Assignment No :- 3

Q. (1) Explain with example . Aggregate functions, Builtin functions.

Ans:-

1. Aggregate functions :-

Aggregate functions perform a calculation on a set of rows and return a single value. They are often used with the 'GROUP BY' clause SQL.

- SUM() → adds all values in a column
- AVG() → calculates the avg of values
- COUNT() → counts the avg of no. of rows
- MAX() → returns the largest value
- MIN() → returns the smallest value.

eg:- • Employees :-

EmpID	Name	Department	Salary
1	Riya	IT	50,000
2	Nikita	IT	70,000
3	Akshada	HR	60,000
4	Gauri	IT	85,000

```
select max(salary) as highestsalary,
       min (salary) as lowestsalary
from Employees;
```

o/p → highestsalary = 85,000
lowestsalary = 50,000

2. Built-in Functions:-

Built-in fn are predefined fn provided by SQL to perform operations on individual value (not on groups of rows)

- Tyes :-

1) String functions:-
   UPPER() , LOWER() , CONCAT() , LENGTH() , SUBSTRING() .

2) Numeric functions:-
   ROUND() , CEIL() , FLOOR() , POWER()

3) Date & time functions :-
   NOW() , CURDATE() , YEA() , MONTH()

4) Conversion functions :-
   CAST() , CONVERT() .

eg:- consider same employee table as last.

1) select upper(Name) as UppercaseName from Employees;

O/P → RIYA , NIKITA , AKSHADA , GAURI

2) select Name, length(Name) as Namelength from Employees;

O/P →

Name	NameLength
Riya	4
Nikita	6
Akshada	7
Gauri	5

Q.(2) & Q.(3) Explain use of group by, having, order by, join and its types.

Ans:-

1. Group by :-

The Group by clause groups rows that have the same values in specified columns. It is usually used with aggregate fn to perform calculations on each group.

eg :- Employees :-

Emp ID	Name	Department	Salary
1	Riya	IT	50000
2	Sonam	HR	60000
3	Aarav	IT	70000
4	Isha	IT	75000
5	Nishant	HR	85000

```
select Department, sum(salary) as TotalSalary
from Employees
group by Department;
```

o/p →

Department	TotalSalary
IT	195,000
HR	145000

Groups employees by department and calculates salary sum for each group

2. HAVING :

The 'having' clause is used to filter the results of groups created by 'group by'. It works like 'where' but 'where' filters rows before grouping, while 'having' filters after grouping.

eg:- consider some table employees :-

```
select Department, sum(salary) as Totalsalary
from Employees
group by Department
having sum(salary) > 150,000 ;
```

O/P →

Department	Totalsalary
IT	195,000

Filters out groups after the aggregation

3. ORDER BY

The 'order by' clause is used to sort the result set either

- Ascending (ASC) - default
- Descending (DESC)

eg:- Sort employees by Salary in descending order.

```
slect Name, Salary
From Employee
order by salary DESC;
```

O/P →

Name	Salary
Nishant	85000
Isha	75000
Aarav	70000
Sonam	60000
Riya	50000

4. JOIN :-

A join is used to combine rows from two or more tables based on related column betn them.

Example Tables :-

- Employees :-

EmpID	Name	Dept ID
1	Riya	101
2	Aman	101
3	Pooja	102
4	Aarav	102
5	Meena	103

- Departments

DeptID	Dept Name
101	IT
102	HR
104	Finance

- Types of join :-

1) INNER JOIN :-
- returns only the rows that have matching values in both tables

query → select Employees.Name, Department.DeptName
from Employees
inner join Departments
on Employees.DeptID = Departments.DeptID

o/p →	Name	DeptName
	Riya	IT
	Aman	IT
	Pooja	HR
	Aarav	HR
	~~Meena~~	

- 'Meena' (DeptID 103) is excluded because there is no matching dept in 'Departments'.

2) Left join (Left outer join) :-

- Returns all rows from the left table and the matching rows from the right table. If there's no match, it returns NULL for the right table's column

```
select Employees.Name, Departments.DeptName
from Employees
left join Departments
on Employees.DeptID = Departments.DeptID;
```

O/P →

Name	DeptName
Riya	IT
Aman	IT
Pooja	HR
Aarav	HR
Meena	NULL

3) Right outer join :-

Returns all rows from the right table and the matching rows from the left table. If there's no match, it returns NULL for the left table's colums.

```
select Employees.Name, Department.DeptName
from Employees
right join Departments
on Employees.DeptID = Departments.DeptID
```

O/P →

Name	DeptName
Riya	IT
Aman	IT
Pooja	HR
Aarav	HR
NULL	Finance

4. Full join (full outer join)

Returns all rows from both tables, and fills with NULL where there's no match

5. CROSS JOIN :-

Returns the Cartesian product of two tables - every row of the first table is combined with every row of the second table.

```
select Employees.Name, Departments.DeptName
from Employees
cross join Departments;
```

O/P :- If there are 5 employees and 3 departments, the result will have 15 rows (5×3).

(4) What is meant by Triggers in SQL? Explain with suitable example

ns :-

A Trigger in SQL is special stored prg that automatically runs (or fires) when a specific event happen in a table or a view in the database

- Triggers are not called manually - they can run automatically when the specified occurs
- They are mainly used for:
  - Enforcing buissness rules
  - Maintaining audit logs
  - Validating or modifying data before or after changes
- Types of triggers depend on when they execute.
  - BEFORE Trigger - executes before the event happen.
  - AFTER Trigger - executes after the event happens.

- Syntax:-

```
create trigger trigger_name
{ BEFORE | AFTER } { insert | update | delete }
on table_name
for each row
begin
  -- SQL statements to execute
end;
```

Q(5) Consider following schema:
account (acct-no, branch-name, balance)
depositor (cust-name, acct-no)
borrower (cust-name, loan-no)
loan (loan-no, branch-name, amount)
Write following queries using SQL
i) Find names of all customers who have a loan at swargate branch
ii) Find all customers who are having an account and loan or both.

Ans:-

i) →
```
select distinct b.cust-name
from borrower b
join loan l on b.loan_no = l.loan-no
where l.branch-name = 'swargate';
```

- we join borrower and loan using loan-no.
- filter those whose branch is 'swargate'
- Use 'DISTINCT' to avoid duplicate customer names.

ii) →
- we combine customers from depositor and borrower table.

```
select distinct cust-name
from depositor
union
select distinct cust-name
from borrower;
```

- UNION = returns all unique customers from both sides.
- Customers having both account and loan will appear only once because of UNION.

Q(5) Consider the following database :-
Student (Roll-no, name, motrine-no, Address)
Subject (sub-Code, Sub-name)
Marks (Roll-no, Sub-code, marks)
Write following queries in SQL.

i) [Max. Marks : 70 Average marks of each student along with Roll-no. of student

→

- select Roll-no, avg(Marks) as Average_Marks from Marks group by Roll-no;
- avg(Marks) calculates the average marks of each ~~element~~ student.
- GROUP BY Roll-no. groups all marks of the same student.

ii) Find how many students have failed in the subject 'DBM'.

→ • select count(distinct M.Roll-no) as Failed-Students
from Marks M
join subjects S on M.sub_code = S.sub_code
where S.sub_name = 'DBM'
and M.Marks <40;

• We join 'Marks' and 'subject' to match the subject name 'DBM'.
• Filter students whose Marks < 40.
• Use 'count(distinct Roll-no)' to count each student only once.

Q.(7) Write syntax of following.

(1) Create Table :-

syntax :- create table table_name (
column1 datatype [constraint],
column2 datatype [constraint],
.....
);

example :-
create table student (
Roll-no int primary key,
Name varchar(50),
Address varchar(100),
Marks int
);

② ALTER TABLE

used to modify an existing table

syntax :-

```
alter table table_name
    add column_name datatype;

alter table table_name
    modify column_name new-datatype;

alter table table-name
    drop column column-name;
```

example :-

```
alter table student
ADD DOB DATE;
```

③ DROP TABLE

syntax :-

```
DROP TABLE table_name;
```

used to delete the entire table (structure + data)

example :-

```
DROP TABLE Student;
```

④ DELETE :-

used to delete rows (records) from a table

syntax :-
delete from table_name
where condition ;

example :-
1) delete from Student
where Roll_no = 101;

2) delete from student;
Deletes all rows but keeps the table structure.

⑤ UPDATE :-

used to modify existing records in a table

syntax :-
UPDATE table_name
SET column1 = value1 , column2 = value2, ...
WHERE cond^n ;

example :-
UPDATE Student
SET Marks = 85
WHERE Roll_no = 102;

Q (8) Explain following terms with example

(1) Function :-

- A 'Function' in SQL is stored prg that performs a specific task and returns a single value.
- It is often used for calculations or returning processed data.

- syntax :-

```
CREATE FUNCTION function_name (parameter_list)
RETURNS data-type
DETERMINISTIC
BIGIN
    -- calculation or logic
    RETURN value;
END;
```

- example :-

```
create function calc-total (price DECIMAL(10,2))
returns DECIMAL(10,2)
deterministic
begin
    declare total DECIMAL(10,2);
    set total = price +(price * 0.18);
    return total;
end;
```

- use function :-

```
select calc-total (1000) as TotalPrice;
```

- o/p → 1180.00

(2) PROCEDURE :-

- A 'procedure' in SQL is a stored prg that performs one or more actions in the database
- It is often used for tasks like inserting, updating, deleting or retrieving data.

- syntax :-

```
CREATE PROCEDURE procedure_name (parameter_list).
BEGIN
    --SQL statements
END;
```

- example :-

```
create procedure get_passed_students()
begin
    select * from student
    where marks > 50;
end;
```

- call the procedure

```
call get_passed_students();
```

- o/p → Displays all students with marks greater than 50