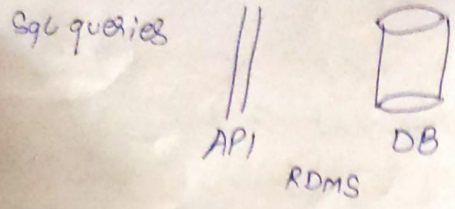
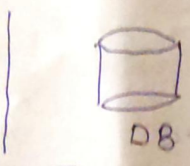
SQL

Structured query language → structured query lang.

RDMS (types) → MySql, Oracle, My access etc.



- CRUD → create
  update
  Read
  Delete



RDMS → provide CRUD oper. to DB

RDMS (types) ⇒ MySQL → It's a software which uses SQL lang.
- It's a open source RDMS

MS SQL server
Oracle.
IBM

- SQL
  - Query lang.
  - Way to access data

- MySQL
  - MySQL: itself a RDMS
  - CRUD done init using SQL.

→ My SQL Community Server
→ My SQL workbench

7

```
CREATE DATABASE temp;

USE temp;
```

→ table create karne se phle btana hoga ki konsa database use karenge

```
CREATE TABLE student(
  id INT PRIMARY KEY,
  name VARCHAR(255)
);
```

← SQL spec. key word capital rakhna taki baad wale aasani se dikh jaae

← SQL command ki student table bna do → RDMS

```
INSERT INTO student VALUES(1, 'Antik');
```

→ adds one entry to table.

```
SELECT * FROM student;
```

→ to fetch data

* → for all data of table

SQL Data types

- CHAR → fixed DT
- VARCHAR → variable DT

VARCHAR(255) → [B|A|M] → this uses 3 bytes only

CHAR(255) → [B|A|M| ...] (0 to 254) → this uses 255 bytes.

RAM → effectively uses 3 bytes

- DATE → YYYY-MM-DD
- DATE TIME → YYYY-MM-DD HH:MM:SS

ENUM → one of the preset value

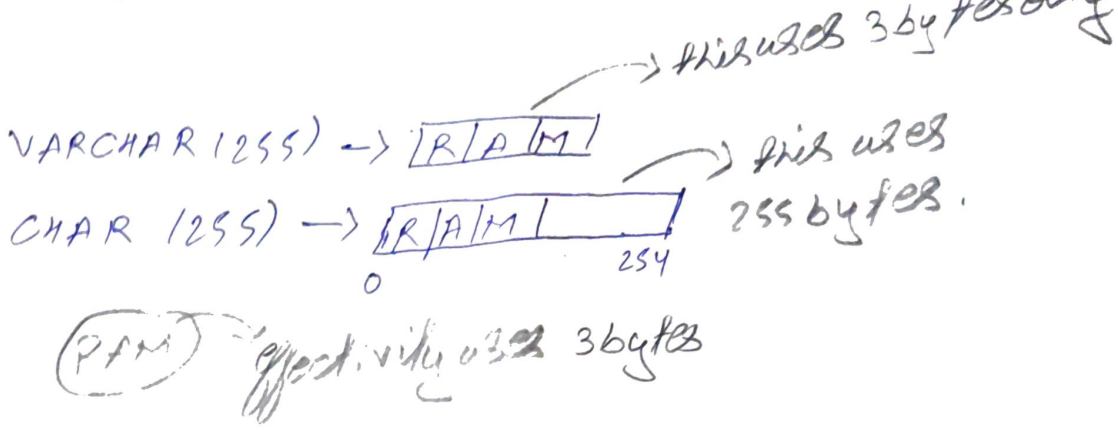
(A, B, C) → koi ek assign kar sakte h.

Refer W3SCHOOLS.com/sql for clarity.

simple → now

SQL Data types

- CHAR → fixed DT
- VARCHAR → variable DT



- DATE → YYYY-MM-DD
- DATE TIME → YYYY-MM-DD HH:MM:SS
- ENUM → one of the preset value
  - (A, B, C) → koi ek assign kar skta h.

Refer W3schools.com/sql for clarity.

tuple → mean → row

- SIGNED, unsigned

TINY INT ⇒ (-128 to 127)

UNSIGNED TINY INT ⇒ (0 → 255)

Create TABLE tablename (

col INT,

col2 INT UNSIGNED → col 2 mae 0-255 ki value vela put ho sakti h

);

* SQL Types of command

1. DDL (data def. lang): defining relation schema
   - CREATE : create table, DB, view
   - ALTER TABLE : modification in table structure. eg change col. dt.
   - DROP : delete table, DB, view
   - TRUNCATE : remove all the tuples from the table.
   - RENAME : rename DB name, table name etc

2. DRL / DQL (data retrieval lang / data query lang):
   retrieve data from tables.
   - SELECT

3. DML (data modification lang): use to perform modification in DB

- INSERT: insert data into a relation.
- UPDATE: update relation data.
- DELETE: delete row(s) from relation.

4. DCL (Data control lang): grant & revoke authorities from user.

- GRANT: access privileges to DB
- REVOKE: revoke user access privileges.

5. TCL (Transaction control lang): To manage transaction done in the DB

- START TRANSACTION
- COMMIT;
- ROLLBACK;
- SAVE POINT

* Managing DB (DDL)

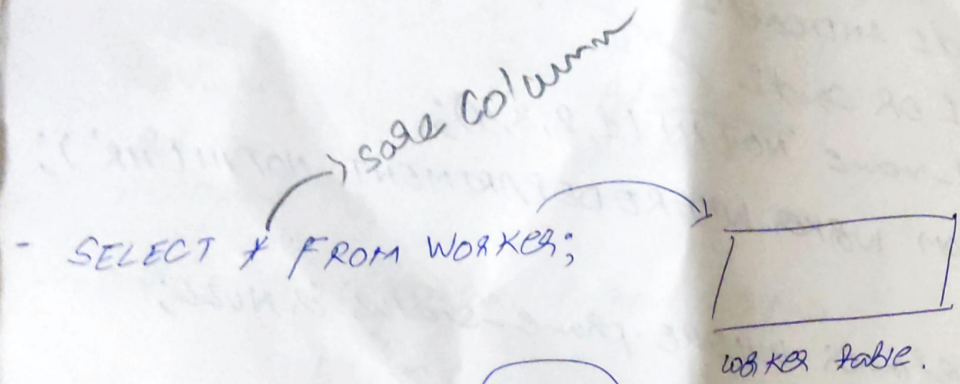
1. Creation of DB

- CREATE DATABASE IF NOT EXISTS db-name;
- USE db-name;
- DROP DATABASE IF EXISTS db-name;
- SHOW DATABASES;
- SHOW TABLES;

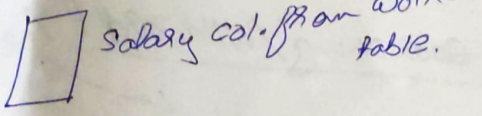
* DQL

① DB
② table

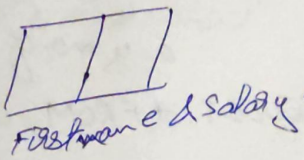
* DATA RETRIEVAL LANGUAGE (DRL)
- Syntax: SELECT <set of col names> FROM <table_name>;
- Order of execution from RIGHT to LEFT.
-



SELECT SALARY FROM WORKER;



SELECT FIRST_NAME, SALARY FROM WORKER;



separated
comma keyboard next word alag

• WHERE
- reduce rows based on given conditions.
- Eg SELECT * FROM CUSTOMER WHERE age>18;

• BETWEEN
- SELECT * from customer WHERE age between 0 AND 100;

- IN
- Reduces OR conditions;
- SELECT * FROM ~~officers~~ WORKER WHERE officer_name IN ('Lakshay', 'Arun');

SELECT * FROM WORKER WHERE DEPARTMENT = 'HR' OR DEPARTMENT = 'Admin';

- AND/OR/NOT
- AND: WHERE cond1 AND cond2
- OR: WHERE cond1 OR cond2
- NOT: WHERE Ed_name NOT IN (1, 2, 3, 4); → NOT ke baad IN
  - SELECT * FROM WORKER WHERE DEPARTMENT NOT IN ('HR');
- IS NULL
- SELECT * FROM customer WHERE prime_status is NULL;
- Pattern Searching/Wildcard ('%', '_')
  - '_' → only one character
  - like '*' → any no. of character (0-∞)
- SELECT * FROM Customer ~~ORDER BY name DESC~~ WHERE name LIKE '%p_';
  - all name whose 2nd last letter is p.

WILDCARDS

ex-

'%pa%'

- 'abcpadb'
- 'pabc'
- 'adjpa'

mean pa ke bagal me kewal ek char. aaya

'pa_' → apab

* Sorting using ORDER BY

- SELECT * from worker order by salary; → default asc h
  ↓
  " " salary DESC; → for desc.

* Distinct values

- Department / distinct

HR
HR
Admin
Admin
Account

→ Total distinct dep.
HR
Admin
Account

- SELECT department from worker;
  ↓
  DISTINCT

* Data Grouping

- find no. of employes working in diff department.

HR → ? kitne employee h
Account → ? kitne employee h
Admin → ? "

Grouping → Aggregation (count)

GROUP BY → Aggregation function (count, SUM, AVG ...)

-- GROUP BY

select department from worker group by department;

→ ek group bna data h
2HR ka, 2 Admin ka

This acts distinct as no aggregation applied

before from and after Group by word same hoga

select department, COUNT(*) from worker group by department;
↳ department

O/P —
HR 2
Admin 4
Account 2

internally SQL ek dummy table bana kiya or count kiya ki
HR → 2 h. Admin → 4 h ...

-- AVG SALARY per department
select department, AVG(salary) from worker group by department;

O/P →

HR	2L
Admin	1L
Account	1L

→ Can be
MIN
MAX
for other operation

*HAVING

→ Select ke saath WHERE use hota h to filter
→ For GROUP BY filter
HAVING

-- department, count, having more than 2 worker?
-- GROUP BY HAVING } → aggregation

Select department, COUNT(department) from worker group by department HAVING COUNT(department) > 2;

O/P → Admin | 4

• WHERE VS HAVING

WHERE	HAVING
- filter table based on specified condition.	- filter groups based on specified condⁿ
- WHERE Can be used with SELECT, UPDATE & DELETE keywords	- Having Can be used with SELECT.

* DDL Key constraints

① primary key.
- Not NULL
- unique
- only one PK

Primary
PK → INT total

```
CREATE TABLE customer(
id INT PRIMARY KEY,
branch_id INT,
first name VARCHAR(50),
last name " ",
DOB DATE,
Gender CHAR(6),
)
```

② Foreign key.
- refers to PK of other table.
- Each relation can having any no. of FK.
- CREATE TABLE ORDER(
  id INT PRIMARY KEY,
  delivery-date DATE,
  order-placed-date DATE,
  FOREIGN KEY (cust_id) REFERENCES customer(id)
  );

③ Unique

```
Create table customer (
  :
  Name VARCHAR (255) UNIQUE,
  :
);
```

④ CHECK

- Consistency constrain (

```
  :
  CONSTRAINT acc_balance_chk CHECK( balance >1000),
  :
);
```

⑤ General assertion

```
CREATE ASSERATION assertion
  :
CHECK assertion cond";
```

↳ checks if each modification that could potentially violate it.

ex - member.uid references user.uid

```
CREATE Assertion member-user-integrity
CHECK (not Exists (select * from member
where uid not in (select uid from user)))
;
```

```
Create table account (
 id int primary key,
 name varchar (255) UNIQUE
acc_balance INT,
 Constraint acc_balance_chk CHECK (balance >1000)
);
```

(id int,
exp int check(exp > 5));

ye me bas ek check statement

```
Insert INTO account (id, name, balance)
values (1, 'A', 10000);
Insert INTO account (id, name, balance)
values (2, 'B', 100);
```

→ run → error

```
select * from account;
```

: DEFAULT

```
create table account (
  :
  balance INT NOT NULL Default 0,
  :
)
```

```
create table account(
id int primary key,
name varchar(255) UNIQUE,
balance INT NOT NULL DEFAULT 0
);
```

O/P

id	name	balance
1	A	0

```
INSERT INTO account(id, name)
values(1,'A');
```

→ balance hiding with balance was default value jaega.

```
INSERT INTO account(id,name)
values(2,'B');
```

~~DR~~. select * from account; → run → O/P

* ALTER operations

→ changes schema

→ ADD
- Add new col
- ALTER TABLE table_name ADD new_col_name datatype, ADD new_col_name_2 datatype;
- ex- ALTER TABLE customer ADD age INT NOT NULL;

→ MODIFY
- change datatype of an attribute
- ALTER TABLE table_name MODIFY col_name col_datatype;
- ex- VARCHAR TO CHAR.
  ALTER TABLE customer MODIFY name CHAR(1024);

→ CHANGE COLUMN → column name changes.

- Rename column name.
- ALTER TABLE table-name CHANGE COLUMN old col-name new col-name new-col-datatype
- Ex- ALTER TABLE customer CHANGE COLUMN name customer-name varchar(1024);

→ DROP COLUMN

- Drop a column completely.
- ALTER TABLE table-name DROP COLUMN col-name;
- Ex- ALTER TABLE customer DROP COLUMN middle name;

→ RENAME → table name changes

- Rename table name itself
- ALTER TABLE table-name RENAME TO new-table name;

~~Ex- Al~~

```
create table account (
id int primary key,
name varchar(255) UNIQUE,
balance INT NOT NULL DEFAULT 0
);
select * from account;
-- Add new column
ALTER TABLE account ADD interest FLOAT
NOT NULL DEFAULT 0;
```

-- MODIFY

alter table account MODIFY interest DOUBLE NOT NULL DEFAULT 0;

DESC account; → gives description of account table column

Row

id	int
name	varchar
balance	int
interest	double

-- CHANGE COLUMN - Rename the col

alter table account change colomn interest Saving - interest FLOAT NOT NULL DEFAULT 0;

-- RENAME THE TABLE

alter table account Rename TO account_detail8;

* DATA MANIPULATION LANGUAGE (DML)

→ INSERT

- INSERT INTO table-name (col1, col2, col3)
  values (V1, V2, V3), (val1, val2, val3);

→ UPDATE → change values of a tuple

- UPDATE table-name
  SET col1 = p, col2 = 'abc' WHERE id = 1;

id	col1	col2
1	p	abc

- update multiple rows

  Q-
  update student SET standard = standard + 1;
  (SET → is compulsory) (standard column ki sari value +1 ho jaegi)

⇒ ON UPDATE CASCADE

can be added to table while creating constraints
Suppose there is a situation where we have two tables such that primary key of one table is the foreign key for another table. if we update the primary key of the first table then using the ON UPDATE CASCADE foreign key of the second table automatically get updated.

→ DELETE

- Delete FROM table-name where id = 1;
- " " " ; // all row will be deleted.

- DELETE CASCADE
  (to overcome DELETE constraint of referential constraints)

?Q what would happen to child entry if parent entry is deleted?

```
CREATE TABLE ORDER(
order_id int PRIMARY KEY,
delivery_date DATE,
cust_id INT,
FOREIGN KEY (cust_id) REFERENCES customer(id)
ON DELETE cascade
);
```

→ ON DELETE NULL
(can FK have all null values?)

```
Create table order(
order_id int PRIMARY KEY,
delivery_date DATE,
cust_id INT,
FOREIGN KEY(cust_id) REFERENCES
customer(id) ON DELETE SET NULL
);
```

→ REPLACE
- primarily used for already tuple in a table.
- As UPDATE, using REPLACE with the help of WHERE clause in PK, then that row will be replaced
- As insert, if there is no duplicate data new tuple will be created.

REPLACE INTO student (id, class) values (4, 3);

REPLACE INTO table SET col1 = val1, col2 = val2;

```
Create database temp;
USE temp;
CREATE Table customer1
id integer Primary Key,
Cname varchar(225),
Address varchar(225),
Gender char(2),
City varchar(225),
Pincode integer
);
Select * from customer;
INSERT INTO customer (id, cname, Address, Gender, City, Pincode)
values (1300, 'Shaym Singh', 'Ludhiana H.O', 'M', 'Ludhiana',
                                                  141001),
(1210, 'Rohan', 'ASHOK Nagar', 'M', 'Jalandhar', 144002);
```

1st way to insert

```
insert into customer values
(1, 'codeHelp', 'delhi', 'M', 'Delhi', '110000');
```

2nd way.

```
insert into customer (id, cname) values (121, 'Bob');
```

now as only 2 details are available

3rd way.

Th bakiyo mae default value jaega nhi th null.

-- UPDATE → set mat bhulna use karna

UPDATE customer SET Address = 'Mumbai',
Gender = 'M' WHERE id = 121;

① update multiple rows

UPDATE customer SET PinCode = 11000; → sari pinCode ko 11000 kardo

✗ (error)

MySQL ek pass kuch hai jo bol raha hai ki kuch bhi gadbad hai jo ek baar mai sari rows ko change & del kar raha hai.

✓ SET SQL_SAFE_UPDATES = 0;

✓ UPDATE customer SET PinCode = 110000;

✓ update customer set PinCode = PinCode + 1;

-- DELETE

DELETE FROM customer id = 123;

Deletes a tuple

→ Referential Constrain (over come methods

① INSERT → value cannot be inserted in child if tab value is not existing in parent table

② DELETE Constrain

to overcome
- → ON DELETE CASCADE → agar parent se value delete karunga to child se bhi delete ho jaega
- → ON DELETE SET NULL → agar parent se entry delete kardi th uske corresponding entry jo child table mai h uske foreign ki jo value h woo Null set ho jaega

-- ON DELETE CASCADE

```
create table customer (

);

select * FROM customer

Insert into customer
values (1, 'Ram Kumar3', Dilbagh Nagar, 'M', Jalandhar, NULL);

create table order_details (
order id integer Primary key,
delivery_date DATE,
cust-id INT,
Foreign KEY(cust_id) references customer(id) ON DELETE CASCADE
);

Insert order_details
values (3, '2019-03-11', 1);

DELETE from Customer where id = 1;
```

→ imp statement

Jb ye nhi hoga jb deletion nhi hoga parent or child se

Jb ye h jb entry parent or child dono se delete ho jaega

-- ON DELETE set NULL

written at same place as of ON Delete cascade.

Same tables & everything just
ON Delete cascade ke jagah on Delete set NULL

agar mai Rajkumar ko delete karta hu parent table customer table se delete karta hu to child table mae foreign key mae NULL set hojaega

// order details ki baki entries ko kuch nhi hoga

-- REplace

- Data already present, Replace
- Data not present, INSERT

-- Replace
REPLACE INTO customer (id, city)
values (1251, 'Colony');

PK dena is necessary.

or baki entries mae null hoga

REPLACE INTO customer (id, cname, city)
values (1333, 'codehelp', 'Colony');

behave as insert as 1333 ki koi PK h he nhi

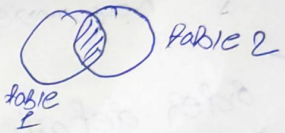
★ JOINS

- All RDMS are relational in nature,
  FK → Relations

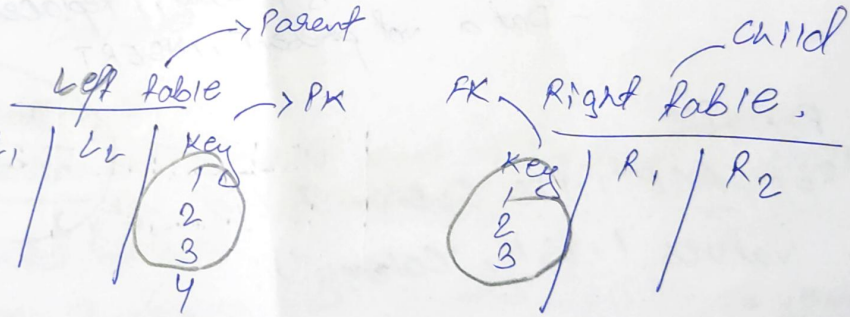
To fetch data → use → JOINS

→ INNER JOIN (intersection)

- Returns a resultant table that has matching values from both the tables & all the tables



SELECT column list FROM table1 INNER JOIN table2 ON condition 1.
INNER JOIN table3 ON condition 2.
...;



Jb Left table or Right table match hoga to ye dhkega kon kon match hua hai 1, 2, 3 match hoga hai to sbhi ye wale rows return hoga

O/P

Key	L1	L2	R1	R2
1				
2				
3				

- JOIN → To apply JOINS, there should be a common attribute

Syntax

→ customer table ke sare col.
→ order table ke sare
→ aliasing

```
select C.*, O.* FROM customers (AS) C INNER JOIN
orders AS O
(ON) C.id = O.cust_id
```

Syntax.

Child after INNER JOIN (Right)

Parent before INNER JOIN (Left)

- Alias in MYSQL (AS)
  - Aliases in MYSQL used to give temporary name to a table & a column in a table for the purpose of a particular query. It works as a nickname for expressing & column names. It makes the query short and neat

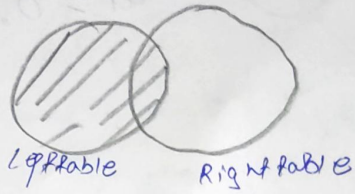
- SELECT col_name AS alias_name FROM table_name;
- SELECT col_name1, col_name2, ... FROM table_name AS alias_name;

→ OUTER JOIN

* LEFT JOIN

- This returns a resulting table that all the data from left table matched data from right table.

- SELECT column's FROM table LEFT JOIN table2 ON JOIN_condition;



Left Table

L1	L2	Key
		1
		2
		3
		4
		5
		6

Right Table

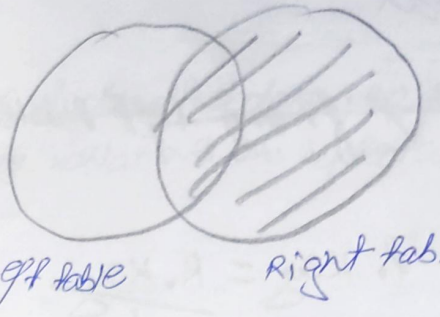
Key	R1	R2
1		
2		
3		

Resultant

Key	L1	L2	R1	R2
1				
2				
3				
4				
5			NULL	NULL
6			NULL	
			NULL	

- SELECT C.*, O.* FROM customers as C LEFT JOIN orders as O ON C.id = O.cust_id;

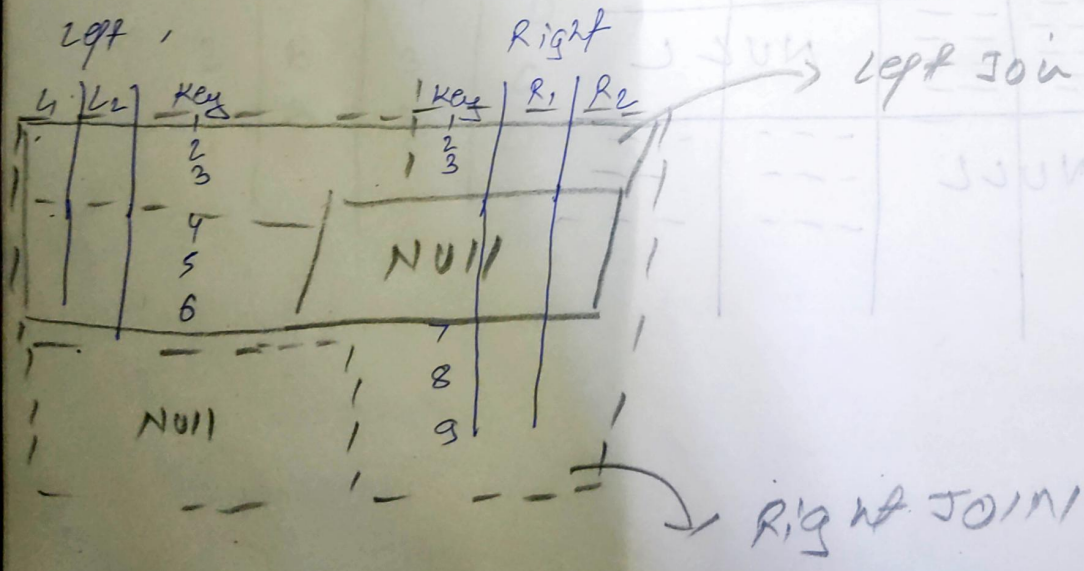
→ Right JOIN



Approx
Same as
LEFT JOIN

→ FULL JOIN

- This returns a resulting table that all the data from right table and matched data from left table data.
- Emulated in MYSQL using LEFT and RIGHT JOIN
- LEFT JOIN UNION RIGHT JOIN.



MYSQL FULL JOIN ~~X~~
Keyword ~~X~~

Syntax

Select * from left table as L LEFT JOIN Right table as R
ON L.Key = R.Key → with table R table ka key colo

UNION

Select * from left table as L RIGHT JOIN Right table as R
ON L.Key = R.Key;

Resultant

Left | Right

Key	L_1	L_2	R_1	R_2
1	--	--	--	--
2	--	--	--	--
3	--	--	--	--
4	--	--	NULL	
5	--	--		
6	--	--		
7	NULL		--	--
8			--	--
9			--	--

* CROSS JOIN

- This returns all the cartesian products of the data present in both tables. Hence all possible variations are reflected in the output.

→ - rarely used in practical.

T_L	T_R
5 rows	10 rows

Resultant Table: $5 \times 10 = 50$ rows.

Ex -

L_1	L_2
1	A
2	B

R_1	R_2
3	C
4	D

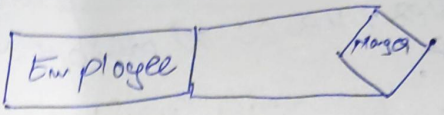
Resultant

L_1	L_2	R_1	R_2
1	A	3	C
1	A	4	D
2	B	3	C
2	B	4	D

SELF JOIN

imp

unary relation.



- It is used to get output from a particular table on the same table joined to itself.
- used very less.

- Emulated using INNER JOIN
- ALIAS 'AS'

e.g. –

```
select e1.id, e2.id, e2.name
FROM
    employee as e1
INNER JOIN
    employee as e2
ON e1.id = e2.id;
```

references behave as diff tables

Example

* Project

id	empID	name	startdate	clientID
1	1	A	2021-04-2'	3
2	2	B	2021-03-12	2
3	3	C	2021-01-16	5
4	4	D	2021-04-27	2
5	5	E	2021-05-01	4

* Employee

id	fname	lname	AGE	emailID	PhoneNo	City
1	Aman	Patel	32	aman@—	898	Delhi
2	Yogya	Nagpure	44	Yogya@—	222	Patna
3	Rahul	BD	22	Rahul@—	444	Kolkata
4	Jatin	Hemnit	31	jatin@—	666	Raipur
5	PK	Pandey	21	PK@—	555	Jaipur

* CLIENT

id	first-name	last-name	age	emailID	PhoneNO	City	empID
1	Mac	Rogers	47	mac@—	333	Kolkata	3
2	Max	Poirier	27	max@—	222	Kolkata	3
3	Peter	Jain	24	peter@—	111	Delhi	1
4	Sushant	Aggarwal	23	sushant@	45454	Hyderabad	5
5	Pratap	Singh	36	P@XYZ.com	77767	Mumbai	2

-- INNER JOIN
-- Enlist all the employees ID's names along with project allocted to them.

```
Select e.id, e.fname, e.lname, p.id, p.name from
employee as e
INNER JOIN project as p ON e.id = p.empID;
```

e.id → employee ka ID col.

O/P:

id	fname	lname	id	name
1			1	A
2			2	B
3			3	C
3			4	D
5			5	E

-- Fetch out all the employee ID's and their contact details who have been working from jaipur with clients name working in Hyderabad.

```
Select e.id, e.email ID, e.phone NO, c.first_name, c.last_name from
Employee as e
INNER JOIN client as c on e.id = c.emp ID where
c.city = 'Jaipur' AND c.city = 'Hyderabad';
```

O/P: 5 | PK@_1555 | Sushant | Aggarwal

-- LEFT JOIN
-- Fetch out each project allocated to each employee.

```
select * from Employee as e
LEFT JOIN project as p ON e.id = p.emp ID;
```

id
1
2
3
3
4
5

-- CROSS JOIN

-- LIST out all the combinations possible for the employee's name and projects that can be exist.

```
select e.fname, e.lname, pid, p.name from Employee as e
CROSS JOIN project as p;
```

(9) Can we use JOIN w/o using JOIN keyword?

- Yes

Syntax:

```
select * FROM left table, Right table WHERE
left table.id = Right table.id;
```

@- previous INNER JOIN '@' can be written as

```
select e.id, e.fname, e.lname, p.id, p.name from
employee as e,
project as p WHERE e.id = p.empID;
```

★ SET operations

- used to combine multiple select statement
- Always gives distinct rows.

Table 1

Col 1	Col 2
A	1
B	1
C	2

Table 2

Col 1	Col 2
A	1
B	2
D	3

→ UNION $T_1 \cup T_2$ → same type ke column same chahiye or same no. of column chahiye

Col 1	Col 2
A	1
B	1
C	2
B	2
D	3

T1 or T2 ke rows ko union karna or "bhi" duplicates ko remove karna

T1

Col 1	Col 2
A	1

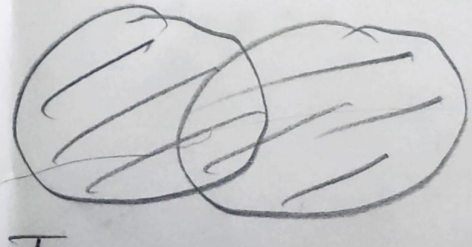
T2

Col 1	Col 2
A	1

duplicate present in both

Syntax:

```
select * FROM Table1
UNION
select * FROM Table2;
```



JOIN VS SET operations

JOIN

- Combine multiple tables based on matching condition.
- Column wise combination
- Data type of two table can be different
- Can generate both distinct & duplicate rows
- The number of column selected may or may not be same from each table
- Combine results horizontally

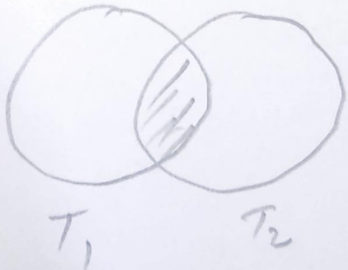
SET operations

- Combination is resulting from two or more SELECT statements
- Row wise comb^n.
- Datatype of corresponding columns from each table should be same.
- Generate distinct rows.
- The number of column(s) selected must be the same from each table
- Combines results vertically.

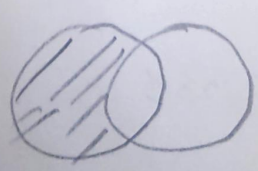
→ INTERSECT

- Emulated

```
Select DISTINCT id from T1
INNER JOIN T2 using(id);
```



→ MINUS



```
Select id FROM T1
    LEFT JOIN T2 using(id)
    WHERE T2.id is NULL;
```

*Example

ABC → company

Dept 1

empid	name	role
1	A	engineer
2	B	Salesman
3	C	manager
4	D	Salesman
5	E	engineer

Dept 2

empid	name	role
3	C	manager
6	F	marketing
7	G	Salesman

-- SET operations

-- List out all the employees in the company

```
Select * from dept 1
UNION
Select * from dept 2;
```

o/p

empid	name	role
1	A	engineer
2	B	
3	C	
4	D	
5	E	
6	F	
7	G	

-- List out all the employees in all departments who work as Salesman

```
Select * from dept 1
UNION
Select * from dept 2;
```

galat

sahi likho

Q- list out all the employees who work for both departments. ?

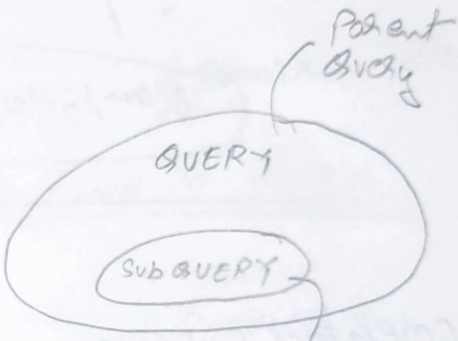
```
select dept1.* from dept1 INNER JOIN dept2 using (empid);
```

-- list out all the employees working in dept1 but not in dept2 ??

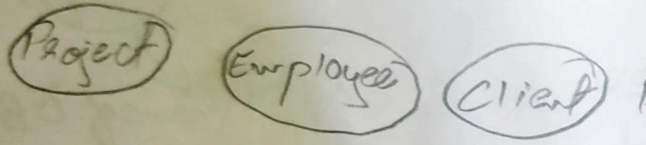
```
select dept1.* from LEFT JOIN dept2 using (empid)
WHERE dept2.empid is null;
```

* SubQueries

- alternative to JOIN
- outer query depends on INNER query



Example:




```
-- SUB queries
-- WHERE clause same table
-- employees with age >30
select * from employee where age in (select age from
  employee where age>30);
```

executed first

```
-- WHERE clause different table
-- empo details working in more than 1 project
select * from employee where id in (
   select empID from project groupby empID having count
                                          (empID)>1
);
```

-- Single value subquery
-- emp details having age > avg(age)

```
select * from employee where age>(select avg(age) from employee);
```

-- FROM Clause - derived tables
-- select max age person whose first name contains 'a'

```
select max(age) from (select * Employee where fname like '%a%') AS temp;
```

temp → name of DB

* Corelated Subquery

outer (inner)

- inner query that refers the outer query.

-- find third oldest employee → wrt type of Q.

```
select * FROM employee e1
WHERE 3 = (SELECT COUNT(e2.age)
FROM Employee e2
WHERE e2.age >= e1.age
);
```

for each e1.age Inner → Completly one time.

ek employee age ke liye inner query pura ek baar chlega

Now,

① e1.age = 32

↳ now inner query.

No of employees jo employee I ke age ke equal ya usse bada = ②

age 32, 44

32
44
22
31
21

② e1.age = 44
↳ inner query.
= (1)

③ e1.age = 22
↳ inner query
= (4)

④ e1.age = 31
↳ inner query. → inner query output
= (3)

fb

WHERE (3 = (3) ko fb)

e1.age = 33 jiska bhi ko output mae aa jaega.

* JOINS V/S SUBQUERIES

JOINS
- Faster
- Joins maximise calculation based on DBMS
- complex, difficult to understand and implement
- choosing optimal join for optimal use case is difficult

SUBQUERIES
- slower
- keeps responsibility of calculation on user
- comparatively easy to understand and implement
- Easy

* SQL views

- customers => id / name / age / address.

view → name / age → ek template.

→ custom view

- A view is a database object that has no values. Its contents are based on the base table. It contains rows and columns similar to real table.
- IN MYSQL, the view is a virtual table created by a query by joining one & more tables. It is operated similarly to the base table but does not contain any data of its own
- The view and table have one main difference that the views are definitions built on top of other tables (& views). If any changes occur in the underlying table, the same changes reflected in view also.

-- VIEW
select * from Employee;

-- creating a view
create VIEW custom_view AS SELECT fname, age from Employee;

-- viewing from VIEW → now we run
select * from custom_view;

o/p:

fname	age
Aman	32
Yogya	44
Rahul	22
Jain	31
PK	21

-- ALTERING the VIEW
ALTER VIEW custom_view AS SELECT fname, lname, age FROM Employee;

-- Droping the VIEW
DROP view IF Exists custom_VIEW;