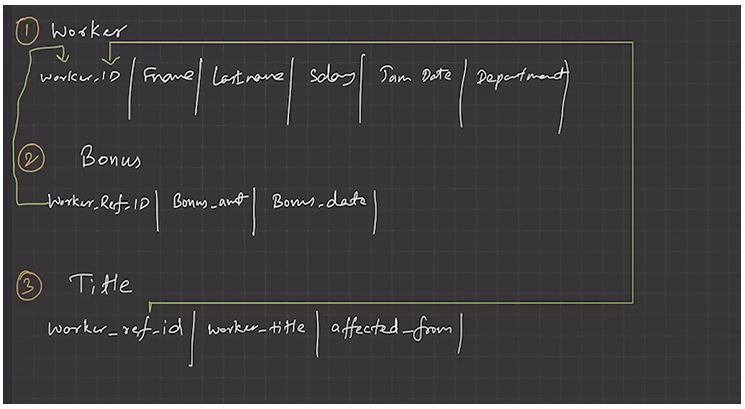
SQL

SQL: Structured Query Language, used to access and manipulate data.

- SQL used CRUD operations to communicate with DB.
  - CREATE - execute INSERT statements to insert new tuple into the relation.
  - READ - Read data already in the relations.
  - UPDATE - Modify already inserted data in the relation.
  - DELETE - Delete specific data point/tuple/row or multiple rows.
- **SQL is not DB, is a query language.**
- What is **RDBMS?** (Relational Database Management System)
  - Software that enable us to implement designed relational model.
  - e.g., MySQL (**Open Source RDBMS)**, MS SQL, Oracle, IBM etc.
  - Table/Relation is the simplest form of data storage object in R-DB.
  - MySQL is open-source RDBMS, and it uses SQL for all CRUD operations.
- **MySQL** used client-server model, where client is CLI or frontend that used services provided by MySQL server.
- **Difference between SQL and MySQL**
  - SQL is Structured Query language used to perform CRUD operations in R-DB, while MySQL is a RDBMS used to store, manage and administrate DB (provided by itself) using SQL.

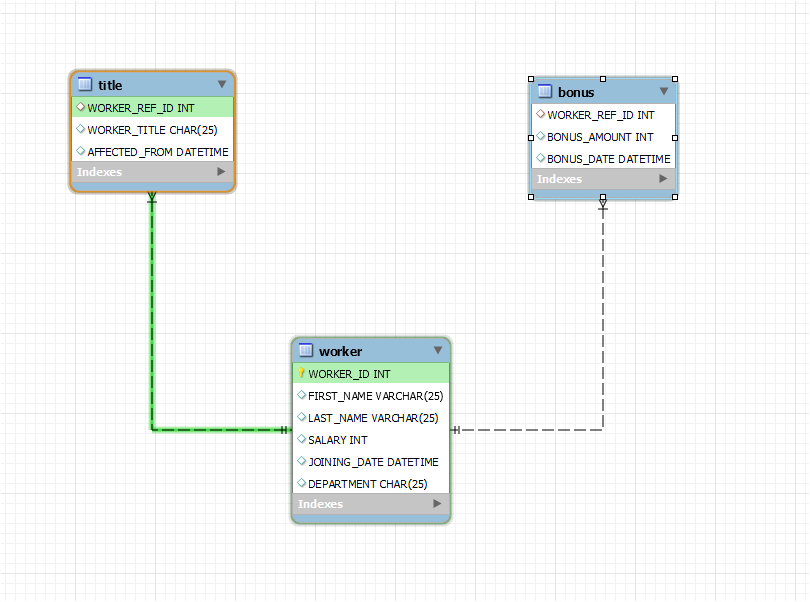
SQL	MySQL
Query Language	MySQL itself a RDMS
Way to access data	CRUD done on it using SQL

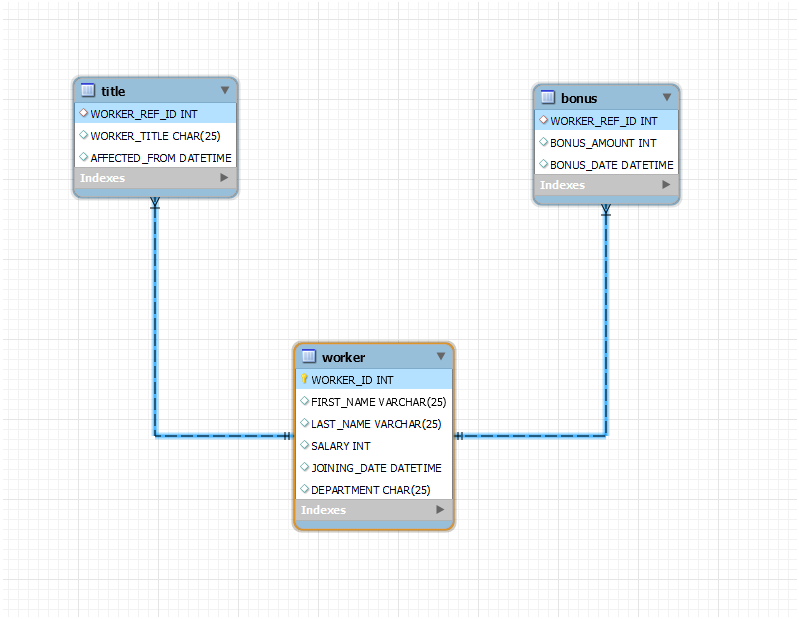


Database:

```
CREATE DATABASE ORG;
SHOW DATABASES;
USE ORG;

CREATE TABLE Worker(
    WORKER_ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    FIRST_NAME VARCHAR(25),
    LAST_NAME VARCHAR(25),
    SALARY INT(15),
    JOINING_DATE DATETIME,
    DEPARTMENT CHAR(25)
);


INSERT INTO Worker VALUES (001, 'Monika', 'Arora', 100000, '14-02-20 09.00.00', 'HR');
INSERT INTO Worker VALUES (002, 'Niharika', 'Verma', 80000, '14-06-11 09.00.00', 'Admin');
INSERT INTO Worker VALUES (003, 'Vishal', 'Singhal', 300000, '14-02-20 09.00.00', 'HR');
INSERT INTO Worker VALUES (004, 'Amitabh', 'Singh', 500000, '14-02-20 09.00.00', 'Admin');
INSERT INTO Worker VALUES (005, 'Vivek', 'Bhati', 500000, '14-06-11 09.00.00', 'Admin');
INSERT INTO Worker VALUES (006, 'Vipul', 'Diwan', 200000, '14-06-11 09.00.00', 'Account');
INSERT INTO Worker VALUES (007, 'Satish', 'Kumar', 75000, '14-01-20 09.00.00', 'Account');
INSERT INTO Worker VALUES (008, 'Geetika', 'Chauhan', 90000, '14-04-11 09.00.00', 'Admin');

SELECT * FROM Worker;


CREATE TABLE Bonus(
    WORKER_REF_ID INT,
    BONUS_AMOUNT INT(10),
    BONUS_DATE DATETIME,
    FOREIGN KEY (WORKER_REF_ID)
        REFERENCES Worker(WORKER_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

INSERT INTO Bonus
    (WORKER_REF_ID, BONUS_AMOUNT, BONUS_DATE) VALUES
        (001, 5000, '16-02-20'),
        (002, 3000, '16-06-11'),
        (003, 4000, '16-02-20'),
        (001, 4500, '16-02-20'),
        (002, 3500, '16-06-11');


CREATE TABLE Title(
    WORKER_REF_ID INT,
    WORKER_TITLE CHAR(25),
    AFFECTED_FROM DATETIME,
    FOREIGN KEY (WORKER_REF_ID)
        REFERENCES Worker(WORKER_ID)
```

```
        ON DELETE CASCADE
        ON UPDATE CASCADE
);


INSERT INTO Title
    (WORKER_REF_ID, WORKER_TITLE, AFFECTED_FROM) VALUES
        (001, 'Manager', '2016-02-20 00:00:00'),
        (002, 'Executive', '2016-06-11 00:00:00'),
        (008, 'Executive', '2016-06-11 00:00:00'),
        (005, 'Manager', '2016-06-11 00:00:00'),
        (004, 'Asst. Manager', '2016-06-11 00:00:00'),
        (007, 'Executive', '2016-06-11 00:00:00'),
        (006, 'Lead', '2016-06-11 00:00:00'),
        (003, 'Lead', '2016-06-11 00:00:00');
```





Like Operator

```
Select all customers that starts with the letter "a":

SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';

----

Return all customers from a city that starts with 'L' followed by one wildcard character, then 'nd' and then two wildcard cha

SELECT * FROM Customers
WHERE city LIKE 'L_nd__';

----

Return all customers from a city that contains the letter 'L':

SELECT * FROM Customers
WHERE city LIKE '%L%';

---

Return all customers that ends with 'a':

SELECT * FROM Customers
WHERE CustomerName LIKE '%a';

For More details -> https://www.w3schools.com/sql/sql_like.asp
```

WHERE vs HAVING

- Both have same function of filtering the row base on certain conditions.
- WHERE clause is used to filter the rows from the table based on specified condition
- HAVING clause is used to filter the rows from the groups based on the specified condition.
- HAVING is used after GROUP BY while WHERE is used before GROUP BY clause.
- If you are using HAVING, GROUP BY is necessary.
- WHERE can be used with SELECT, UPDATE & DELETE keywords while GROUP BY used with SELECT

```
CREATE DATABASE ORG;
SHOW DATABASES;
USE ORG;

CREATE TABLE Worker(
    WORKER_ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    FIRST_NAME VARCHAR(25),
    LAST_NAME VARCHAR(25),
    SALARY INT(15),
    JOINING_DATE DATETIME,
    DEPARTMENT CHAR(25)
);


INSERT INTO Worker VALUES (001, 'Monika', 'Arora', 100000, '14-02-20 09.00.00', 'HR');
INSERT INTO Worker VALUES (002, 'Niharika', 'Verma', 80000, '14-06-11 09.00.00', 'Admin');
INSERT INTO Worker VALUES (003, 'Vishal', 'Singhal', 300000, '14-02-20 09.00.00', 'HR');
INSERT INTO Worker VALUES (004, 'Amitabh', 'Singh', 500000, '14-02-20 09.00.00', 'Admin');
INSERT INTO Worker VALUES (005, 'Vivek', 'Bhati', 500000, '14-06-11 09.00.00', 'Admin');
INSERT INTO Worker VALUES (006, 'Vipul', 'Diwan', 200000, '14-06-11 09.00.00', 'Account');
INSERT INTO Worker VALUES (007, 'Satish', 'Kumar', 75000, '14-01-20 09.00.00', 'Account');
INSERT INTO Worker VALUES (008, 'Geetika', 'Chauhan', 90000, '14-04-11 09.00.00', 'Admin');

SELECT * FROM Worker;


CREATE TABLE Bonus(
```

```
    WORKER_REF_ID INT,
    BONUS_AMOUNT INT(10),
    BONUS_DATE DATETIME,
    FOREIGN KEY (WORKER_REF_ID)
        REFERENCES Worker(WORKER_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

INSERT INTO Bonus
    (WORKER_REF_ID, BONUS_AMOUNT, BONUS_DATE) VALUES
        (001, 5000, '16-02-20'),
        (002, 3000, '16-06-11'),
        (003, 4000, '16-02-20'),
        (001, 4500, '16-02-20'),
        (002, 3500, '16-06-11');


CREATE TABLE Title(
    WORKER_REF_ID INT,
    WORKER_TITLE CHAR(25),
    AFFECTED_FROM DATETIME,
    FOREIGN KEY (WORKER_REF_ID)
        REFERENCES Worker(WORKER_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);


INSERT INTO Title
    (WORKER_REF_ID, WORKER_TITLE, AFFECTED_FROM) VALUES
        (001, 'Manager', '2016-02-20 00:00:00'),
        (002, 'Executive', '2016-06-11 00:00:00'),
        (008, 'Executive', '2016-06-11 00:00:00'),
        (005, 'Manager', '2016-06-11 00:00:00'),
        (004, 'Asst. Manager', '2016-06-11 00:00:00'),
        (007, 'Executive', '2016-06-11 00:00:00'),
        (006, 'Lead', '2016-06-11 00:00:00'),
        (003, 'Lead', '2016-06-11 00:00:00');

SELECT NOW(); -- To get current server time

SELECT * FROM Worker WHERE SALARY>100000;

-- SALARY (80000,300000)
SELECT * FROM Worker WHERE SALARY BETWEEN 80000 AND 300000 ORDER BY SALARY;

-- REDUCE OR STATEMENTS
-- HR, ADMIN
SELECT * FROM Worker WHERE DEPARTMENT='HR' OR DEPARTMENT='ADMIN';


-- BETTER WAY: IN
SELECT * FROM Worker WHERE DEPARTMENT IN('HR','ADMIN');


-- NOT
SELECT * FROM Worker WHERE DEPARTMENT NOT IN ('HR','ADMIN');


-- Pattern Searching
--  1. % -> any number of character
--  2. _ => only one character

-- WILDCARD
SELECT * FROM Worker WHERE FIRST_NAME LIKE '%i%';


-- Sorting using ORDER BY
SELECT * FROM Worker ORDER BY SALARY;

-- DISTINCT
```

```
SELECT DISTINCT DEPARTMENT FROM Worker;

-- GROUP BY
SELECT DEPARTMENT,COUNT(*) FROM Worker GROUP BY DEPARTMENT;

-- Find average salary per Department
SELECT DEPARTMENT,AVG(SALARY) FROM Worker GROUP BY DEPARTMENT;

-- MIN
SELECT DEPARTMENT,MIN(SALARY) FROM Worker GROUP BY DEPARTMENT;

-- MAX
SELECT DEPARTMENT,MAX(SALARY) FROM Worker GROUP BY DEPARTMENT;


-- HAVING
SELECT DEPARTMENT,COUNT(SALARY) FROM Worker GROUP BY DEPARTMENT HAVING COUNT(DEPARTMENT)> 2;
```

ALTER

```
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;

-- CHANGE COLUMN - RENAME THE COLUMN
alter table account
CHANGE /COLUMN interest saving_interest FLOAT NOT NULL DEFAULT 0;

-- DROP COLUMN
alter table account
DROP COLUMN saving_interest;

-- RENAME THE TABLE
alter table account
RENAME TO account_details;
```

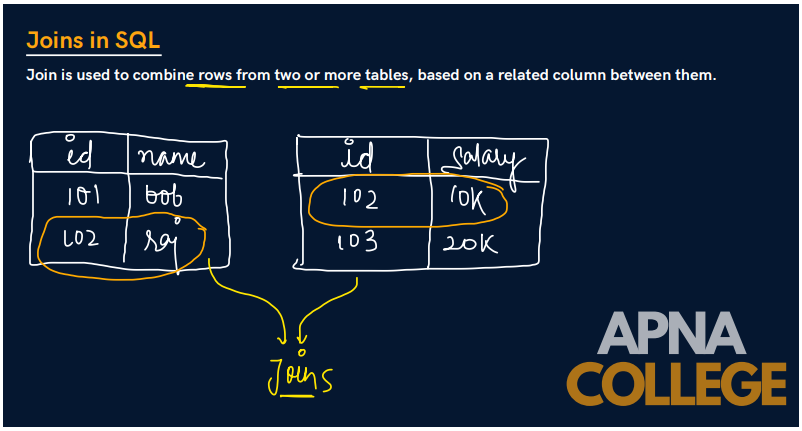
```
Replace Function

1. Data already present, replace
2. Data not present, insert

-- REPLACE
REPLACE INTO Customer (id, City)
VALUES(1251,'Colony');

REPLACE INTO Customer (id, cname, City)
VALUES(1333, 'codehelp', 'Colony');


REPLACE INTO Customer SET id = 1300, Name = 'Mac', City = 'Utah';
```

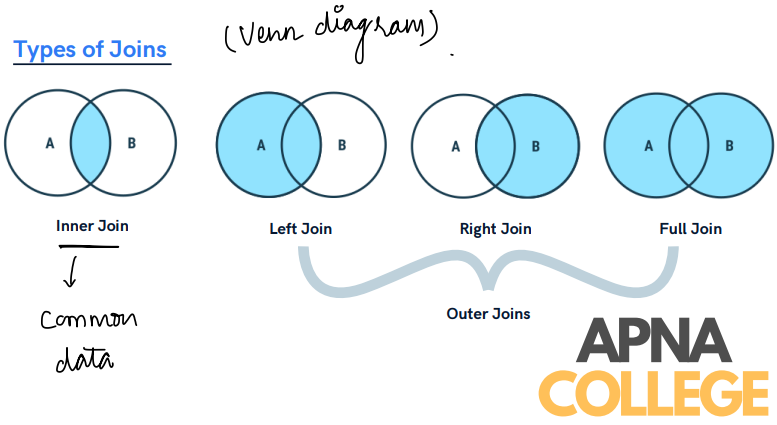
```
REPLACE INTO Customer(Name, City)
SELECT Name, City
FROM Customer WHERE id = 500;
```

Replace vs Update

- If row is not present, replace will add a new row whole update will do nothing.

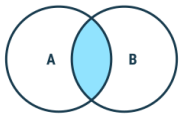
Joins





Inner Join

Returns records that have matching values in both tables



Syntax

```
SELECT column(s)
FROM tableA
INNER JOIN tableB
ON tableA.col_name = tableB.col_name;
```

APNA COLLEGE

Inner Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

```
SELECT *
FROM student
INNER JOIN course
ON student.student_id = course.student_id;
```

alias
↳ alternate name

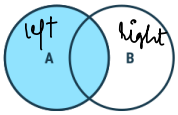
Result

student_id	name	course
102	bob	english
103	casey	science

APNA COLLEGE

Left Join

Returns all records from the left table, and the matched records from the right table



Syntax

```
SELECT column(s)
FROM tableA
LEFT JOIN tableB
ON tableA.col_name = tableB.col_name;
```

APNA COLLEGE

Left Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

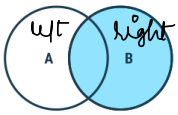
Result

student_id	name	course
101	adam	*null*
102	bob	english
103	casey	science

```
SELECT *
FROM student as s
LEFT JOIN course as c
ON s.student_id = c.student_id;
```

Right Join

Returns all records from the right table, and the matched records from the left table



Syntax

```
SELECT column(s)
FROM tableA
RIGHT JOIN tableB
ON tableA.col_name = tableB.col_name;
```

Right Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

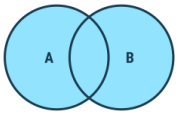
Result

student_id	course	name
102	english	bob
105	math	*null*
103	science	casey
107	computer science	*null*

```
SELECT *
FROM student as s
RIGHT JOIN course as c
ON s.student_id = c.student_id;
```

Full Join

Returns all records when there is a match in either left or right table



Syntax in MySQL

```
SELECT * FROM student as a
LEFT JOIN course as b
ON a.id = b.id
UNION
SELECT * FROM student as a
RIGHT JOIN course as b
ON a.id = b.id;
```

LEFT JOIN
UNION
RIGHT JOIN

APNA COLLEGE

Full Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

Result

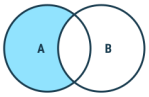
student_id	name	course
101	adam	*null*
102	bob	english
103	casey	science
105	*null*	math
107	*null*	computer science

APNA COLLEGE

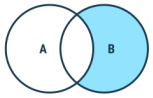
Think & Ans

Full exclusion Join → Left Exclusion
UNION
Right Exclusion

Qs: Write SQL commands to display the right exclusive join :



Left Exclusive Join



Right Exclusive Join

```
SELECT *
FROM student as a
RIGHT JOIN course as b
ON a.id=b.id
WHERE a.id IS NULL;
```

```
SELECT *
FROM student as a
LEFT JOIN course as b
ON a.id = b.id
WHERE b.id IS NULL;
```

APNA COLLEGE

Self Join

It is a regular join but the table is joined with itself.

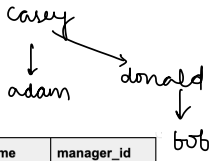
Syntax

```
SELECT column(s)
FROM table as a
JOIN table as b
ON a.col_name = b.col_name;
```

APNA COLLEGE

Self Join

Example

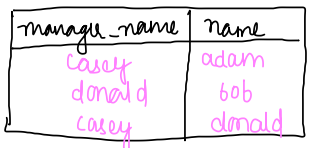


Employee

id	name	manager_id
101	adam	103
102	bob	104
103	casey	*null*
104	donald	103

```
SELECT a.name as manager_name, b.name
FROM employee as a
JOIN employee as b
ON a.id = b.manager_id;
```

Result



APNA COLLEGE

Union

It is used to combine the result-set of two or more SELECT statements.
Gives UNIQUE records.

To use it :

- **every SELECT should have same no. of columns**
- **columns must have similar data types**
- **columns in every SELECT should be in same order**

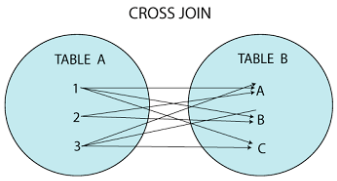
Syntax

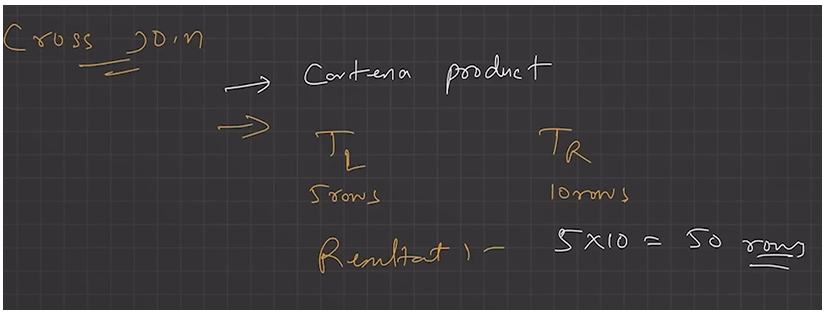
```
SELECT column(s) FROM tableA
UNION
SELECT column(s) FROM tableB
```

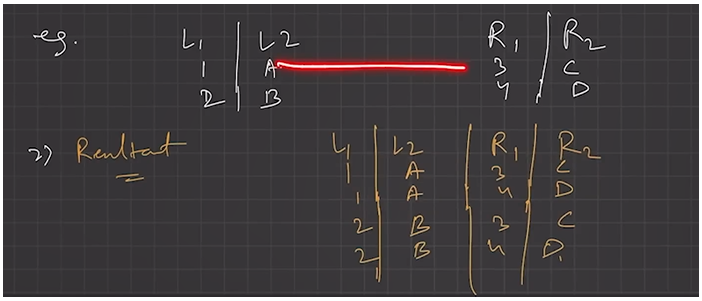
APNA COLLEGE

CROSS JOIN

- This returns all the cartesian products of the data present in both tables. Hence, all possible variations are reflected in the output.
- Used rarely in practical purpose.
- Table-1 has 10 rows and table-2 has 5, then resultant would have 50 rows.
- SELECT column-lists FROM table1 CROSS JOIN table2;







JOINS CODE

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

id	fname	lname	Age	emailID	PhoneNo	City
1	Aman	Proto	32	aman@gmail.com	898	Delhi
2	Yagya	Narayan	44	yagya@gmail.com	222	Palam
3	Rahul	BD	22	rahul@gmail.com	444	Kolkata
4	Jatin	Hermit	31	jatin@gmail.com	666	Raipur
5	PK	Pandey	21	pk@gmail.com	555	Jaipur
NULL	NULL	NULL	NULL	NULL	NULL	NULL

id	first_name	last_name	age	emailID	PhoneNo	City	empID
1	Mac	Rogers	47	mac@hotmail.com	333	Kolkata	3
2	Max	Poirier	27	max@gmail.com	222	Kolkata	3
3	Peter	Jain	24	peter@abc.com	111	Delhi	1
4	Sushant	Aggarwal	23	sushant@yahoo.com	45454	Hyderabad	5
5	Pratap	Singh	36	p@xyz.com	77767	Mumbai	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Database

```
CREATE DATABASE COMPANY;
USE COMPANY;

CREATE TABLE PROJECT(
    ID INT NOT NULL PRIMARY KEY,
    EMPID INT,
    NAME VARCHAR(25),
    STARTDATE DATE,
    CLIENTID INT
);

INSERT INTO PROJECT VALUES (1,1,'A','2021-04-21',3);
INSERT INTO PROJECT VALUES (2,2,'B','2021-03-12',1);
INSERT INTO PROJECT VALUES (3,3,'C','2021-01-16',5);
INSERT INTO PROJECT VALUES (4,3,'D','2021-04-27',2);
INSERT INTO PROJECT VALUES (5,5,'E','2021-05-01',4);

SELECT * FROM PROJECT;

CREATE TABLE EMPLOYEE(
    ID INT NOT NULL PRIMARY KEY,
    FNAME VARCHAR(25),
    LNAME VARCHAR(25),
    AGE INT NOT NULL,
    EMAILID VARCHAR(50),
    PHONENO VARCHAR(15),
    CITY VARCHAR(25)
);

INSERT INTO EMPLOYEE VALUES (1,'Aman','Proto',32,'aman@gmail.com',898,'Delhi');
INSERT INTO EMPLOYEE VALUES (2,'Yagya','Narayan',44,'yagya@gmail.com',222,'Palam');
INSERT INTO EMPLOYEE VALUES (3,'Rahul','BD',22,'rahul@gmail.com',444,'Kolkata');
INSERT INTO EMPLOYEE VALUES (4,'Jatin','Hermit',31,'jatin@gmail.com',666,'Raipur');
INSERT INTO EMPLOYEE VALUES (5,'PK','Pandey',21,'pk@gmail.com',555,'Jaipur');


SELECT * FROM EMPLOYEE;

CREATE TABLE CLIENT(
    ID INT NOT NULL PRIMARY KEY,
    FNAME VARCHAR(25),
    LNAME VARCHAR(25),
    AGE INT NOT NULL,
    EMAILID VARCHAR(50),
    PHONENO VARCHAR(15),
    CITY VARCHAR(25),
    EMPID INT
);

INSERT INTO CLIENT VALUES (1,'Mac','Rogers',47,'mac@hotmail.com',333,'Kolkata',3);
INSERT INTO CLIENT VALUES (2,'Max','Poirier',27,'max@gmail.com',222,'Kolkata',3);
INSERT INTO CLIENT VALUES (3,'Peter','Jain',24,'peter@abc.com',111,'Delhi',1);
INSERT INTO CLIENT VALUES (4,'Sushant','Aggarwal',23,'sushant@yahoo.com',45454,'Hyderabad',5);
INSERT INTO CLIENT VALUES (5,'Pratap','Singh',36,'p@xyz.com',77767,'Mumbai',2);

SELECT * FROM CLIENT;
```

```
CREATE DATABASE COMPANY;
USE COMPANY;

CREATE TABLE PROJECT(
    ID INT NOT NULL PRIMARY KEY,
    EMPID INT,
    NAME VARCHAR(25),
    STARTDATE DATE,
    CLIENTID INT
);

INSERT INTO PROJECT VALUES (1,1,'A','2021-04-21',3);
```

```
INSERT INTO PROJECT VALUES (2,2,'B','2021-03-12',1);
INSERT INTO PROJECT VALUES (3,3,'C','2021-01-16',5);
INSERT INTO PROJECT VALUES (4,3,'D','2021-04-27',2);
INSERT INTO PROJECT VALUES (5,5,'E','2021-05-01',4);

SELECT * FROM PROJECT;

CREATE TABLE EMPLOYEE(
    ID INT NOT NULL PRIMARY KEY,
    FNAME VARCHAR(25),
    LNAME VARCHAR(25),
    AGE INT NOT NULL,
    EMAILID VARCHAR(50),
    PHONENO VARCHAR(15),
    CITY VARCHAR(25)
);

INSERT INTO EMPLOYEE VALUES (1,'Aman','Proto',32,'aman@gmail.com',898,'Delhi');
INSERT INTO EMPLOYEE VALUES (2,'Yagya','Narayan',44,'yagya@gmail.com',222,'Palam');
INSERT INTO EMPLOYEE VALUES (3,'Rahul','BD',22,'rahul@gmail.com',444,'Kolkata');
INSERT INTO EMPLOYEE VALUES (4,'Jatin','Hermit',31,'jatin@gmail.com',666,'Raipur');
INSERT INTO EMPLOYEE VALUES (5,'PK','Pandey',21,'pk@gmail.com',555,'Jaipur');


SELECT * FROM EMPLOYEE;

CREATE TABLE CLIENT(
    ID INT NOT NULL PRIMARY KEY,
    FNAME VARCHAR(25),
    LNAME VARCHAR(25),
    AGE INT NOT NULL,
    EMAILID VARCHAR(50),
    PHONENO VARCHAR(15),
    CITY VARCHAR(25),
    EMPID INT
);

INSERT INTO CLIENT VALUES (1,'Mac','Rogers',47,'mac@hotmail.com',333,'Kolkata',3);
INSERT INTO CLIENT VALUES (2,'Max','Poirier',27,'max@gmail.com',222,'Kolkata',3);
INSERT INTO CLIENT VALUES (3,'Peter','Jain',24,'peter@abc.com',111,'Delhi',1);
INSERT INTO CLIENT VALUES (4,'Sushant','Aggarwal',23,'sushant@yahoo.com',45454,'Hyderabad',5);
INSERT INTO CLIENT VALUES (5,'Pratap','Singh',36,'p@xyz.com',77767,'Mumbai',2);

SELECT * FROM CLIENT;

-- INNER JOIN
-- Enlist all the employee ID's, names along with the Project allocated to them.
SELECT EMPLOYEE.ID, EMPLOYEE.FNAME, EMPLOYEE.LNAME, PROJECT.ID, PROJECT.NAME
FROM EMPLOYEE INNER JOIN PROJECT ON EMPLOYEE.ID=PROJECT.EMPID;

-- Fetch out all the employee ID's and their contact details who have been working
-- from Jaipur with the clients name working in Hyderabad
SELECT EMPLOYEE.ID, EMPLOYEE.EMAILID, EMPLOYEE.PHONENO, CLIENT.FNAME AS CLIENT_FNAME, CLIENT.LNAME AS CLIENT_LNAME
FROM EMPLOYEE INNER JOIN CLIENT ON EMPLOYEE.ID=CLIENT.EMPID
WHERE EMPLOYEE.CITY='Jaipur' AND CLIENT.CITY='Hyderabad';

-- LEFT JOIN
-- Fetch out each project allocated to each employee
SELECT P.ID, P.NAME, E.FNAME, E.LNAME, E.EMAILID FROM EMPLOYEE AS E
LEFT JOIN PROJECT AS P ON E.ID=P.EMPID;


-- RIGHT JOIN
-- List out all the projects along with the employee's name and their respective allocated email ID.
SELECT * FROM EMPLOYEE AS E
RIGHT JOIN PROJECT AS P ON E.ID=P.EMPID;



-- CROSS JOIN
-- List out all the combinations possible for the employee's name and project that can exist.
```

```
SELECT E.FNAME, E.LNAME, P.ID, P.NAME FROM EMPLOYEE AS E
CROSS JOIN PROJECT AS P;
```

Can we use join without using JOIN Keyword?

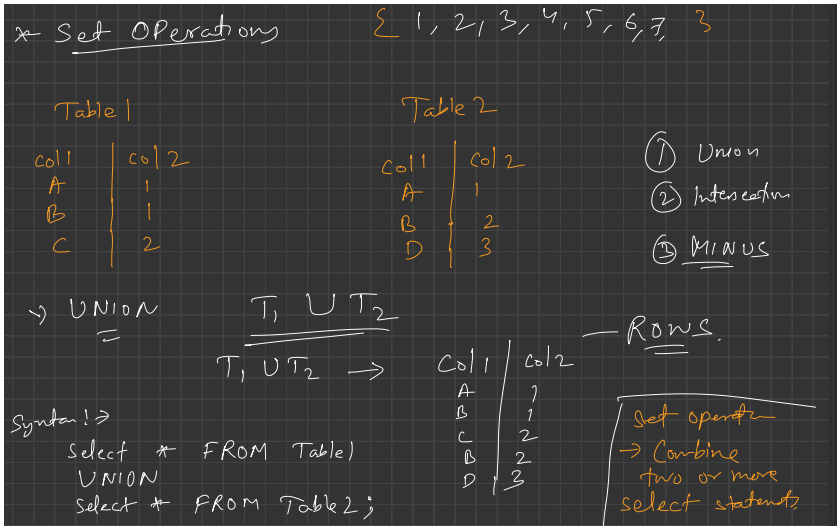
Yes

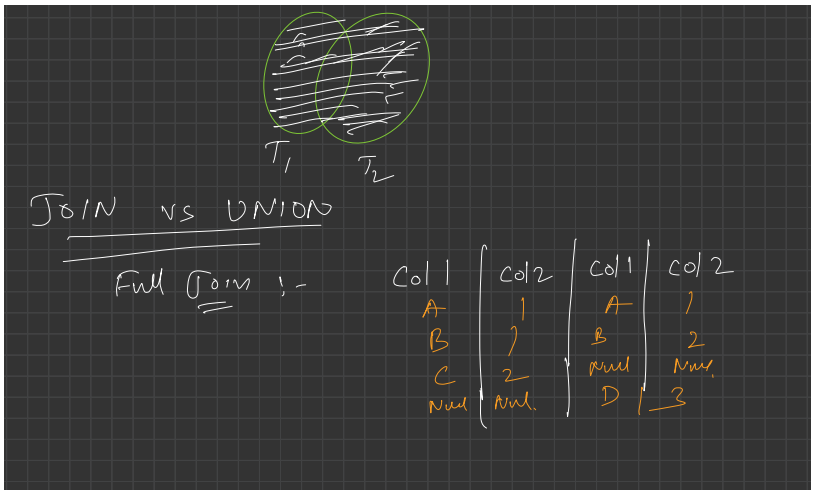
Syntax

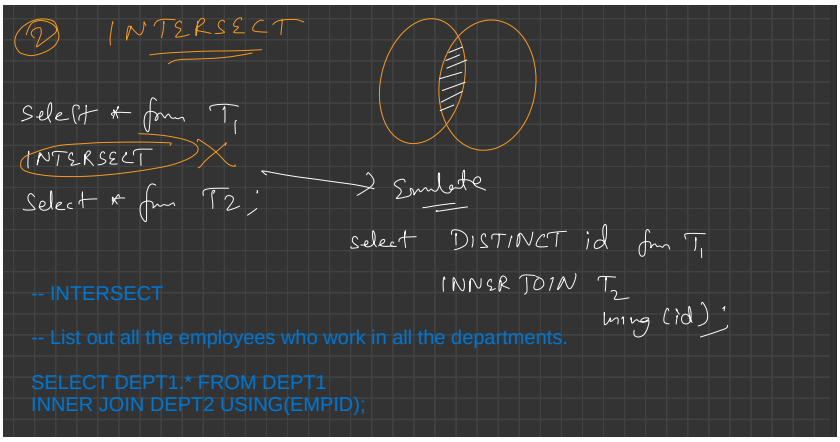
```
SELECT * FROM LEFTTABLE, RIGHTTABLE WHERE
LEFTTABLE.ID=RIGHTTABLE.ID;
```

JOIN	SET Operation
Combines multiple tables based on matching condition.	Combination is resulting set from two or more SELECT statements
Column wise combination	Row wise combination.
Data types of two tables can be different.	Datatypes of corresponding columns from each table should be the same.
Can generate both distinct or duplicate rows.	Generate distinct rows.
The number of column(s) selected may or may not be the same from each table.	The number of column(s) selected must be the same from each table.
Combines results horizontally.	Combines results vertically.

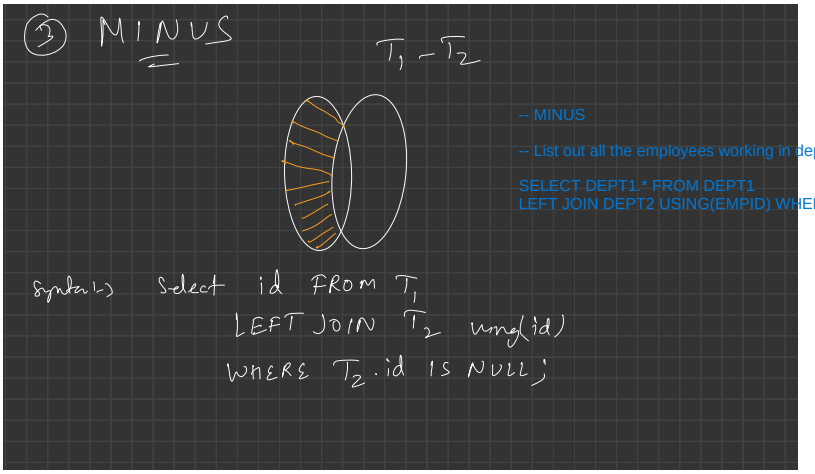
SET OPERATION








```
-- INTERSECT
-- List out all the employees who work in all the departments.
SELECT DEPT1.* FROM DEPT1
INNER JOIN DEPT2 USING(EMPID);
```

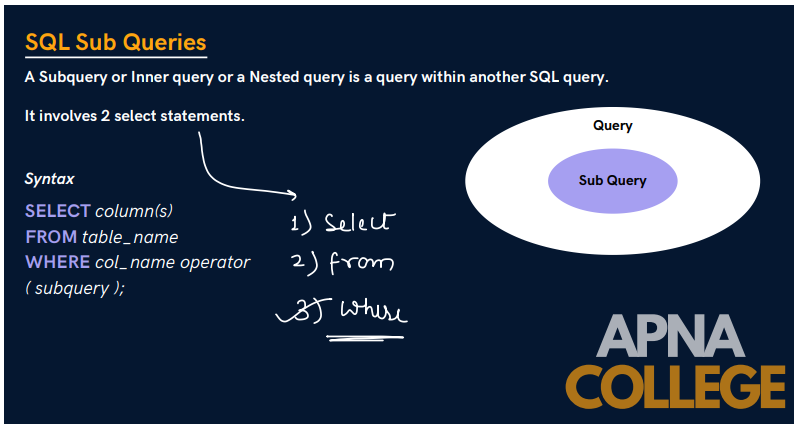



```
-- MINUS
-- List out all the employees working in dept1 but not in dept2
SELECT DEPT1.* FROM DEPT1
LEFT JOIN DEPT2 USING(EMPID) WHERE DEPT2.EMPID IS NULL;
```

```
CREATE DATABASE DEPARTMENT;
USE DEPARTMENT;

CREATE TABLE DEPT1(
    EMPID INT NOT NULL PRIMARY KEY,
    NAME VARCHAR(25),
    ROLE VARCHAR(50)
);
```

```
INSERT INTO DEPT1 VALUES (1,'A','Engineer');
INSERT INTO DEPT1 VALUES (2,'B','Salesman');
INSERT INTO DEPT1 VALUES (3,'C','Manager');
INSERT INTO DEPT1 VALUES (4,'D','Salesman');
INSERT INTO DEPT1 VALUES (5,'E','Engineer');

SELECT * FROM DEPT1;

CREATE TABLE DEPT2(
    EMPID INT NOT NULL PRIMARY KEY,
    NAME VARCHAR(25),
    ROLE VARCHAR(50)
);

INSERT INTO DEPT2 VALUES (3,'C','Manager');
INSERT INTO DEPT2 VALUES (6,'F','Marketing');
INSERT INTO DEPT2 VALUES (7,'G','Salesman');

SELECT * FROM DEPT2;


-- SET OPERATION

-- UNION

-- List out all the employees in the company
SELECT * FROM DEPT1
UNION
SELECT * FROM DEPT2;

-- List out all the employees in all departments who work as salesman
SELECT * FROM DEPT1 WHERE ROLE='Salesman'
UNION
SELECT * FROM DEPT2 WHERE ROLE='Salesman';


-- INTERSECT
-- List out all the employees who work in all the departments.
SELECT DEPT1.* FROM DEPT1
INNER JOIN DEPT2 USING(EMPID);


-- MINUS
-- List out all the employees working in dept1 but not in dept2
SELECT DEPT1.* FROM DEPT1
LEFT JOIN DEPT2 USING(EMPID) WHERE DEPT2.EMPID IS NULL;
```

SUB QUERIES



```
-- 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
-- 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 has 'a'
SELECT MAX(AGE) FROM (SELECT * FROM EMPLOYEE WHERE FNAME LIKE '%a%') AS temp;
```

Corelated Sub Queries

Outer(Inner)

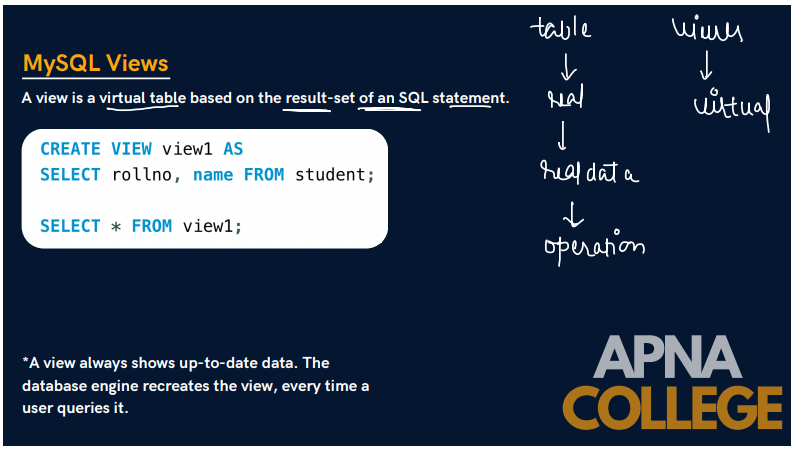
— Inner query that refers the outer query.

```
-- Corelated subquery
-- find 3rd oldest employee
SELECT *
FROM EMPLOYEE E1
WHERE 3 = (
    SELECT COUNT(E2.AGE)
    FROM EMPLOYEE E2
    WHERE E2.AGE >=E1.AGE
);
```

Difference between JOINS And SUBQUERIES

JOINS	SUBQUERIES
Faster	Slower
Joins maximize calculation burden on DBMS	Keeps responsibility of calculation on user
Complex, difficult to understand and implement	Comparatively easy to understand and implement
Choosing optimal join for optimal use case is difficult.	Easy

SQL VIEW




```
-- VIEW
SELECT * FROM EMPLOYEE;

-- creating a view
CREATE VIEW CUSTOM_VIEW AS SELECT FNAME,AGE FROM EMPLOYEE;

-- VIEWING FROM VIEW
SELECT * FROM CUSTOM_VIEW;


-- Altering the view
ALTER VIEW CUSTOM_VIEW AS SELECT FNAME, LNAME, AGE FROM EMPLOYEE;

-- DROPPING THE VIEW
DROP VIEW IF EXISTS CUSTOM_VIEW;
```