SQL → Structured Query Language.

Q What is Database?

→ A database is an organized, electronic collection of data, stored in a structured format (usually tables) allowing for easy access, management & updating.

Key features:

- Structured storage: uses tables
- Consistency: Uniform data format
- Integrity: Ensures accuracy & reliability
- Scalability: can handle growing data efficiently.

Q] What is DBMS?

A Database management system is a software that lets users create, manage & interact with dB. It acts as a bridge between users & the data. ensuring secure & consistent operations like insert, delete, update & retrieve.

Q] What is RDBMS.

→ It is a Relational DBMS, it is a type of DBMS that stores data in table & manages relationships between them using primary & foreign keys

Key Concepts:

- Tables (rows = records, columns = attributes)
- primary key = unique identifier
- foreign key = Establishes relation between tables
- SQL is used for managing RDBMS
- Normalization reduces redundancy.

Examples: MySQL, PostgreSQL, Oracle, SQL server

Q] What is SQL ?

- SQL is standard language used to manage and manipulate relational databases.

1] DQL → Query data (eg - select)
2] DML → Modify data (eg - Insert, update, delete)
3] DDL → Define structure (eg, create, Alter, Drop)
4) DCL → control access (eg - GRANT, Revok)
5) TCL → Manage Transaction (Commit, Rollback)

In MySQL or sql we can give comments by 3 ways

1] Single line Comment (using --)

```
-- This is single-line comment.
select * from employee; -- This commen is on the same line of query.
```

2] Single line comment using "#"

```
# This is comment
Select *from employee;
```

3] multiline comment (/* */)

```
/* This is a
multi line
comment */
Select *from employee;
```

Module 1 → Create db, table, show table, diff showing ideas.

```
create database employees;

use employees;

Create table employee (
  emp-id int primary key,
  firstname Nvarchar (60),
  Lastname Nvarchar (60),
  dept varchar (100),
  salary decimal (10,2)
  hiredate ~~int~~ date
);
```

Nvarchar is used to display things other than english also like - प्रतिक
प्रतिक 😊 - emoji also
any lang like - hindi, chinese, emoji

```
insert into employee
(emp-id, firstname, Lastname, dept, salary, hiredate)
values

(1, "pratik", "Patil", "data analyst", "120000.00", "2025-11-22"),
(2, "Pushkar", "Patil", "Product engg", 86000.95, "2024-12-30"),
(3, "Akshay", "Patil", "full stack dev", 76000.83, "2025-12-19"),
(4, "Mayur", "Patil", "data analyst", 80000.00, "2025-11-22"),
(5, "Jayesh", "Patil", "data engg", 70000.00, "2025-12-12");
```

select * from employee;

This will give table

select emp-id, firstname from employee;

This will just retrieve two columns

emp_id	firstname
1	Pratik
2	Pushkar
3	Akshay
4	mayur
5	Jayesh

select emp-Id, concat(firstname, " ", lastname)

as fullname from employee;

(New column name)

This method is used to combine two columns and display them

Hint → Use Single (' ') quotes instead of two.

SQL-Select distinct

```
use students;

select distinct name from students;

select distinct dept from students;

select distinct * from student;
```

This distinct keyword is used to display unique records only from table.

SQL Temporary Tables

```
create Temporary table temp1 as
select * from student;
```

The above query is used in MYSQL

1)
```
select * into #temp1.
from [dbo].[Employees]
```

This Single hash is used to create local variable table that means this is not going to run/ execute in another query windows

2)
```
select * into ##temp2.
from [dbo].[Employees]
```

This double hash (##temp2) is used to create global table where it can be used in another query also.

SQL Where Clause.mp4

```
1) select * from Employees
2) where id = 1.
```

o/p → prank's flow →

```
3) select * from Employees
4) where salary > 80000
```

```
5) select distinct firstname, lastname, department. from Employees
   where salary > 80000
```

5. SQL ORDER By Clause

```
use employees

select * from Employees
orderby firstname

select * from Employees
order by first name asc, salary desc.
```

first the table will be sorted by name & then the salary will be sorted according to name.

eg

Name	Salary
Akshay	700000
Jayesh	600000
Mayur	720000
Pratik	1200000
Pushkar	800000

Salary sorting by descending

name is sorted by ascending order

SQL Operators

~~Select~~

There are several operators in SQL. Operators are used to perform operations of data.

Operators	Example
1) = (Equal to)	where age = 25
2) != (Not equal to)	where name != 'John' / where name <> 'John'
3) > (Greater than), < (less than)	where salary > 7000 / where salary < 70000
4) >= , <=	~~Sel~~ where salary <= 700000 / salary >= 10000
5) Between (Range exc.)	where age Between 18 and 25
6) In (Value in list)	where city In ('Pune', 'Mumbai')
7) Like (Pattern match)	where name like 'A%'
8) Is Null (Check for Null)	where address is Null
9) AND (True if both true)	where name = 'Pratik' AND age = 21
10) OR (True if any one condition is True)	where name = 'Pratik' or age = '22'
11) Not (Reverse the condition)	where ~~name~~ Not city = "Mumbai"
12) + (String Concatenation)	Select 'Hello' + 'World';

Delete Drop & Truncate

1] Delete

- Used to remove specific rows from a table
- Can use where clause to filter which rows to delete.
- Slower than Truncate for large datasets.

Example

```
Delete from Employees
where Department = 'HR';
```

2] Truncate

- Delete all rows from table - no filtering
- Faster than Delete because it doesn't log each row deletion.
- Cannot use where clause.
- Does not activate triggers

eg →

```
Truncate table Employees.
```

3] Drop

- Completely removes the table (structure + data).
- After Drop, the table is no longer available.
- Cannot be used with where.

- Deletes everything permanently - structure, data, constrains indexes, etc.

Example→

Drop table Employees

Drop Database Company

Top records views top n

---

select top2 * from Employees

select top 2 firstname, lastname from Employees.

select top4 Department from Employees.

Update SQL

```
Select * into #1
from Employees
```

```
update #1
set department = 'HR'
where department is Null
```

```
update #1
set salary = 112000
where ID = 1
```

```
update #1
set salary = 120000 , department = finance
where id = 7.
```

```
updak #1
set salary = 90000 , Hiredate = '2025-08-04'
where salary is null or hiredate is null
```

Use to update table name

```
exec sp_rename 'oldname', 'NewName'
```

SQL MAX & GroupBy & Min - GroupBy.

```
Select max(quantity) from sales

Select max(paymentmethod) from sales
selet max(paymentmethod) [max.payment method] from Sales.
                                                  alias
Select productid, max(quantity) from sales
Groupby productid.


select min(quantity) from Sales

select min (quantity) [minimum quantity] from sales.
                                          alias

Select productId, min(TotalSales) from sales
Group by productID.
```

SQL Sum, Avg, Count function

```
Select sum(totalamount) as [sum of total amount]
from sales

select avg (totalamount) as [avg total amount], sum(total amount)
[sum total amount]
from sales.

Select
productid, storeid,
sum(quantity) as [sum quantity],
sum (totalamount) as [Sum Amount],
avg (quantity) as [avg quantity],
avg (total amount) as [totavg amount].
from sales
Group by productid, storeid.
```

--Count function

```
select * from sales

select count(*)[no of rows] from sales

select productid, count(productid) from sales
group by productid.

select paymentmethod, count(*) from sales
groupby paymentmethod.
```

Having Clause

where	Inidival rows	Total amount 7200
Having	Group after aggregation.	Sum(Total amount)>500

```
Select
Productid, storedid,
Sum(Total amount),
avg(quantity)
from Sales
Group by productid, storeid
having sum(Total amount) < 700.
```

Q show number of sales per storeid, only for stores that had more than 3 sales.

```
Select
storeid, count(*)
from sales.
group by storeid
having count(*) > 3
```

This counts number of rows.

Joins

- Inner Join
- Left Join
- Right Join
- Left Anti Join & Right Anti Join
- Full Outer Join.
- Self Join
- Union & union All Join

Table A

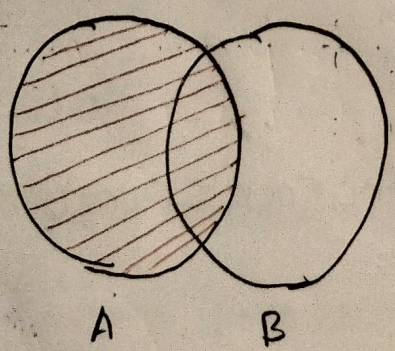
1	A
1	B
2	C
null	D
3	E
7	DA

Table B

1	xA
2	mB
2	NX
null	MO
4	MY
5	TF

1] Left Inner Join →

- Suppose we have two tables say A & table B then by using inner join we will retrieve all the element from left and common element from right.

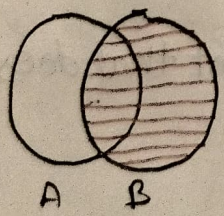


	C1	C2	C3
1	1	A	XA
2	1	B	XA
3	2	C	MB
4	2	C	NX
5	Null	D	Null
6	3	E	Null
7	7	DA	Null

Select
a.C1, a.C2, b.C3
from table1 as a
Left Join table2 as b
ON a.C1 = b.C1

2] Right Join

- It is vice versa of Left Join or say similar to Left Join. In Right Join Right table is retrieved and common elements from left table is retrieved.

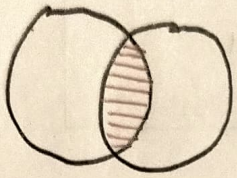


Select
b.C1, a.C2, b.C3
from table1 as a
left Join table2 as b
on a.C1 = b.C1

	C1	C2	C3
1	1	A	XA
2	1	B	XA
3	2	C	MB
4	2	C	NX
5	Null	Null	MO
6	4	Null	XY
7	5	Null	TF

3] Inner Join

- In Inner Join only common elements from both table are retrived.



	C1	C2	C3
1	1	A	XA
2	1	B	XA
3	2	C	MB
4	2	C	MX

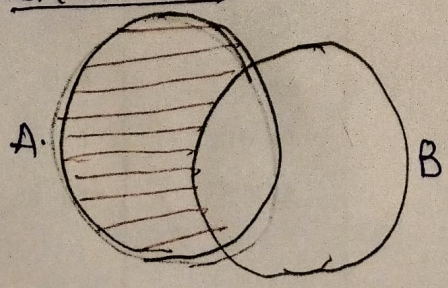
```
select
a.C1, a.C2, b.C3
from table1 as a
inner Join table2 as b
on a.C1 = b.C1
```

4] Left Anti Join & Right Anti Join

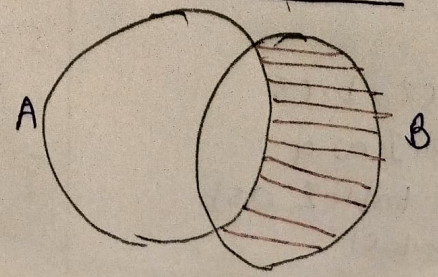
- Left ANTI Join → Only records from left with no match in right

- Right ANTI Join → Only records from Right that does not match in left.

Left anti Join



Right Anti Join



Left anti Join

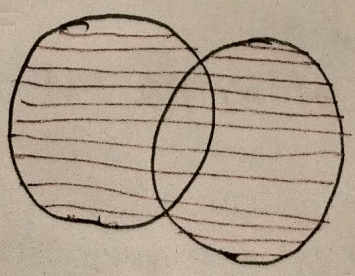
```
select
a.c1 , a.c2 , b.c1 , b.c2
from table1 as a
left Join table2 as b
on a.c1 = b.c1
where b.c3 is null
```

Right anti Join

```
Select
b.c1 , b.c3 , a.c2
from table1 as a
Right Join table2 as b
on a.c1 = b.c1
where a.c2 is null
```

5] full outer Join

- It combines both the table and the retrieve the data if no match found then Retrieves "NULL"
- It does not remove duplicate data.



```
Select * from table1 as a ~~from~~ ~~fo~~
full outer join table2 as b
on a.c1=b.c1
```

6] ~~Inner~~ Self Join

```
select * from table1.

Select
a.city, ~~b.city~~ a.name, b.name
from people as a
Join people as b
on a.city = b.city and a.id < b.id.
```

id	Name	City
1	Alice	Pune
2	BOB	Pune
3	Carol	Mumbai
4	David	Pune
5	Eve	mumbai

Consider above table for understanding Self Join

O/P →

	City	name	Name
1	Pune	Alice	BOB
2	Pune	Alice	David
3	Pune	BOB	David
4	Mumbai	Carol	Eve

7. UNION & UNION ALL

- UNION
  - Combines rows from two tables
  - Table must have same number of columns
  - Think of it like copy-pasting rows from one table below the other
  - Removes duplicate (use UNION All to keep duplicates)

Feature	Union	Full outer Join
1) Combines data by	Rows	Columns
2) Based on matching	No matching	Based on 'ON' condition
3) Duplicates	Reved (by default is not used onion all)	Not applicable.
4) Null for No match	Not applicable	Yes
5] Use case	Merging results for two similar queries	Combining related data from two tables.

```
Select * from append1
union All
Select * from append2
```

```
Select * from append1
union
Select * from append2
```

```
select c1,c2,c3 from append1
union
select c1,c2,c3 from append2
```

Names given to colm. in first select statement will be used for table.

```
select c1 as [c
```

Like Operator

① - find Employees whose last name starts with 'S'

→ select * from employees where lastname like 'S%'

② - find Employees whose last name ends with a

→ select * from employees where lastname like '%a'

③ find ~~op~~ department whose ~~first name ends with 'a'~~

→ select * from employes

3) Find employees whose department contains Eng

→ Select * from employees where department like '%Eng%'

4) find employees whose last name is exactly 5 chars long

→ select * from employees where ~~de~~lastname like '____'

5) find employee whose first Name contains the letter 'i' as the second character

→ select * from employees where firstname like '_i%'

6) find employees whose firstname starts with C or D

→ select * from employees where firstname like '[CD]%'

7) find employees whose last name starts with any char between A & L

→ Select * from employees where last name like '[A-L]%'

8] find Employees whose firstname does not contain 'a'

→ select * from employees where firstname not like '%a%'

9] find Employees whose firstname starts with vowel

→ select * from employes where firstname like '[aeiou]%'

10] find Employees whose firstname does not start with consonants.

→ select * from employees where firstname not like '[^aeiou]%'

11] find employees whose firstname start with consonants

→ select * from employees where first name like '[^aeiou]%'

Case in Select Statement

```
Select * from products.
```

Q. Add a column to a categorize each product into category of high, low or medium.

1st approach

```
Select
*,
 case
    when price >500 then 'high'
    when price <= 500 and price >= 200 then 'medium'
    when price < 200 then 'low'
 end.
from products.
```

2nd approach.

```
Select
*,
   case
       when price >500 then 'high'
       when price <=500 and price >200 then 'medium'
      else 'low'
   END
from products
```

9 Sort the data according to priority first assign priority to them

```
select * from products
order by
  case
    when category = 'electronics' then 1
    when category = 'furniture' then 2
    when category in ('accessories') then 3
  end asc.
```

Nested Case Statement

```
Select
*,
   case
      when category = 'electronics' then
         case
           when price > 300 then 'Premium Product'
           else 'Affordable'
         END
      when category in ('furniture') then
         case
           when price > 300 the 'premium Product'
           else 'Affordable'
         END
      else
         case
           when price > 300 then 'premium product'
           else 'Affordable'
         end
   end
from products.
```

Not Null Constraints

Constraints → Conditions that can be applied on columns of a table & these conditions are to be followed while inserting records into the table.

Not Null Constraints → This is the condition that allows to follows certain roles on the particular column that values cannot be null or left empty on the column on which Not null constraints is applied.

Case 1 → We have to create a new table.

```
create table not_null_constrain (
  id int not null,
  firstname varchar(256),
  age tinyint.
)

insert into not_null_constrain
values
(1, 'Zenitsu', null), ✓
(2, 'Inuzan', 23), ✓
(3, Null, 24), ✓
(Null, 'Nezuko', 14); ✗
```

Alter table not-null_constraint.
Alter column age tinyint not null

UNIQUE CONSTRAINT

```
create table unique_table (
id int unique,
firstname varchar(256),
age tinyint,
Phone varchar(15) unique.
)
```

This avoids the duplication while inserting data into ~~tot~~ table.

Check Constraint.

It checks certain conditions that can be applied on the columns of a table, if this condition is not fulfilled, we will not be able to insert the records into the table.

```
create table check_test (
id int,
firstname varchar(256),
age tinyint check(age>18)
)
```

Alter table ~~check_test~~.
~~Alter add check~~.
Alter table test_check
add check(eid >5)

Default constraint.

```
Create ~~defo~~table test_default (
 id int unique,
 firstname varchar(25),
 age tinyint default(18)
)

insert into test_default (id, firstname)
values ( 1, 'Krish' )
select * from test_default.
```

o/p

id	firstname	~~vote~~ age
1	Krish	18

Primary Keys

- Primary Key is a column in a table that uniquely identifies each row in that table.

```
create table test_primary_key(
  id int primary key,
  name varchar(256)
)
```

```
insert into test_primary_key values
('1', 'Krish') ✓
('2', 'Jayant') ✓
('3', 'Pratik') ✓
(null, 'Heralal') Error null not alloweed.
```

If table already exists !

```
Alter table test_primary_key_2
add primary key (id)
```

Foreign Key

```
select * from test_primary_key
```

```
create table test_foreign_key (
    id int foreign key refrenses test_primary_key(id),
    course varchar(50)
)
```

Foreigns key can have duplicate keys, null keys, but they cannot be other than one in primary key table.

```
insert into test_foreign_key values
    (1, 'Data analyst'),
    (2, 'Statistics')
    (9, 'Cloud eng')
```

Error → because 9 is not in primary key table.

If table already exists.

```
Alter table key_foreign_table
add foreign key(id) references test_primary_key(id)
```

SQL order

Order of execution should be.

```
Select distinct top 1 Department, AVG(salary) as
[AVG Salary]
from EmployeeSalaries
where Salary > 50000
group by Department
having AVG(salary) > 50000
order by department Desc.
```

order for query

From & Joins
where
group by
having
select
distinct.
order by
top

Order by System execution