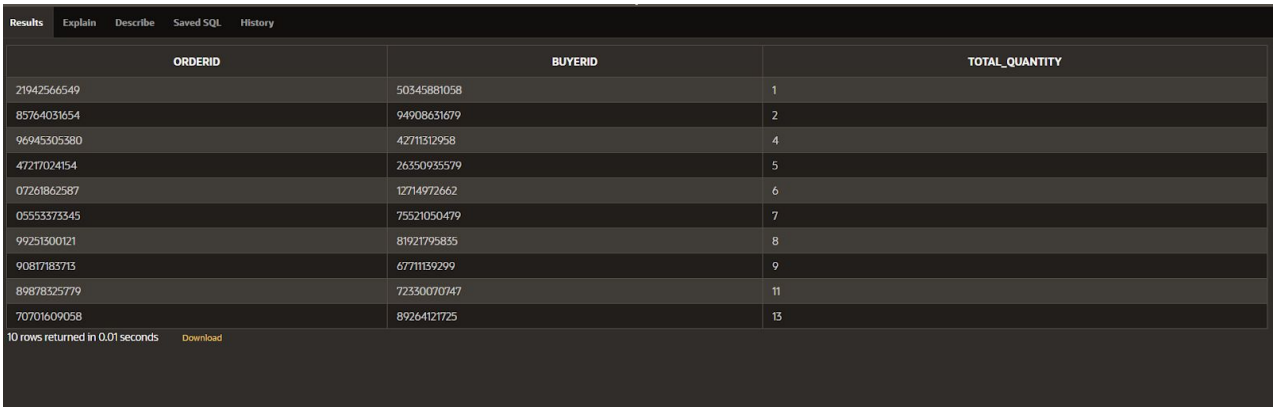
```
SELECT column1, column2, … FROM table_name
        ORDER BY column1, column2, ... ASC|DESC;
```

EXAMPLE:-

QUERY:

```
//////ORDER BY
SELECT OrderID, BuyerID, Total_quantity FROM Orders ORDER BY Total_quantity;
```

Output:



Results | Explain | Describe | Saved SQL | History

ORDERID	BUYERID	TOTAL_QUANTITY
21942566549	50345881058	1
85764031654	94908631679	2
96945305380	42711312958	4
47217024154	26350935579	5
07261862587	12714972662	6
05553373345	75521050479	7
99251300121	81921795835	8
90817183713	67711139299	9
89878325779	72330070747	11
70701609058	89264121725	13

10 rows returned in 0.01 seconds Download

Group By: The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".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.

SYNTAX:-

SELECT column_name(s) FROM table_name WHERE condition

GROUP BY column_name(s) ORDER BY column_name(s);

EXAMPLES:-

Query:

```
///////////////  ALIAS / GROUP BY
select CardType,count(*) as popular from Payment group by CardType;
```

Output:

Results | Explain | Describe | Saved SQL | History

CARDTYPE	POPULAR
America Express	2
Visa	3
JCR	3
Discover	2

4 rows returned in 0.03 seconds Download

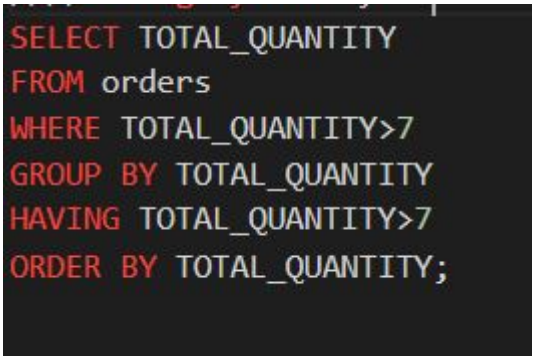
Having: The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

SYNTAX:-

SELECT column_name(s) FROM table_name WHERE condition GROUP BY column_name(s) HAVING condition ORDER BY column_name(s);

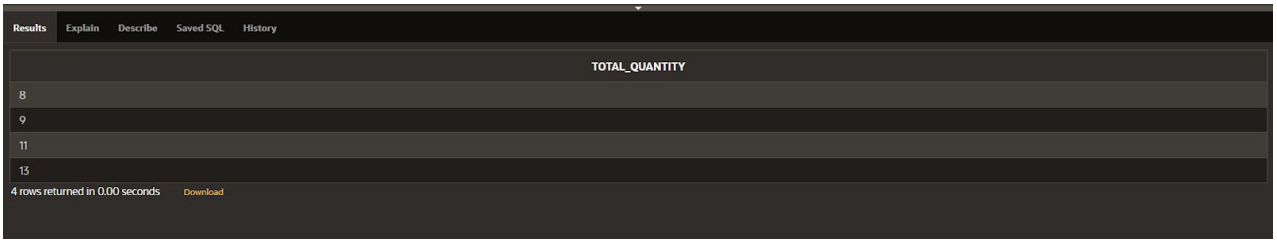
EXAMPLES:-

Query:



```
SELECT TOTAL_QUANTITY
FROM orders
WHERE TOTAL_QUANTITY>7
GROUP BY TOTAL_QUANTITY
HAVING TOTAL_QUANTITY>7
ORDER BY TOTAL_QUANTITY;
```

Output:



Results | Explain | Describe | Saved SQL | History

TOTAL_QUANTITY
8
9
11
13

4 rows returned in 0.00 seconds Download

5. SUBQUERIES

A Subquery is a query within another SQL query and embedded within the WHERE clause.A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

There are a few rules that subqueries must follow −

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

(i) Nested Queries

In nested queries, a query is written inside a query. The result of the inner query is used in the execution of the outer query.

SYNTAX:-

Single

SELECT column(s) FROM table_name WHERE column IN/NOT IN(query);

Multiple

SELECT column(s) FROM table_name WHERE column IN/NOT IN(query2 IN/NOT IN(query(s));

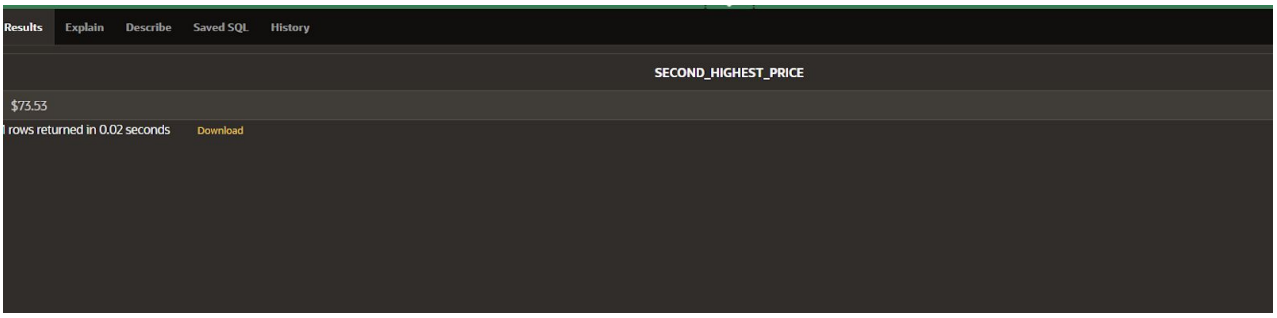
EXAMPLES:-

Query:

```
////////////// NESTED QUERIES


select max(UnitPrice) as second_highest_price
from Products
where Products.UnitPrice<(select max(UnitPrice)
                          from Products);
```

Output:



(ii)Correlated Queries

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a **SELECT**, **UPDATE**, or **DELETE** statement.

SYNTAX:-

SELECT column1, column2, ... FROM table1 outer

WHERE column1 operator

(SELECT column1 FROM table2

WHERE expr1 = outer.expr2);

EXAMPLES:-

Query:

```
SELECT *
FROM Products
WHERE QuantityOnHand >
(SELECT AVG(QuantityOnHand)
FROM Products
);
```

Output:

Results | Explain | Describe | Saved SQL | History

PRODUCTID	PRODUCTNAME	SUPPLIERID	CATEGORY	UNITPRICE	COLOR	QUANTITYONHAND
206908-3232	Remover	31704489846	Lip	$29.58	green	898
096419-4823	Beauty	30918186609	Cheek	$26.70	red	551
197710-3876	Lotion	70268860023	Eye, Face	$26.98	green	794
533169-3365	Shampoo	30751277561	Body, Hair	$84.35	yellow	916
998757-5991	Sets	43636501698	-	$59.44	orange	827

rows returned in 0.01 seconds Download

Nested Subqueries Versus Correlated Subqueries :

With a normal nested subquery, the inner SELECT query runs first and executes once, returning values to be used by the main query. A correlated subquery, however, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query.

6. JOINS

A.Cross Join:

The CARTESIAN JOIN is also known as CROSS JOIN. In a CARTESIAN JOIN there is a join for each row of one table to every row of another table. This usually happens when the matching column or WHERE condition is not specified.

Syntax:

```
SELECT ColumnName_1,
        ColumnName_2,
        ColumnName_N
FROM [Table_1]
        CROSS JOIN [Table_2];
```

EXAMPLE:

Query:

```
///////  CROSS JOIN
SELECT *
FROM Products
CROSS JOIN Orders
WHERE Products.ProductName = 'Lotion' OR Products.ProductName = 'Primer';
```

Output:

Results | Explain | Describe | Saved SQL | History

PRODUCTID	PRODUCTNAME	SUPPLIERID	CATEGORY	UNITPRICE	COLOR	QUANTITYONHAND	ORDERID	BUYERID	TOTAL_AMOUNT	TOTAL_QUANTITY	PAYMENTID	PAYMENTDATE	ORDERDATE	C
748130-5105	Lotion	12241497023	Hair	$6.83	yellow	109	07261862587	12714972662	177	6	26660945790	10/20/2001	10/20/2001	0
748130-5105	Lotion	12241497023	Hair	$6.83	yellow	109	89878325779	72330070747	134	11	83308456699	11/05/2021	11/05/2021	0
748130-5105	Lotion	12241497023	Hair	$6.83	yellow	109	70701609058	89264121725	238	13	38387495295	12/15/2021	12/15/2021	0
748130-5105	Lotion	12241497023	Hair	$6.83	yellow	109	05553373345	75521050479	22	7	79922775587	12/30/2021	12/30/2021	0
748130-5105	Lotion	12241497023	Hair	$6.83	yellow	109	21942566549	50345881058	48	1	57775141099	10/12/2021	10/12/2021	0
748130-5105	Lotion	12241497023	Hair	$6.83	yellow	109	99251300121	81921795835	68	8	92384433306	11/17/2021	11/17/2021	0
748130-5105	Lotion	12241497023	Hair	$6.83	yellow	109	96945305380	42711312958	56	4	36297339285	10/30/2021	10/30/2021	0
748130-5105	Lotion	12241497023	Hair	$6.83	yellow	109	90817183713	67711139299	54	9	49871562602	11/17/2021	11/17/2021	0
748130-5105	Lotion	12241497023	Hair	$6.83	yellow	109	85764031654	94908631679	928	2	31863280913	11/25/2021	11/25/2021	0
748130-5105	Lotion	12241497023	Hair	$6.83	yellow	109	47217024154	26350935579	662	5	39477519625	11/25/2021	11/25/2021	0

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.05 seconds Download

B.Natural Join:

A NATURAL JOIN is a JOIN operation that creates an implicit join clause for you based on the common columns of the two tables that are being joined. Common columns are columns that have the same name in both the tables.

Syntax:

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

EXAMPLE:

Query:

```
/////natural join
SELECT *
FROM Orders
NATURAL JOIN Buyers
Where Total_amount>=6;
```

Output:

Results | Explain | Describe | Saved SQL | History

BUYERID	ORDERID	TOTAL_AMOUNT	TOTAL_QUANTITY	PAYMENTID	PAYMENTDATE	ORDERDATE	CANCEL	PAID	FULFILLED	SHIPDATE	SHIPPINGID	TELNUMBER	EMAIL
94908631679	85764031654	928	2	31863280913	11/25/2021	11/25/2021	0	1	1	11/28/2021	44764907944	1-772-737-2701	molestie@idantedictum.ca
89264121725	70701609058	238	13	38387495295	12/15/2021	12/15/2021	0	1	1	12/20/2021	73874731131	1-905-188-8108	mi.pede@non.edu
50345881058	21942566549	48	1	57773141099	10/12/2021	10/12/2021	0	1	1	10/15/2021	81892563843	1-850-899-9543	euismod.Etiam@etultrices.edu
81921795835	99251300121	68	8	92384433306	11/17/2021	11/17/2021	0	1	1	11/22/2021	14265493321	1-583-615-6760	velit.dui@sapien.org
26350935579	47217024154	662	5	39477519625	11/25/2021	11/25/2021	0	1	1	11/28/2021	43358910115	1-865-217-7844	ac.mattis@felispurus.org
12714972662	07261862587	177	6	26660945790	10/20/2001	10/20/2001	0	1	1	10/27/2021	86649590825	1-102-293-5153	Quisque@elitdictumeu.net
75521050479	05553373345	22	7	79922775587	12/30/2021	12/30/2021	0	1	1	01/05/2021	90750814057	1-959-804-3019	id.libero.Donec@gmail.com
72330070747	89878325779	134	11	83308456699	11/05/2021	11/05/2021	0	1	1	11/16/2021	93438458936	1-184-195-5645	gravida.sagittis.Duis@magnaa.ca

C.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 the same.

Syntax:

SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;

Example:

Query:

```
/////INNER JOIN

select Buyers.BuyerID,Orders.ShipDate from Buyers inner join Orders on Orders.BuyerID = Buyers.BuyerID
```

Output:

Results | Explain | Describe | Saved SQL | History

BUYERID	SHIPDATE
12714972662	10/27/2021
72330070747	11/16/2021
89264121725	12/20/2021
75521050479	01/05/2021
50345881058	10/15/2021
81921795835	11/22/2021
42711312958	10/31/2021
67711139299	11/25/2021
94908631679	11/28/2021
26350935579	11/28/2021

10 rows returned in 0.01 seconds Download

D.Outer Join:

1.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 the join. The rows for which there is no matching row on the right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax:

SELECT column_name(s)
FROM table1
LEFT JOIN table2

ON table1.column_name = table2.column_name;

Example:

Query:

```
////LEFT JOIN
select ShipCountry,count(*) as bestsell from Buyers left join Orders on Buyers.BuyerID = Orders.BuyerID group by ShipCountry;
```

Output:

Results | Explain | Describe | Saved SQL | History

SHIPCOUNTRY	BESTSELL
Finland	1
Liberia	1
Morocco	1
Singapore	2
Marshall Islands	1
Christmas Island	1
Guinea-Bissau	1
Mongolia	1
Seychelles	1

9 rows returned in 0.02 seconds Download

2.Right Outer Join:

RIGHT JOIN is similar to LEFT 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 the join. The rows for which there is no matching row on the left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax:

SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;

Example:

Query:

```
select *
from Order_Details
right outer join Orders ON
Orders.OrderId = Order_Details.OrderId;
```

Output:

Results | Explain | Describe | Saved SQL | History

ORDERID	PRODUCTID	ITEMQUANTITY	ORDERID	BUYERID	TOTAL_AMOUNT	TOTAL_QUANTITY	PAYMENTID	PAYMENTDATE	ORDERDATE	CANCEL	PAID	FULFILLED	SHIPDATE	SHIPPINGID
05553373345	748130-5105	57	05553373345	75521050479	22	7	79922775587	12/30/2021	12/30/2021	0	1	1	01/05/2021	90750814057
07261862587	206908-3232	26	07261862587	12714972662	177	6	26660945790	10/20/2001	10/20/2001	0	1	1	10/27/2021	86649590825
21942566549	332338-2345	34	21942566549	50345881058	48	1	57773141099	10/12/2021	10/12/2021	0	1	1	10/15/2021	81892563843
47217024154	096419-4823	57	47217024154	26350935579	662	5	39477519625	11/25/2021	11/25/2021	0	1	1	11/28/2021	43358910115
70701609058	585230-0077	13	70701609058	89264121725	238	13	38387495295	12/15/2021	12/15/2021	0	1	1	12/20/2021	73874731131
85764031654	197710-3876	29	85764031654	94908631679	928	2	31863280913	11/25/2021	11/25/2021	0	1	1	11/28/2021	44764907944
89878525779	533169-3365	11	89878525779	72330070747	134	11	83308456699	11/05/2021	11/05/2021	0	1	1	11/16/2021	93438458936
90817183713	588681-0034	9	90817183713	67711139299	54	9	49871562602	11/17/2021	11/17/2021	0	1	0	11/25/2021	09462963286
96945305380	998757-5991	46	96945305380	42711312958	56	4	36297339285	10/30/2021	10/30/2021	0	1	1	10/31/2021	02573273025
99251300121	375163-6428	81	99251300121	81921795835	68	8	92384433306	11/17/2021	11/17/2021	0	1	1	11/22/2021	14265493321

3.Full Outer Join:

FULL JOIN creates the result-set by combining the 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.

Syntax:

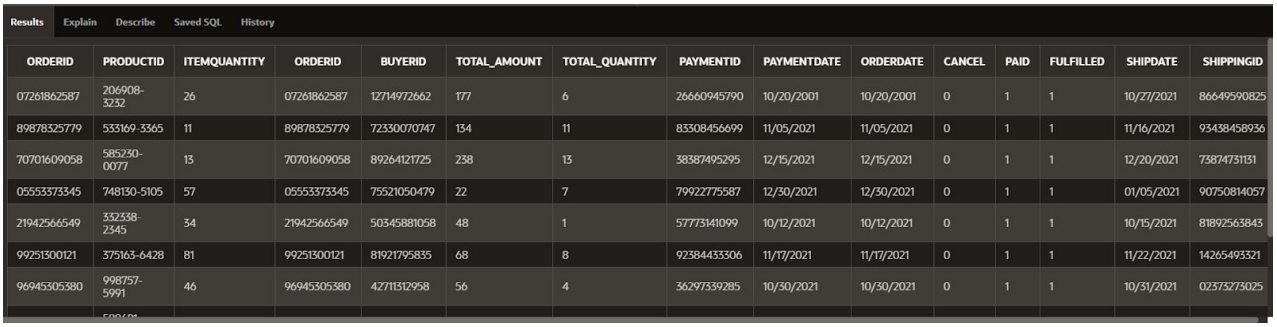
```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Example:

Query:

```
select *
from Order_Details
full outer join Orders ON
Orders.OrderId = Order_Details.OrderId;
```

Output:



Results | Explain | Describe | Saved SQL | History

ORDERID	PRODUCTID	ITEMQUANTITY	ORDERID	BUYERID	TOTAL_AMOUNT	TOTAL_QUANTITY	PAYMENTID	PAYMENTDATE	ORDERDATE	CANCEL	PAID	FULFILLED	SHIPDATE	SHIPPINGID
07261862587	206908-3232	26	07261862587	12714972662	177	6	26660945790	10/20/2001	10/20/2001	0	1	1	10/27/2021	86649590825
89878325779	533169-3365	11	89878325779	72330070747	134	11	83308456699	11/05/2021	11/05/2021	0	1	1	11/16/2021	93438458936
70701609058	585230-0077	13	70701609058	89264121725	238	13	38387495295	12/15/2021	12/15/2021	0	1	1	12/20/2021	73874731131
05553373345	748130-5105	57	05553373345	75521050479	22	7	79922775587	12/30/2021	12/30/2021	0	1	1	01/05/2021	90750814057
21942566549	332338-2345	34	21942566549	50345881058	48	1	57773141099	10/12/2021	10/12/2021	0	1	1	10/15/2021	81892563843
99251300121	375163-6428	81	99251300121	81921795835	68	8	92384433306	11/17/2021	11/17/2021	0	1	1	11/22/2021	14265493321
96945305380	998757-5991	46	96945305380	42711312958	56	4	36297339285	10/30/2021	10/30/2021	0	1	1	10/31/2021	02373273025

7.VIEWS

A.Introduction:

Views in SQL are a kind of virtual table. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain conditions.

B.TYPES

Materialized Views:

Definition of Materialized views(called as MV) has been stored in databases.Materialized views are useful in Data-warehousing concepts.When you create a Materialized view, Oracle Database creates one internal table and at least one index, and may create one view, all in the schema of the materialized views. Oracle Database uses these objects to maintain the materialized views in SQL data.

View Resolution:

When you read from a view, or join a view to an existing query the SQL server will then execute the query in the view and join it to your data set. When it does that, that would be view resolution.

Features/Advantages of views:

1.Security

Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data

2.Query Simplicity

A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.

3.Structural simplicity

Views can give a user a "personalized" view of the database structure, presenting the database as a set of virtual tables that make sense for that user.

4.Consistency

A view can present a consistent, unchanged image of the structure of the database, even if the underlying source tables are split, restructured, or renamed.

5.Data Integrity

If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets the specified integrity constraints.

6.Logical data independence.

Views can make the application and database tables to a certain extent independent. If there is no view, the application must be based on a table. With the view, the program can be established in view of above, to view the program with a database table to be separated.

D.CREATE/DROP VIEWS

Syntax 1 :

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

```
CREATE VIEW view_name AS
   SELECT column1,column2,.....
   FROM ((table_name1 FULL OUTER JOIN table_name2)
   NATURAL FULL OUTER JOIN table_name3);
```

Example:

Query:

```
//////////////////  VIEW
create view pop as
select ProductName,ItemQuantity from Products inner join Order_details on Products.ProductID = Order_Details.ProductID;
select ProductName, sum(ItemQuantity) from pop group by ProductName;
```

Output:

Results | Explain | Describe | Saved SQL | History

PRODUCTNAME	SUM(ITEMQUANTITY)
Lotion	86
Moisturizer	9
Primer	81
Shampoo	11
Sets	59
Remover	26
Beauty	91

7 rows returned in 0.04 seconds Download

Syntax 2:

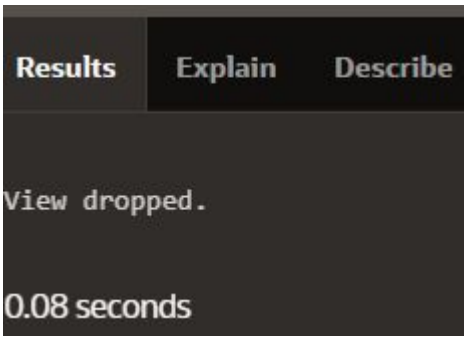
```
DROP VIEW view_name;
```

Example:

Query:

```
DROP view pop;
```

Output:



E.INSERT / UPDATE / DELETE IN VIEWS

There are certain conditions needed to be satisfied to update a view. If any one of these conditions is not met, then we will not be allowed to update the view.

1. The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.

2. The SELECT statement should not have the DISTINCT keyword.

3. The View should have all NOT NULL values.

4. The view should not be created using nested queries or complex queries.

5. The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.

1. Inserting a row in a view -

We can insert a row in a View in the same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View.

Syntax:

INSERT INTO view_name(column1, column2 , column3,..) VALUES(value1, value2, value3..);

This will insert a row in a view.

2.Deleting a row from a View -

We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first deletes the row from the actual table and the change is then reflected in the view.

Syntax:

DELETE FROM view_name WHERE condition;

This will delete a row from the view.

3.Updating a row from a View -

We can use the UPDATE statement of SQL to update rows from a view.

syntax:

UPDATE view_name SET column1 = value1, column2 = value2 ... WHERE condition;

This statement will update a row from the view.

8.Indexes

A.Introduction:

An index is a schema object. It is used by the server to speed up the retrieval of rows by using a pointer. It can reduce disk I/O(input/output) by using a rapid path access method to locate data quickly. An index helps to speed up select queries and where clauses, but it slows down data input, with the update and the insert statements. Indexes can be created or dropped with no effect on the data.

where the index is the name given to that index and TABLE is the name of the table on which that index is created and column is the name of that column for which it is applied.

Syntax:

```
CREATE INDEX index ON
    TABLE column;
```

B.Types:

1.Unique:

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table.

Syntax:

```
CREATE UNIQUE INDEX index_name
on table_name (column_name);
```

2.Simple:

A single-column index is created based on only one table column.

Syntax:

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

3.Composite:

A composite index is an index on two or more columns of a table.

Syntax:

```
CREATE INDEX index_name
on table_name (column1, column2);
```

C. CREATE/DROP INDEX

1.CREATE INDEX:

For creating a index on single column:

Syntax:

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

2.DROP INDEX:

Synat to perform this statement or execute this statement:

Syntax:

```
DROP INDEX index_name;
```

9.SEQUENCES

A.Introduction:

Sequence is a set of integers 1, 2, 3, ... that are generated and supported by some database systems to produce unique values on demand.

A sequence is a user defined schema bound object that generates a sequence of numeric values.

Sequences are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provide an easy way to generate them.

The sequence of numeric values is generated in an ascending or descending order at defined intervals and can be configured to restart when exceeds max value.

B.CREATE/DROP SEQUENCE:

CREATE

Syntax to create a sequence is,

```
CREATE SEQUENCE sequence-name
    START WITH initial-value
    INCREMENT BY increment-value
    MAXVALUE maximum-value
    CYCLE | NOCYCLE;
```

- The **initial-value** specifies the starting value for the Sequence.
- The **increment-value** is the value by which sequence will be incremented.
- The **maximum-value** specifies the upper limit or the maximum value upto which sequence will increment itself.
- The keyword `CYCLE` specifies that if the maximum value exceeds the set limit, sequence will restart its cycle from the begining.
- And, `NO CYCLE` specifies that if sequence exceeds `MAXVALUE` value, an error will be thrown.

Example:

Let's start by creating a sequence, which will start from `1`, increment by `1` with a

maximum value of 999.

```
CREATE SEQUENCE seq_1
START WITH 1
INCREMENT BY 1
MAXVALUE 999
CYCLE;
```

Now let's use the sequence that we just created above.

Below we have a **class** table,

ID	NAME
1	abhi
2	adam
4	alex

The SQL query will be,

```
INSERT INTO class VALUE(seq_1.nextval, 'anu');
```

Resultset table will look like,

ID	NAME
1	abhi

2	adam
4	alex
1	anu

Once you use nextval the sequence will increment even if you don't Insert any record into the table.

DROP

Syntax:

DROP SEQUENCE schema_name.sequence_name;

```
DROP SEQUENCE [ IF EXISTS ] { database_name.schema_name.sequence_name | schema_
 [ ; ]
```

C.INSERT USING SEQUENCE:

Example:

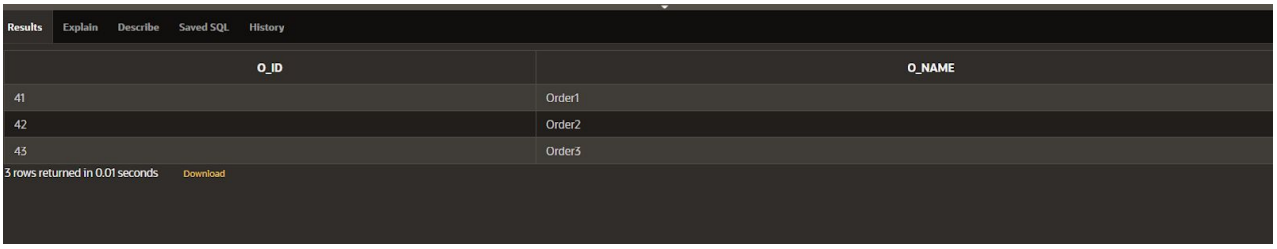
Query:

```
CREATE TABLE ordprod ( o_id NUMBER NOT NULL, o_name VARCHAR2(20) );
--Creating a sequence to insert values in ordprod
CREATE SEQUENCE sequence_1
START WITH 1
INCREMENT BY 1
MINVALUE 0
MAXVALUE 100 CYCLE;

INSERT INTO ordprod VALUES(sequence_1.nextval, 'Order1');
INSERT INTO ordprod VALUES(sequence_1.nextval, 'Order2');
INSERT INTO ordprod VALUES(sequence_1.nextval, 'Order3');

SELECT * FROM ordprod;
drop table ordprod;
```

Output:



Results | Explain | Describe | Saved SQL | History

O_ID	O_NAME
41	Order1
42	Order2
43	Order3

3 rows returned in 0.01 seconds Download

D.ALTER SEQUENCE:

ALTER SEQUENCE statement can be used to change the increment, minimum and maximum values, cached numbers, and behavior of an existing sequence. This statement affects only future sequence numbers.

Syntax:

```
ALTER SEQUENCE [ IF EXISTS ] name
    [ AS data_type ]
    [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE
    ] [ START [ WITH ] start ]
    [ RESTART [ [ WITH ] restart ] ]
    [ CACHE cache ] [ [ NO ] CYCLE ]
    [ OWNED BY { table_name.column_name | NONE } ]
```

Example:

Query:

```
CREATE TABLE ordprod ( o_id NUMBER NOT NULL, name VARCHAR2(20) );
--Creating a sequence to insert values in o_id
CREATE SEQUENCE sequence_1
START WITH 1
INCREMENT BY 1
MINVALUE 0
MAXVALUE 100 CYCLE;
--Inserting into ordprod
INSERT INTO ordprod VALUES(sequence_1.nextval, 'Order1');
INSERT INTO ordprod VALUES(sequence_1.nextval, 'Order2');
INSERT INTO ordprod VALUES(sequence_1.nextval, 'Order3');
--Displaying the ordprod table
SELECT * FROM ordprod;
--Altering the sequence 'sequence_1'
ALTER SEQUENCE sequence_1
INCREMENT BY 2;
--Inserting into ordprod using the altered sequence
INSERT INTO ordprod VALUES (sequence_1.nextval, 'Order3');
INSERT INTO ordprod VALUES (sequence_1.nextval, 'Order4');
--Displaying ordprod table
SELECT * FROM ordprod;
```

Output:

Results | Explain | Describe | Saved SQL | History

O_ID	NAME
22	Order3
1	Order1
2	Order2
3	Order3
62	Order4

5 rows returned in 0.01 seconds Download

10. Introduction to PL/SQL

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

A. Syntax of PL/SQL:

PL/SQL is a block-structured language; this means that the PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts -

1. Declarations - This section starts with the keyword DECLARE. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.

2. Executable Commands - This section is enclosed between the keywords BEGIN and END and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.

3. Exception Handling - This section starts with the keyword EXCEPTION. This optional section contains exception(s) that handle errors in the program. Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END. Following is the basic structure of a PL/SQL block -

Syntax:

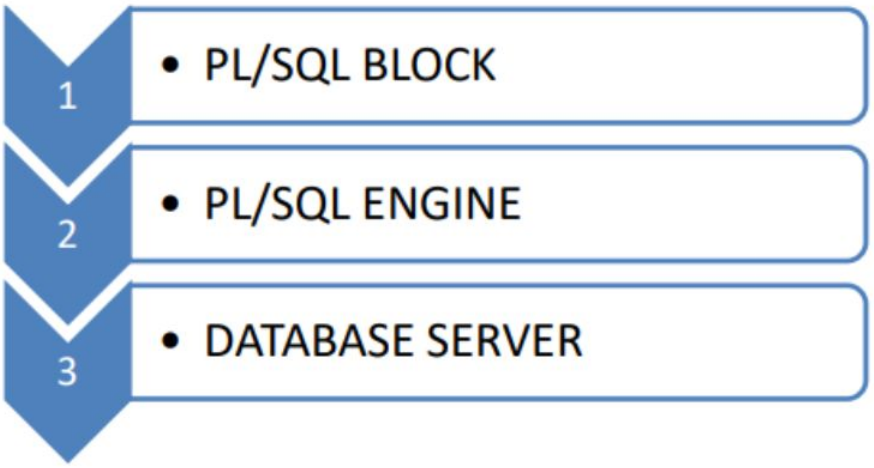
```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling>
END;
```

The END; line signals the end of the PL/SQL block.

PL/SQL Identifiers - PL/SQL identifiers are constants, variables, exceptions, procedures, cursors, and reserved words. By default, identifiers are not case sensitive

PL/SQL Architecture

PL/SQL architecture consist of following three components, displayed in the diagram below:



B.Control Structures in PL/SQL:

1.Sequential Control:

Sequential control statements, which are not crucial to PL/SQL programming. The sequential control statements are GOTO, which goes to a specified statement, and NULL, which does nothing.

Syntax:

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

2.Conditional Control:

Conditional selection statements, which run different statements for different data values. The conditional selection statements are IF and CASE.

Syntax:

```
IF < Condition > THEN
    < Action >
ELSE IF <Condition> THEN
    < Action >
```

ELSE < Action >
END IF;

i. IF-THEN Statement - The IF statement associates a condition with a sequence of statements enclosed by the keywords THEN and END IF. If the condition is true, the statements get executed and if the condition is false or NULL then the IF statement does nothing.

```
Syntax for IF-THEN statement is -

IF condition THEN
    S;
END IF;
```

ii. IF-THEN-ELSE Statement - IF statement adds the keyword ELSE followed by an alternative sequence of statement. If the condition is false or NULL, then only the alternative sequence of statements get executed. It ensures that either of the sequence of statements is executed.

```
Syntax for the IF-THEN-ELSE statement is -

IF condition THEN S1;
ELSE
    S2;
END IF;
```

iii. IF-THEN-ELSIF Statement - It allows you to choose between several alternatives.

```
Syntax for the IF-THEN-ELSIF Statement is -

IF(boolean_expression 1)THEN
    S1; -- Executes when the boolean expression 1 is true ELSIF(
boolean_expression 2) THEN
    S2; -- Executes when the boolean expression 2 is true ELSIF(
boolean_expression 3) THEN
    S3; -- Executes when the boolean expression 3 is true
ELSE
1
2
3
4
5
6
7
8
9
    S4; -- executes when the none of the above condition is true
END IF;
10
```

Example:

query:

```
DECLARE e_id Employee.employee_id%type := 'emp1';
    e_firstName Employee.first_name%type;
    e_lastName Employee.last_name%type;
    e_hourlyWage Employee.hourly_wage%type;
                                                                    83
BEGIN
    SELECT first_name, last_name, hourly_wage INTO                  8   e_firstName,
    FROM Employee                                                       e_lastName,
    WHERE employee_id = e_id;                                           e_hourlyWage
    IF (e_hourlyWage < 150) THEN dbms_output.put_line('Employee ' ||
        e_firstName ||
        ' ' || e_lastName || ' earns less than 150 per hour.');
    ELSIF (e_hourlyWage < 200) THEN dbms_output.put_line('Employee ' ||
        e_firstName ||
        ' ' || e_lastName || ' earns less than 200 per hour.');
    ELSE dbms_output.put_line('Employee ' || e_firstName ||
        ' ' || e_lastName || ' earns more than 199 per hour.'); END IF;
END;
```

Output:

```
Output:-
Employee Robert Smith earns more than 199 per hour.
```

iv. CASE Statement - Like the IF statement, the CASE statement selects one sequence of statements to execute. However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions. A selector is an expression, the value of which is used to select one of several alternatives.

```
Syntax for the case statement is -

CASE selector
    WHEN 'value1' THEN S1;
    WHEN 'value2' THEN S2;
```

```
    WHEN 'value3' THEN S3; ...
    ELSE Sn; -- default case
END CASE;
```

v. CASE Statement - The searched CASE statement has no selector and the WHEN clauses of the statement contain search conditions that give Boolean values.

Syntax for the searched case statement is -

```
CASE
    WHEN selector = 'value1' THEN S1;
    WHEN selector = 'value2' THEN S2;
    WHEN selector = 'value3' THEN S3; ...
    ELSE Sn; -- default case
END CASE;
```

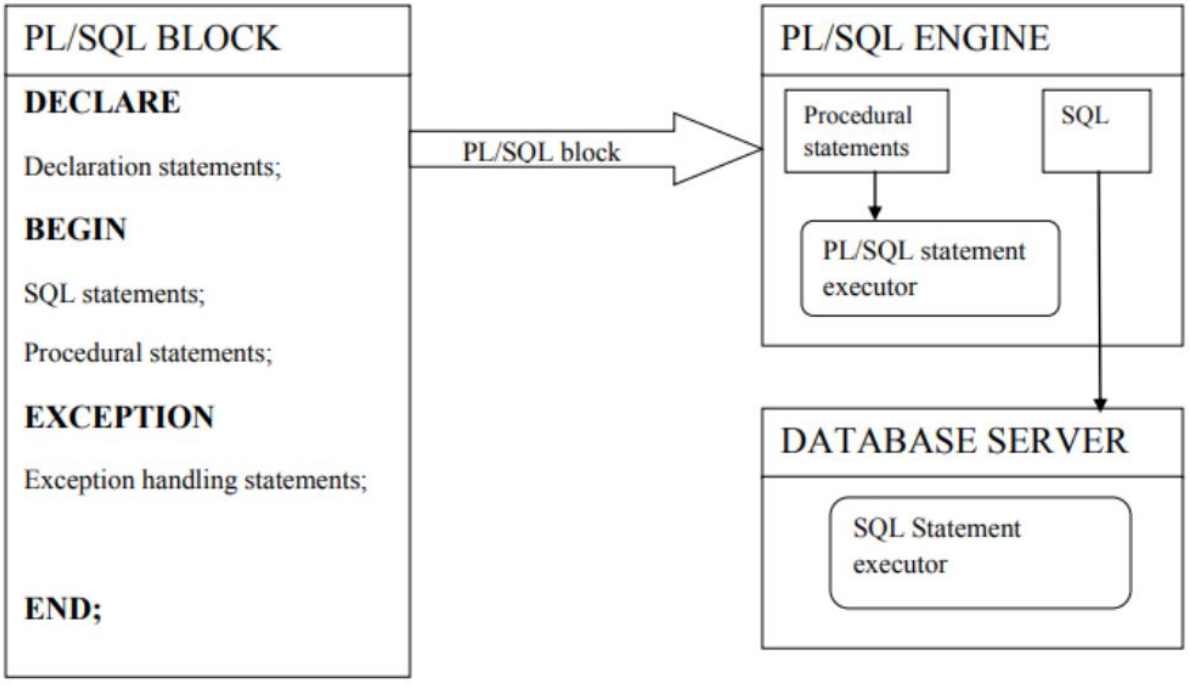
Example of a case statement:-

```
DECLARE e_id Employee.employee_id%type := 'emp1';
    e_firstName Employee.first_name%type;
    e_lastName Employee.last_name%type; e_gender
    Employee.gender%type;

BEGIN
    SELECT first_name, last_name, gender INTO e_firstName, e_lastName, e_gender FROM
    Employee
    WHERE employee_id = e_id;
    CASE e_gender
        WHEN 'Male' THEN dbms_output.put_line('Employee ' || e_firstName || ' ' ||
         e_lastName || ' is a male.');
        WHEN 'Female' THEN dbms_output.put_line('Employee ' || e_firstName
        || ' ' || e_lastName || ' is a female.'); ELSE
        dbms_output.put_line('Employee ' ||
        e_firstName || ' ' || e_lastName || ' is neither male or female.'); END CASE;
END;
```

```
Employee Robert Smith is a male.
```

Below we have a simple pictorial representation of the 3 components:



3.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.

Simple Loop:

Syntax:

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

Example:

```
-- Syntax of a basic loop in is -

-- LOOP
-- Sequence of statements;
-- END LOOP;

-- EXAMPLE-
DECLARE x number :=
    10;
BEGIN
    LOOP dbms_output.put_line(x);
        x := x + 10;
        IF x > 50 THEN
        exit; END IF;
    END LOOP;
    -- after exit, control resumes here dbms_output.put_line('After Exit x is: ' || x);
END;
```

```
10
20
30
40
50
After Exit x is: 60
```

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:

```
-- Syntax
-- WHILE condition LOOP
-- sequence_of_statements
-- END LOOP;
```

condition is true. It tests the condition before executing the loop body.

```
-- Example

DECLARE a number(2) :=
    10;
BEGIN
    WHILE a < 20 LOOP dbms_output.put_line('value of a: '
        || a);
        a := a + 1;
    END LOOP;
END;
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

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

Syntax:

FOR variable IN [REVERSE] start..end
LOOP

< Action >
END LOOP;

Example:

```
-- Syntax
-- FOR counter IN initial_value .. final_value LOOP
-- sequence_of_statements; -- END
LOOP;

DECLARE a
    number(2);
BEGIN
    FOR a in 10 .. 20 LOOP
        dbms_output.put_line('value of a: ' || a); END LOOP;
END;
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
```

4.)Loop control statements:

Loop control statements change execution from its normal sequence.

i. EXIT statement - The Exit statement completes the loop and control passes to the

statement immediately after the END LOOP.

ii. CONTINUE statement - Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

iii. GOTO statement - Transfers control to the labeled statement.

Example:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

P1. To check if a year is a leap or not.

```
-- To check if a
-- given year is leap year or not
DECLARE year NUMBER :=
   1600;
BEGIN
   -- true if the year is a multiple -- of 4 and not
   multiple of 100.
   -- OR year is multiple of 400.
   IF MOD(year, 4)=0
      AND
      MOD(year, 100)!=0
      OR
      MOD(year, 400)=0 THEN dbms_output.Put_line(year
      || ' is a leap year ');
   ELSE dbms_output.Put_line(year
      || ' is not a leap year.');
   END IF;
END;
```

Output:-

```
2000 is a leap year
```

P2.To find the area and circumference of a circle using a case statement.

```
--Find the area and perimeter of circle
DECLARE

    -- Variables to store area and perimeter area
    NUMBER(6, 2) ; perimeter NUMBER(6, 2) ;

    --Assigning the value of radius radius
    NUMBER(1) := 3;

    --Constant value of PI pi CONSTANT

    NUMBER(3, 2) := 3.14; to_find NUMBER :=

    1;

BEGIN

    --Formula for area and perimeter of a circle area := pi *
    radius * radius; perimeter := 2 * pi * radius;
    CASE to_find
    WHEN 1 THEN dbms_output.Put_line('Area = ' || area);
    WHEN 2 THEN dbms_output.Put_line(' Perimeter = ' || perimeter);

    END CASE;

END;
```

Output:-

```
Area = 28.26
```

P3.To check whether a number is prime or not using while loop and for loop.

```
declare

-- declare variable n, i -- and temp of
datatype number n number; i
number;
```

for loop. Code:-

```
temp number; begin

-- Here we Assigning 10 into n n := 13;

-- Assigning 2 to i i := 2;

-- Assigning 1 to temp temp := 1;

-- loop from i = 2 to n/2 for i in
   2..n/2 loop
          if mod(n, i) = 0
          then temp := 0;
               exit;
          end if;
     end loop;

     if temp = 1
     then dbms_output.put_line('true'); else
     dbms_output.put_line('false');
     end if;
end;
```

Output:

Output:-

```
true
```

P4: To generate fibonacci series

```
declare

-- declare variable first = 0,
-- second = 1 and temp of datatype number first
number := 0; second number := 1;
```

Code:-

```
temp number;

n number := 5; i number; begin

dbms_output.put_line('Series:');

--print first two term first and second dbms_output.put_line(first);
      dbms_output.put_line(second);

-- loop i = 2 to n for i in
      2..n
      loop temp:=first+second;

first := second; second :=
temp;

--print terms of fibonacci series dbms_output.put_line(temp);
end loop; end;
```

Output:-

```
Series:
0
1
1
2
3
5
```

P5: To generate factorial of a given number

```
declare
-- it gives the final answer after computation fac number :=1;
```

Code:-

```
-- given number n -- taking
input from user n number := 6;

-- start block begin

-- start while loop while n > 0
loop

-- multiple with n and decrease n's value fac:=n*fac;
n:=n-1; end loop;
-- end loop

-- print result of fac
dbms_output.put_line(fac);

-- end the begin block end;
```

Output:

```
720
```

P6: To Check Palindrome

```
declare

-- declare variable n, m, temp
-- and temp of datatype number n
      number; m number; temp
      number:=0; rem number;

begin n:=5432112345;
      m:=n;

      -- while loop with condition till n>0 while n>0
```

Code:-

```
      loop rem:=mod(n,10);
            temp:=(temp*10)+rem;
            n:=trunc(n/10);
      end loop; -- end of while loop here

      if m = temp
      then dbms_output.put_line('true'); else
      dbms_output.put_line('false');
      end if;
end;
```

Output:-

```
true
```

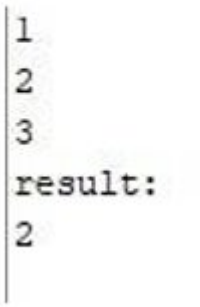
P7: To create a dummy calculator taking two numbers as input as performing selected operation(Addition, subtraction, multiply,etc) and displaying output

---

```
declare n1 number := &1; n2
      number := &2; selector
      number := &1;

begin dbms_output.put_line(n1);
      dbms_output.put_line(n2);
      dbms_output.put_line(selector);
      dbms_output.put_line('result: '); case
      selector
            WHEN 1 THEN dbms_output.put_line(n1 + n2);
            WHEN 2 THEN dbms_output.put_line(n1 - n2);
            WHEN 3 THEN dbms_output.put_line(n1 * n2);
            WHEN 4 THEN dbms_output.put_line(n1 / n2);
            ELSE dbms_output.put_line('give valid input'); end case;
end;
```

Output:



11.Subprograms in PL/SQL

A.Introduction to Procedures:

A PL/SQL subprogram is a PL/SQL block that can be invoked with a set of parameters. A subprogram can be either a procedure or a function. Typically, you use a procedure to perform an action and a function to compute and return a value. At the schema level, subprogram is a standalone subprogram. It is created with the CREATE PROCEDURE or the CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement. A subprogram created inside a package is a packaged subprogram. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement.

Syntax:

Without Parameters

```
CREATE [OR REPLACE] PROCEDURE procedure_name
IS
BEGIN
sql_statement
END;
EXEC procedure_name;
```

With Parameters

```
CREATE [OR REPLACE] PROCEDURE
procedure_name(parameter1,parameter2,...)
IS
```

BEGIN
sql_statement
END;
EXEC **procedure_name**;
A standalone procedure is deleted with the DROP PROCEDURE statement.

1.)PROCEDURE FOR P1

CODE:

```
declare yr
      number;
procedure checkLeap(a IN number) IS begin
if mod(a, 4)=0 and mod(a, 100)!=0 or mod(a,
400)=0 THEN dbms_output.Put_line(a
      || ' is a leap year ');
   else dbms_output.Put_line(a
      || ' is not a leap year.'); end if;
end; begin yr :=
2000;
      checkLeap(yr);
end;
```

Output:-

```
2000 is a leap year
```

2.)PROCEDURE FOR P2

CODE:

```
declare radius number(2); to_find number(1);
      pi_val constant number(3,2) := 3.14;
procedure perArea(a IN number, b IN number, pi IN number) IS begin case b when 1
then dbms_output.Put_line('Area = ' || pi * a * a); when 2 then
dbms_output.Put_line(' Perimeter = ' || 2 * pi * a); end case;
end; begin

      radius := 3; to_find := 1; perArea(radius,
      to_find, pi_val);
end;
```

Output:-

```
Area = 28.26
```

3.)PROCEDURE FOR P3

CODE:

```
declare n
      number;
procedure checkPrime(n IN number) IS i
number := 2; temp number := 1; begin
   for i in 2..n/2 loop
            if mod(n, i) = 0
            then temp := 0;
                  exit;
            end if;
      end loop;

      if temp = 1
      then dbms_output.put_line('true'); else
      dbms_output.put_line('false');
      end if;
end; begin
      n := 19;
      checkPrime(n);
end;
```

Output:-

```
true
```

4.)PROCEDURE FOR P4

CODE:

```
declare n
      number(3);
procedure fib(n IN number) IS first
number := 0; second number := 1; temp
number; begin
dbms_output.put_line('Series:');

      dbms_output.put_line(first);
      dbms_output.put_line(second); for i in
      2..n
      loop temp:=first+second;

first := second; second := temp;
dbms_output.put_line(temp); end
loop; end; begin
      n := 5; fib(n);
end;
```

Output:-

```
Series:
0
1
1
2
3
5
```

5.)PROCEDURE FOR P5

CODE:

```
declare n
      number(3);
```

```
procedure fac(a IN number) IS n
number := a; fac number := 1; begin
while n > 0 loop
fac:=n*fac; n:=n-1; end loop;
dbms_output.put_line(fac); end;
begin
      n := 5; fac(n);
end;
```

Output:-

```
120
```

6.)PROCEDURE FOR P6

CODE:

```
declare n
      number;
procedure checkPalindrome(a IN number) IS n
number := a; m number := a; temp number:=0; rem
number; begin while n>0 loop rem:=mod(n,10);
temp:=(temp*10)+rem; n:=trunc(n/10);
      end loop;

      if m = temp
      then dbms_output.put_line('true'); else
      dbms_output.put_line('false'); end if;
```

```
end; begin n := 123321;
checkPalindrome(n);
end;
```

Output:-

```
true
```

7.)PROCEDURE FOR P7

CODE:

```
declare n1 number := &1; n2
      number := &2; selector
      number := &1;
procedure calc(n1 IN number, n2 IN number, n3 IN number) IS begin
dbms_output.put_line(n1); dbms_output.put_line(n2);
dbms_output.put_line(selector); dbms_output.put_line('result: '); case
selector
            WHEN 1 THEN dbms_output.put_line(n1 + n2);
            WHEN 2 THEN dbms_output.put_line(n1 - n2);
            WHEN 3 THEN dbms_output.put_line(n1 * n2);
            WHEN 4 THEN dbms_output.put_line(n1 / n2);
            ELSE dbms_output.put_line('give valid input'); end case;

end; begin calc(n1, n2, selector);
end;
```

Output:-

```
1
2
1
result:
3
```

B.Introduction to Functions:

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

Syntax:

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

1.)FUNCTION FOR P1

CODE:

```
declare yr
      number;
function checkLeap(a IN number)
return boolean IS isLeap boolean; begin
if mod(a, 4)=0 and mod(a, 100)!=0 or
mod(a, 400)=0 THEN isLeap := true;
   else
      isLeap := false;
   end if; return
   isLeap;
end; begin yr :=
2000;
      dbms_output.put_line(sys.diutil.bool_to_int(checkLeap(yr)));
end;
```

Output:-

```
1
```

2.)FUNCTION FOR P2

CODE:

```
declare radius number(2); to_find number(1);
      pi_val constant number(3,2) := 3.14; res
      number;
function perArea(a IN number, b IN number, pi IN number) return number IS begin
case b when 1 then dbms_output.Put_line('Area = ' || pi * a * a); when 2 then
dbms_output.Put_line(' Perimeter = ' || 2 * pi * a); end case; return null;
end; begin radius := 3; to_find := 1; res :=
perArea(radius, to_find, pi_val);
end;
```

Output:-

```
Area = 28.26
```

3.)FUNCTION FOR P3

CODE:

```
declare n
      number;
function checkPrime(n IN number)
return boolean IS
isPrime boolean; i
number := 2; temp
number := 1; begin
   for i in 2..n/2 loop
            if mod(n, i) = 0 then
            temp := 0;



                  exit;
            end if;
      end loop;

      if temp = 1
      then isPrime := true;
      else isPrime := false;
      end if; return
      isPrime;
end; begin
      n := 53;
       dbms_output.put_line(sys.diutil.bool_to_int(checkPrime(n)));
end;
```

Output:-

4.)FUNCTION FOR P4

CODE:

```
declare n
      number(3); res
      number;
function fib(n IN number)
return number IS
first number := 0; second number := 1;
temp number; begin
dbms_output.put_line('Series:');

      dbms_output.put_line(first);
      dbms_output.put_line(second); for i in
      2..n
      loop temp:=first+second;
first := second;
```

```
second := temp;
dbms_output.put_line(temp); end
loop; return 0; end; begin
      n := 5; res :=
      fib(n);
end;
```

Output:-

```
Series:
0
1
1
2
3
5
```

5.)FUNCTION FOR P5

CODE:

```
declare n
        number(3); res
        number;
function fac(a IN number)
return number IS
n number := a; fac
number := 1; begin
while n > 0 loop
fac:=n*fac;
n:=n-1; end
loop; return
fac; end;
begin
        n := 7; res := fac(n);
        dbms_output.put_line(res);
end;
```

Output:-

```
5040
```

6.)FUNCTION FOR P6

CODE:

```
declare n
      number;
function checkPalindrome(a IN number) return
boolean IS isPalindrome boolean; n number :=
a; m number := a; temp number:=0; rem
number; begin while n>0 loop rem:=mod(n,10);
temp:=(temp*10)+rem; n:=trunc(n/10);
      end loop;

      if m = temp
      then isPalindrome := true;
      else isPalindrome := false;
      end if; return
      isPalindrome;
end; begin n := 123321;
dbms_output.put_line(sys.diutil.bool_to_int(checkPalindrome(n)));
end;
```

Output:-

```
1
```

7.)FUNCTION FOR P7

CODE:

```
declare n1 number := &1; n2
      number := &2; selector
      number := &1; res number;
function calc(n1 IN number, n2 IN number, n3 IN number) return
number IS result number; begin dbms_output.put_line(n1);
dbms_output.put_line(n2); dbms_output.put_line(selector);
dbms_output.put_line('result: '); case selector
            WHEN 1 THEN result := n1 + n2;
            WHEN 2 THEN result := n1 - n2;
            WHEN 3 THEN result := n1 * n2;
            WHEN 4 THEN result := n1 / n2;
            ELSE dbms_output.put_line('give valid input'); end case;
      return result;
end; begin res := calc(n1, n2, selector);
dbms_output.put_line(res);
end;
```

Output:-

```
2
4
3
result:
8
```

12.Cursors in PL SQL

A.Introduction:

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

A cursor is a pointer that points to a result of a query.

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

```
CURSOR c_customers IS
    SELECT id, name, address FROM customers;
```

Opening the Cursor - Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows.

```
OPEN c_customers;
```

Fetching the Cursor - Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows.

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor - Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows:-

```
CLOSE c_customers;
```

B.Advantages and Disadvantages of Cursor:

Advantages:

- Using Cursor we can perform row by row processing so we can perform row wise validation or operations on each row.
- Cursors can provide the first few rows before the whole result set is

assembled. Without using cursors, the entire result set must be delivered before any rows are displayed by the application. So using a cursor, better response time is achieved.

- If we make updates to our without using cursors in your application then we must send separate SQL statements to the database server to apply the changes. This can cause the possibility of concurrency problems if the result set has changed since it was queried by the client. In turn, this raises the possibility of lost updates. So using cursor, better concurrency Control can be achieved.
- Cursors can be faster than a while loop but at the cost of more overhead.

Disadvantages:

- Cursor in SQL is a temporary work area created in the system memory, thus it occupies memory from your system that may be available for other processes. So occupies more resources and temporary storage.
- Each time when a row is fetched from the cursor it may result in a network round trip. This uses much more network bandwidth than the execution of a single SQL statement like SELECT or DELETE etc that makes only one round trip.
- Repeated network round trips can degrade the speed of the operation using the cursor.

C.Types of Cursor:

Implicit Cursor: 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.

In PL/SQL, you can refer to the most recent implicit cursor as the SQL cursor, which always has attributes such as %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT.

S.No	Attribute & Description

1	**%FOUND** Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
2	**%NOTFOUND** The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	**%ISOPEN** Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Example:

```
--cursor
DECLARE
   p_id Products.ProductID%type;
   p_name Products.ProductName%type;
   p_catg Products.Category%type;
   CURSOR p_products is
      SELECT ProductID, ProductName, Category FROM Products;
BEGIN
   OPEN p_products;
   LOOP
   FETCH p_products into p_id, p_name, p_catg;
      EXIT WHEN p_products%notfound;
      dbms_output.put_line(p_id || ' ' || p_name || ' ' || p_catg);
   END LOOP;
   CLOSE p_products;
END;
```

Output:

```
748130-5105 Lotion Hair
206908-3232 Remover Lip
332338-2345 Beauty Face
096419-4823 Beauty Cheek
585230-0077 Sets Cheek
197710-3876 Lotion Eye, Face
533169-3365 Shampoo Body, Hair
588681-0034 Moisturizer Lip
998757-5991 Sets
375163-6428 Primer

Statement processed.
```

Explicit Cursor: 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. Explicit cursors can be declared, opened, fetched and closed as in the preceding section

Syntax:

```
CURSOR cursor_name IS select_statement;
```

Program1 :To count the number of records using implicit cursor.

```
DECLARE
  CURSOR move IS SELECT * FROM EMPLOYEES;
BEGIN
  OPEN move;
  dbms_output.put_line('Number of records='||move%ROWCOUNT);
  CLOSE move;
END;
```

Program2:To display records of 5 least paid employees using an explicit cursor.

```
DECLARE
   CURSOR paid IS
                SELECT EID,Name,Salary FROM EMPLOYEES
                                ORDER BY Salary ASC;
    sal EMPLOYEES.Salary%type;
    eid EMPLOYEES.EID%type;
    name EMPLOYEES.Name%type;
BEGIN
  OPEN paid;
  FOR i IN 1..5
     LOOP
        FETCH paid INTO eid,name,sal;
        dbms_output.put_line(eid||'  '||name||'  '||sal);
    END LOOP;
  CLOSE paid;
```

```
END;
```

Program3:To display records of 5 highest paid employees using explicit cursor.

```
DECLARE
    CURSOR paid IS
                SELECT EID,Name,Salary FROM EMPLOYEES
                                ORDER BY Salary DESC;
     sal EMPLOYEES.Salary%type;
     eid EMPLOYEES.EID%type;
     name EMPLOYEES.Name%type;
BEGIN
  OPEN paid;
  FOR i IN 1..5
     LOOP
        FETCH paid INTO eid,name,sal;
        dbms_output.put_line(eid||'  '||name||'  '||sal);
     END LOOP;
   CLOSE paid;
END;
```

13.Triggers in PL/SQL

A.Introduction :

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events:-

1. A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)

2. A database definition (DDL) statement (CREATE, ALTER, or DROP)

3. A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN) Triggers can be defined on the table, view, schema, or database with which the event is associated

B.Advantages and disadvantages of Triggers:

Advantages of SQL Triggers

1) It helps in maintaining the integrity constraints in the database tables, especially when the primary key and foreign key constraints are not defined.
2) It sometimes also helps in keeping the SQL codes short and simple as I show in the real-life example.
3) It helps in maintaining the track of all the changes (update, deletion and insertion) occurs in the tables through inserting the changes values in the audits tables.
4) Sometimes if the code is not well managed, then it can help in maintaining the database constraints defined on the tables on which the trigger is defined. For example, suppose there is a situation where there is an online learning system in which a user can register in multiple courses.

Disadvantages of Triggers

1) Hard to maintain since this may be a possibility that the new developer isn't able to know about the trigger defined in the database and wonder how data is inserted, deleted or updated automatically.
2) They are hard to debug since they are difficult to view as compared to stored procedures, views, functions, etc.
3) Excessive or over use of triggers can slow down the performance of the application since if we defined the triggers in many tables then they kept automatically executing every time data is inserted, deleted or updated in the tables (based on the trigger's definition) and it makes the processing very slow.
4) If complex code is written in the triggers, then it will slow down the performance of the applications.
5) The cost of creation of triggers can be more on the tables on which frequency of DML (insert, delete and update) operation like bulk insert is high.

C.Syntax of Triggers:

```
CREATE TRIGGER [trigger name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

D.Types of Triggers:

1.)Before event Triggers: BEFORE triggers run the trigger action before the triggering statement is run. This type of trigger is commonly used in the following situations:

When the trigger action determines whether the triggering statement should be allowed to complete. Using a BEFORE trigger for this purpose, you can eliminate unnecessary processing of the triggering statement and its eventual rollback in cases where an exception is raised in the trigger action.

To derive specific column values before completing a triggering INSERT or UPDATE statement.

2.)After event Triggers: AFTER triggers run the trigger action after the triggering statement is run.

3.)Row Trigger:
A row trigger is fired each time the table is affected by the triggering statement. 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 a triggering statement affects no rows, a row trigger is not run. Row triggers are useful if the code in the trigger action depends on data provided by the triggering statement or rows that are affected.

4.)Statement Trigger: A statement trigger is fired once on behalf of the

triggering statement, regardless of the number of rows in the table that the triggering statement affects, even if no rows are affected. For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once. Statement triggers are useful if the code in the trigger action does not depend on the data provided by the triggering statement or the rows affected.

For example, use a statement trigger to:

Make a complex security check on the current time or user

Generate a single audit record.

E.Write programs to demonstrate D:

(a) Before Trigger:

```
CREATE TRIGGER Doctor_Changed
  BEFORE INSERT OR UPDATE OR DELETE
        ON DOCTOR
  AS
IF INSERTING THEN
  BEGIN
```

(b) After Trigger:

```
CREATE TRIGGER Employee_Changed
  AFTER INSERT OR UPDATE OR DELETE
       ON EMPLOYEE
  AS
 IF INSERTING THEN
    BEGIN
```

14.Exception Handling

An exception is an error condition during a program execution. PL/SQL

supports programmers to catch such conditions using EXCEPTION block in the program and an appropriate action is taken against the error condition. There are two types of exceptions −

- System-defined exceptions
- User-defined exceptions

A.System Defined Exception Handling:

PL/SQL provides many pre-defined exceptions, which are executed when any database rule is violated by a program. For example, the predefined exception NO_DATA_FOUND is raised when a SELECT INTO statement returns no rows.

System-defined exceptions are further divided into two categories:

Named system exceptions - They have a predefined name by the system like ACCESS INTO NULL, DUP VAL ON INDEX, LOGIN DENIED etc.

Unnamed system exceptions - Oracle doesn't provide names for some system exceptions called unnamed system exceptions.These exceptions don't occur frequently.These exceptions have two parts code and an associated message. The way to handle these exceptions is to assign names to them using Pragma EXCEPTION_INIT.

B.User Defined Exception Handling:

PL/SQL allows you to define your own exceptions according to the need of your program. A user-defined exception must be declared and then raised explicitly, using either a RAISE statement or the procedure DBMS_STANDARD.RAISE_APPLICATION_ERROR.

Syntax:

```
DECLARE
   my-exception EXCEPTION;
```

C.Write Programs to demonstrate A and B:

:Dealing with System Defined Exceptions:-

1. NO DATA FOUND: It is raised WHEN a SELECT INTO statement returns no rows.

Code:

```
DECLARE temp
    VARCHAR(20);

BEGIN
    SELECT employee_id INTO temp
    FROM Employee WHERE first_name='Gordon' AND last_name='Freeman';

EXCEPTION
    WHEN no_data_found THEN dbms_output.put_line('ERROR');

        dbms_output.put_line('there is no name as');
        dbms_output.put_line('Gordon Freeman in the Employee table');
END;
```

Output:

```
ERROR
there is no name as
Gordon Freeman in the Employee table
```

2. TOO MANY ROWS:It is raised WHEN a SELECT INTO statement returns more than one row

Code:

```
DECLARE temp
    VARCHAR(20);

BEGIN

-- raises an exception as SELECT
-- into trying to return too many rows SELECT first_name
    INTO temp FROM Employee;
    dbms_output.put_line(temp);

EXCEPTION
    WHEN too_many_rows THEN dbms_output.put_line('error trying to SELECT too
        many rows');

END;
```

Output:

```
error trying to SELECT too many rows
```

User Defined Exceptions:-

Code:

```
DECLARE new_wage NUMBER :=
    &x; negative_wage
    EXCEPTION;

BEGIN


    IF new_wage < 0
    THEN RAISE negative_wage;

    UPDATE Employee
    SET hourly_wage = new_wage
    WHERE employee_id = 'emp5'; END
    IF;
EXCEPTION
   WHEN negative_wage THEN dbms_output.put_line('ERROR');
       dbms_output.put_line('hourly_wage cannot be negative');
END;
```

output:

```
ERROR
hourly_wage cannot be negative
```

15.Packages in PL/SQL

A.Introduction to Package:

Packages are schema objects that groups logically related PL/SQL types,

variables, and subprograms.

A package will have two mandatory parts −

- Package specification
- Package body or definition

Packages support the development and maintenance of reliable, reusable code with the following features:

Modularity - Packages let you encapsulate logically related types, variables, constants, subprograms, cursors, and exceptions in named PL/SQL modules. You can make each package easy to understand, and make the interfaces between packages simple, clear, and well defined. This practice aids application development.

Easier Application Design - When designing an application, all you need initially is the interface information in the package specifications. You can code and compile specifications without their bodies. Next, you can compile standalone subprograms that reference the packages. You need not fully define the package bodies until you are ready to complete the application.

Hidden Implementation Details - Packages let you share your interface information in the package specification, and hide the implementation details in the package body.

Added Functionality - Package public variables and cursors can persist for the life of a session. They can be shared by all subprograms that run in the environment. They let you maintain data across transactions without storing it in the database.

Better Performance - The first time you invoke a package subprogram, Oracle Database loads the whole package into memory. Subsequent invocations of other subprograms in the same package require no disk I/O.

B.Package Specification:

The specification is the interface to the package. It just DECLARES the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package.

All objects placed in the specification are called public objects. Any subprogram not in the package specification but coded in the package body is called a private object.

Syntax:

CREATE [OR REPLACE] PACKAGE <package_name>

IS

<sub_program and public element declaration>

.

.
END <package name>

C.Package Body:

The package body has the codes for various methods declared in the package specification and other private declarations, which are hidden from the code outside the package.

The CREATE PACKAGE BODY Statement is used for creating the package body.

It consists of the definition of all the elements that are present in the package specification. It can also have a definition of elements that are not declared in the specification, these elements are called private elements and can be called only from inside the package.

Syntax:

CREATE [OR REPLACE] PACKAGE BODY <package_name>
IS
<global_declaration part>
<Private element definition>
<sub_program and public element definition>
.
<Package Initialization>
END <package_name>

16.Miscellaneous Queries(REFER Annexure 1)

>tables and insertion of data for this question:

```
CREATE TABLE Department
(
Dept_NO number(12) primary key,
Dept_Name varchar2(100) not null,
Dept_Location varchar2(100) not null
);
```

```
CREATE TABLE Employee(
 Emp_ID varchar2(30) primary key,
 Emp_Name varchar2(100) not null,
 Job varchar2(20),
 Salary number(12),
 Manager_ID varchar2(5) check (Manager_ID='NO' OR Manager_ID='YES'),
 Dept_NO number(12),
 constraint fkey Foreign key(Dept_NO) REFERENCES Department(Dept_NO)
);
```

```
insert into Department values(1,'Finance','Bangalore');
insert into Department values(2,'Research','Mumbai');
insert into Department values(3,'Marketing','Ludhiana');
insert into Department values(4,'HR','Chandigarh');
```

```
insert into Employee values('E006','Amit','Worker',100000,'NO',2);
insert into Employee values('E018','Ashish','Worker',40000,'NO',3);
insert into Employee values('E010','Aniket','HR',130000,'YES',4);
insert into Employee values('E003','Akanksha Jha','Worker',20000,'NO',4);
insert into Employee values('E012','Ankita Thakur','Manager',100000,'YES',1);
insert into Employee values('E001','Abhishek','Manager',130000,'YES',2);
insert into Employee (Emp_ID,Emp_Name,Salary) values ('E029','Ayush',3000000);
insert into Employee values('E004','Akanksha Sharma','Manager',100000,'YES',4);
```

```
insert into Employee values('E004','Akanksha Sharma','Manager',100000,'YES',4);
insert into Employee values('E015','Anuj','HR',180000,'YES',3);
insert into Employee values('E016','Archit','Manager',140000,'YES',3);
insert into Employee values('E008','Ananya','HR',175000,'YES',2);
insert into Employee values('E014','Anshal','HR',200000,'YES',1);
insert into Employee values('E007','Amrit','Worker',60000,'NO',1);
```

Now there are some queries that we have to perform:

QUERY 1 : Give Total Salary issued by each Department

QUERY2 : Average salary of each job

QUERY3 : Give the total salary paid to each job within each Department QUERY4 : Give Total, minimum, maximum and average salary of employees job wise for department numbers 1 and 2 and displaying rows having average salary > 1,00,000

QUERY5 : Give name, salary of employee having salary greater than minimum salary of department ‘Research’.

QUERY6 : Give name, salary of employees having salary greater than average salary of Department ‘HR’.

QUERY7 : List all details of the department whose ManagerID = ‘E006’.

QUERY8 : List all employees belonging to Department Finance

QUERY9 : List name of employees who do the same job as that of employee having EmployeeID = ‘E008’.

QUERY10 : List the name of employees who do the same job as an employee having EmployeeID = ‘E012’ and get the salary greater than the salary of that employee.

QUERY11 : List the name and salary of Employees who earn more than the Department of ‘Research’.

QUERY12 : List the name of employees who do not manage any Employee QUERY13 : List name of employees who have at least one person reporting to them. QUERY14 : Give the name of employees having salary (a)less than / (b)greater than minimum salary of ‘HR’ Department and are not of ‘HR’ department. QUERY15 : List the name of the highest paid employee.

QUERY16 : List the name of lowest paid employee.

Query1:

```
SELECT Dept_NO, SUM(Salary) AS Total_Salary FROM Employee GROUP BY Dept_NO;
```

Output:

Results | Explain | Describe | Saved SQL | History

DEPT_NO	TOTAL_SALARY
1	360000
2	405000
4	250000
-	300000
3	360000

5 rows returned in 0.02 seconds Download

Query2:

```
////2
SELECT Job, AVG(Salary) AS Average_Salary FROM Employee GROUP BY Job;
```

Output:

Results | Explain | Describe | Saved SQL | History

JOB	AVERAGE_SALARY
HR	171250
Worker	55000
-	300000
Manager	117500

4 rows returned in 0.00 seconds Download

Query3:

```
////3
SELECT dept_no ,job, SUM(salary) Total_Salary FROM  employee
GROUP BY  dept_no,job order by dept_no,job;
```

Output:

DEPT_NO	JOB	TOTAL_SALARY
1	HR	200000
1	Manager	100000
1	Worker	60000
2	HR	175000
2	Manager	130000
2	Worker	100000
3	HR	180000
3	Manager	140000
3	Worker	40000
4	HR	130000
4	Manager	100000
4	Worker	20000
		300000

Query4:

```
////4
select Job,sum(Salary) as Total,min(Salary) as Min,max(Salary) as Max,avg(Salary) as Avg from Employee
where Dept_NO=1 or Dept_NO=2 group by Job having avg(Salary)>100000;
```

Output:

Results | Explain | Describe | Saved SQL | History

JOB	TOTAL	MIN	MAX	AVG
HR	375000	175000	200000	187500
Manager	230000	100000	130000	115000

2 rows returned in 0.00 seconds Download

Query5:

```
/////5
select e.Emp_Name,e.Salary from Employee as e left join Department as d ON e.Dept_NO=d.Dept_NO
where e.Salary>(select min(Salary) from Employee as e left join Department as d ON e.Dept_NO=d.Dept_NO where Dept_Name='Research');
```

Output:

Results Messages

	Emp_Name	Salary
1	Abhishek	130000
2	Ananya	175000
3	Aniket	130000
4	Anshal	200000
5	Anuj	180000
6	Archit	140000
7	Ayush	3000000

Query6:

```
/////6
select e.Emp_Name,e.Salary from Employee as e left join Department as d ON e.Dept_NO=d.Dept_NO
where e.Salary>(select avg(Salary) from Employee as e left join Department as d ON e.Dept_NO=d.Dept_NO where Dept_Name='HR');
```

Output:

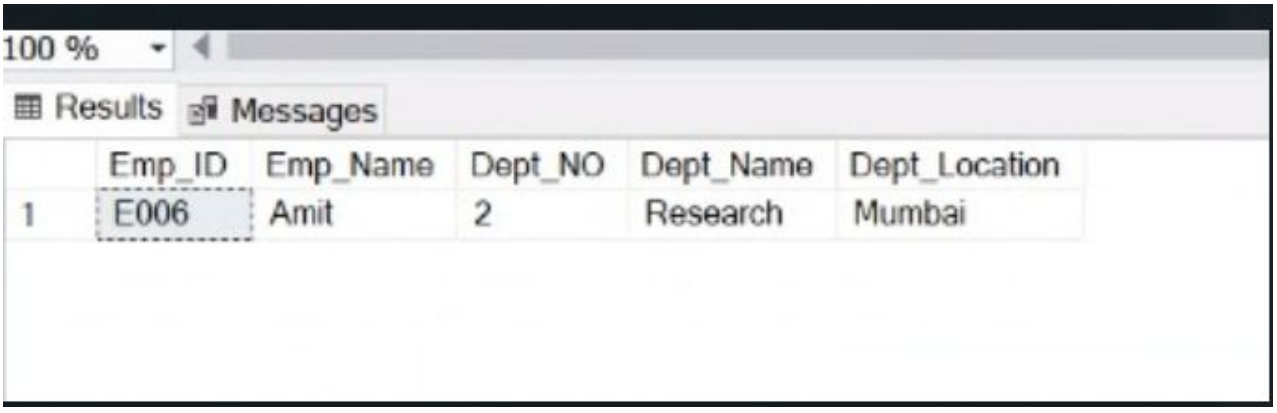
Results Messages

	Emp_Name	Salary
1	Abhishek	130000
2	Akanksha Sharma	100000
3	Amit	100000
4	Ananya	175000
5	Aniket	130000
6	Ankita Thakur	100000
7	Anshal	200000
8	Anuj	180000
9	Archit	140000
10	Ayush	3000000

Query7:

```
//////7
select e.Emp_ID,e.Emp_Name,d.Dept_NO,d.Dept_Name,d.Dept_Location FROM Employee as e join Department as d ON e.Dept_NO=d.Dept_NO
where e.Emp_ID=6;
```

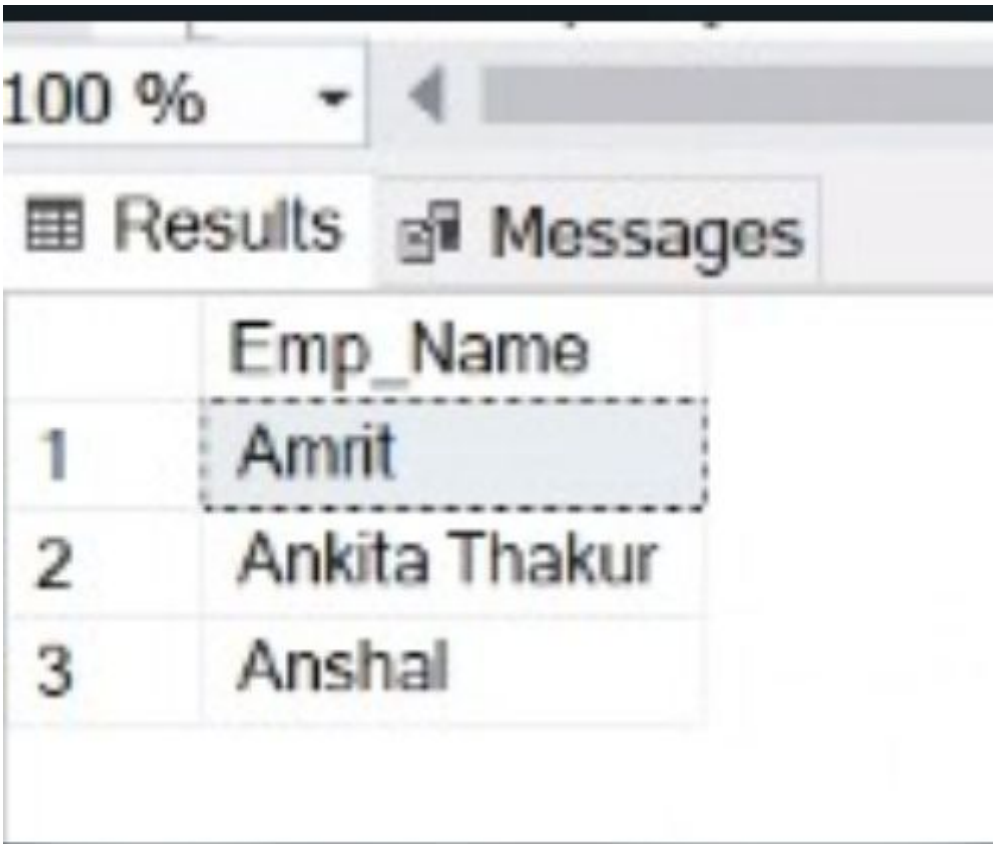
Output:



Query8:

```
////8
select e.Emp_Name from Employee as e join Department as d on e.Dept_NO=d.Dept_NO where d.Dept_Name='Finance';
select * from Employee
```

Output:



Query9:

```
/////9
select Emp_ID,Emp_Name from Employee where Job=(select Job from Employee where Emp_ID='E008');
```

Output:

Results | Explain | Describe | Saved SQL | History

EMP_ID	EMP_NAME
E010	Aniket
E008	Ananya
E014	Anshal
E015	Anuj

4 rows returned in 0.02 seconds Download

Query10:

```
//////10
select Emp_ID,Emp_Name from Employee where Job=(select Job from Employee where Emp_ID='E012') AND Salary>(select Salary
from Employee where Emp_ID='E012');
```

Output:

Results | Explain | Describe | Saved SQL | History

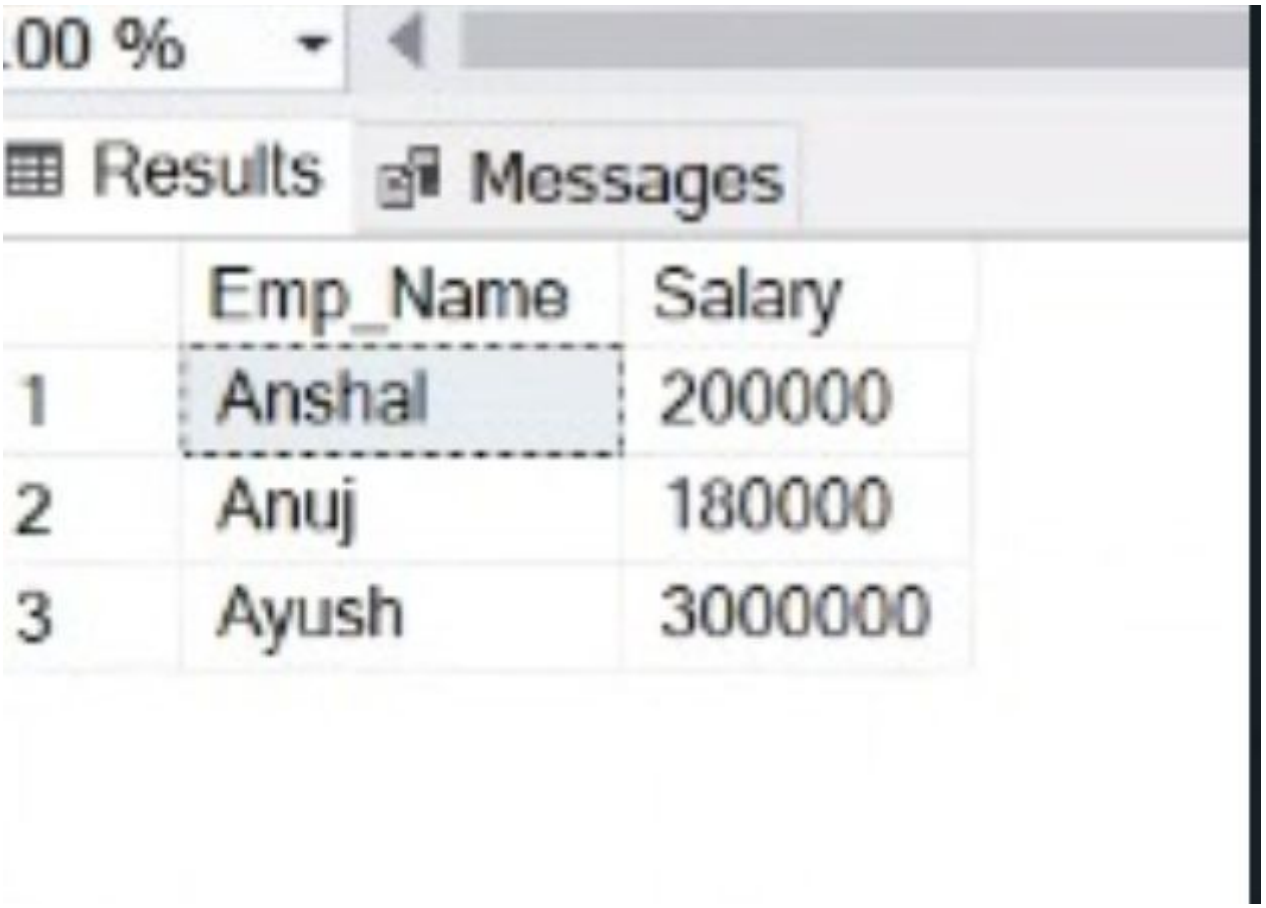
EMP_ID	EMP_NAME
E001	Abhishek
E016	Archit

2 rows returned in 0.02 seconds Download

Query11:

```
///////11
select e.Emp_Name,e.Salary from Employee as e left join Department as d ON e.Dept_NO=d.Dept_NO
where e.Salary>(select max(Salary) from Employee as e left join Department as d ON e.Dept_NO=d.Dept_NO where Dept_Name='Research');
```

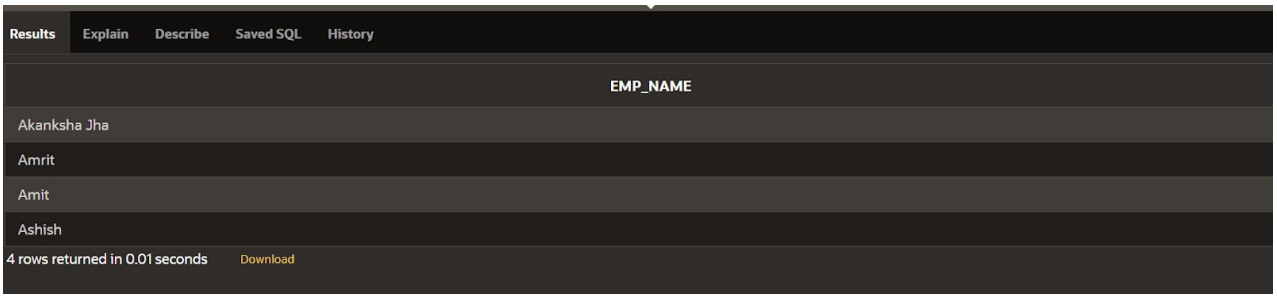
Output:



Query12:

```
/////////12
select Emp_Name from Employee where Manager_ID='NO';
```

Output:



Query13:

```
///////13
select Emp_Name from Employee where Manager_ID='YES';
```

Output:

EMP_NAME
Aniket
Ananya
Ankita Thakur
Abhishek
Archit
Anshal
Akanksha Sharma
Anuj

Query14:

```
////14
select e.Emp_Name,e.Salary from Employee as e left join Department as d ON e.Dept_NO=d.Dept_NO
where d.Dept_Name !='HR' AND
e.Salary>(select min(Salary) from Employee as e left join Department as d ON e.Dept_NO=d.Dept_NO where Dept_Name='HR');
```

Output:

00 %

Results Messages

	Emp_Name	Salary
1	Abhishek	130000
2	Amit	100000
3	Amrit	60000
4	Ananya	175000
5	Ankita Thakur	100000
6	Anshal	200000
7	Anuj	180000
8	Archit	140000
9	Ashish	40000

Query15:

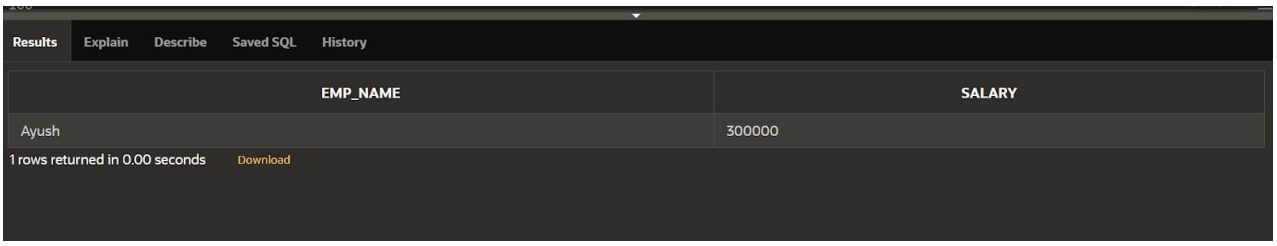
```
////15
select Emp_Name,Salary from Employee where Salary=(select max(Salary) from Employee);
```

Output:

Query16:

```
////16
select Emp_Name,Salary from Employee where Salary=(select min(Salary) from Employee);
```

Output:



So, we completed all the 16 queries on the database given in the problem statement.

Conclusion and Recommendations

The purpose and objective of creating a shopping management System, that is to create a management system that allows buyers to manipulate the database records of various products and order,buyers,shipment,payments related information and various other tasks performed by them and to store any new record that has relation to the shop in any way.

Creating a database and management system for a shop helped us to be a better programmer in SQL and gave us a better idea of the subject. This project has made us more skilful and proficient in SQL and working with the environment of oracle server(apex). After completing this project we are able to understand much better about the field of DBMS and how it works.
Soon after completing this project we can perform well also we can try to do tasks related to databases and its management due to the immense knowledge after this wonderful project of Shopping Management System.

Thank you….