Database Languages

- **Data definition language (DDL)**
  - Create
  - Alter
  - DROP
  - Truncate
  - Comment
  - Rename
- **Data manipulation language (DML)**
  - Insert
  - update
  - delete
  - Merge
  - Call
  - Explain plan
  - Select (DQL)
- **Data Control language (DCL)**
  - Grant
  - Revoke
- **Transaction Control language (TCL)**
  - Commit
  - RollBack
  - Save point
  - Auto commit.

Data Types Supported in MySQL:-

1. Char :-(size) A fixed length string can contains letters, numbers and special characters.

The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1.

2. Varchar (size) :- A variable length string can contains letters, numbers and special characters) The size parameter specifies the maximum column length in characters - can be from 0 to 65535

Integer (size):- (small width data)

INT (size) :- It represent non fractional data (large width data)

Float (size, d):- A floating point number. The total number of digits is specified in size. the number of digits after the decimal point is specified in parameter d. This syntax is deprecated in mysql 8.0.17 and it will be remove in future mysql.

Double (size, d):- A normal size floating point number. The total number of digits is specified in size. the number of digits after the p decimal point is specified in d parameter.

Date:- It specifies date use (' ')

format: yyyy : mm : dd.

time (fsp): A time

format: hh:mm:ss.

the supported range is from -838:59:59 to 838:59:59.

Year:- A year in 4 digit format.

DDL Commands :-

1. Create :-

Syntax :- Create table tablename (column name1 datatype (size), col2 datatype(size), - - - coln datatype(size))

Eg:- Create table student (id integer, name char(10));

To verify table structure

describe tablename;

(or)

desc tablename.

Eg:- describle student.

To insert values in a table :-

1. insert into tablename values (val1, val2 - - - valn);
2. insert into tablename (col names) values (values);
3. insert into tablename values(id, 'name'), (id, 'name');

Ex:- 1. insert into student values (1267, 8, 'Anuradha');

2. insert into student (col1, col2) values (val1, val2);

To retrieve the data :-

Select * from tablename (tells the attributes)

(or)

Select only one column or reverse the columns.

Ex:-

Select id as rollname, name

Here it can changes column name.

→ S, %.S, --a:-

mysql> Select * from student where name like 'S%'

↓

Selects names with S.

'%a' - Ends with a.

'-a' - position of a.

Where (to retrieve Specialtype)

Alter:-

Alter is used to add, Create, Remove the Column or change the width (or) change the data size of a Column.

To add the Column:-

alter table tablename add Columnname datatype (size);

Ex:- alter table Student add location varchar(20);

Update:-

Update tablename set columnname = value;

Update tablename set Columnname = value where Condition;

Ex:- Update Student set location = 'hyderabad' location where id=128;

Delete :-
using this command only data removed.

Syntax :-

delete from tablename where condition;

delete from tablename ; (All data removed from table)

Ex:- delete from Student where id = 1285

Delete a Column :-

alter table tablename drop (permanently deleted.) Column Columnname;

Ex:- alter table Student drop Column location;

Change the datasize of a Column :-

alter table tablename modify columnname datatype (size);

Ex:- alter table Student modify name char(3);

Cycle-I :-

Aim:-

Creating, Altering and dropping of Tables and inserting rows into a Tables (use Constraints while Creating Constructing Tables). Examples using Select Command

mysql > Show databases ;

Database
information-schema
mysql
Performance-schema
Sample
sys
world.

mysql > Create database Sample ;

Query ok, 1 row affected (0.01 sec)

mysql > use Sample ;

Database changed

Creating Table :-

mysql > Create table Employee (id int (30), name char(30), occupation char (30), working hours int (30));

Query ok, 0 rows affected, 2 warnings (0.06secs)

mysql > Show Tables;

Tables_in_Sample
Employee.

1 row in Set (0.04 secs)

mysql > desc Employee ;

Field	Type	Null	Key	Default	Extra.
Id	int	Yes		Null	
name	char(30)	yes		Null	
occupation	char(30)	yes		Null	
working_hours	int	Yes		Null.	

4 rows in set (0.01 secs).

Inserting :-

mysql > insert into Employee values (1, 'Joseph', 'Bussiness', 12), (2, 'stephen', 'Doctor', 11), (3, 'mark', 'Engineer', 10), (4, 'peter', 'Teacher', 8), (5, 'Alice', 'police', 8);

Select :-

mysql > Select * from Employee ;

Id	name	occupation	working_hours.
1	Joseph	Bussiness	12
2	stephen	Doctor	11
3	mark	Engineer	10
4	peter	Teacher	8
5	Alice	police	8.

mysql > Select name, occupation from Employee ;

name	Occupation
Joseph	Bussiness
Stephen	Doctor
mark	Engineer
Alice	Police
Peter	Teacher

mysql> Select name from Employee where occupation = 'Doctor';

name
stephen

mysql> Select occupation from Employee where occupation like '%r';

Occupation
Doctor
Engineer
Teacher.

ALTER:- mysql> Select * from Employee;

Id	name	occupation	working_hours.
1	Joseph	Bussiness	12
2	stephen	Doctor	11
3	Mark	Engineer	10
4	Peter	Teacher	8
5	Alice	police	8

MySQL> alter table Employee add City varchar(30);

Id	name	occupation	working-hours	City
1	Joseph	Bussiness	12	Null
2	Stephen	Doctor	11	Null
3	mark	Engineer	10	Null
4.	peter	Teacher	8	Null
5.	Alice	police	8	Null.

Update :-

mySQL> update Employee Set city = mysore;
mySQL> Select * from Employee;

Id	name	occupation	working-hours	city.
1	Joseph	Bussiness	12	mysore
2	Stephen	Doctor	11	mysore
3	mark	Engineer	10	mysore
4	peter	Teacher	8	mysore
5	Alice	police	8	mysore.

Drop :-

Before : Deleting / Dropping.

mySQL > Select * from Employee;

Id	name	occupation	working-hours	City
1	Joseph	Bussiness	12	mysore
2	stephen	Doctor	11	mysore
3	mark	Engineer	10	mysore
4	Alice	Police	8	mysore
5	Peter	Teacher	8	mysore.

After Dropping;-

mysql> alter table Employee drop column working-hours.

mysql> Select * from Employee;

Id	name	occupation	working hours	City
1	Joseph	Bussiness		mysore
2	Stephen	Doctor		mysore
3	mark	Engineer		mysore
4	Peter	Teacher		mysore
5	Alice	police		mysore

Task-1 :-

Create a Table student with appropriate data types And performs the following Queries.

Roll number, student name, date of birth, branch And year of study.

```
mysql> Create database Sample;
Query OK, 1 row affected (0.01secs)
mysql > show databases;
```

Database.
information_schema
mysql
performance_schema
Sakila
sample
sys
world.

```
mysql> use Sample;
Database changed.
mysql> Create table student (rollno int(40),
       name char(40), Date of Birth,
       Branch Varchar(50), Year of study int(40));
Query OK, 0 rows affected, 3 warnings.
mysql > show Tables;
```

Tables_in_sample
student.

1. Insert 8 rows in a table.

mysql> Insert into student (Rollnumber, student name, Date of Birth, Branch, Year of study)

Values (1201, 'Ramya', '25-04-2005', 'IT', 2);
(1202, 'surya', '28-03-2003', 'CSM', 4);
(1203, 'John', '19-09-1999', 'ECE', 2);
(1204, 'Sushutha', 05-05-2005, NULL, 2);
(1205, 'stephen', '08-08-2008', 'ECE', 4),
(1206, 'Susan', '25-11-1995', 'CSE', 2);
(1207, 'marcus', '20-12-2000', 'IT', 1);
(1208, 'manoj', '27-1-2003', 'CSM', 3);

mysql> Select * from student;

Rollno	Student name	Date of Birth	branch	Year of study.
1201	Ramya	2005	IT	2
1202	Surya	2003	CSM	4
1203	John	1999	ECE	2
1204	Sushutha	2005	NULL	2
1205	Stephen	2008	ECE	4
1206	Susan	1995	CSE	2
1207	marcus	2000	IT	1
1208.	manoj.	2003.	CSM	3.

2. List all students of all Branches.

mysql> select name, branch from student;

Student Name	Branch
Ramya	IT
surya	CSE
John	ECE
Susrutha	Null
stephen	ECE
Susan	CSE
marcus	IT
manoj	CSE

3. List student name starts with 's'.

mysql> select name from student where name like 'S%';

Studentname
Surya
Susrutha
stephen
Susan.

mysql> Select * from Student;

rollno	Student name	Date of birth	branch	year of study.
1201	Ramya	2005	IT	2
1202	Surya	2003	CSE	4
1203	John	1999	ECE	2
1204	Susrutha	2005	Null	2
1205	stephen	2008	ECE	4
1206	Susan	1995	CSE	2
1207	marcus	2000	IT	1
1208	manoj.	2003.	Csm.	3.

4. List names whose name Contains 's' as second

mysql> select Studentname from student where Studentname like '_s%';

Studentname
Surya
Susrutha
stephen
Susan.

5. List student names whose name Contains 's' as the third literal:-

mysql> Select Studentname from student where Studentname like '__s%';

student name
Susrutha
Susan.

6. List student names whose name Contains two 's' anywhere in the name:

mysql> Select Studentname from student where Studentname like '%s%s%';

Student name
Susrutha
Susan.

7. List students whose branch is Null:

```
mysql> Select * from student where Branch
       is Null;
```

Student name.
Susrutha.

8. List students of CSE and ECE who were born after 1980:

```
mysql> select * from student where (Branch = 'CSE' or
       Branch = 'ECE') and Date of Birth > '1980-01-01';
```

rollno	Student name	Date of birth	branch	Year of Study
1203	John	1999	ECE	2
1205	stephen	2008	ECE	4
1206	susan	1995	CSE	2.

9. List all students in reverse order of their names:

```
>select * from student order by studentname
 Desc;
```

rollno	student name	Date of Birth	branch	Year of study.
1205	stephen	2008	ECE	4
1202	surya	2003	CSE	4
1206	susan	1995	CSE	2
1204	Susrutha	2005	Null	2
1201	Ramya	2005	IT	2
1208	manoj	2003	csm	3
1207	marcus	2000	IT	1
1203	John	1999	ECE	2

10. Delete students of any branch whose name starts with 's':

mysql> Delete From student where Studentname like 's%';

Rollno	student name	Date of Birth	branch	year of study
1201	Ramya	2005	IT	2.
1208	John	1999	ECE	2
1207	marcus	2000	IT	1
1208	manoj	2003	CSE	3.

11. Update the branch of CSE student to ECE:

mysql> Update student
Set branch = 'ECE'
where Branch = 'CSE';

before updating;

Rollno	Student name	Date of birth	branch	Year of study
1201	Ramya	2005	IT	2
1202	Surya	2003	CSE	4
1203	John	1999	ECE	2
1204	Sushrutha	2005	NuLL	2
1205	stephen	2008	ECE	
1206	Susan	1995	CSE	4
1207	marcus	2000	IT	2
1208	manoj	2003	CSE	1 3

After updating :-

Rollno	Student name	Date of birth	branch	Year of study.
1201	Ramya	2005	IT	2
1202	Surya	2003	ECE	4
1203	John	1999	ECE	2
1204	Susrutha	2005	NULL	2
1205	Stephen	2008	CSE	4
1206	Susan	1995	ECE	2
1207	marcus	2000	IT,	1
1208	manoj	2003	csm CSE	3.

12. Display student names padded with '*' after their names

mysql> select Concat(studentname, '*****') As PaddedName From Student;

Student name
Ramya ****
Surya*****
John ****
Susrutha ****
Stephen *****
Susan ****
Marcus ****
manoj *****

10

mysql> show databases;

Database
information schema
mysql
Performance-schema.
ramya
Sample
it24765a1211.
Sys
world

mysql > use it24765a1211
Database changed.

Sailor's Table Creation:-

mysql> Create Table sailors (sid integer primary key, sname varchar (10), rating integer, age ~~float~~ integer (4));

Query OK, 0 rows affected, 0 warnings (0.2secs)

mysql > desc sailors;

field	type	Null	key	Default	Extra.
sid	int	No	PRI	Null	
sname	varchar (10)	yes		Null	
rating	integer	yes		Null	
age	float(4)	yes		Null.	

mysql> insert Sailor values (12, 'ramya', 5, 30),
(21, 'dustin', 7, 45), (22, 'brutus', 8, 46), (23, 'lubber', 9, 48)
(24, 'Andy', 9, 50), (25, 'rusty', 10, 36), (26, 'Horatio', 7, 35)
(27, 'Zobra', 10, 20);
Query ok, 3 row affected (0.00sec)

mysql> Select * from sailors;

sid	Sname	Rating	Age
12	Ramya	5	30
21	Dustin	7	45
22	Brutus	8	46
23	Lubber	9	48
24	Andy	9	50
25	rusty	10	36
26	Horatio	7	35
27	Zobra	10	20.

Boats table Creation :-

mysql> Create table boats (bid integer primary key,
bname varchar(10), color varchar(10);
Query ok, 0 rows affected (0.02secs)

mysql> desc boats;

Field	Type	Null	Key	Default	Extra
bid	int	no	PRI	NULL	
bname	varchar(10)	Yes		NULL	
color	varchar(10)	Yes		NULL	

mysql> insert into boats values (101, 'Interlake', blue'), (102, 'Interlake', 'Red'), (103, 'Clipper', 'green'), (104, 'marine', 'red');

Query ok, 4 rows affected.

mysql> Select * from Boats;

bid	bname	Color.
101	Interlake	blue
102	Interlake	red
103	clipper	green
104	marine	red.

4 rows in a set (0.00sec)

Creation of reserves table:-

Create Table reserves (sid integer, bid integer, day date, primary key (sid, bid, day));

Query ok, 0 rows affected (0.02sec).

mysql> Show tables;

Tables in it24765a1211
Boats
reserves.
Persons
Sailors
Student
5 rows in Set (0.00 sec)

mysql> describe reserves;

mysql> alter table reserves add foreign key(sid)
references sailors (sid);

Query ok, 10 rows affected

mysql> alter table reserves add foreign key (bid)
references boats (bid);

Query ok, 10 rows affected (0.07sec)

mysql> describe reserves;

Field	Type	Null	Key	Default	Extra
Sid	int	No	PRI	Null	
Bid	int	NO	PRI	NULL	
day	date	No	PRI	NULL	

3 rows in set (0.00sec).

23/1

Task-2

1. find the names of the Sailors.

mysql> Select Sname from Sailors;

Sname
Ramya
Dustin
brutus
Lubber
Andy
Rusty
Horatio
Zobra

8 rows in Set (0.00 sec).

2. print the Sname as Sailorsname from the Sailors table.

mysql> Select Sname as Sailorsname from Sailors;

Sailorsname
Ramya
Dustin
brutus
lubber
Andy
Rusty
Horatio
Zobra

3. find the Sailors information whose rating is 7.

mysql> select * from Sailors where rating = 7;

Sid	Sname	rating	age
21	Dustin	7	45
26	Horatio	7	35

4. Find the sailors information whose rating is better than 7.

mysql> Select * from Sailors where rating > 7;

Sid	Sname	rating	age
22	brutus	8	46
23	Lubber	9	48
24	Andy	9	50
25	Rusty	10	36
27	Zobra	10	20

5. find the Sailors information whose rating is better than 7.

mysql> Select * from Sailors where rating >= 7;

Sid	Sname	Rating	age.
21	Dustin	7	45
26	Horatio	7	35
22	brutus	8	46
23	Lubber	9	48
24	Andy	9	50
25	Rusty	10	36
27	Zobra	10	20

6. Find the information all Sailors whose rating is less than 7.

mysql> Select * from Sailors where rating < 7;

sid	Sname	rating	age.
12	Ramya	5	30

7. Find the information of sailors whose rating is less than or Equal to 7.

```
mysql> Select * from Sailors where rating <=7;
```

sid	sname	rating	age
12	Ramya	5	30
21	Dustin	7	45
26	Horatio	8	46.

8. find the average age of the sailors.

```
mysql> Select avg(age) from Sailors;
```

avg(age)
38.75

9. find the youngest age of the sailors.

```
mysql> Select min(age) as youngest_age from
Sailors;
```

youngest_age.
20

10. print all the information of youngest sailor.

```
mysql> Select * from Sailors where age=(select
min(age) from Sailors);
```

sid	sname	rating	age.
27	Zobra	10	20.

11. Find the oldest age of the Sailors.

mysql> Select max(age) as oldest_age from Sailors;

oldest_age
50.

12 Print all the information of oldest Sailor.

mysql> Select * from Sailors where age = (select max(age) from Sailors);

Sid	Sname	Rating	age.
24	Andy	9	50

13. find the Sum of the ages of Sailors.

mysql> Select Sum(age) from Sailors;

Sum(age)
310.

14. Print all the Sailors information whose age is better than the average age of the Sailor.

mysql> Select * from Sailors where age > (select avg(age) from Sailors);

Sid	Sname	Rating	age
21	dustin	7	45
22	brutus	8	46
23	Lubber	9	48
24	Andy	9	50.

15. Print all the Sailors name in sorted order.

mysql> Select Sname from Sailors order by Sname;

Sname
Andy
brutus
dustin
Horatio
Lubber
Ramya
rusty
Zobra.

16. Print all the Sailors information based on the rating increasing order

mysql> select * from Sailors order by rating;

Sid	Sname	rating	age
12	Ramya	5	30
21	dustin	7	45
26	Horatio	7	35
22	brutus	8	46
23	Lubber	9	48
24	Andy	9	50
25	rusty	10	36
27	Zobra	10	20

17. Print all the Sailors information based on the rating in descreasing order.

mysql> select * from sailors order by rating Desc;

sid	sname	rating	age
25	rusty	10	36
27	zobra	10	20
23	lubber	9	48
24	Andy	9	50
22	brutus	8	46
21	dustin	7	45
26	Horatio	7	35
12	ramya	5	30

18. print all the sailors information based on the age increasing order.

mysql> Select * from sailors order by age;

sid	sname	rating	age
27	zobra	10	20
12	ramya	5	30
26	Horatio	7	35
25	rusty	10	36
21	dustin	7	45
22	brutus	8	46
23	lubber	9	48
24	Andy	9	50

19. Compute and print present age and age after 2 years of the Each Sailor.

```
mysql> Select age as present_age, age+2 as age_after_2 years from Sailors;
```

present_age	age_after_2 years.
30	32
45	47
46	48
48	50
50	52
36	38
35	37
20	22.

Vaps
23/1

Group By

⇒ Purpose :- Used to group rows with the same values in specified columns into summary rows, like "count of reservations per Boat".

⇒ Works with Aggregate functions :- (count(), Sum(), Avg(), max(), min() etc..)

Syntax :-

```
Select column1, aggregate_function (column2)
  From table
  group by column1;
```

Having :-

⇒ Purpose :- Filters grouped results based on a condition. It is similar to where clause but works on aggregated data.

⇒ Used after grouped By.

Syntax :-

```
Select column1, aggregate_function (column2)
 from table
Group By column1
Having aggregate_function (column2) condition;
```

Creation of Sales Table :-

mysql> Create Table Sales (Product id int, Category Varchar(100), quality int, price int);

mysql> desc Sales ;

Field	Type	Null	Key	Default	Extra
Productid	int	Yes		Null	
Category	Varchar(100)	Yes		NULL	
quantity	int	Yes		NULL	
price	int	Yes		NULL	

4 rows in Set (0.01 secs)

mysql> insert into Sales values (1, 'electronics', 10, 500), (2, 'furniture', 9, 400), (3, 'organic', 5, 100), (4, 'plastic', 8, 1200), (5, 'gadgets', 10, 1000), (6, 'Organic', 8, 900), (7, 'Electronics', 9, 500), (8, 'furniture', 10, 900);

Query ok, 8 rows effected (0.015secs)

Productid	Category	Quantity	Price
1	Electronics	20	500
2	furniture	9	400
3	organic	5	100
4	plastic	8	1200
5	gadgets	29	1000
6	organic	8	900
7	Electronics	9	500
8	furniture	10	900

(1) Group by Category and Calculate total quantity sold:

```
Select Category, Sum(Quantity) AS total quantity
  from Sales group by Category;
```

Category	Total Quantity
Electronics	28
Furniture	19
organic Kitchen Accessories	13
Plastic	8
Gadgets	29

(2) Filter grouped results (using HAVING) to show Categories where total quantity sold > 20:

```
Select Categories, Sum(Quantity) AS Total Quantity
  from Sales
  Group by Category;
  Having Sum(Quantity) > 20;
```

Category	Total quantity
Electronics	28
Gadgets	29

3. Group by Category and find the average price of products in Each Category, but only for Categories with an average price > 400.

```
Select Category, AVG(price) AS averageprice
  from Sales
  Group by category
  Having AVG(price) > 400;
```

Category	Average price.
Electronics	500.000
furniture	750.00
organic	500.00
gadgets	1000
plastics	1200

1. Find the number of reservations made by Each Sailor:

Select Sid, Count(*) AS reservation_count.
→ From reserves
→ Group by Sid;

Sid	reservation_count.
22	4
31	3
64	3.

2. Find Sailors who have made more than 3 reservations:

Select Sid, Count(*) As reservation_count.
→ From reserves
→ group by sid
→ Having Count(*) > 3;

sid	reservation_count.
22	4.

3. Find the number of reservations for Each boat:

Select bid, Count(*) AS reservation_count
→ from ~~Customers~~ reserves
→ Group by bid;

Bid	reservation_count.
101	2
102	3
103	3
104	2.

4. find Boats that have been reserved more than 5 times.

Select bid, Count(*) AS reservation_count.
→ From reserves
→ Group by bid
→ Having Count(*) > 5;

Empty set.

5. Find the average age of Sailors who have reserved boats

```
Select R.bid, Avg (S.age) AS avg_age.
→ From Reserves R
→ join Sailors S on R.sid = S.sid
→ group by R.bid;
```

Bid	avg_age.
101	40.25
102	45.333333336
103	45.333333336
104	50.25.

6. Find Sailors with an average rating of reservations greater than 8:

```
Select R.sid, Avg (S.Rating) AS avg_rating
→ from Reserves R
→ join Sailors S on R.sid = S.sid
→ group by R.sid
→ Having Avg (S.rating) > 8;
```

Sid	Avg_rating
74	9.0000

7. List the Boats reserved on more than 2 Unique dates:

```
Select Bid, count (Distinct day)
  AS Unique_days
  From Reserves
  group by bid
Having Count (Distinct day) > 2;
```

Bid	Unique_days
102	3
103	3.

8. Find the total number of reservations for Each Sailors who reserved a specific:

```
Select R.sid, Count (*) AS Reservation_count
→ From Reserves R
→ join boats B on R.bid = B.bid
→ where B.color = 'Red'
→ group by R.sid;
```

Sid	Reservation_count
22	2
31	2
64	1.

Task:-

1. Orders :-

Order Id :- Unique ID for Each other

Customer Id :- ID of the Customer who replaced the order

Order Date :- Date of the Order

Total Amount :- Total Amount of the order.

2. order Details :- ID of the order

Order Id (foreign key) :- ID of the product in Order

Product Id :- Quantity of the product Ordered

Quantity :-

Price :- price per unit of the product

3. Customers :-

Customer Id (primary key) :- Unique ID for Each Customer

Customer Name :- Name of the Customer

City :- City where the customer lives.

Creation :-

Customers table :-

Create Table Customers (Customer id int primary key,
, Cname varchar(20), city varchar(20));

Query ok, 0 rows affected.

Orders Table :-

Create Table Orders (orderid int, customerid int,
orderdate date, totalamount float, primary key(Order
id), foreign key (Customer id) references customers
(customer id));

Query ok, 0 rows affected.

Orders details Table :-

Create Table Ordersdetails (orderid int, productid int, quantity int, price float, primary key (productid), foreign key (orderid) references Orders(orderid));

Query ok, 0 rows affected.

Insertion :-

insert Customers Values (101, 'Ramya', 'Vij'), (102, 'vinnu', 'mylavaram'), (103, 'surya', 'hyd'), (104, 'Pavani', 'Tiruvuru'), (105, 'honey', 'landon');

Query ok, 5 rows affected.

Customerid	Cname	City.
101	Ramya	Vij
102	Vinnu	mylavaram
103	Surya	Hyd
104	Pavani	Tiruvuru
105	honey	landon

insert Orders Values (1, 101, '2025-4-12', 1000), (2, 102, '2024-06-11', 1500), (3, 103, '2025-04-25, 1300), (4, 104, '2025-02-04', 2500), (5, 105, '2025-04-12', 500);

Query ok, 5 rows affected.

(6, 102, '2025-02-04', 200), (7, 102, '2025-04-25, 300), (8, 102, '2025-02-04, 100), (9, 102, '2025-02-04, 150), (10, 104, '2025-11-05, 550);

Select * from Orders;

Orderid	Customerid	Orderdate	Totalamount
1	101	2025-4-12	1000
2	102	2024-6-1	1500
3	103	2025-4-25	1300
4	104	2025-2-4	2500
5	105	2025-4-12	500
6			

insert Ordersdetails values (1,25,15,1000), (2,24,8, 1500), (3,23,5,1300), (4,22,50,2500), (5,21,10,500), (6,26, (8,269), (7,27,9,300), (8,28,5,100), (9,29,10,150), (10,30,8,550).

Select * from Orderdetails;

Orderid	productid	quantity	price
5	21	10	500
4	22	50	2500
3	23	55	1300
2	24	8	15000
1	25	15	1000
6	26	8	200
7	27	9	300
8	28	5	100
9	29	10	150
10	30	8	550

Queries Using Groupby and Having :-

1. find the total Amount Spent by Eachother.

mysql> Select CustomerId, Sum (totalAmount) from Orders groupby (CustomerID);

CustomerID	TotalAmount.
101	1000 2250.
102	2250
103	1300
104	2500
105	500

2. List Customers who spent more than Rs. 500.

Select Customers.Customerid, Sum (Orders.totalAmount) from Customers join Orders on Customers.CustomerID = Orders.CustomerID group by (Customers.CustomerID) having Sum (Orders.totalAmount) > 500;

Customerid	Sum (Orders.totalAmount)
101	1000
102	2250
103	1300
104	2500.

3. Find the average orders value for each city.

Select Customers.city, avg(orders.totalAmount) from Customers join orders on Customers.CustomerID = orders.customerID group by (customers.city);

city	avg(orders.totalAmount
vij	833.6
mylavaram	933.3
hyd	42.25
Tiruvuru	319.00
London	488.00

4. List cities where the average orders value is greater than Rs.300.

Select Customers.city, avg(orders.totalAmount) from Customers join orders on Customers.CustomerID = orders.customerID group by (Customers.city) having avg(orders.totalAmount) > 300;

city	avg(orders.totalAmount
vij	833.6
hyd	319
mylavaram	933.3
London	488

5. Find products ordered more than 50 times.

Select productID, sum(quantity) from orderdetails group by productID having sum(quantity)>50;

productID	sum(quantity)
23	55

6. List Customers who placed more than 5 orders.

Select C.CustomerID, Customers.cname, count(orders.orderID) AS TotalOrders from Customers C join orders o on C.customerID = o.CustomerID group by C.CustomersID, C.CustomersName Having Count(O.OrderID) >5;

CustomerID	cname	Total Order.
101	Ramya	10
102	vinnu	50
103	Surya	55
104	Pavani	8
105	honey	15

select * from orders;

orderid	customerid	orderdate	Totalamount.
1	101	2025-4-12	1000
2	102	2024-6-12	1500
3	103	2025-4-25	1300
4	102	2025-2-4	2500
5	105	2025-4-12	500
6	102	2025-2-4	200
7	102	2025-2-4	300
8	102	2025-2-4	100
9	102	2025-2-4	150.
10	104	2024-11-5	100

6. List Customers who placed more than 5 orders.

Select Customers CustomerID, Customers.Cname, Count (Orders.OrderID) AS Total order from Customers C join Orders O on C.CustomerID = O.CustomerID group by C.CustomerID, C.Customersname having Count O.Order ID) > 5;

Customer ID	Cname	Total order
102	Vinnu.	6.

Cycle-4

Single row functions :-

Single row functions can be categorized into five. These will be applied for each row and produces individual output for each row.

- ⇒ Numeric functions
- ⇒ String functions
- ⇒ Date functions

⇒ Numeric functions :-

⇒ Abs	⇒ Log	⇒ Least.
⇒ Sign	⇒ Ceil	
⇒ Sqrt	⇒ Floor	
⇒ mod	⇒ Round	
⇒ Power	⇒ Trunc.	
⇒ Exp	⇒ Greatest	

(a) ABS :-

→ Absolute value is the measure of magnitude of value

→ Absolute value is always a positive number.

Syntax :- abs (value).

Ex :- Select abs (5), abs(-5), abs(0), abs (null) from dual;

Abs (5)	abs(-5)	abs(0)	abs (null)
5	5	0	Null.

abs(5)	abs(-5)	abs(0)	abs(null)
5	5	0	Null
5	5	0	Null
5	5	0	Null
5	5	0	Null
5	5	0	Null

(b) Sign :- Sign gives the sign of a value.

Syntax :- Sign(value)

Ex:-

mysql> Select sign(5), sign(-5), sign(0), sign(null), sign(a) from dual;

sign(5)	sign(-5)	sign(0)	sign(Null)
1	-1	0	Null.

(c) SQRT :- This will give the square root of the given value.

Syntax :- Sqrt (value) — here value must be positive.

Ex:- mysql> Select sqrt(4), sqrt(0), sqrt(null), sqrt(1) from dual;

Sqrt(4)	Sqrt(0)	sqrt(Null)	sqrt(1).
1	0	Null	1

(d) MOD :- This will give the Remainder

Syntax :- mod (value, divisor)

Ex:- Sql> Select mod(7,4), mod(1,5), mod(null, null), mod(0,0), mod(-7, 4) from dual;

mod(7,4)	mod(1,5)	mod(null,null)	mod(0,0)	mod(-7,4)
3	1	Null	0	-3

f. Power:- Power is the ability to raise a value given Exponent.

Syntax:- Power (value, Exponent)

Ex:- SQL> select power(2,5), Power(0,0), Power(1, Power(null,null), Power(2,-5) from dual;

Power(2,5)	Power(0,0)	power(null,null)	power(2,-5)
32	1	Null	-32.

(g) EXP:- This will raise e value to the given power.

Syntax:- exp(value)

Ex:- SQL> select exp(1), exp(2), Exp(0), Exp(null) Exp(-2) from dual;

EXP(1)	EXP(2)	EXP(0)	EXP(null)	EXP(-2)
2.7182818	7.389056	1	Null	0.135335

(h) Log:- This is based on 10 based logarithm.

log(10,value) - here value must be greater than zero which is positive only.

Ex:-

SQL> select log(10,100), log(10,2), log(10,1), log(10, null) from dual;

log(10,null)	log(10,2)	log(10,1)	log(10,100)
Null	0.3d02999566	0	2.

(i) CEIL :- This will produce a whole number that is greater than or Equal to the specified value.

Syntax :- Ceil(value)

Ex :- SQL> Select Ceil(5), Ceil(5.1), Ceil(-5) Ceil(-5.1), Ceil(0) from dual;

Ceil(5)	Ceil(5.1)	Ceil(-5)	Ceil(-5.1)	Ceil(0)
5	6	-5	-5	0.

(j) Floor :- This will produce a whole number that is lesser than or Equal to the specified value.

Syntax :- floor (value)

Ex :- Select floor(5), floor(5.1), floor(-5), floor(-5.1), floor(0) from dual;

Floor(5)	Floor(5.1)	Floor(5)	Floor(-5.1)	Floor(0)
5	5	-5	-6	0.

(K) Round :- This will rounds number to given number of digits of precision.

Syntax :- Round (value, precision)

Ex :- Select Round (123.2345), Round (23.2345, 2), Round (123.2354, 2) from dual;

Round (123.2345)	Round(123.2345, 2)	Round(123.234, 2
123	123.23	123.24.

(L) Trunc :- This will truncates or chops off digits of precision from a number.

Syntax :- trunc (value, precision).

Sql> trunc (123.2345), trunc (123.2345,2), trunc (123.2354,
from dual;

Trunc (123.2345)	Trunc (123.2345,2)	trunc (123.235
123	123.23	123.23

(M) Greatest :- This will give the greatest number

Syntax:- greatest (value1, value2, .. valuen)

Ex:- SQL> Select greatest (1,2,3), greatest (-1,-2,
from dual;

Greatest (1,2,3)	Greatest (-1,-2,-3)
3	-1.

→ if all the values are zero then it will display zero

→ If all the parameters are nulls then it will display nothing

→ If any of the parameters is null it will display nothing.

(N) least :- This will gives the least number.

Syntax:- Select least (value1, value2 . . . valuen).

Ex:- SQL> Select least (1,2,3), least (-1,-2,-3) from dual;

least (1,2,3)	least (-1,-2,-3)
1	-3

STRING functions :-

→ Init cap	→ LTrim
→ Upper	→ Rtrim
→ Lower	→ Trim
→ Length	→ Concat
→ Rpad	→ substr
→ Lpad	→ Instr.

(a) Initcap :- This will Capitalize the initial letter of the string.

Syntax :- init cap (string)

Ex :- mysql> select init cap ('computer') from dual;

Initcap
Computer

(b) Upper :- This will Convert the string into uppercase

Syntax :- upper (string)

Ex :- mysql> select upper ('computer') from dual;

Upper
COMPUTER

(c) Lower :- This will Convert the string into lowercase

Syntax :- ~~upper~~ lower (string)

mysql> select Lower ('COMPUTER') from dual;

Lower
computer

(d) Length :- This will give length of the string.

Syntax :- length (string)

mysql> Select length ('Ramya') from dual;

length.
5

(e) Rpad:- This is allows you to pad the right side of a column with any set of characters.

Syntax:- Select Rpad ('string/computer', length, [padding_char])

Ex:- mysql> select Rpad ('Computer', 15, '@'), Rpad ('Ramya', 15, '*') from dual;

Rpad ('Computer', 15, '@')	Rpad ('Ramya', 15, '*')
Computer@@@@@@@@	Ramya**********

(f) Lpad:- This is allows you to pad the left side of a column with any set of characters.

Syntax:- Select Lpad ('string, length, padding [unwanted char]),

Ex:- mysql> Select Lpad ('Computer', 15, '@'), Lpad ('Ramya', 15, '*') from dual.

Lpad ('Computer', 15, '@')	Lpad ('Ramya', 15, '*')
@@@@@@@@Computer	**********Ramya

(g) LTrim:- This will trim off unwanted characters from the left End of the string.

Syntax:- Ltrim (string, [unwanted_chars])

Ex:- SQL> select LTrim ('computer', 'co'), LTrim ('computer', 'com') from dual;

LTrim (LTrim)	
mputer	puter

mysql> select LTrim ('computer', puter'), LTrim ('computer', 'computer') from dual;

LTrim ('C')	LTrim ('c',
Computer	Computer

→ If you haven't specify any unwanted characters it will display Entire string.

(b) RTrim :- This will trim off unwanted characters from the right End of the string.

Syntax:- RTrim (string, (string [unwanted_chars])

Ex:- mysql> select RTrim ('computer', 'er'), RTrim ('computer', 'ter') from dual;

RTrim	RTrim
Computer	Compu

(i) Trim :- This will trim of unwanted characters from both sides of the string.

SQL> select trim ('i' from indian) from dual;

Trim
ndian

(j) Concat :- This will be used to combine two string only.

select Concat ('computer', 'operator') from dual;

Concat ('computer', 'operator')

Computeroperator.

(k) Substr :- This will be used to Extract sub strings.

Syntax :- substr (string, strat_char_count [no_of_chars])

Ex:- SQL> Select substr ('computer', 2), substr ('computer', 2, 5), substr ('computer', 3, 7) from dual;

substr (subst	substr).
omputer	omput mputer

→ If no_of chars parameters is negative then it will display nothing.

→ If both parameters expect string are null & zeros then it will display nothing.

→ If no_of_ chars parameter is greater than the length of string then it ignores and calculates based on the original string length.

→ If start_chr_count is negative then it will extract the substring from right End.

1	2	3	4	5	6	7	8
C	o	m	p	u	t	e	r
-8	-7	-6	-5	-4	-3	-2	-1

(I) Instr :- this will allows you for searching through a string for set of characters

Syntax:- instr(string, search_str [start_chr_count [Occurence]])

Eg:- Select instr('information', 'o', 4,1), instr('information', 'o', 4,2) from dual;

Instr('Information', 'o', 4,1)	Instr('Information', 'o', 4,2)
4	10.

→ If you are not specifying start_chr_count and Occurence then it will start search from beginning and finds first occurences only.

→ If both parameters start_chr_count and occurence are null, it will display nothing.

DATE Functions :-

Mysql offers a wide range of functions to work with dates and times.

⇒ Getting Current Date and Time :-

- Curdate() or Current_Date() : Returns the current date (YYYY-MM-DD).

Eg:- Select Curdate();

Curdate()
2025-02-13.

- Curtime() & Current_time() : Returns the current time (HH:mm:ss).

Select Curtime();

Curtime()
18:40:23.

- Now() or sysdate() or current_timestamp():
Returns the current date and time (YYYY-MM-DD HH:MM:SS)

Select Now();

Now
2025-02-13 18:41:40.

Extracting Parts of a Date or time :-

⇒ Year(date): Extracts the year.

Eg: Select Year('2025-2-13');

Year('2025-2-13)
2025

⇒ month(date): Extracts the month (1-12).

Eg: Select month('2025-02-13');

month('2025-2-13')
2.

⇒ Day(date) or Day of month(date): Extracts the day of the month (1-31).

Eg: Select Day('2025-02-13');

Day('2025-02-13')
13

⇒ Dayname (date) : Returns the name of the weekday (E.g:- 'Monday').

Eg: Select dayname (now());

Dayname (now())
Thursday.

⇒ Day of week (date): Returns the weekday index (1=Sunday, 2= monday, ... 7= Sunday).

Eg:- Select dayof week (now());

Day of week (now())
5

⇒ Day of year (date): Returns the day of year (1-366).

Eg:- Select Day of year (now());

Day of year (now())
44

⇒ Hour (time): Extracts the Hour.

Eg:- Select Hour (now());

Hour (now())
18

⇒ minute (time): Extracts the minute

Eg: Select minute(now());

minute(now())
44

⇒ second (time): Extract the second.

Eg: Select second(now());

Second now())
49

Formatting Dates and times :-

⇒ Date_format(date, format) : Formats a date according to a specified format string.

Eg:- Select date_format('2025-02-13', '%w, %m, %y');

date_format('2025-02-13', '%w, %m, %y')
4, 02, 13, 25

⇒ Time_format(time, format) : Formats a time according to a specified format string.

Eg:- Select time_format('14:30:00', '%H:%i:%s');

Time_format('14:30:00)
12:00:25.

Modifying Dates and Times :-

⇒ Date_add(date, interval Expr unit) or Adddate (date, interval Expr Unit): Adds a time interval to a date.

Eg:- Select Date_add('2025-02-13', interval 5 Day)

date_add('2025-02-13', interval 5 day)
2025-02-18.

⇒ Date_Sub(date, interval Expr unit) or Subdate(date, interval Expr unit): Subtracts a time interval from a date.

Eg:- Select date_sub('2025-02-13', interval 5 month);

date_sub('2025-02-13', interval 5 month);
2024-09-13

⇒ Addtime(time, time_exprs): Adds a time value to a time or datetime value.

Eg:- select addtime('14:30:00', '02:00:00'); Add 2 hours to a time:

Addtime('14:30:00', '02:00:00')
16:30:00

Calculating differences blw dates and times.

⇒ Datediff(date1, date2): Returns the number of days blw two dates.

Eg:- Select datediff('2023-10-25', '2023-10-20');

Datediff('2023-10-25', '2023-10-20')
5

⇒ Timediff(time1, time2): Returns the difference blw two times.

Eg:- Select timediff('12:30:00', '10:00:00');

Timediff('12:30:30', '10:00:00)
02:30:00.

Converting blw date/time types and strings.

⇒ str_to_date(str, format): converts a string to a date according to specified format.

Eg:- Select str_to_Date ('13/02/2025', '%d/%m/%

str_to_Date ('13/02/2025', '%d/%m/%Y')
2020-02-13

⇒ Date(Expr): Extracts the date part from a datetime Expression.

Eg:- Select date(now());

date(now())
2025-02-13

⇒ Time(expr): Extracts the time part from a datetime Expression

Eg:- Select time(now());

time(Now())
18:56:09

Other useful Functions :-

⇒ Last_day(date): Returns the last day of the month for a given date.

Eg:- Select Last_day(Now());

last_day(now())
2025-02-28

⇒ Week(date): Returns the week numbers of given date

Eg:- Select Week(now());

Week(now())
6

⇒ Quarter(date):- Returns the quarter of the year (1-4) for a given data.

Eg:- Select quarter(now());

quarter(now())
1

Cycle-5

Views in Sql:-

⇒ A view in Sql is a virtual that is based on result of a SELECT query.

→ It doesn't store the data itself but provides a way to look at data from one or more tables in structured way.

→ We can think of a view as a stored query that can be treated like a table for querying purpose.

Syntax for Creating a view:-

Create view view_name AS SELECT Column1, Col2 ... from table_name WHERE Condition;

Ex:-

Create Table "clients". This Table will Contain all the basic information about the client such as ID, name, and Email ID.

clients Table:-

ID	Name	Email_ID.
1	George	ge.com
2	David	da.com
3	Chris.	ch.com.
4	Morrison	mo.com
5	Brian	Br.com

Create table "client's_location". This Table will have information about the clients-location, including ID, Country, and Country_code.

clients_location Table:-

ID	Country	Country-Code.
1	INDIA	IND
2	SPAIN	ESP
3	FRANCE	FRA
4	ENGLAND	ENG
5	POLAND	POL.

1. Create a Simple view from our clients table.
This view will store all the columns from the clients_Table.

```
mysql > Create view V_clients AS Select * fromclients;
mysql > Select * from V_clients.
```

output:

ID	Name	Email_Id.
1	George	ge.com
2	David	da.com
3	chirs	ch.com
4	morrison	mo.com
5	Brain	br.com.

2. Creating a view from multiple Table. We can also Create Views from multiple tables. In this example, we will Create a view that will store ID, name, Country, and Country_code Columns data for all five Elements.

```
mysql> Create view V_clients_Loc_All as Select clients.ID, clients
.Name, clients_Location.country, clients_Location.Countrycode
from clients, clients_Location where clients.ID = clients_Location.
ID;
mysql > Select * from V_clients_Loc_All;
```

output:

ID	Name	Country	Country_code
1	George	INDIA	IND
2	David	SPAIN	ESP
3	chirs	FRANCE	FRA
4	morrison	ENGLAND	ENG
5	Brian	POLAND	POL.

Inserting a new row in a view:-

For the inserting Example, we will use the v_clients view that we had Created earlier. The view already had three rows and we will insert the fourth one using the Insert INTO statement.

```
mysql> insert into v_clients values (6,'Akash','update_later');
mysql> Select * from v_clients;
```

Output:-

ID	Name	Email_Id.
1	George	ge.com
2	David	da.com
3	Chris	ch.com
4	Morrison	mo.com
5	Brian	br.com
6	Akash	update-later.

Updating a row in a view:-

Like inserting, we Can also Update a row in a view. If you notice the previous Example where we inserting a new row, we have inserted Email_id Column value as 'update_later'. That's what we will be Updating here.

```
mysql> Update v_clients Set Email_id = 'aa.com' where ID=6;
mysql> Select * from v_clients;
```

Output:-

ID	Name	Email_ID
1	George	ge.com
2	David	da.com
3	Chris	ch.com
4	Morrison	mo.com
5	Brian	br.com
6	Aakash	aa.com.

Deleting a row in a view:-

Just like how we are inserted and updated a row, we can also deleting one using the delete statement. for this, we will Explain delete the row with ID=6 using the where Condition.

```
mysql> Delete From V_clients where ID= 6;
mysql> Select * from V_clients;
```

Output:-

ID	Name	Email_Id.
1	George	ge.com
2	David	da.com
3	Chris	ch.com
4	Morrison	mo.com
5	Brian	br.com

Dropping a view:-

Just like a table, we can also drop a view using the 'DRop statement'. Let's Drop our V_clients View and then try to Execute it to see the results.

```
mysql> DRop view V_clients;
mysql > Select * from V_clients;
```

Output:-

Table 'V_clients' doesn't Exist.

As we can see, the Output says that the table V_clients does not Exist. This Confirms that the views was dropped.

Task-1

You have an Employees table, departments table and Salaries Table.

Tables:-

Employees: Employee Id, name, department id.

departments: department_id, department_name.

Salaries: Employee_id, salary.

mysql> ~~Create table Employees (e_id int, Name varchar(~~

~~dept_id~~

mysql> Create table dept (deptID int primary key, deptname varchar(20));

insert into dept values (11, 'IT'), (22, 'Sales'), (33, 'fin

(44, 'customer care'); (55, 'feedback');

mysql> Select * from dept;

deptID	deptName
11	IT
22	Sales
33	finance
44	Customer Care.
55	feedback.

Create Table ~~Emp~~ Salary (EmpID int primary key, Salary float);

insert into salary values (1, 100000), (2, 600000), (3, (700000), (4, 2500000), (5, 2500000), (6, 500000), (7, 600000), (8, 200000);

mysql> Select * from Salary;

EmpID	Salary.
1	100000
2	600000
3	700000
4	250000
5	2500000
6	5000000
7	600000
8	200000.

mysql> Create Table Employ (EmpID int, name varchar(50), dept ID int);

mysql> alter table Employ add foreign key (EmpID) References Pay (EmpID);

mysql> alter table Employ add foreign key (deptID) References dept (deptID);

mysql> insert into Employ ~~values~~ (EmpID, name, deptID) values (1, 'Ramya', 11), (2, 'Surya', 22), (3, 'vinnu', 33) (4, 'honey 44), (5, 'Sai', 55), (6, 'Pavani', 55), (7, 'manoj', 44), (8, 'lakshmi', 33);

mysql> select * from Employ;

EmpID	name	deptID.
1	Ramya	11
2	surya	22
3	vinnu	33.
4	honey	44.
5	Sai	55
6	Pavani	55
7	manoj	44
8	lakshmi	33

1. Create a view to list Employees name with Department name.

```
mysql> Create view v_name as Select Employ.name,
dept.name from dept, Employ where dept.deptID=
Employ.deptID;
mysql> select * from v_name;
```

Name	deptName
Ramya	IT
Surya	Sales
vinna	finance
Lakshmi	finance.
manoj	Customes care
Honey	Customes Care
Sai	feedback
Pavani	feedback

2. Create a view to show Employees with Salaries >750000

```
mysql> Create view largesalary as Select employ.name
Employ.EmpID, salary.salary from Employ, salary where
Employ.EmpID=salary.EmpID AND salary.salary>500000;
mysql> select * from largesalary;
```

Name	EmpID	salary.
Surya	2	600000
vinna	3	700000
Sai	5	2500000
Pavani	6	500000
manoj	7	600000
Lakshmi	8	200000.

3. Create a view to Calculate Average Salary per department.

mysql> Create view Average as select dept.dept name, avg(pay.salary) from dept join Employ on dept.deptID = Employ.deptID join salary on salary.EmpID = Employ.EmpID group by dept.deptname;

mysql> Select * from Average;

Dept name	avg(pay.salary)
IT	100000
Sales	600000
finance	1350000
Customer Case feedback	3012500
	37500000.

Task-2

You have an orders, order_items and products Table.

Tables:-

→ product: product_id, product_name, price

→ orders: order_id, order_date

→ order_items: order_id, product_id, quantity.

mysql> Create table products (product_id int, Product_name varchar(100), price decimal(10,2));

mysql> insert into products values (111, 'Oreo', 20.00), (222, 'dark fantasy', 30.00), (333, 'dairy milk', 60.00), (444, 'Kitkat', 15.00), (555, 'bagbon', 40.00); (666, 'Perk', 10.00);

mysql> select * from products;

Product_id	product_name	price
111	8ico	20.00
222	dark fantasy	30.00
333	dairy milk	60.00
444	kit kat	15.00
555	Bagbon	40.00
666	Pesk	10.00

mysql> Create Table Orders (order_id, int, order_date date);

mysql> insert into orders values (101, '2025-03-01'), (102, '2025-03-02'), (103, '2025-03-03'), (104, '2025-03-04'), (105, '2025-03-05');

mysql> select * from Orders;

Orderid	order_date
101	2025-03-01
102	2025-03-02
103	2025-03-03
104	2025-03-04
105	2025-03-05

mysql> Create Table order_items (order_id int, product_id int, quantity int);

mysql> Insert into order_items Values (101, 111, 2), (101, 333, 1), (103, 555, 1), (104, 444, 4);

mysql> select * from order_items

order_id	product_id	quantity
101	111	2
102	333	1
103	222	3
104	555	1
105	444	4

1. Create a view to show Total sales by product.

mysql> Create view total_sales_by_product as select p.product_id, p.product_name, Sum(oi.quantity * p.price) as total_sales from product p join order_items oi on p.product_id = oi.product_id group by p.product_id, p.product_name;

mysql> select * from total_sales_by_product;

product_id	product_name	Total_sales
111	oreo	40.00
333	dairymilk	60.00
222	dark fantasy	90.00
555	bournbon	40.00
444	kitkat	60.00
666	~~kitkat~~	60.00

2. View to show products not ordered.

mysql> Create view product_not_ordered As select p.product_id, p.product_name from products p left join order_items oi on p.product_id = oi.product_id where oi.order_id is null;

mysql> Select * from product_not_ordered;

product_name
Desk

Cycle-6

MYSQL STORED procedure:-

→ A procedure (often called a stored procedure) is a collection of pre-compiled SQL statements stored inside the database.
→ It is a subprogram in regular computing language
→ A procedure always contains a name, parameter lists and SQL commands.

Stored procedure features:-

→ stored procedures increases the performance are created, they are compiled and stored in the database.
→ stored procedures reduced the traffic b/w application and database servers.
→ A procedure is always secure.

Syntax for Creating a stored procedure:

```
Delimiter //--changing the delimeter to allow for semicolon
                in the procedure definition.

Create procedure procedure_name (parameter1 datatype, parameter
2, datatype);
BEGIN
  -- SQL statements to go here
     SQL - statement-1;
     SQL - statement-2;
     -- other SQL logic
  -- END //
DELIMITER ; --Reset the delimeter back to the default
Semicolon.
```

Example 1:- Simple stored procedure.

Let's create a stored procedure to display a student's name and marks from a student Table.

Step1: Create the Students table and insert records

```
mysql> CREATE TABLE STUDENTS(
        student_id int primary key,
        Student_name varchar(30),
        marks INT
      );
```

Step2: Create the stored procedure

```
DELIMITER $$
mysql> CREATE PROCEDURE get_student_marks()
    BEGIN
        Select student_name, marks
        From Students;
    END $$
mysql> DELIMITER;
```

Step3: Call the stored procedure

```
CALL get_student_marks();
```

Inserting rows:-

```
mysql> Insert into student values (101,'Ramya',99), (102,'Surya',
98), (103,'Vinnu',97), (104,'honey',96), (105,'pavani',97);
Query OK, 5 rows affected
mysql> select * from students;
```

Student_id	Student_name	marks.
101	Ramya	99
102	Surya	98
103	Vinnu	97
104	Honey	96
105	Pavani	97

mysql> Call get_student_marks();

Student_name	marks.
Ramya	99
Surya	98
Vinnu	97
Honey	96
Pavani	97.

Example 2:- stored procedure with IN parameter.

Let's create a procedure that accepts a student ID and returns their name and marks.

Create the stored procedure with IN parameter.

```
mysql> DELIMTER $$
→ Create procedure get_student_by_id(IN p_id INT)
BEGIN
    SELECT S_name, marks
      FROM students
    WHERE s_id = p_id;
  END $$
```

Call the stored procedure :-

CALL get_student_by_id (102);

Student_name	marks.
Surya	98

Example-3:-

Stored procedure with multiple IN parameters.

Let's create a procedure that accepts the student ID and the two marks, and update the student's marks in database.

Create the stored procedure with parameters.

```
mysql> DELIMITER $$
CREATE PROCEDURE update_student_marks(IN p_id INT,
IN new_marks INT)
BEGIN
    Update students
        SET marks = new marks
        WHERE S_id = p_id;
    SELECT 'student marks updated successfully'. As message;
    END $$
mysql> DELIMITER ;
```

Cal the stored procedure:

```
CALL update_student marks (101, 95);
```

message.
Student marks updated successfully.

Example 4:- Stored procedure with OUT parameter.

Let's create a stored procedure where:

The OUT parameter will return the maximum marks from the Students Table.

```
mysql> DELIMITER &&
CREATE procedure display_max_mark (OUT highestmark
    INT)
BEGIN
    SELECT MAX (marks) INTO highest mark from students;
    END &&
mysql> DELIMITER ;
```

Call the stored procedure:

```
CALL display_max_mark(@m);
```

mysql> select @m;

@m
98.

Example 5:- Stored procedure with INOUT parameter

→ In this procedure, we can use INOOT Parameter 'var1' of Integer type.

→ Its body part first fetches the marks from the table with the specified id and then stores it into same variable var1.

→ the var1 first acts as the IN parameter and then OUT parameter.

```
mysql> DELIMITER &&
CREATE PROCEDURE display_marks(INOUT var1 INT)
BEGIN
select marks INTO var1 From students where s_id =
  var1;
END &&
DELIMITER ;
```

After successful Execution we can call the procedure as follows:

Set the input parameter:

```
Set @M = '103';
```

Query ok, 0 rows affected.

Call the stored procedure by having passing @m as parameter

```
Call display_marks (@m);
```

Output:

mysql> select @m;

@m
97.

Task 7

1. Create the following Tables:

Create table products (Product_id INT, AUTO_Increment primary key, product_Name Varchar(100), price Decimal (10,2), quantity INT

);

— Create Customers Table.

Create table customers (customer_id int Auto_increment primary_key, customer_name Varchar (100), Email Varchar (100));

Create Table Orders (Orderid int Auto_Increment primary key, customer_id INT, Order_date date, total_amount Decimal (10,2), Foreign key (customer_id) References Customers (customer_id));

-- Create Order_items Table.

Create Table order_items (order_id int, product_id int, quantity int, price Decimal (10,2), primary key (order_id, Product id), Foreign key (order_id) References Orders (order_id), Foreign key (product_id) References Product (product_id));

2. Insert ~~into~~ relevant data into above Tables.

mysql> insert into products values (11, 'laptop', 50000.60, 2300), (22, 'mobile', 30000.67, 1200), (33, 'charger', 400.67, 300), (44, 'chair', 300.56, 5000), (55, 'sofa', 40000.76, 4000));

mysql> select * from products

Product_id	product_name	price	Quantity
11	dap TOP	50000.56	2300
22	mobile	30000.67	1200
33	charger	400.67	300
44	Chair	300.56	5000
55	Sofa	40000.76	4000.

mysql> insert into customers values (1011, 'Ramya', 'Ra.com'), (1022, 'surya', 'su.com'), (1033, 'vinnu', 'vn.com'), (1044, 'honey', 'hn.com'), (1055, 'Pavani', 'pv.com'));

mysql> Select * from customers;

Customer_id	Customer_name	Email
1011	Ramya	Ra.com
1022	surya	su.com
1033	vinnu	vn.com
1044	honey	hn.com
1055	Pavani	pv.com

mysql> insert into orders values (1, 1011, '2023-09-05', 200.76), (2, 1022, '2023-08-5', 2000.96), (3, 1033, '2024-05-05', 300.87), (4, 1044, '2022-09-05', 100.76), (5, 1055, '2023-04-05', 2100.45);

Query ok, 5 rows affected (0.01 secs).

mysql> Select * from orders;

Orderid	Customer_id	Order_date	TotalAmount.
1	1011	2023-09-05	200.76
2	1022	2023-08-05	2000.98
3	1033	2024-05-05	300.87
4	1044	2022-09-05	100.76
5	1055	2023-04-05	2100.45.

mysql> Insert into Order_items values (1, 11, 4, 2000), (2, 12, 5, 3000), (3, 103, 2, 1000), (4, 104, 3, 3500), (5, 105, 2, 1500);
Query ok, 5 row affected (0.01sec).
mysql> select * from Order_items;

Order_id	Product_id	Quantity	price
1	11	4	2000.00
2	12	5	3000.00
3	13	2	1000.00
4	14	3	3500.00
5	15	2	1500.00.

Cycle-7

PL/SQL:

PL/SQL stands for procedural language Extension of SQL
PL/SQL is a Combination of SQL along with the procedural features of programming languages.
→ It was developed by oracle corporation in the Early 90's to Enhance the Capabilities of SQL.

PL/SQL block Consists of three Sections:
→ The Declaration Section (optinal)
→ The Execution Section (mandatory)
→ The Exception handling section (optional).

Declaration Section:— This section is optional and starts with keyword DECLARE. It is used to declare any place holders like Variables, Constants, Records And Cursors which are used to manipulate data in Execution Section. placeholders may be any of Variables, Constants and Records, which stores data temporarily

Execution Section:— The Execution Section of a PL/SQL block starts with the reserved keyword BEGIN and Ends with END. This is a mandatory Section and is the Section where the program logic is written to perform any task. The programmatic Constructs like loops, Conditional Statement, And SQL statements form the part of Execution Section.

Expection Section:— The Exception section of a PL/SQL block starts with reserved keycord EXCEPTION. this Section is optinal An Errors in the program Can be handled in this section, So, that PL/SQL blocks terminates gracefully. If PL/SQL block Contains exceptions that cannot be handled, the block terminates abruptly with Errors.

Every statement in the above three sections must end with a semicolon ; PL/SQL blocks can be nested within other PL/SQL blocks. Comments can be used to document code.

Create a PL/SQL program which includes declaration section, executable section and exception.

```
DECLARE
    -- Declaration Section
     num1 NUMBER := 10;
     num2 NUMBER := 0;
     result NUMBER;
BEGIN
   -- Executable Section
     result := num1 / num2;

  DBMS_Output.put_line('Result :' || result);
EXCEPTION
    -- Exception handling section
  When zero_DIVIDE THEN
     DBMS_OUTPUT.PUT_LINE('Error : Division by zero
                  is not allowed.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An Unexpected Error occured');
  END;
```

Output:-

ERROR : DIVISION by zero is not Allowed.

Program 8:-

```
DECLARE
   --Declaration Section
     Emp_id NUMBER := 101;
     Emp_name VARCHAR2 (20);
     Emp_Salary NUMBER;
BEGIN
    -- Executable Section
   SELECT name, Salary INTO Emp_name, Emp_Salary
     FROM Employees
   WHERE Employee_id = Emp_id;
DBMS_OUTPUT.PUT_LINE('Employee Name: ' || Emp_name);
DBMS_OUTPUT.PUT_LINE ('Employee Salary: ' ||Emp_Salary);
EXCEPTION
        -- Exception Handling Section
  WHEN NO_DATA_FOUND THEN
     DBMS_OUTPUT.PUT_LINE('No Employee found with ID' ||
                                  Emp_id);
WHEN TOO_MANY_ROWS THEN
     DBMS_OUTPUT.PUT_LINE ('multiple Employees found with
ID' || Emp_id);
WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE('An UnEncepted Error occured');
END;
```

1. Creation of Employees Table :-

```
Create Table Employees (Employee_id number primary key,
Name varchar2 (50), Salary number );
INSERT INTO Employees Values (101, 'John',
   50000);
Insert into Employees values (102, 'Bob', 60000);
   Commit;
```

Output

Employee_ID	Name	Salary.
101	John	50000.
102	Bob	60000.

Output: Employee Name : John
Employee Salary : 50000.

Program 3:- Write a PL/SQL block using which student marks can be selected from the table and printed for those who secured first class (>70) and an Exception can be raised if no records were found.

Creation of Students Table:-

```
Create Table students (S_id Number primary key, S_name
Varchar2(50), S_marks NUMBER);
Insert into Students values (1, 'Alice', 85); (2, 'Bob', 90),
(3, 'Charlie', 65), (4, 'David', 72), (5, 'Eve', 50);
COMMIT;
```

Output

S_id	Sname	S_marks.
1	Alice	85
2	Bob	90
3	Charlie	65
4	David	72
5	Eve	50.

```
DECLARE
--Declare variables to Hold student details.
  v_student_name varchar2(50);
  v_student_marks. Number;
--Declare an Exception for when no records are found
  no_records_found_ EXCEPTION;

BEGIN
--Open a Cursor to Select students who secured firstclass
 FOR STUDENT IN
    (SELECT s_name, s_marks
      FROM students
      WHERE s_marks >=70)
   loop
    -- fetch student details And Print them.
   v_student_name := student.s_name ;
    v_student_marks := student.s_marks;
DBMS_OUTPUT_PUT_LINE ('student name : ' || v_student_name
|| ' ; marks : ' || v_student_marks)
 END loop ;
-- Raise an Exception if no records were found
 IF SQL %ROWCOUNT = 0 Then
    RAISE no_records_found ;
  END IF ;
EXCEPTION
  -- Exception handles if no records were found
When no_records_found then
DBMS_OUTPUT. PUT_LINE (' No students found who secured
firstclass.');
WHEN OTHERS THEN
-- Generic Exception handles for any other Errors
DBMS_OUTput. PUT_LINE ('An error occured : ' || SQLERRM);
   END ;
/
```

Output:-

Student Name : Alice, marks: 85
Student Name : Bob, marks : 90
Student name : David, marks : 72.

Program 4:- Write a PL/SQL block which inserts into student table and use COMMIT, ROLLBACK and SAVEPOINT.

Student Table:-

```
Create Table students (s_id number generated by default
As Identity primary key , S_name Varchar2(50), S_marks
number);

DECLARE
  -- Declare variables
V_student_name varchar2(50);
V_student_marks_number;
BEGIN
-- Insert first student Record
INSERT INTO Students (student_name, marks) Values ('Rahul', 85);
DBMS_Output.put_line('Inserted Rahul with 85 marks.');
--Create a Save point after Inserting first Record
SAVEPOINT after_first_insert;
-- Insert second student record
INSERT INTO Students (student_name, marks) values(priya, 78
DBMS_OUTPUT_PUT_Line ('Inserted priya with 78 marks');
savepoint after_ second_ Insert;
-- Insert third student student Record
Insert into students ( Student_name, marks) values ('Amit', 45);
DBMS_OUTput_ PUT_Line( Inserted Amit with 45 marks');
ROLLBACK TO after second_ insert;
DBMS_ OUT Put_ put_ Line(Rolled Back Amit Insert');
```

-- Insert another student after rollback.

```
INSERT INTO students (student_name, marks) VALUES ('Neha',
92);
DBMS_OUTPUT.PUT_LINE ('Inserted Neha with 92 marks after
RollBack.');
-- COMMIT the changes.
COMMIT;
DBMS_OUTPUT.PUT_LINE ('Transaction Commited');
EXCEPTION
--Exception handling in case of any Error
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE ('An Error occured :' || SQLERRM);
ROLLBACK; -- Rollback entire transaction in case of Error
DBMS_OUTPUT.PUT_LINE ('Transaction Rolled back due to
Error.');
END;
/
```

Output :—

Inserted Rahul with 85 marks
Inserted priya with 78 marks
Inserted Amit with 45 marks
Rolled Back Amit's Insertion.
Inserted Neha with 92 marks after RollBack
Transaction commited.

Task:-

Create tables Sailors, boats and reserves.

Insert relevant data into the tables, write a PL/SQL program to display the names of sailors who reserved red color boats.

```
Create Table Sailors(
    Sailor_id Number primary key,
    Sailor_name varchar(100),
    Rating Number,
    age Number
);

Create Table boats (
    boat_id Number primary key,
    boat_name varchar2(100),
    Color Varchar(50)
);

Create table reserves(
    Sailor_id Number,
    boat_id Number,
    Reservation_date DATE,
primary key (sailor_id, boat_id, Reservation date);
Foreign key (boat_id) References Sailors (sailor_id),
foreign key (boat_id) References boats (boat_id)
);

// Inserting relevant values into the Tables.
    BEGIN
        For sailors_rec IN (
        SELECT Distinct S.sailor_name
        from Sailors s
        Join reserves r on S.sailors_id = r.sailor_id.
```

```
join boats b ON s.boat_id = b.boat_id
where b.color = 'red'
)
loop
DBms_Output.put_line('sailor name': || sailor_rec.
                                    sailor_name);
End Loop;
IF SQL%Rowcount = 0 THEN
   DBms_output.Put_Line('No sailors found who
                        reserved a red boat :);
   END IF;
   END;
//
```

Output:-

```
Sailor name : Dustin
Sailor name : RUBBER
```

Cycle-5

1) Develop a program that involves the following features NESTED IF, CASE and CASE Expression. The program can be extended using the NESTED IF and COALSECE functions.

NESTED IF :- To handle conditions based on various attributes.

CASE statement :- To choose between multiple conditions.

CASE EXP :- To determine values based on expressions

NULLIF :- To prevent division by zero errors

COALESCE :- To handle null values by providing a feedback

Creating Employees Table :-

Create Table Employees (Employeeid, number, Employee_dept varchar2(10), Performance varchar(10), salary number);

insert into Employees (2547, 'HR', 'Excellent'), (2548, 'IT', 'Good'); (2549, 'HR', Good), (2550, 'IT', Good);

mysql > select * from Employees.

Employee_id	department	performance	Salary
2547	HR	Excellent	10,000
2548	IT	Excellent.	80000
2549	HR	Good	7000
2550	IT	Good	8000.

Program :-

```
DECLARE
v_Employee_id number := 2547;
v_department varchar2(50);
v_performance varchar2(10);
v_salary number;
v_bonus number := 0;
v_totalsalary number;
BEGIN
```

```
Select dept, Performance, salary into v_department, v
v_Performance, v_Salary from Employees where
Employee_id = v_Employee_id :
  if v_department = 'HR' then
    if v_Performance = 'Excellent' then
      v_bonus := 1000;
    Elseif v_Performance = 'Good' then
      v_bonus := 500;
    Else
      v_bonus := 0;
    Endif;
  Elseif v_department = 'IT' then
  If v_Performance = 'Excellent' then
      v_bonus := 1200;
  Elseif v_Performance = 'Good' then
      v_bonus := 0;
    Endif;
  Else
  v_bonus := 0;
Endif:
Case v_department
      when 'HR' then
  Case v_Performance
  when 'Excellent' then v_bonus := 1000;
  when 'Good' then v_bonus := 500;
  Else v_bonus := 0;
  END CASE;
  When 'IT' then
  Case v_Performance
```

```
when 'Excellent' then v_bonus := 1200;
when 'Good' then v_bonus := 600;
else v_bonus := 0;
End case;
when 'IT' then
  Case v_Performance
    when 'Excellent' then v_bonus := 1200;
    when 'Good' then v_bonus := 600;
      else v_bonus := 0;
  End case;
Else
  v_bonus := 0;
  End case;
v_totalsalary := v_salary
    Case
    when v_Performance = 'Excellent' then 1000
    when v_Performance = 'Good' then 500
      else 0
End;

If Nullif (v_bonus 0) IS NOT Null Then
DBMS_Output.put_line('salary to Bonus Ratio:' || round
                    ((v_salary(v_bonus, 2));
Else
PBMS_output.putline('Bonus is Zero Cannot Calculate
                                    Ratio');
  END IF;
DBMS_Output.Put_Line("Bonus :" || Condesce(v_bonus, 0));
DBMS_Output.put_line('Total Salary :' || v_total_salary);
```

Output:-

Employee id	department	performance	Salary
2547	HR	Excellent	11,000
2548	IT	Excellent	8000
2549	HR	good	7000
2550	IT	good	6000

Salary to Bonus ratio : 10,000

Bonus : 1000

total salary : 11000.

2. PL/SQL program development using while loops, numbers, for loops nested loops.

While Loop : To execute a loop with condition

Numeric Loop : A loop that iterates over a range of numbers

Nested Loop : loop inside other loops.

Program:-

```
DECLARE
    V_Employee_id Number(10);
    V_Employee_name varchar(50);
    V_project_id Number;
    V_Project_count Number := 3;
    V_Counter_Number := 1;
BEGIN
DBMS_Output.put_line ('Processing Employees and their
                        projects - - -');
    While V_Counter := 5 loop
    if V_Counter = 1 then
        V_Employee_name : 'Alice';
```

```
Elseif v_counter=2 then
v_Employee_name := 'Bob';
Elseif v_counter := 3 then
v_ Employee_name := 'Charlie';
Elseif v_counter=4 then
   v_Employee_name := 'Eve';
Endif;

DBms_Output.Put_line('Employee ID:' || "Employee_id||
        'Name:' ||v_ Employee e_name);
DBms_Output.Put_line('Projects assigned');
For project IN 1.. v_project count loop
'Project id : 2000 + project;
DBms_Output.put_line ('→ projectID :' || v_projectid||
        ('project' || Project (i)');
END loop;
   v_Employee_id= v_ Employee id+1;
      v_counter := v_ Counter++;
   DBms_Output.Put_line('- - -- - - -');
   END loop;
   END;
```

Output:-

```
Processing Employees and their projects....
Employee ID=1001, Name: Alice
   Projects Assigned
  →Project ID : 2001 (Project1)
  →Project ID : 2002 (Project 2)
  →Project ID : 2003 (Project 3)
```

Employee ID : 1002, Name : Bob
Projects assigned
→ Project ID : 2001 (Project 1)
→ Project ID : 2002 (Project 2)
→ Project ID : 2003 (Project 3)

Employee ID : 1003, Name : Charlie
Projects assigned
→ Project ID : 2001 (Project 1)
→ Project ID : 2002 (Project 2)
→ Project ID : 2003 (Project 3)

Employee ID : 1004, Name : Eve
Projects assigned
→ Project ID : 2001 (Project 1)
→ Project ID : 2002 (Project 2)
→ Project ID : 2003 (Project 3).

(3) PL/SQL program development using Error handling Built IN Exceptions, Userdefined functions, RAISE Application Error.

→ Error handling :— Exception blocks to handle Errors.

→ Built in Exception :— Handling standard Exceptions like NO_DATA_FOUND, TOO_MANY_ROWS etc:,

→ User defined Exceptions :— Defining Custom Exceptions to handle Specific Errors.

→ RAISE_APPLICATION_ERROR : Raising Custom application Errors with Specific Error Codes.

Program:

```
DECLARE
V_Employee_id Number := 101;
V_Employee_Name Varchar2(50);
V_salary Number;
en_low_salary Exception;
BEGIN
if v_ Employee_id = 101 then
V_Employee_name := 'John Doe';
V_salary := 5000;
Elseif V_Employee_id = 102 then
V_Employee_name := 'Jane Smith';
  V_salary := 0;
Else
Raise No_data_Found;
 Endif;
if v_ salary < 3000 Then
Raise en_low_salary;
 End If;
V_salary := 1000/ v_salary;
DBMS_Output.Put_Line('Employeename : '|| V_Employee_name);
DBMS_Output.Put_Line('updated salary : '|| v_ salary);
 Exception
 When no_data_found then
    DBMS_ Output.Put_Line ('Error : Employee Not found!!');
When Zero_Divide then
DBMS_Output.Put_Line('Error : Division by Zero Occured.
                      Invalid salary data.!);
```

```
When e_low_salary then
Dbms_output.put_line ('Error: Employee salary is bel
        the minimum of threshold.');
When others then
Dbms_output.put_line ('Error: ' || SQLERRM);
Raise_Application_Error(-20001, 'Unknown error occured:
        Employee processing');
END;
/.
```

Output:-

```
Employee Name : John Doe
Updated Salary : 2.
```

Cycle-9

Stored functions in PL/SQL:-

A stored function in PL/SQL is a subprogram that returns a single value and can be used as SQL queries & procedural statements.

Syntax of a stored function:

```
CREATE OR REPLACE FUNCTION function_name(
  Parameter_1 DATATYPE,
  Parameter_2 DATATYPE
) RETURN return_datatype IS
  --variable declarations
   v_variable DATATYPE;
BEGIN
  --function logic
RETURN Some_value;
 END function_name ;
/
```

- CREATE OR REPLACE FUNCTION → Creates & updates a function
- RETURN → Specifies the return data type
- IS → Declares local variables.
- BEGIN... END → Function body with logic
- RETURN → Returns a single variable.

1. Simple Function: Get Employee Age.

This function returns an Employee's age based on their ID

```
CREATE OR REPLACE FUNCTION get_employee_age(
    P_Emp_id IN Number
) Return Number IS
    v_age NUMBER;
BEGIN
    SELECT age Into v_age
```

```
FROM Employees
WHERE eid = P_Emp_id ;
RETURN v_age ;
END ;
/.
```

Employees Table :-

```
mysql> Select * from Employees;
```

Employee id	Emp_Name	Salary	Age.
101	Ramya	5000	20
102	surya	6000	30
103	Vinnu	7000	40
104	Honey	8000	50..

Calling the function in SQL :-

```
Select get_Employee_age (102) As Employee_Age from
    dual;
```

Output :-

Employee_Age.
30.

2. Function : Calculate Bonus.

```
CREATE OR REPLACE FUNCTION Calculate_Bonus(
    P_Emp_id IN NUMBER,
    P_Salary IN NUMBER
)RETURN NUMBER IS
    v_age NUMBER
    V_Bonus NUMBER
BEGIN
    SELECT age INTO v_age
    FROM Employees
    WHERE eid = P_Emp_id;
```

- Calculate Bonus: 20% if age > 50, otherwise 10%

```
IF v_age > 30 THEN
   v_Bonus := p_salary * 0.2;
ELSE
   v_Bonus := p_salary * 0.1;
END IF ;
RETURN v_bonus;
END;
```

Calling the function in SQL:-

```
SELECT Calculate_bonus(103, 7000) AS Bonus from dual;
```

Output:-

Bonus.
14008.

3. Function to Determine Employee Category.

This function categories Employees based on their age:

- "Senior" (if age > 50)
- "Mid-Level" (if age between 30 and 50)
- "Junior" (if age < 30).

Function Creation:-

```
CREATE OR REPLACE FUNCTION get_Employee_Category (
    p_Emp_id IN Number
)RETURN VARCHAR2 IS
    v_age NUMBER;
    v_Category VARCHAR2(20);
BEGIN
 -- get Employee age
   SELECT age INTO v_age
    FROM Employee
   WHERE eid = p_Emp_id;
 -categories based on age.
```

```
If V_age>30 THEN
  V_category := 'Senior';
ELSE IF V_age BETWEEN 25 AND 30 THEN
  V_category := 'mid_level';
ELSE
  V_category := 'Junior';
  END IF;
RETURN V_category ;
  END ;
/
```

select get_Emplogee_category (103) AS Category from dual;

Output:-

Category
Senior

Select get_Employee_category (101) AS Category from dual;

Output:-

Category
Junior

Select get_Employee_category (102) As Category from dual;

Output:-

Category
mid-Level.

Task:-
Create a function that returns No. of Employees above a given Age.

```
CREATE OR REPLACE FUNCTION
 Count_Employees_Above_age(
   P_age IN Number
 )RETURN Number IS
      V_count NUMBER;
 BEGIN
   SELECT COUNT(*) INTO V_count
    FROM Employees
    WHERE age > P_age;
  RETURN V_count;
 EXCEPTION
    WHEN OTHERS THEN
       RETURN 0;
END Count_Employees_above_age;
/
```

Calling the function in SQL:-

```
SELECT Count_Employees_above_age(30) AS Employee_
COunt from dual;
```

Output:-

Employee ID	Name	Age.
103	Vinnu	40
104	Honey	50

Cycle-10

Cursors in PL/SQL

In PL/SQL, cursors are used to handle multiple rows returned by query. A cursor acts as a pointer that allows you to process each row from the query result set one by one.

Types of cursors in PL/SQL:-

1. Implicit Cursors:- * Automatically created by Oracle when a SELECT INTO, INSERT, UPDATE or DELETE operation is executed.

2. Explicit Cursors:- * Defined by the programmer to help handle queries that return multiple rows.
* Require explicit DECLARE, OPEN, FETCH and CLOSE operations.

Cursor Attributes:-

PL/SQL provides the following attributes for cursor to check the cursor's state:

Attribute	Description	Example
%FOUND	Returns true if a row was fetched successfully.	IF Emp_Cursor%Found Then
%NOT FOUND	Returns TRUE if no rows was fetched	EXIT WHEN Emp_cursor% NOTFOUND;
%ISOPEN	Returns TRUE IF the cursor is open	IF Emp_cursor%ISOPEN Then....
%ROW COUNT	Returns the no. of rows fetched so far.	DBMS_OUTPUT.put_line (Emp_Cursor%ROWCOUNT);

Employees Table:-

Emp_ID	Emp_name	Emp_Salary	Age	dept.
101	Ramya	5000	20	IT
102	Surya	16000	22	HR
103	Vinod	8000	19	HR
104	Honey	14000	30	IT

1. Impliat Cursor Example:-

```
DECLARE
  V_total_Emps NUMBER;
BEGIN
  SELECT COUNT(*) INTO
    V_total_Emps FROM Employee;
DBMS_OUTPUT.put_Line('Total Employees: '|| V_total_Emps);
END;
/
```

Output:-

Total Employees: 4

2. Explict Cursor Example:-

```
DECLARE
CURSOR Emp_Cursor IS
SELECT Emp_id, emp_name, Emp_salary FROM Employee;
V_Emp_id Employee.Emp_id%TYPE;
V_Emp_name Employee.employee_name%TYPE;
V_Emp_salary Employee.Employee_Salary%TYPE;
BEGIN
  OPEN Emp_Cursor; --Open the cursor
LOOP
FETCH Emp_cursor INTO V_Emp_id, V_Emp_name, V_Emp_salary;
EXIT WHEN Emp_Cursor%NOTFOUND;
DBMS_OUTput.put_Line('Emp_id :'||V_Emp_id||', Emp name:
||V_Emp_name||', Salary: '||V_Emp_salary);
  END LOOP;
  CLOSE Emp_Cursor; --close the cursor
```

END;
/

Output:-

Emp_id :101, Name : Ramya, Salary : 5000
Emp_id :102, Name : Surya, Salary : 16000
Emp_id :103, Name : Vinna, Salary : 8000
Emp_id :104, Name : Honey, Salary : 18000.

3. Parameterized Cursor Example:-

```
DECLARE
CURSOR dept_cursor (p_dept VARCHAR) IS
 SELECT Emp_id, Emp_name, Emp_salary
  From Employee
 WHERE dept= p_dept;
V_Emp_id Employee.Empid%.TYPE ;
V_Emp_name Employee.Emp_name %.TYPE;
V_Emp_salary Employee.Emp_salary %.TYPE;
BEGIN
 OPEN dept_cursor('HR');
 Loop
 FETCH dept_Cursor INTO V_Emp_id, V_Emp_name, V_Emp_Salary;
 EXIT WHEN dept_Cursor %.NOTFOUND;
DBMS_OUTput.put_line('Emp_id :'|| V_Emp_id ||', Name :'||
 V_Emp_name ||', Salary :'|| V_Emp_salary);
 END Loop ;
  Chose dept_cursor ;
  END;
```

Output:-

Emp_id : 102, Name : Surya, Salary : 16000
Emp_id : 103, Name : Vinna, Salary : 8000.

4. For Update Cursor :-

In PL/SQL, a FOR UPDATE cursor is used to lock the rows retrieved by the cursor, preventing other transactions from modifying them until lock is released. The lock remains until the transaction is committed or rolled back.

```
DECLARE
 CURSOR Emp_Cursor IS
 SELECT emp_id, Emp_name, Emp_salary.
  From Employee
  WHERE salary < 15000.
   FOR UPDATE;
 V_Emp_id Employee.Emp_id %TYPE;
 V_Emp_name Employee.Emp_name %TYPE;
 V_Emp_salary Employee.Emp_salary %TYPE;
  BEGIN
 OPEN Emp_Cursor;
  Loop
  FETCH Emp_Cursor INTO V_Emp_id, V_Emp_name, V_Empsalary;
  EXIT WHEN Emp_Cursor%NOTFOUND;
  Update Employees
   SET Salary = salary + 500
   WHERE Current of Emp_Cursor;
DBMS_output.Put_Line ('salary Updated for Emp_id :' ||V_
      Emp_id||', Name :' || V_Emp_name);
 END Loop;
 Chose Emp_Cursor;
    END;
/
```

Output:-

Salary updated for Emp_id : 101, Emp_name : Ramya.
Salary update for Emp_id : 103, Emp_name : Vinnu.
Salary updated for Emp_id : 104, Emp_name : Honey.

Emp_id	Emp Name	Emp_salary
101	Ramya	5500
102	Surya	16000
103	Vinnu	8500
104	Honey	14500.

5. Cursor Variables (REF CURSOR) in PL/SQL:—

A REF CURSOR in PL/SQL is a cursor variable that points to a query result set and allows dynamic SQL execution. Unlike a static cursor, a REF CURSOR can be dynamically assigned different SQL queries at runtime.

```
DECLARE
    TYPE Emp_Ref_Cursor IS REF CURSOR;
    Emp_Cursor Emp_Ref_Cursor;
    V_Emp_id Employee.emp_id %TYPE;
    V_Emp_name Employee.Emp_name %TYPE;
BEGIN
    OPEN Emp_Cursor FOR SELECT Emp_id, Emp_Name
FROM Employee WHERE dept = 'IT';
 LOOP
    FETCH Emp_Cursor INTO V_Empno, V_Emp_name;
    EXIT WHEN Emp_Cursor %NOTFOUND;
DBMS_output.put_line ('Emp_id :' || V_Emp_id || ', Name:
                      || V_Emp_name);
END LOOP;
    CLOSE Emp_Cursor;
    END;
```

Output:—

Emp_id : 101, Name: Ramya

Emp_id: 104, Name : Honey.

Cycle-1

Triggers in PL/SQL :-

A trigger is a stored PL/SQL block that automatically executes (fires) in response to specific events on a table or view, such as INSERT, UPDATE or DELETE. Triggers are used to enforce business rules, mentioned audit trails, replicate data and more.

Types of Triggers in PL/SQL :-

1. Row-Level Trigger :-

- Fires once for each row affected by the triggered event.
- Typically used for operations that need to check or modify individual rows.

2. Statement-Level Trigger :-

- Fires once for a entire SQL statement, regardless of how many rows are affected.

Timing of Triggers :-

Triggers can fire :

1. Before the triggering event : useful for validation.
2. After the triggering event : useful for logging or cascading actions.
3. Instead of Trigger : Handles DML operations on views

Syntax for Creating triggers :-

```
CREATE [OR REPLACE] TRIGGER Trigger_name
(BEFORE|AFTER|INSTEAD OF|INSERT|DELETE)
 ON TABLE_Name
  [FOR EACH Row]
  [WHEN (condition)]
   BEGIN
   END; -- Trigger logic goes here.
/
```

Employee Table :-

Emp_id	Emp Name	Emp_salary	dept.
101	Ramya	5000	IT
102	Surya	1600	HR
103	vinny	8000	HR
104	Honey	1400	IT

Example 1:- Row-Level Before insert trigger

This trigger automatically sets a default salary if it is not provided during the INSERT operation

```
CREATE OR REPLACE TRIGGER Set_default_salary
BEFORE INSERT ON Employee
  FOR EACH ROW
  BEGIN
    IF: New.salary IS NULL THEN
          :New.salary:=1000;
    END IF;
    END;
/
```

Inserting of New Employee:-

1. INSERT INTO Employee(Emp_id, Emp_name, dept)
   Values (105,'charlie','HR');
2. Insert into Employee (Emp_id, Emp_name, Emp_salary)
   Values(106,'David',25000,'IT');

Checking Trigger Information:-

```
Select trigger_name, trigger_type, triggering_event,
status FROM user_triggers;
```

Output: select emp_id, Emp_name, dept, Emp_salary from Employee;

Emp_id	Emp_Name	Emp_salary	dept.
101	Ramya	5000	IT
102	Surya	1600	HR
103	Vinnu	8000	HR
104	Honey	4000	IT
105	charlie	1000	HR
106	David	25000	IT.

Output:-

Checking Trigger information :-

| Trigger_name | Trigger_TYPE | Trigger Event | status

| set_default_salary | Before Each row | Insert | Enabled.

Example : Statement-Level Before Delete Trigger.

This Example demonstrates a Before Delete statement level trigger that prevents any deletion on the Employees table if the total no. of Employees is less than 10.

```
CREATE OR REPLACE TRIGGER
  Prevent_bulk_delete
BEFORE DELETE ON Employees .
DECLARE
   V_Emp_Count NUMBER ;
BEGIN
SELECT COUNT(*) INTO V_Emp_Count From Employee ;
IF V_Emp_count <10 THEN
RAISE_APPLICATION_ERROR (-20001, 'Deletion not allowed !
   Employee Count is too Low.');
        END IF ;
, End;
```

Output:-

```
ORA-20001 : Deletion not allowed !
Employee Count is too low .
ORA-06512 : at 'prevent-Bulk Delete', line 6
```

Instead of Trigger in PL/SQL:-

An Instead of Trigger is used to perform Specific actions in place of the DML operations (insert, Delete, Update) on views. This is useful because views do not directly support DML operations unless they are inherently updatable. with a INSTEAD OF-trigger, you can define what happens.

Example:-

We have two Tables:

- Department (dept) with Column: dept_id, dept_name
- Employee (Emp) with columns: Emp_id, Emp_name, dept_id

We also Create a view that joins these two tables to show Employee details along with their dept names. Since this view is not inherently updatable, we will Create an INSTEAD OF trigger to handle INSERT operations on this view.

```
--Create Department Table----
CREATE TABLE Dept(
  dept_id NUMBER PRIMARY KEY,
  dept_name VARCHAR (50)
  );
--Create Employee Table---
  CREATE TABLE Emp(
  Emp_id NUMBER PRIMARY KEY,
   Emp_name VARCHAR2(50),
   dept_id NUMBER,
  CONSTRAINT fk_dept FOREIGN KEY(dept_id) REFERENCE
  dept (dept_id)
   );
```

~~Replace TRIGGER Emp~~

```
--Create a view that joins Emp and dept ---
CREATE VIEW Emp_dept_view AS SELECT e.Emp_id,
e.Emp_name, d.dept_name
   From Emp e
  JOIN dept d on e.dept_id=d.dept_id;
```

Creating INSTEAD OF INSERT TRIGGER.

```
CREATE OR REPLACE TRIGGER
Emp_dept_insert trigger
INSTEAD OF INSERT ON Emp_dept_view
FOR Each Row
  BEGIN
```

```
MERGE INTO dept d
USING (SELECT :new.dept_id AS dept_id, :new.dept_name
AS dept_name FROM dual) src
  ON (d.dept_id = src.dept_id)
    WHEN NOT MATCHED THEN
    INSERT (dept_id, dept_name)
    values(src.dept_id, src.dept name);
INSERT INTO Emp (Emp_id, Emp_name, dept_id) Values
  (:new.emp_id, :new.Emp_name, :new.dept_id);
    END;
```

Inserting into the view:-

```
INSERT INTO Emp_dept_view (Emp_id, Emp_name, dept_name)
Values (101, 'Ramya', 'HR');
select * from dept;
```

dept_id	dept_name
101	HR

```
Select * from Emp;
```

Emp_id	Emp_name	dept_id
101	Ramya	101

- New Keyword:-

: New keyword is used to refer to the new values being inserted into the view.

What happens:-

The INSTEAD of Trigger fires and performs the following operations.

1. Insert a new department into the dept Table.

```
INSERT INTO dept (dept_id, dept_name) value(101, 'HR');
```

& INSERT a new Employee into the Emp Table.

```
INSERT INTO Emp (Emp_id, Emp_name, dept_id) values (101,
'Ramya', 101);
```

Cycle-12

Create Table and perform the search operations on table using indexing and non-indexing techniques.

To demonstrate the difference blw indexed and non-indexed searches, here's an Example involving a database Table. Lets create a hypothetical table and walk through the process for both techniques.

Example table :- Student.

SID	Sname	Marks	Major	GPA.
101	Ramya	90	maths	9.1
102	Rishika	92	Chemistry	8.8
103	Vinnu	89	maths	9.3
104	Charlie	95	maths	9.0
105	David	88	Chemistry	8.8.

Search operation without indexing :-

If no index Exists, the database must perform a full table scan to find the record. It Examines Each row Sequentially until it finds the match.

Select * from student where major = 'Chemistry.';

SID	Sname	marks	major	GPA
102	Rishika	92	Chemistry	8.8
105	David	88	chemistry	8.8.

1. Creating an Index :-

To create an Index, use the Create INDEX statement.

Syntax :- Create Index index name ON table_name (column_name);

Example :-

Create index idx_major ON student (major);

Output :-

Index Created.

2. Creating a Unique index

If you want to ensure that values in a columns are Unique, use a unique index:

Create Unique Index unique_index_name ON table_name (column_name);

Example:-

~~CREATE Unique INDEX Unique index_name on table_name(column name~~

Create Unique index idx_~~stu~~s_id on student(ID);

Output:- idx_studentID Created.

3. Creating a Composite index:-

A Composite index is created on multiple columns. It's useful for queries involving conditions on multiple columns.

Create index index_name ON Table_name (col1, col2);

Example:-

If you often Search by major and Age together.

Create INDEX idx_major_masks ON Student (major, masks);

4. viewing indexes:- To check existing indexes in your database, many SQL database systems have specific Commands or utilities for instance:-

mysql:-

show index from table_name;

postgresql:-

\d table_name

you can verify that the index was used by checking the execution plan.

* Check Execution plan:

Explain plan for

Select * from student Where ~~age~~ masks > 90;

Output :-

SID	SName	marks	major	GPA
101	Ranjan	92	chemistry	8.8
102	Rishika	92	maths	9.0
104	Charlie	95		

Select * from table(dbms_xplan.display());

Output :-
You will see that the query uses an INDEX Range Scan, indicating that the index is being used to improve performance.

Id	Operation	Name
0	select statement	Rishika
1	INDEX Range SCAN	charlie
IDX_major_marks		

Additional Program

Develop a program to demonstrate JDBC Connectivity.

(a) mysql :-

Step 1 :- Register the Drives class.

Step 2 :- Create a Connection

Step 3 :- Create a statement.

Step 4 :- Execute a Queries.

Step 5 :- close Connection

(b) Oracle:-

Step 1:- import required packages.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLStatement;
```

Step 2:- Load the Oracle JDBC Driver.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Step 3:- Establish Connection:-

```
String dbURL = "jdbc:oracle:thin:@localhost:1521:xe";
String username = "your_username";
String password = "your_password";
Connection Conn = DriverManager.getConnection(dbURL, username,
                                              password);
```

Step 4:- Create Statement & Execute query:-

```
Statement stmt = Conn.CreateStatement();
String SqlQuery = "Select * from yourTablename";
ResultSet rs = Stmt.ExecuteQuery(sqlquery);
```

Step 5:- Process the result:-

```
while(rs.next()){
  int id = rs.getint("id");
  String name = rs.getstring("name");
  int age = rs.getint("age");
System.out.println("ID:"+id+", Name:"+name+", Age:"+age);
}
```

Step 6:- Close the Resources:-

```
rs.close();
stmt.close();
Conn.close();
```

Complete program:—

```
Public class import java.sql.Connection;
        import java.sql.DriverManager;
        import java.sql.ResultSet;
        import java.sql.Statement;
        import java.sql.SQLException;
Public class OracleJDBCExample {
  Public static void main(String[] args){
String dbURL = "Jdbc:Oracle:thin:@localhost:1521:x
String username = "24765A1211";
String password = "student";
 Connection Conn = null;
  Statement stmt = null;
  ResultSet rs = null;
try{
   class.forname("oracle.jdbc.drives.Oracledrives");
System.out.println("Oracle JDBC Drives loaded Successfully");
 Conn = DriverManager.getConnection(dbURL, Username,
                                    Password);
System.out.println("Connection Established Successfully");
 stmt = conn.createstatement();
String SQLQuery = "select * from student";
 rs = stmt.executeQuery(sqlquery);
 while(rs.next()){
    int id = rs.getint("id");
String name = rs.getstring("sname");
 int marks = rs.getint("marks");
System.out.println("ID:"+sid+", Name:"+sname+",
   marks :"+ marks);
   }  }
Catch(ClassNotFoundException e){
```

```
System.out.println("
        Jar file ");
    e.printStackTrace();
}catch (SQLException e){
  System.out.println("Database Error occured ");
      e.printStackTrace();
} finally {
   try{
      if (rs!=null) rs.close();
      if (stmt !=null) stmt.close();
      if (conn !=null) conn.close();
   }catch(SQLException e) {
        e.printStackTrace();
   }
  }
}
```

Output:—

Oracle JDBC Driver loaded Successfully.
Connection Established Successfully.
ID : 103, Name : Vinnu, Marks : 89.
ID : 104, Name : Charlie, Marks : 95
ID : 105, Name : David, Marks : 88
ID : 106, Name : Eleven, Marks : 78
ID : 107, Name : Fatima, Marks : 89
ID : 102, Name : Rishika, Marks : 92
ID : 101, Name : Ramya, Marks : 90.