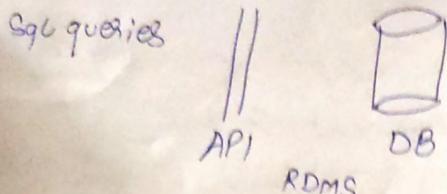
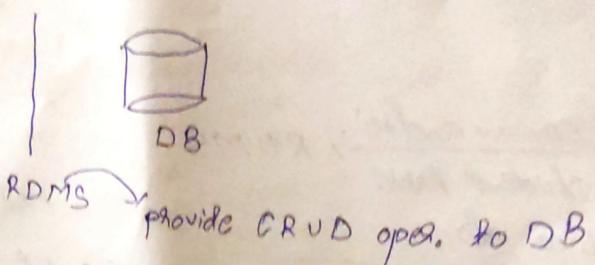


SQL

Structured query language
 RDMS → MySQL, Oracle, Microsoft Access etc.



- CRUD →
 - Create
 - Update
 - Read
 - Delete



RDMS → MySQL
 MySQL is a software which uses SQL lang.
 MS SQL server
 Oracle.
 IBM

SQL
 - Query lang.
 - Way to access data

MySQL
 - MySQL itself a RDMS
 - CRUD done out using SQL.

- MySQL Community Server
- MySQL Workbench

* CREATE DATABASE Temp;

table create kرنge phle

USE Temp;

Now logo ki koi database
use hoga

CREATE TABLE student(

id INT PRIMARY KEY,

name VARCHAR(255)

);

SQL spec. key word

capital letter rahi

baik waqasani; 33

dk joo

- SQL

command ki, RDMS

bua do

→ add one entry to table.

INSERT INTO student VALUES (1, 'ankit');

SELECT * FROM student;

→ fetch data

→ for all data of table

7. SQL Data Types

- CHAR → variable DT

- VARCHAR → variable DT.

CHAR(100) → [RAM] → 100 bytes

VARCHAR(100) → [RAM] → 100 bytes
CHAR(100) → [RAM] → 100 bytes

(RAM) capacity used 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 skta h.

Refer W3 Schools.com / SQL

for clarity.

use now
impl

- > SQL Data types
 - CHAR → ~~variable~~^{fixed} DT
 - VARCHAR → variable DT
 - VARCHAR(1255) → R|A|M| → this uses 3 bytes only
 - CHAR(1255) → R|A|M| → this uses 255 bytes
 - (P.M.) effectively 032 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 saktah.
- # Refer W3Schools.com/SQL
for clarity.
- tuple → row
near

- SIGNED, Unsigned

TINYINT \Rightarrow (-128 to 127)

UNSIGNED TINYINT \Rightarrow (0 to 255)

CREATE TABLE tablename (

col1 INT,

col2 INTUNSIGNED

);

col 2 max 0 - 255
values put > 0
skip

+ SQL Types of Command

1. DDL (data def. lang): defining relation schema

- CREATE : Create table, DB, view

- ALTERTABLE : modification in table structure. e.g. change col. col-T

- DROP : delete table, DB, view

- TRUNCATE : remove all the tuples from the table.

- RENAME : rename DB name, table name etc

2. DML/DQL (data manipulation language):
retrieve data from tables.

- SELECT

3. DML (Data modification lang): used 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. TCM (Transaction control lang): To manage transaction done in the DB

- START TRANSACTION
- COMMIT:
- ROLL BACK:
- SAVE POINT

* Managing DB (DDL)

I. Creation of DB

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

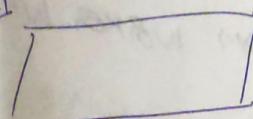
* DQL

- ① DB
- ② Table

- DATA RETRIEVAL LANGUAGE (DRL)
- Syntax: `SELECT <set of col names> FROM <table name>;`
- Order of execution from RIGHT TO LEFT.
-

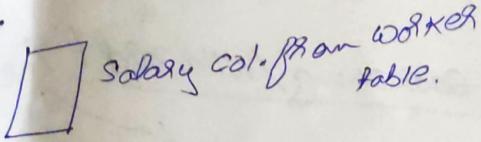
single column

- `SELECT * FROM WORKER;`



WORKER table.

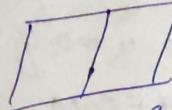
`SELECT SALARY FROM WORKER;`



`SELECT FIRST_NAME, SALARY FROM WORKER;`

is separated

comma is read next word after



First_name & salary col.

• 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 workers WHERE officer-name IN ('Lakshay', 'Ranu')

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

AND/OR/NOT

- AND: WHERE cond1 AND cond2
- OR: WHERE cond1 OR cond2
- NOT: WHERE col-name NOT IN (1, 2, 3, 4);
 \rightarrow SELECT * FROM WORKER WHERE DEPARTMENT NOT IN ('HR');
- IS NULL

- SELECT * FROM customer WHERE prime-status IS NULL;

Pattern searching/Wildcard ('%', '_')

like '%', _
any no. of character

\cup 0-

- SELECT * FROM customer ORDER BY name DESC
WHERE name LIKE '% p %';

\downarrow
all name whose 2nd last
letter is p.

WILDCARDS

Ex -

'% pa %'

\rightarrow 'abc pad b'

'pabc'

'adpa'

'pa -'
 \rightarrow apab

mean pa ke bagal mae
kewal ek chog loga

* Sorting using ORDER BY
- SELECT * from WORKER Order by Salary;
 ↓
 ↓
 Salary → Desc;
 ↓
 ↓
 default asc.

* Distinct values

- Department / distinct
 HR } → Total distinct dep.
 HR HR
 Admin Admin
 Admin Account
 Account

- SELECT ↓ department from WORKER;
DISTINCT

* Data Grouping

- find no. of employee working in diff department.
 HR → ? Kifne employee 2
 Account → ? Kifne employee 2
 Admin → ? ..

Grouping → Aggregation (count)

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

-- GROUP BY
Select department from WORKER group by department;

Trickles
distinct & no
aggregation applied

before and after group by add same logo

→ check group by data
2 HR ka, 2 Admin ka

Select department, COUNT () from worker group by department;

O/p → HR 2 Admin 4 Account 2

internally SQL is doing this from RDBMS on count(K1)

HR → 2 L. Admin → 4 L.

-- 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 * from worker WHERE use note > 1000per

→ FB 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

- filter table based on specified condition.	- filter groups based on specified cond
- WHERE can be used with SELECT, UPDATE & DELETE Key words	- HAVING can be used with SELECT

DDL Key Constraints

① Primary Key.

- Not NULL

- unique

- only one PR

normally

PR → INT data

```
CREATE TABLE customer (
    id INT PRIMARY KEY,
    branch_id INT,
    first_name VARCHAR(50),
    last_name  ||  || ,
    DOB      DATE,
    gender   CHAR(16),
);
```

② Foreign Key.

- refers to PK of other table.

- Each relation can have 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 constraint (

:

CONSTRAINT acc_balance - CK CHECK (balance > 0),

:

);

Create Table account (id int,

id int primary key,);

name varchar (255) UNIQUE,

balance INT,

Constraint acc_balance - CK CHECK (balance > 1000);

);

Insert INTO account (id, name, balance)

values (1, 'A', 10000);

Insert INTO account (id, name, balance)

values (2, 'B', 100);

Select * from account;

• DEFAULT

Create Table account (

:

balance INT NOT NULL Default 0,

:

);

⑤ General discussion

CREATE ASSERTION customer

CHECK (customer card);

↳ Checks if each modification that could potentially violate it.

Ex - number uid appears unique
CREATE ASSERTION acc NO DUPS
CHECK (NOT EXISTS (SELECT * FROM acc
WHERE uid NOT IN (SELECT uid FROM acc)));

CONSTRAINT acc_balance - CK CHECK (balance > 0),

CONSTRAINT acc_balance - CK CHECK (balance > 1000);

```
create table account (
    id int primary key,
    name varchar(100) unique,
    balance int not null default 0
);
```

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

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

```
DR. select * from account;
```

id	name	balance
1	A	0

balance whidhe
with balance
one default value
jaega.

* Alter

* ALTER operations

→ changes schema

→ ADD

- Add new col
- ALTER TABLE table-name ADD new_col-name
datatype, ADD new_col-name -> 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);
```

④ CHANGE COLUMN → column name changes.

- Rename column name,
- ALTER TABLE table-name CHANGE COLUMN old_col-name new_col-name new-col-data-type
- Ex - ALTER TABLE customer CHANGE COLUMN name customer_name varchar(1024);

→ DROP COLUMN

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

⑤ RENAME

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

→ EX-AL

```
create table account (
    id int primary key,
    name varchar(1255) 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 columns*

accn	id	int
	name	varchar
	balance	int
	interest	double

-- CHANGE COLUMN - Rename the col

alter table account change column interest
saving - interest float not null default 0;

-- RENAME THE TABLE

alter table account rename to account_details;

* DATA MANIPULATION LANGUAGE (DML)

→ INSERT

- INSERT INTO table-name (col1, col2, col3)
values (v₁, v₂, v₃), (val1, val2, val3);

→ UPDATE → change value of a tuple

- UPDATE table-name set col1, col2, col3

SET col1=p, col2='abc' WHERE id=1;

id	col1	col2	col3
1	p	a	b c

→ Update multiple rows

a - update student SET standard = Standard + 1;
standard column
type compulsory. If no value +1
no jaisi

⇒ 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 rows will be deleted.

- DELETE CASCADE

(to overcome DELETE constraint of referential
constraints)

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)

;

i) 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

- previously 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 inserted.

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

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

Create database temp;

USE temp;

CREATE TABLE customer
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, 'Shayna Singh', 'Ludhiana H.O', 'M', 'Ludhiana',
141001)

1st way.

insert
1210, 'Rohan', 'Abhoknagar', 'M', 'Jalandhar', 144002);

insert into customer values

('1', 'codehelp', 'delhi', 'M', 'delhi', '110000');

2nd way.

insert into customer (id, cname) 2 details
values (121, 'Bob'); as only
are available

3rd way.

th bakiyo wao default wao jaega
whi th null.

-- UPDATE

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

⑦ Update multiple rows

UPDATE customer SET PinCode = 110000;

X

error

Sohi pincode ko
110000 ka do

MySQL ek poos kuch njo bol raha h
ki kuch ph gadbad h jo ek baad
wao sohi row ko change & del kar
rha h.

✓ 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 Constraint (over child with odds)

- ① INSERT → value can not be inserted in child if
that value is not present in parent
table
- ② DELETE Constraint →
to cascade → ON DELETE CASCADE
- parent survive delete
parent to child relationship
delete to cascading
- ON DELETE SET NULL
→ now parent can be deleted
and all the rows corresponding
entry to child table will
also change to null
now null set to cascading

-- ON DELETE Cascade

Create Table customer,

;

Select * From customer

Insert into customer

values (1, 'Ranu Kumar', '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.o

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

insert statement

Is yeh hi
yoga tb
deletion hi
loga parent
or child se

ON DELETE
CASCADE

Is yeh tb entry
parent or child
dono se delete ho jega

DELETE from customer where id = 1;

-- ON DELETE SET NULL

written at same place as of ON Delete cascade.

Same articles & everything just

ON Delete cascade ke jagh on Delete set null

Agar wai sarkare ke delete karta suparht table
Customer table se delete karta hu to child table me
foreign key me null let hogega

// order details ki baki entries ko kuch nahi hoga

-- REplace

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

-- Replace

REPLACE INTO customers (id, city)
values (11251, 'Colony');

PK data is
necessary.

or baki entries
me null hoga

REPLACE INTO customers (id, name, city)
values (11333, 'codetop', 'Colony');

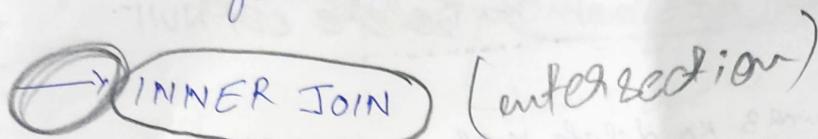
behaves as
insert as

1333 ki koi PK + 10
nhi

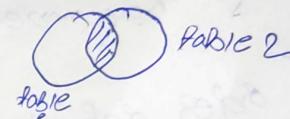
* JOINS

- All RDBMS are relational in nature,
FK → Relations

To fetch data $\xrightarrow{\text{use}}$ JOINS

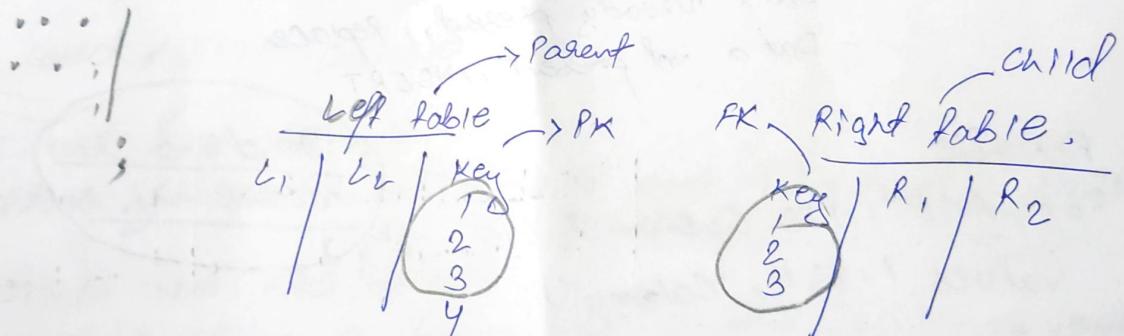


- Returns a resultant table that has matching values from both the tables & all the tables



SELECT column list FROM table1 INNER JOIN table2 ON Condition 1.

INNER JOIN table3 ON Condition 2.

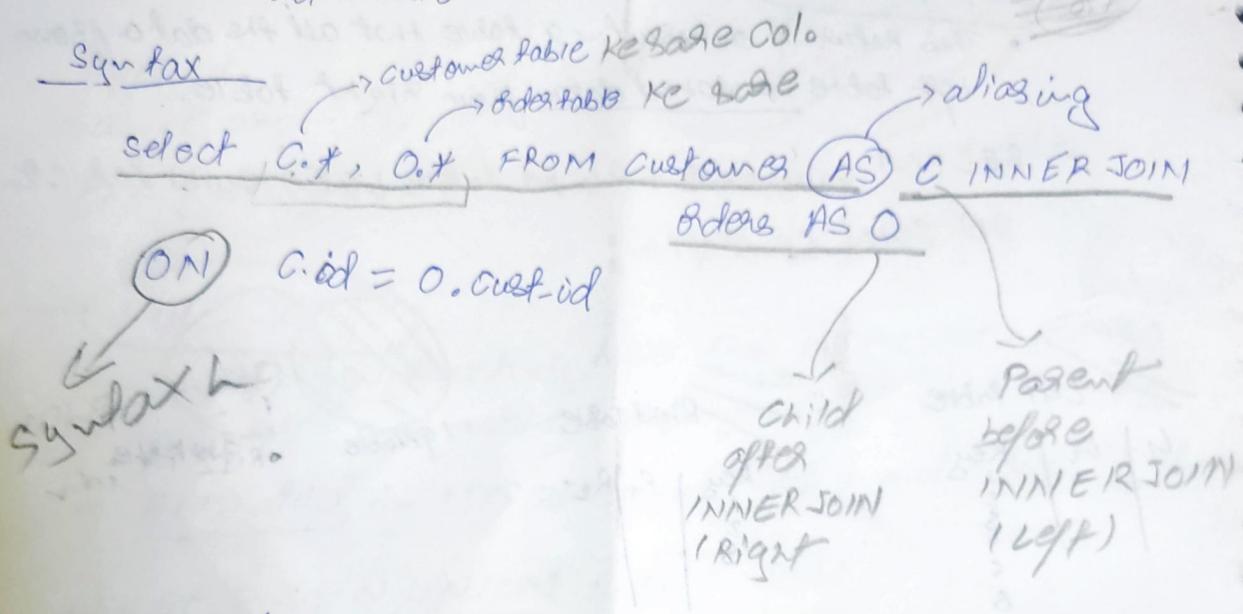


To Left table or Right table match hoga to ye dhikrega koi koi match nahi aata h 1, 2, 3 match hoga lekki se wale rows return hoga

O/P

Key	L1	L2	R1	R2
1				
2				
3				

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



- Alias in MySQL (AS)

- Aliases in MySQL base to give temporary name to a table or 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 columns FROM Table LEFT JOIN Table2
ON JOIN-condition;

new

Left Table		
Key	L ₁	L ₂
1
2
3
4
5
6

new

Right Table	
Key	R ₁ R ₂
1	...
2	...
3	...



Resultant

new

Key	L ₁	L ₂	R ₁	R ₂
1
2
3
4
5	NULL	NULL
6	NULL	NULL

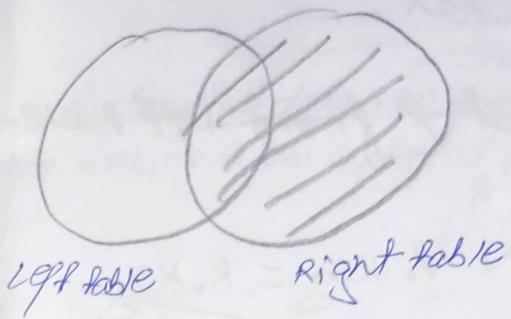
- SELECT C.* , O.* FROM customer AS C LEFT JOIN
orders AS O
ON C.id = O.cust_id;

→ Right regular?

→ Right JOIN

Approx
Same as

LEFT JOIN



→ FULL JOIN

- This represents 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



Left		Right		Result	
Key	Value	Key	Value	R ₁	R ₂
1	1	1	1	1	1
2	2	1	2	2	2
3	3	1	3	3	3
4	4	2	4	4	4
5	5	2	5	5	5
6	6	2	6	6	6
	NULL	3	7	7	7
	NULL	4	8	8	8
	NULL	5	9	9	9
	NULL	6	10	10	10

MySQL FULL JOIN X
Key word of X

Syntax

Select * from leftTable as P LEFT JOIN RightTable
as R

ON P.Key = R.Key

(UNION)

~~mtable R table ka
key colo~~

Select * from leftTable as P RIGHT JOIN Right
Table as R

ON P.Key = R.Key;

Resultant

Key	Left				Right
1	L_1	L_2	R_1	R_2	
2	--	--	--	--	
3	--	--	--	--	
4	--	--			
5	--	--	NULL		
6	--	--			
7			--	--	
8	NULL		--	--	
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.

\Rightarrow rarely used in practical.

T_L
5 rows

T_R
10 rows

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

Ex -

L_1 | L_2
1 | A
2 | B

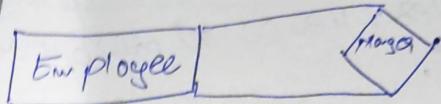
R_1 | R_2
3 | C
4 | D

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

in P using relation.



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

- Emulated using INNER JOIN
- ALIAS 'AS'

~~Set~~ e.g -

select e1.id, e2.id, e2.name
FROM

Employee AS e1

INNER JOIN

Employee AS e2

ON e1.id = e2.id;

... references behaved as
diss tables

Self join no?
confused?

Example

* Project

id	empID	name	startdate	clientID
1	1	A	2021-04-21	3
2	2	B	2021-03-12	2
3	3	C	2021-01-16	5
4	4	D	2021-05-27	2
5	5	E	2021-03-01	4

* Employee

id	fname	lname	AGE	availID	PhoneNo	City
1	Anand	Proto	32	anand@—	898	Delhi
2	Vogya	Nandan	44	vogya@—	222	Palam
3	Rahul	BD	22	rahul@—	444	Kolkata
4	Jatin	neeraj	31	jatin@—	666	Patiala
5	PK	Panday	21	PK@—	555	Jaipur

* CLIENT

id	first-name	last-name	AGE	availID	Phone No	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	Surbant	Aggarwal	23	surbant@—	45454	Hyderabad	5
5	Pratap	Singh	36	P@XYZ.COM	77767	Mumbai	2

2

-- INNER JOIN

②
-- INNER JOIN
-- Enlist all the employees ID's names along with project allocated 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;

O/P:

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

→ employeer
ka ID col.

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

Select c.id, c.email ID, c.phone NO, c.first_name, c.last_name from Employee as e
INNER JOIN

INNER JOIN Client AS C ON C.id = Employee ID WHERE
C.city = 'Jaipur' AND C.city = 'Hubballi'

O/P: 5 | PK @ - LEFT JOIN 1555 (Subrant) Aggarwal
-- Fetch out result

-- Fetch out each project allocated to each employee.
-- select * from Employee as
1 EFT -

Select * from Employee
LEFT JOIN

LEFT JOIN project as p ON e.id = p.emp ID;

id | Fwd asp on e.ed = p.asp ID;

id | Fwd asp on e.ed = p.asp ID;

-- CROSS JOIN

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

Select e.fname, e.lname, p.id, p.name from Employee
CROSS JOIN project as p;

Q) Can we use JOIN w/o using JOIN keyword?

- Yes

Syntax:

Select * FROM leftTable, rightTable WHERE
leftTable.id = RightTable.id;

Q- Previous INNER JOIN 'Q' 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

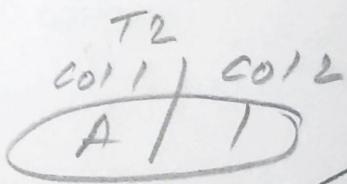
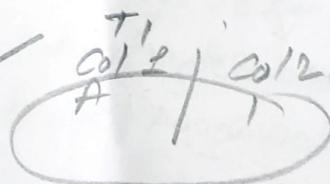
→ UNION

T₁ ∪ T₂

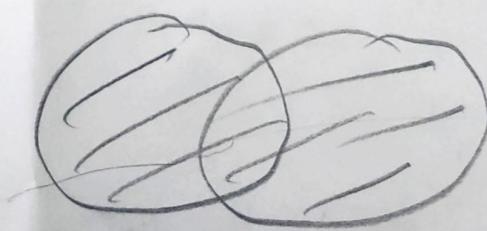
→ same type ke column
rone chahiye or same
no. of column chahiye

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

T₁ or T₂ ke rows
ko union krdiya or
jabhi duplicates hote
hain ko remove krdiya



duplicate
present in both



T₁

T₂

Syntax:

Select * FROM TABLE1

UNION

Select * FROM TABLE2;

JOIN

V/S

- 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 columns selected may or may not be same from each table
- combine results horizontally

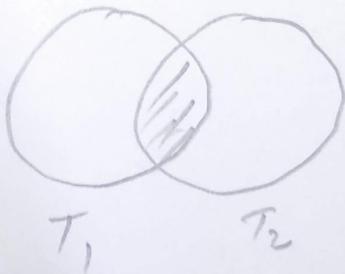
INTERSECT

- Emulated

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

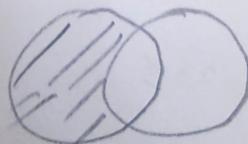
SET operations

- Combination is resulting from two or more **SELECT** statements
- Row wise comb.
- Data type of corresponding columns from each table should be same.
- Generate distinct rows.
- The number of columns selected must be the same from each table
- combines results vertically.



MINUS

Select Ed FROM T₁,
LEFT JOIN T₂ using(id)
WHERE T₂.Ed 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 who work as Salesman in all departments

Select * from Dept 1

UNION

Select * from Dept 2;

7

X g/f/h

sahil khan

Q- list out all the employees who work for both departments.
o select dept1_id from dept1 INNER JOIN dept2 using (empid);

-- list out all the employees working in dept1 but not in dept2
select dept1_id from LEFT JOIN dept2 using (empid)
WHERE dept2.empid is null;

* Subqueries

- alternative to JOIN
- outer query depends on INNER query

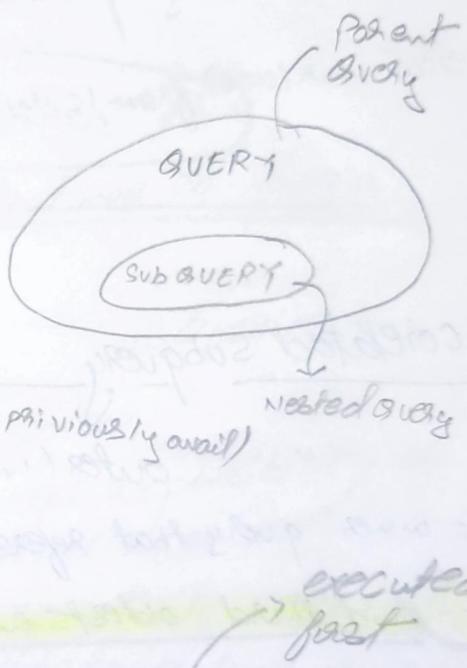
Example:

Project

Employee

Client

DB



-- SUB QUERIES

-- WHERE clause same table

-- employees with age > 30

select * from employee where age in (select age from employee where age > 30);

* WHERE clause different table

o emp details working in more than 1 project

select * from employee where id in ()

select empID from project group by 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);

name of DB

* correlated subquery

outer (inner)

- inner query that refers the outer query.

- Find third oldest employee → w/p 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 → completely one time.

ex employee age ke liye inner query pura ek baar

chlega

Now,

① e1.age = 32

(→ does inner query.

No of employees to employee 2 ke age
ke equal ya use joda = ②

age 32, 44

32
44
22
31
21

$$\textcircled{2} \quad c_1.\text{age} = 44$$

↳ inner query
= \textcircled{1}

$$\textcircled{3} \quad c_1.\text{age} = 22$$

↳ inner query
= \textcircled{4}

$$\textcircled{4} \quad c_1.\text{age} = 31$$

↳ inner query → inner query output
= \textcircled{3}

fb
WHERE $3 = (3)$ \rightarrow fb

$c_1.\text{age} = 33$ jska bhi to output wala aa jaega.

* JOINS

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

VS

SUBQUERIES

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

SQL views

Customer \Rightarrow id | name | age | address.

view \rightarrow name | age \rightarrow ex template.
 \rightarrow custom view

- A view is a database object that has no values. If 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 or 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 (or 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 name, age
FROM Employee;

-- viewing from view

select * from custom-view;

Q/P:

name	age
Anmol	32
Yogya	44
Rahul	22
Jain	31
PK	21

NOW
8 rows

-- ALTERING the view

ALTER VIEW custom-view AS SELECT name, name,
age FROM Employee;

-- Dropping the view

DROP view IF EXISTS custom-view;