**Database Management System Notes**

**1. File Storage**

* **Definition**:  
  File storage, also called file-level or file-based storage, is a hierarchical storage methodology where data is stored in files, organized in folders and directories.

**Advantages of File Storage**

* Simple to implement – Easy for small-scale applications.
* Low cost – No need for specialized software.
* Direct access – Files can be opened and edited directly.
* Good for static data – Ideal for storing images, videos, logs, or configuration files.

**Common Uses**

* Storing media files (photos, videos, audio)
* Log files for applications
* Configuration files
* Simple record keeping (e.g., CSV or TXT files)

**2. Database Management System (DBMS)**

* **Definition**:  
  A DBMS is software that manages structured data in databases, allowing efficient data definition, storage, retrieval, and manipulation.

**Advantages of DBMS over File Storage**

* Reduced data redundancy – Avoids duplication by storing data once.
* Improved data consistency – Updates are reflected across all applications.
* Data integrity – Ensures valid and accurate data through rules.
* Concurrent access – Multiple users can access and modify data safely.
* Security – Controls access and modifications.
* Efficient searching – Queries allow fast data retrieval.
* Data independence – Storage changes don’t affect applications.

**SQL Queries**

**Table Creation**

create table employee( id int PRIMARY KEY, name VARCHAR(40) NOT NULL , designation VARCHAR(60) NOT NULL, email VARCHAR(60) NOT NULL);

Creates an employee table with fields for ID, name, designation, and email.

create table e\_commerce\_orders(order\_id int PRIMARY KEY, customer\_id int NOT NULL, city VARCHAR(40) NOT NULL, status VARCHAR(50) NOT NULL, order\_date date NOT NULL, amount decimal NOT null);

Creates an e\_commerce\_orders table to store order details.

**Data Insertion**

insert into employee values(101,"Sayali","STE", "powarsayali2002@gmail.com");

Inserts one employee record into the employee table.

INSERT INTO e\_commerce\_orders (...) VALUES (...);

Inserts multiple records into the e\_commerce\_orders table.

**Basic Select Queries**

select \* from e\_commerce\_orders;

Displays all records from the e\_commerce\_orders table.

select order\_id as ID, customer\_id as Customer from e\_commerce\_orders;

Displays order ID and customer ID with aliases.

select \* from e\_commerce\_orders where order\_id=3;

Displays the order with ID 3.

select \* from e\_commerce\_orders order by order\_date desc;

Displays all orders sorted by date in descending order.

**Aggregate Functions**

select COUNT(\*) AS total\_orders, SUM(amount) as Total\_revenue, AVG(amount) as avg\_oder\_value, MAX(amount) as max\_order, MIN(amount) AS min\_order from e\_commerce\_orders where status ="PAID";

Calculates total orders, revenue, average, max, and min order amounts for paid orders.

Select city, SUM(amount) as revenue from e\_commerce\_orders where status="PAID" GROUP BY city;

Shows revenue per city for paid orders.

Select city, SUM(amount) as revenue from e\_commerce\_orders where status="PAID" GROUP BY city having sum(amount)>2500;

Shows cities with revenue greater than 2500 for paid orders.

**Top Orders and Calculations**

Select order\_id, city, amount from e\_commerce\_orders order by amount DESC limit 3;

Displays top 3 highest amount orders.

select order\_id, amount, amount\* 0.18 AS gst\_tax from e\_commerce\_orders;

Calculates GST tax (18%) for each order.

**Advanced WHERE Clause Usage**

**Customer Table Creation and Insertion**

create table customer( customer\_id int PRIMARY KEY, name VARCHAR(40), email VARCHAR(50));

Creates a customer table with customer details.

insert into customer values(101, "raj", "raj@mail.com"),(102, "meera", "meera@mail.com"),(103, "arjun", "arjun@mail.com");

Inserts multiple customer records.

**Join Queries**

select o.order\_id, c.name, o.amount from e\_commerce\_orders o JOIN customer c ON o.customer\_id = c.customer\_id where o.status= "PAID";

Displays paid orders with customer names using inner join.

select o.order\_id, c.name, o.city, o.amount from e\_commerce\_orders o Inner join customer c ON o.customer\_id = c.customer\_id;

Displays order details with customer names using inner join.

select c.customer\_id, c.name, o.order\_id, o.amount from customer c LEFT JOIN e\_commerce\_orders o ON c.customer\_id=o.customer\_id;

Displays all customers and their orders using left join.

**Filtering and Pattern Matching**

select distinct city from e\_commerce\_orders;

Displays unique cities from orders.

select order\_id, city, amount from e\_commerce\_orders;

Displays order ID, city, and amount.

select \* from e\_commerce\_orders where city="Pune" and city="Delhi";

Returns no result as a city cannot be both Pune and Delhi.

select \* from e\_commerce\_orders where city="Pune" or city="Delhi";

Displays orders from Pune or Delhi.

select \* from e\_commerce\_orders where amount between 1000 and 3000;

Displays orders with amount between 1000 and 3000.

select \* from e\_commerce\_orders where city in ("Mumbai" , "Delhi");

Displays orders from Mumbai or Delhi.

select \* from e\_commerce\_orders where city like "M%";

Displays orders from cities starting with 'M'.

select \* from e\_commerce\_orders where city like "\_u\_b%";

Displays cities matching the pattern (e.g., 'Mumbai').

**Join Queries**

select o.order\_id, c.name, o.amount from e\_commerce\_orders o JOIN customer c ON o.customer\_id = c.customer\_id where o.status= "PAID";

Displays paid orders with customer names using inner join.

select o.order\_id, c.name, o.city, o.amount from e\_commerce\_orders o Inner join customer c ON o.customer\_id = c.customer\_id;

Displays order details with customer names using inner join.

select c.customer\_id, c.name, o.order\_id, o.amount from customer c LEFT JOIN e\_commerce\_orders o ON c.customer\_id=o.customer\_id;

Displays all customers and their orders using left join.

select o.order\_id, o.amount, c.name from e\_commerce\_orders o Right JOIN e\_commerce\_orders o ON c.customer\_id=o.customer\_id;

Displays all customers and their orders using right join.  
select c.customer\_id, c.name,o.order\_id, o.amount From customers c RIGHT JOIN orders o

ON c.customer\_id= o.customer\_id;

select c.name, o.order\_id FROM customer c CROSS JOIN e\_commerce\_orders o;

* + Group by and Order by

select Date(order\_date) As order\_day,

count(\*) as e\_commerce\_orders,

SUM(amount) as revenue

from e\_commerce\_orders

where status= "PAID"

GROUP BY DATE(order\_date)

ORDER BY order\_day;

select customer\_id, count(\*) as paid\_orders

from e\_commerce\_orders

where status="PAID"

GROUP BY customer\_id

HAVING COUNT(\*) >=2

Order By paid\_orders desc;

select city, status,count(\*) as cnt

From e\_commerce\_orders

GROUP BY city, status

ORDER BY city, status;

select COALESCE(city, 'unknown') as city\_label, SUM(amount) AS revenue

from e\_commerce\_orders

where status= "PAID"

GROUP BY COALESCE(city, 'Unknown');

Select city, COUNT(DISTINCT customer\_id) as unique\_buyers

from e\_commerce\_orders

where status="PAID"

GROUP BY city

ORDER BY unique\_buyers DESC;  
  
rules regarding group by and order by :

CREATE TABLE employee (

emp\_id INT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

dept\_id INT NOT NULL,

salary INT NOT NULL,

FOREIGN KEY (dept\_id) REFERENCES departments(dept\_id)

);

insert into Employee values(1,"John", 101, 60000),(2, "Alice",102,50000),(3, "Bob",101,50000);

create table departments(dept\_id int PRIMARY KEY, dept\_name Varchar(50));  
  
insert into departments values(101,"IT"),(102,"HR"),(103, "Finance");

* + Subquery -

select name, salary

from employee

where salary > (select avg(salary) from employee);

SELECT e.name

FROM employee e

JOIN departments d ON e.dept\_id = d.dept\_id

WHERE d.dept\_name IN ('IT', 'Finance');

select name

from employee

where dept\_id in (select dept\_id from departments where dept\_name In ("IT", "Finance"));

select dept\_id, MAX(salary) as highest\_salry

from employee

Group By dept\_id;

select e.name, e.salary, d.dept\_name

from employee e

JOIN (SELECT dept\_id, MAX(salary) as Highest\_salary

from employee

GROUP BY dept\_id) temp

ON e.dept\_id = temp.dept\_id AND e.salary = temp.highest\_salary

JOIN departments d ON e.dept\_id = d.dept\_id;

Views :

create view employee\_salaries AS

select name,salary

from employee;

select \* from employee\_salaries;

create view employee\_departments as

select e.name, e.salary, d.dept\_name

from employee e

join departments d on e.dept\_id = d.dept\_id;

select \* from employee\_departments;

create view high\_salary\_employees as

select name,salary

from employee

where salary>40000;

select \* from high\_salary\_employees;

SHOW FULL TABLES where Table\_type= "VIEW";

show create view employee\_department;  
  
Indexing :

* + Creating the index
  + Create index idx\_name on Employee(name);
  + Now searching by name will be much faster

Now search again :

Select \* from employee where name =’sarah’