SQL (Structured Query Language) is used to perform operations on the records stored in the database, such as updating records, inserting records, deleting records, creating and modifying database tables, views, etc.

SQL is not a database system, but it is a query language.

Suppose you want to perform the queries of SQL language on the stored data in the database. You are required to install any database management system in your systems, for example, **Oracle, MySQL, MongoDB, PostgreSQL, SQL Server, DB2, etc.**

What is SQL?

This database language is mainly designed for maintaining the data in relational database management systems. It is a special tool used by data professionals for handling structured data (data which is stored in the form of tables).

You can easily create and manipulate the database, access and modify the table rows and columns, etc.

SQL for storing the data in the back-end.

SQL is widely used in data science and analytics

It also helps them to describe the structured data.

It allows SQL users to create, drop, and manipulate the database and its tables

Some SQL Commands

The SQL commands help in creating and managing the database. The most common SQL commands which are highly used are mentioned below:

1. CREATE command
2. UPDATE command

3. DELETE command
4. SELECT command
5. DROP command
6. INSERT command

.

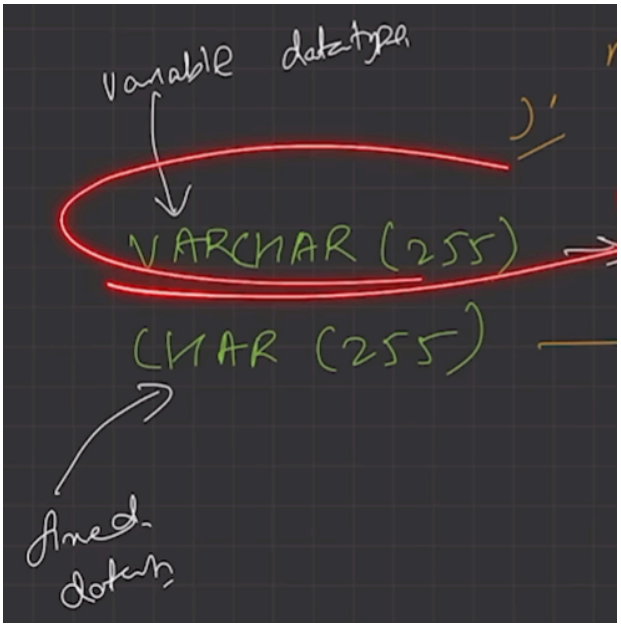
What is MySQL and why is it used?

MySQL is a relational database management system

MySQL stores data in tables made up of rows and columns. Users can define, manipulate, control, and query data using Structured Query Language

What is the difference between SQL and MySQL?

SQL is a query programming language for managing RDBMS.
MySQL is an RDBMS (Relational Database Management System) that employs SQL.
So, the major difference between the two is that **MySQL is software**, but **SQL is a database language.**
Oracle, MySQL, Microsoft SQL Server, PostgreSQL, SQLite, and MariaDB.
MySQL - Open Source Relational Database Management System



1. **SQL**: Structured Query Language, used to access and manipulate data.
2. SQL used **CRUD** operations to communicate with DB.
    1. **CREATE** - execute INSERT statements to insert new tuple into the relation.
    2. **READ** - Read data already in the relations.
    3. **UPDATE** - Modify already inserted data in the relation.
    4. **DELETE** - Delete specific data point/tuple/row or multiple rows.
3. **SQL is not DB, is a query language.**
4. What is **RDBMS**? (Relational Database Management System)
    1. Software that enable us to implement designed relational model.
    2. e.g., MySQL, MS SQL, Oracle, IBM etc.
    3. Table/Relation is the simplest form of data storage object in R-DB.
    4. **MySQL** is open-source RDBMS, and it uses SQL for all CRUD operations
5. **MySQL** used client-server model, where client is CLI or frontend that used services provided by MySQL server.
6. **Difference between SQL and MySQL**
    1. 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.

https://www.digitalocean.com/community/tutorials/sql-data-types

CHAR
VARCHAR
VARCHAR(MAX)
INT
FLOAT
DATE
TIME
DECIMAL
NUMERIC
DATETIME
YEAR
TEXT

SQL DATA TYPES (Ref: https://www.w3schools.com/sql/sql_datatypes.asp)

1. In SQL DB, data is stored in the form of tables.
2. Data can be of different types, like INT, CHAR etc.

DATATYPE	Description
CHAR	string(0-255), string with size = (0, 255], e.g., CHAR(251)
VARCHAR	string(0-255)
TINYTEXT	String(0-255)
TEXT	string(0-65535)
BLOB	string(0-65535)
MEDIUMTEXT	string(0-16777215)
MEDIUMBLOB	string(0-16777215)
LONGTEXT	string(0-4294967295)
LONGBLOB	string(0-4294967295)
TINYINT	integer(-128 to 127)
SMALLINT	integer(-32768 to 32767)

DATATYPE	Description
DECIMAL	Double stored as string
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS
ENUM	One of the preset values
SET	One or many of the preset values
BOOLEAN	0/1
BIT	e.g., BIT(n), n upto 64, store values in bits.

3. Size: TINY < SMALL < MEDIUM < INT < BIGINT.
4. **Variable length Data types** e.g., VARCHAR, are better to use as they occupy space equal to the actual data size
5. Values can also be unsigned e.g., INT UNSIGNED.

Difference between char and varchar?
VARCHAR is variable length, while CHAR is fixed length

Difference between CHAR and VARCHAR datatypes:

In CHAR, If the length of the string is less than set or fixed-length then it is padded with extra memory space. In VARCHAR, If the length of the string is less than the set or fixed-length then it will store as it is without padded with extra memory spaces

6. **Types of SQL commands:**
    1. **DDL** (data definition language): defining relation schema.
        1. **CREATE**: create table, DB, view.
        2. **ALTER TABLE**: modification in table structure. e.g, change column datatype or add/remove columns.
        3. **DROP**: delete table, DB, view.
        4. **TRUNCATE**: remove all the tuples from the table.
        5. **RENAME**: rename DB name, table name, column name etc.
    2. **DRL/DQL** (data retrieval language / data query language): retrieve data from the tables.
        1. **SELECT**
    3. **DML** (data modification language): use to perform modifications in the DB
        1. **INSERT**: insert data into a relation
        2. **UPDATE**: update relation data.
        3. **DELETE**: delete row(s) from the relation.
    4. **DCL** (Data Control language): grant or revoke authorities from user.
        1. **GRANT**: access privileges to the DB
        2. **REVOKE**: revoke user access privileges.
    5. **TCL** (Transaction control language): to manage transactions done in the DB
        1. **START TRANSACTION**: begin a transaction
        2. **COMMIT**: apply all the changes and end transaction
        3. **ROLLBACK**: discard changes and end transaction
        4. **SAVEPOINT**: checkout within the group of transactions in which to rollback.

MANAGING DB (DDL)

1. Creation of DB
    1. **CREATE DATABASE IF NOT EXISTS db-name;**
    2. **USE** db-name; //need to execute to choose on which DB CREATE TABLE etc commands will be executed. //make switching between DBs possible.
    3. **DROP** DATABASE IF EXISTS db-name; //dropping database.
    4. **SHOW** DATABASES; //list all the DBs in the server.
    5. **SHOW** TABLES; //list tables in the selected DB.

```sql
CREATE DATABASE temp;

USE temp;

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

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

SELECT * FROM student;

DROP DATABASE IF EXISTS temp1;
```

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

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

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

DATA RETRIEVAL LANGUAGE (DRL)

1. Syntax: SELECT <set of column names> FROM <table_name>;
2. Order of execution from RIGHT to LEFT.
3. Q. Can we use SELECT keyword without using FROM clause?
    1. Yes, using DUAL Tables.
    2. Dual tables are dummy tables created by MySQL, help users to do certain obvious actions without referring to user defined tables.
    3. e.g., SELECT 55 + 11;
       SELECT now();
       SELECT ucase(); etc.
4. **WHERE**
    1. Reduce rows based on given conditions.
    2. E.g., SELECT * FROM customer WHERE age > 18;
5. **BETWEEN**
    1. SELECT * FROM customer WHERE age between 0 **AND** 100;
    2. In the above e.g., 0 and 100 are inclusive.
6. **IN**
    1. Reduces **OR** conditions;
    2. e.g., SELECT * FROM officers WHERE officer_name IN ('Lakshay', 'Maharana Pratap', 'Deepika');
7. **AND/OR/NOT**
    1. AND: WHERE cond1 AND cond2
    2. OR: WHERE cond1 OR cond2
    3. NOT: WHERE col_name NOT IN (1,2,3,4);
8. **IS NULL**
    1. e.g., SELECT * FROM customer WHERE prime_status is NULL;

Can we use select without using FROM ?

Yes, DUAL Tables → select 4+5;

4. **WHERE**
    1. Reduce rows based on given conditions.
    2. E.g., SELECT * FROM customer WHERE age > 18;
5. **BETWEEN**
    1. SELECT * FROM customer WHERE age between 0 AND 100;
    2. In the above e.g., 0 and 100 are inclusive.
6. **IN**
    1. Reduces **OR** conditions;
    2. e.g., SELECT * FROM officers WHERE officer_name IN ('Lakshay', 'Maharana Pratap', 'Deepika');
7. **AND/OR/NOT**
    1. AND: WHERE cond1 AND cond2
    2. OR: WHERE cond1 OR cond2
    3. NOT: WHERE col_name NOT IN (1,2,3,4);
8. **IS NULL**
    1. e.g., SELECT * FROM customer WHERE prime_status is NULL;
9. **Pattern Searching / Wildcard ('%', '_')**
    1. '%', any number of character from 0 to n. Similar to '*' asterisk in regex.
    2. '_', only one character.
    3. SELECT * FROM customer WHERE name **LIKE** '%p_';

```sql
SELECT * FROM Worker;

SELECT * FROM Worker WHERE SALARY > 100000;

SELECT * FROM Worker WHERE DEPARTMENT = 'HR';

-- salary [80000, 300000]
SELECT * FROM Worker WHERE SALARY BETWEEN 80000 AND 300000;

-- reduce OR statements
-- HR, ADMIN
SELECT * FROM Worker WHERE DEPARTMENT = 'HR' OR DEPARTMENT = 'Admin' OR DEPARTMENT = 'Admin';
```

```sql
-- better way: IN
SELECT * FROM Worker WHERE DEPARTMENT IN ('HR', 'Admin', 'Account');

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

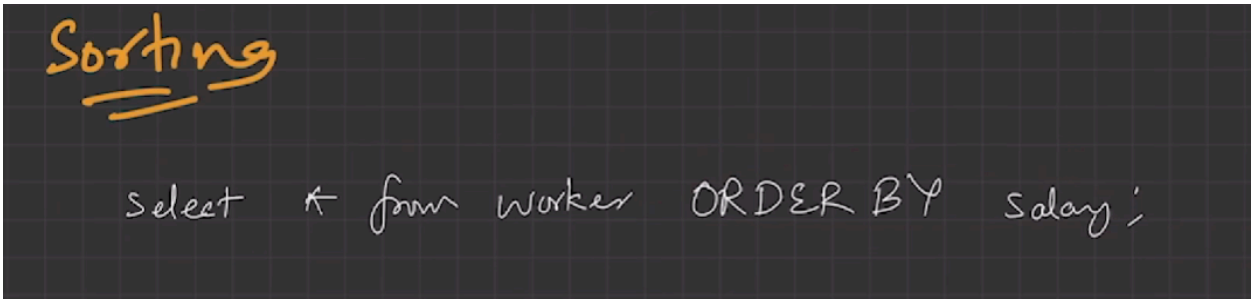
9. **Pattern Searching / Wildcard ('%', '_')**
    1. '%', any number of character from 0 to n. Similar to '*' asterisk in regex.
    2. '_', only one character.
    3. SELECT * FROM customer WHERE name **LIKE** '%p_';

First name contains letter ‘i’

```
-- WILDCARD
select * from worker where first_name LIKE '%i%';
```

First name with second character as ‘i’

```
-- WILDCARD
select * from worker where first_name LIKE '_i%';
```

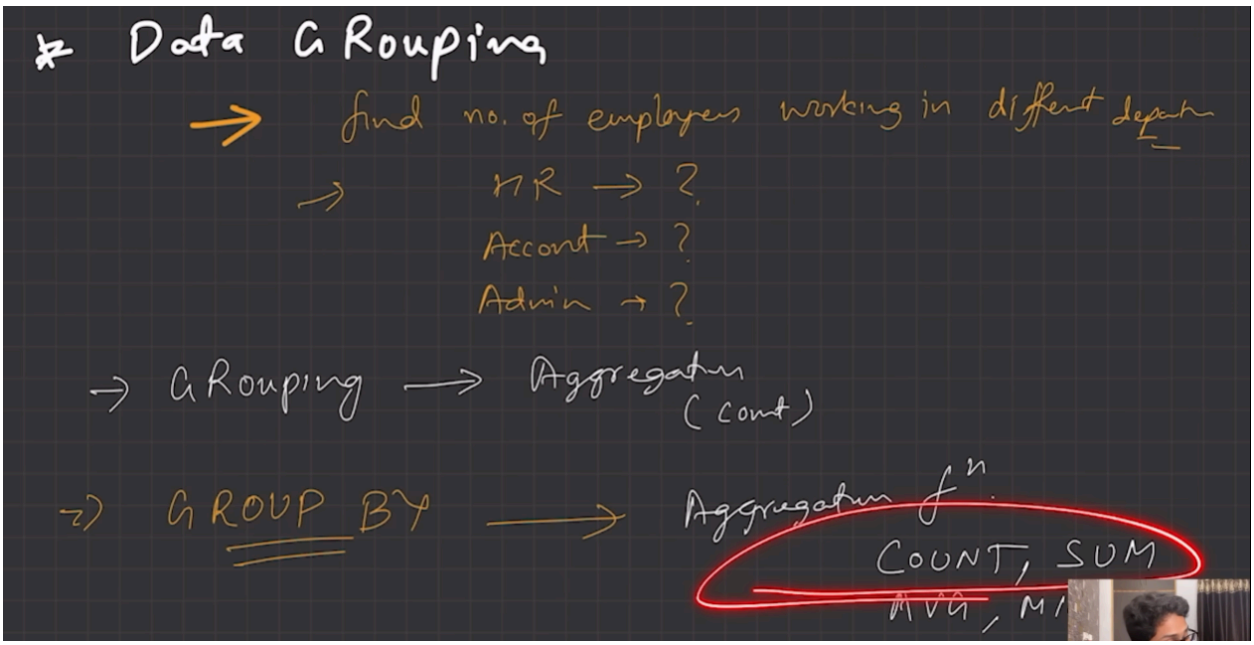



```
-- sorting using ORDER BY
select * from worker order by salary DESC;
```

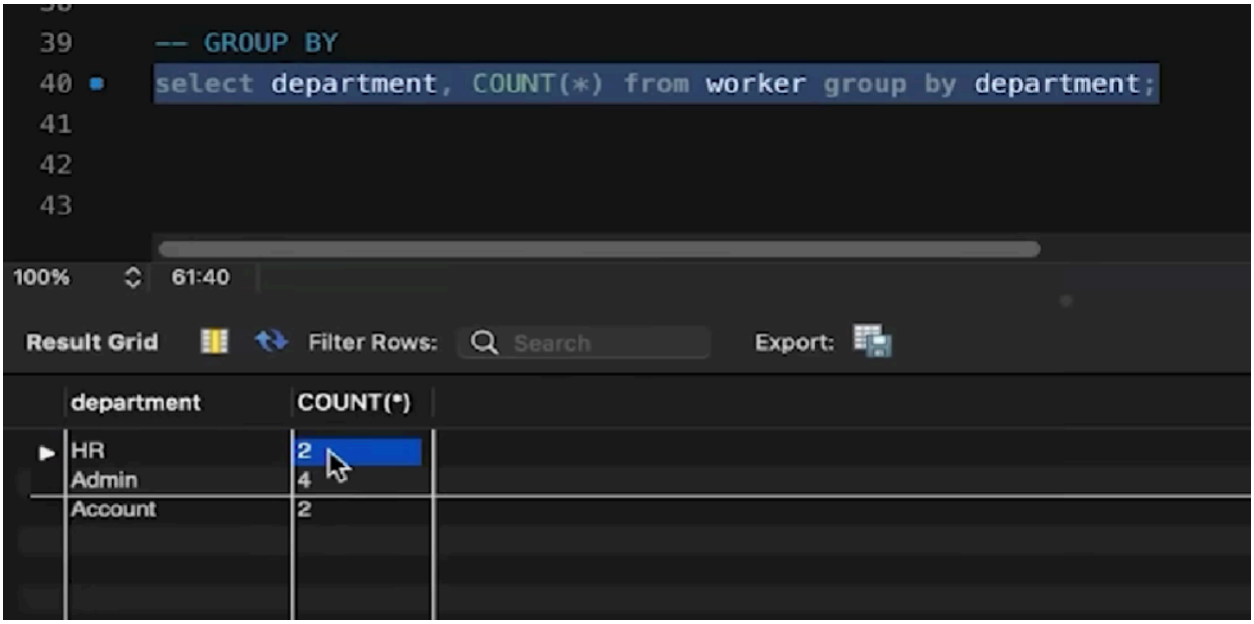
DISTINCT

```
-- DISTINCT
select DISTINCT department from worker;
```

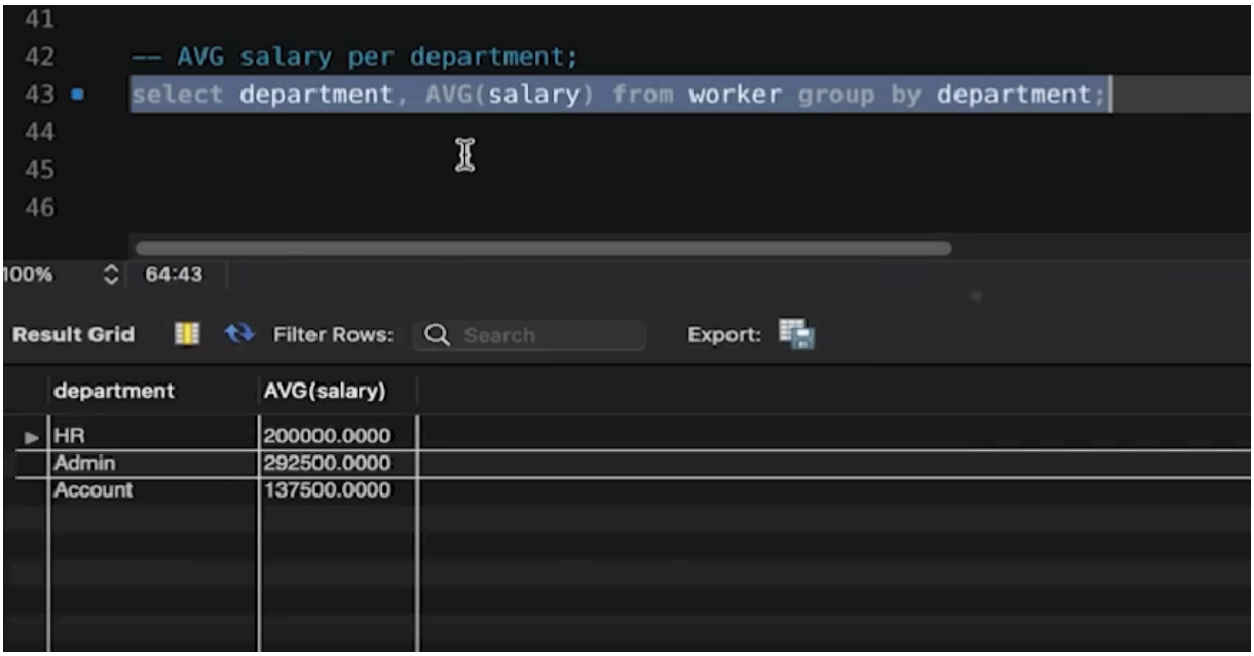
10. **ORDER BY**
    1. Sorting the data retrieved using **WHERE** clause.
    2. ORDER BY <column-name> DESC;
    3. DESC = Descending and ASC = Ascending
    4. e.g., SELECT * FROM customer ORDER BY name DESC;
11. **GROUP BY**
    1. GROUP BY Clause is used to collect data from multiple records and group the result by one or more column. It is generally used in a SELECT statement.
    2. Groups into category based on column given.
    3. SELECT c1, c2, c3 FROM sample_table WHERE cond GROUP BY c1, c2, c3.
    4. All the column names mentioned after SELECT statement shall be repeated in GROUP BY, in order to successfully execute the query.
    5. Used with aggregation functions to perform various actions.
        1. COUNT()
        2. SUM()
        3. AVG()
        4. MIN()
        5. MAX()




```sql
-- GROUP BY
select department from worker group by department;
```



Average salary for each department



Minimum salary for each department

```
45      -- MIN
46  •   select department, MIN(salary) from worker group by department;
47
```

1:34

Result Grid | Filter Rows: Search | Export:

department	MIN(salary)
HR	100000
Admin	80000
Account	75000

Maximum salary for each department

```
47
48      -- MAX
49  •   select department, MAX(salary) from worker group by department;
50
51
```

64:49

Result Grid | Filter Rows: Search | Export:

department	MAX(salary)
HR	300000
Admin	500000
Account	200000

Total salary for each department

```
51      -- SUM
52  •   select department, SUM(salary) from worker group by department;
53
54
```

100% 18:39

Result Grid | Filter Rows: Search | Export:

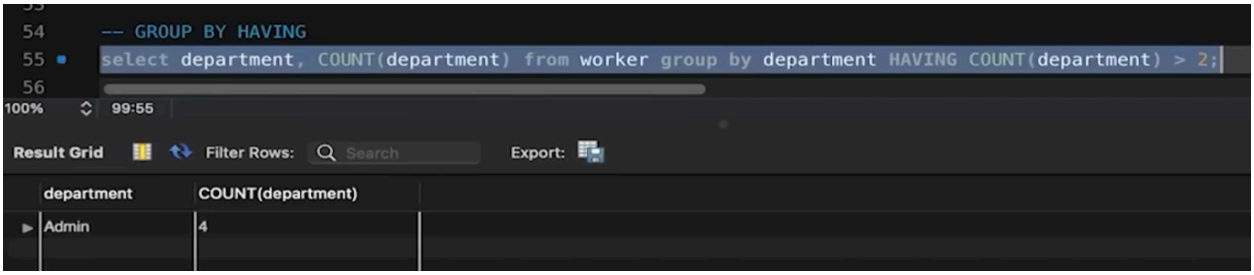
department	SUM(salary)
HR	400000
Admin	1170000
Account	275000

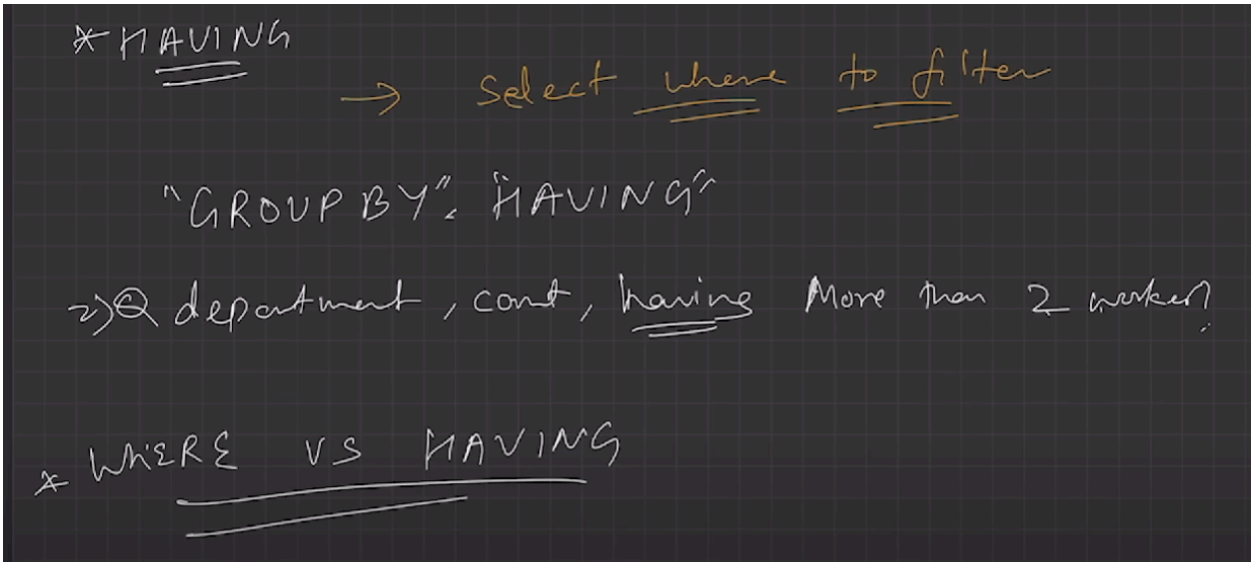
12. **DISTINCT**
    1. Find distinct values in the table.
    2. SELECT DISTINCT(col_name) FROM table_name;
    3. GROUP BY can also be used for the same
        1. "Select col_name from table GROUP BY col_name;" same output as above DISTINCT query.

13. **GROUP BY HAVING**
    1. Out of the categories made by GROUP BY, we would like to know only particular thing (cond).
    2. Similar to WHERE.
    3. Select COUNT(cust_id), country from customer GROUP BY country HAVING COUNT(cust_id) > 50;
    4. WHERE vs HAVING
        1. Both have same function of filtering the row base on certain conditions.
        2. WHERE clause is used to filter the rows from the table based on specified condition
        3. HAVING clause is used to filter the rows from the groups based on the specified condition.
        4. HAVING is used after GROUP BY while WHERE is used before GROUP BY clause.
        5. If you are using HAVING, GROUP BY is necessary.
        6. WHERE can be used with SELECT, UPDATE & DELETE keywords while GROUP BY used with SELECT.

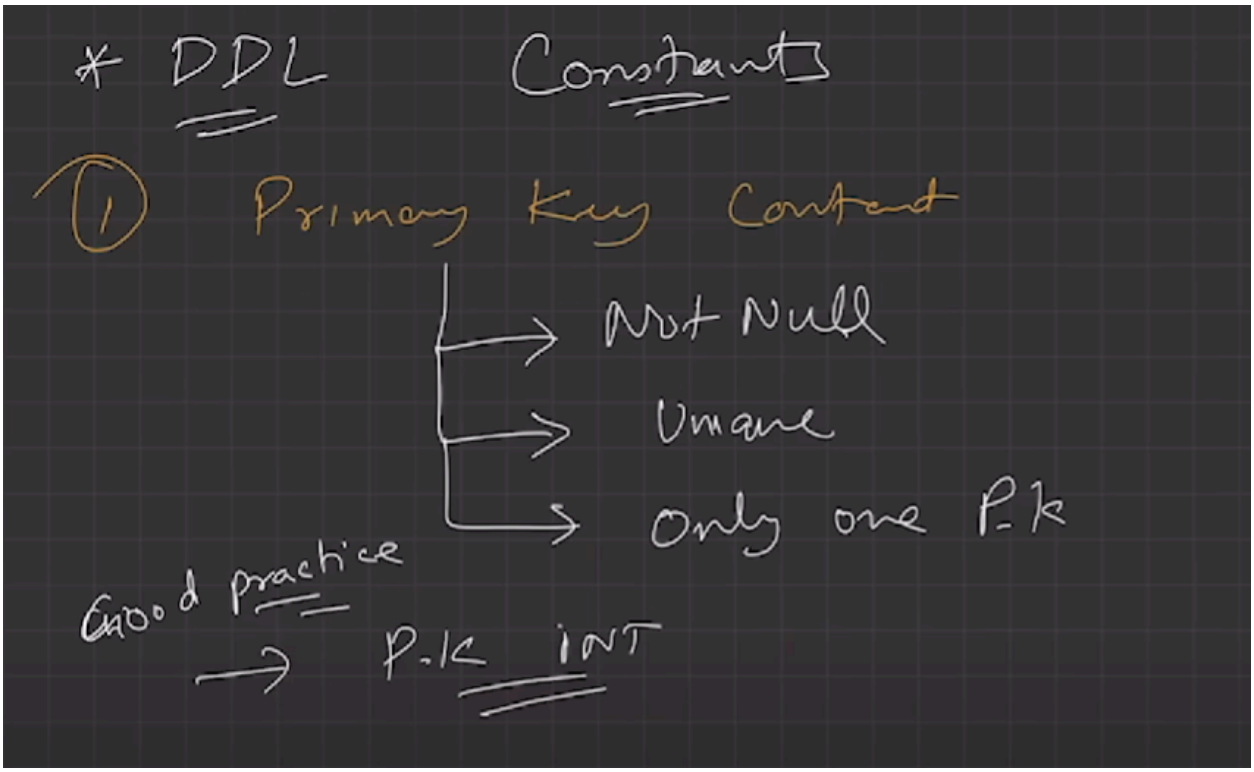
Filtering with group by → Having

Department name whose count is greater than 2



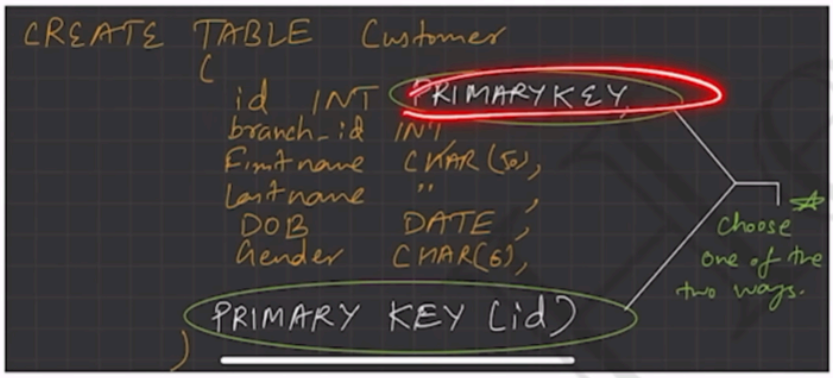


Having used to filter rows from groups
Where used to filter rows from table
Where comes before group by
Having comes after group by
For having , Group by is necessary



ONSTRAINTS (DDL)

Primary Key



1. PK is not null, unique and only one per table.

2. **Foreign Key**
    1. FK refers to PK of other table.
    2. Each relation can having any number of FK.
    3. CREATE TABLE ORDER (
       id INT PRIMARY KEY,
       delivery_date DATE,
       order_placed_date DATE,
       FOREIGN KEY (cust_id) REFERENCES customer(id)
       );

```sql
CREATE TABLE Order_details (
Order_id integer PRIMARY KEY,
delivery_date DATE,
cust_id INT,
FOREIGN KEY(cust_id) references Customer(id)
);
```

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

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

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

SELECT * FROM Worker;
```

```sql
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 DELETE CASCADE

The **ON DELETE CASCADE** clause in MySQL is used to automatically remove the matching records from the child table when we delete the rows from the parent table. It is a kind of referential action related to the foreign key.

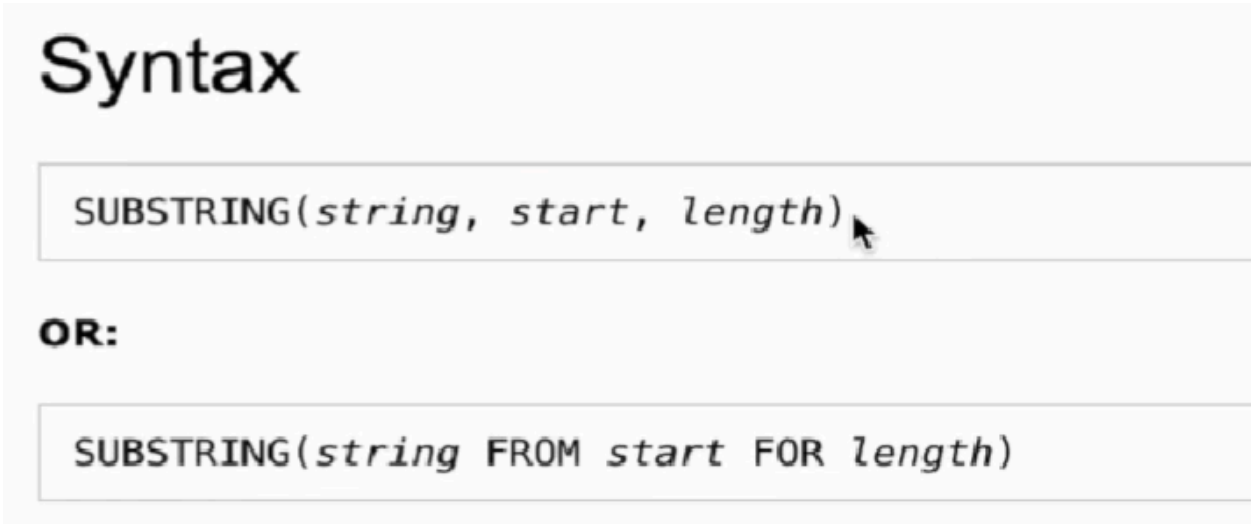
```
35
36  INSERT INTO Bonus
37      (WORKER_REF_ID, BONUS_AMOUNT, BONUS_DATE) VALUES
38          (001, 5000, '16-02-20'),
39          (002, 3000, '16-06-11'),
40          (003, 4000, '16-02-20'),
41          (001, 4500, '16-02-20'),
42          (002, 3500, '16-06-11');
43
44  CREATE TABLE Title (
45      WORKER_REF_ID INT,
46      WORKER_TITLE CHAR(25),
47      AFFECTED_FROM DATETIME,
48      FOREIGN KEY (WORKER_REF_ID)
49          REFERENCES Worker(WORKER_ID)
50          ON DELETE CASCADE
51  );
52
53  INSERT INTO Title
54      (WORKER_REF_ID, WORKER_TITLE, AFFECTED_FROM) VALUES
55      (001, 'Manager', '2016-02-20 00:00:00'),
56      (002, 'Executive', '2016-06-11 00:00:00'),
57      (008, 'Executive', '2016-06-11 00:00:00'),
58      (005, 'Manager', '2016-06-11 00:00:00'),
59      (004, 'Asst. Manager', '2016-06-11 00:00:00'),
60      (007, 'Executive', '2016-06-11 00:00:00'),
61      (006, 'Lead', '2016-06-11 00:00:00'),
62      (003, 'Lead', '2016-06-11 00:00:00');
```

To Get Upper Case → **UPPER()**

To Get Unique Values → **DISTINCT()**

```
-- Q-1. Write an SQL query to fetch “FIRST_NAME” from Worker table using the alias name as <WORKER_NAME>.
select first_name AS WORKER_NAME from worker;

-- Q-2. Write an SQL query to fetch “FIRST_NAME” from Worker table in upper case.
select UPPER(first_name) from worker;

-- Q-3. Write an SQL query to fetch unique values of DEPARTMENT from Worker table.
SELECT distinct department from worker;
```

SUBSTRING(string,start,length)



Example

Extract a substring from a string (start at position 5, extract 3 characters):

```
SELECT SUBSTRING("SQL Tutorial", 5, 3) AS ExtractString;
```

INSTR(string1,string2)

The INSTR() function returns the position of the first occurrence of a string in another string.

This function performs a case-insensitive search.

Syntax

```
INSTR(string1, string2)
```

Parameter Values

Parameter	Description
string1	Required. The string that will be searched
string2	Required. The string to search for in *string1*. If *string2* is not found, this function returns 0

Example

Search for "3" in string "W3Schools.com", and return position:

```
SELECT INSTR("W3Schools.com", "3") AS MatchPosition;
```

```
-- Q-2. Write an SQL query to fetch “FIRST_NAME” from Worker table in upper case.
select UPPER(first_name) from worker;

-- Q-3. Write an SQL query to fetch unique values of DEPARTMENT from Worker table.
SELECT distinct department from worker;

-- Q-4. Write an SQL query to print the first three characters of  FIRST_NAME from Worker table.
select substring(first_name, 1, 3) from worker;

-- Q-5. Write an SQL query to find the position of the alphabet (‘b’) in the first name column ‘Amitabh’ from Worker table.
select INSTR(first_name, 'B') from worker where first_name = 'Amitabh';
```

MySQL RTRIM(),LTRIM() Function

Remove trailing spaces from a string:

```
SELECT RTRIM("SQL Tutorial      ") AS RightTrimmedString;
```

```
-- Q-5. Write an SQL query to find the position of the alphabet (‘b’) in the first name column ‘Amitabh’ from Worker table.
select INSTR(first_name, 'B') from worker where first_name = 'Amitabh';

-- Q-6. Write an SQL query to print the FIRST_NAME from Worker table after removing white spaces from the right side.
select RTRIM(first_name) from worker;

-- Q-7. Write an SQL query to print the DEPARTMENT from Worker table after removing white spaces from the left side.
select LTRIM(first_name) from worker;
```

MySQL LENGTH() Function

Example

Return the length of the string, in bytes:

```
SELECT LENGTH("SQL Tutorial") AS LengthOfString;
```

```sql
-- Q-6. Write an SQL query to print the FIRST_NAME from Worker table after removing white spaces from the right side.
select RTRIM(first_name) from worker;

-- Q-7. Write an SQL query to print the DEPARTMENT from Worker table after removing white spaces from the left side.
select LTRIM(first_name) from worker;

-- Q-8. Write an SQL query that fetches the unique values of DEPARTMENT from Worker table and prints its length.
select distinct department, LENGTH(department) from worker;
```

MySQL REPLACE() Function

Example

Replace "SQL" with "HTML":

```sql
SELECT REPLACE("SQL Tutorial", "SQL", "HTML");
```

CONCAT

```sql
-- Q-8. Write an SQL query that fetches the unique values of DEPARTMENT from Worker table and prints its length.
select distinct department, LENGTH(department) from worker;

-- Q-9. Write an SQL query to print the FIRST_NAME from Worker table after replacing 'a' with 'A'.
select REPLACE(first_name, 'a', 'A')  from worker;

-- Q-10. Write an SQL query to print the FIRST_NAME and LAST_NAME from Worker table into a single column COMPLETE_NAME.
-- A space char should separate them.
select CONCAT(first_name, ' ', last_name) AS COMPLETE_NAME from worker;

-- Q-11. Write an SQL query to print all Worker details from the Worker table order by FIRST_NAME Ascending.

-- Q-12. Write an SQL query to print all Worker details from the Worker table order by
-- FIRST_NAME Ascending and DEPARTMENT Descending.
```

100% 59:30

Result Grid | Filter Rows: Search | Export:

COMPLETE_NAME
Monika Arora
Niharika Verma
Vishal Singhal
Amitabh Singh
Vivek Bhati
Vipul Diwan
Satish Kumar
Geetika Chauhan

Example

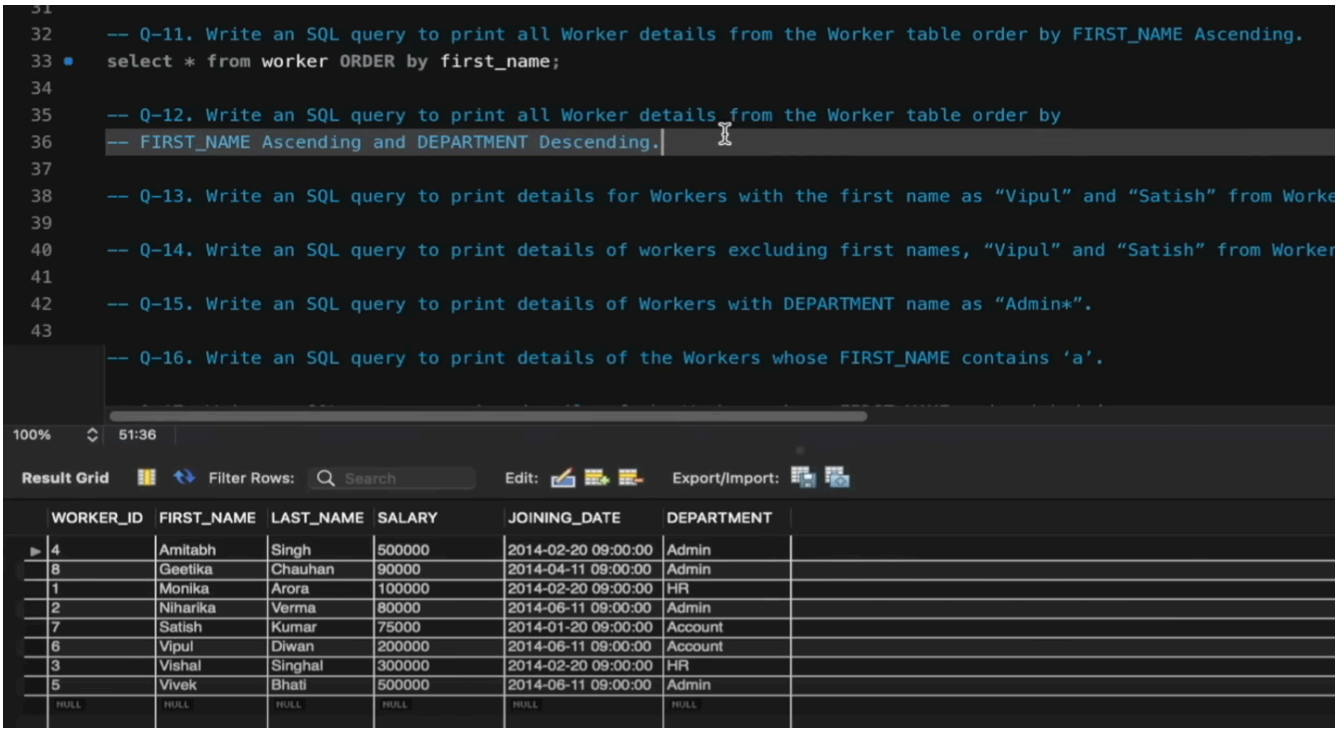
Add several strings together:

```
SELECT CONCAT("SQL ", "Tutorial ", "is ", "fun!") AS ConcatenatedString;
```

Example

Add three columns into one "Address" column:

```
SELECT CONCAT(Address, " ", PostalCode, " ", City) AS Address
FROM Customers;
```




```
-- Q-11. Write an SQL query to print all Worker details from the Worker table order by FIRST_NAME Ascending.
select * from worker ORDER by first_name;

-- Q-12. Write an SQL query to print all Worker details from the Worker table order by
-- FIRST_NAME Ascending and DEPARTMENT Descending.
select * from worker order by first_name, department DESC;
```

IN,NOT IN,LIKE

```sql
-- FIRST_NAME Ascending and DEPARTMENT Descending.
select * from worker order by first_name, department DESC;

-- Q-13. Write an SQL query to print details for Workers with the first name as “Vipul” and “Satish” from Worker table.
select * from worker where first_name IN ('Vipul', 'Satish');

-- Q-14. Write an SQL query to print details of workers excluding first names, “Vipul” and “Satish” from Worker table.
select * from worker where first_name NOT IN ('Vipul', 'Satish');

-- Q-15. Write an SQL query to print details of Workers with DEPARTMENT name as “Admin*”.
select * from worker where department LIKE 'Admin%';
```

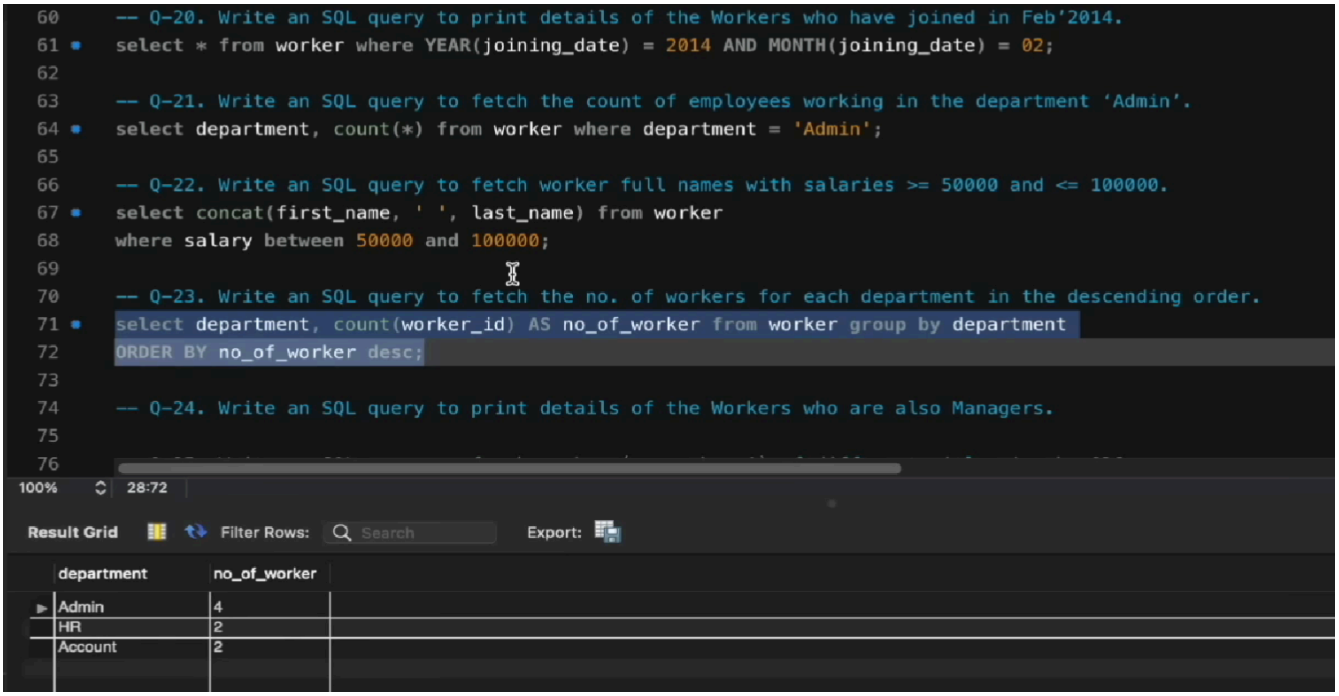
```sql
-- Q-15. Write an SQL query to print details of Workers with DEPARTMENT name as “Admin*”.
select * from worker where department LIKE 'Admin%';

-- Q-16. Write an SQL query to print details of the Workers whose FIRST_NAME contains ‘a’.
select * from worker where first_name LIKE '%a%';

-- Q-17. Write an SQL query to print details of the Workers whose FIRST_NAME ends with ‘a’.
select * from worker where first_name LIKE '%a';
```

MONTH (), YEAR()

```sql
-- Q-18. Write an SQL query to print details of the Workers whose FIRST_NAME ends with ‘h’ and contains six alphabets.
select * from worker where first_name LIKE '_____h';

-- Q-19. Write an SQL query to print details of the Workers whose SALARY lies between 100000 and 500000.
select * from worker where salary between 100000 AND 500000;

-- Q-20. Write an SQL query to print details of the Workers who have joined in Feb’2014.
select * from worker where YEAR(joining_date) = 2014 AND MONTH(joining_date) = 02;
```

```sql
-- Q-20. Write an SQL query to print details of the Workers who have joined in Feb’2014.
select * from worker where YEAR(joining_date) = 2014 AND MONTH(joining_date) = 02;

-- Q-21. Write an SQL query to fetch the count of employees working in the department ‘Admin’.
select department, count(*) from worker where department = 'Admin';

-- Q-22. Write an SQL query to fetch worker full names with salaries >= 50000 and <= 100000.
select concat(first_name, ' ', last_name) from worker
where salary between 50000 and 100000;
```

ALIAS, GROUP BY, ORDER BY



```sql
-- Q-20. Write an SQL query to print details of the Workers who have joined in Feb'2014.
select * from worker where YEAR(joining_date) = 2014 AND MONTH(joining_date) = 02;

-- Q-21. Write an SQL query to fetch the count of employees working in the department 'Admin'.
select department, count(*) from worker where department = 'Admin';

-- Q-22. Write an SQL query to fetch worker full names with salaries >= 50000 and <= 100000.
select concat(first_name, ' ', last_name) from worker
where salary between 50000 and 100000;

-- Q-23. Write an SQL query to fetch the no. of workers for each department in the descending order.
select department, count(worker_id) AS no_of_worker from worker group by department
ORDER BY no_of_worker desc;

-- Q-24. Write an SQL query to print details of the Workers who are also Managers.
```

100% 28:72

Result Grid Filter Rows: Search Export:

department	no_of_worker
Admin	4
HR	2
Account	2

SQL vs NoSQL: 5 Critical Differences

The five critical differences between SQL and NoSQL are:

- SQL databases are relational, and NoSQL databases are non-relational.
- SQL databases use structured query language (SQL) and have a predefined schema. NoSQL databases have dynamic schemas for unstructured data.
- SQL databases are vertically scalable, while NoSQL databases are horizontally scalable.
- SQL databases are table-based, while NoSQL databases are document, key-value, graph, or wide-column stores.
- SQL databases are better for multi-row transactions, while NoSQL is better for unstructured data like documents or JSON.

What is SQL?

SQL is a domain-specific language used to query and manage data. It works by allowing users to query, insert, delete, and update records in relational databases. SQL also allows for complex logic to be applied through the use of transactions and embedded procedures such as stored functions or views.

What is NoSQL?

NoSQL stands for Not only SQL. It is a type of database that uses non-relational data structures, such as documents, graph databases, and key-value stores to store and retrieve data. NoSQL systems are designed to be more flexible than traditional relational databases and can scale up or down easily to accommodate changes in usage or load. This makes them ideal for use in applications

Why NoSQL is Used Over SQL

NoSQL is preferred over SQL in many cases because it offers more flexibility and scalability. The primary benefit of using a NoSQL system is that it provides developers with the ability to store and access data quickly and easily, without the overhead of a traditional relational database. As a result, development teams can focus on delivering features and core business logic faster, without worrying about the underlying data storage implementation.

Which is better SQL or NoSQL?

The decision of which type of database to use - SQL or NoSQL - will depend on the particular needs and requirements of the project. For example, if you need a fast, scalable, and reliable database for web applications then a NoSQL system may be preferable. On the other hand, if your application requires complex data queries and transactional support then an SQL system may be the better choice. Ultimately, there is no one-size-fits-all solution - it all comes down to what you need from your database and which type of system can provide that in the most efficient manner. It's best to research both options thoroughly before making a decision.

Comparison of SQL vs NoSQL

With a basic understanding of what SQL vs NoSQL is, let's take a look at this quick comparison chart to see what sets the two apart:

SQL	NoSQL
Stands for Structured Query Language	Stands for Not Only SQL
Relational database management system (RDBMS)	Non-relational database management system
Suitable for structured data with predefined schema	Suitable for unstructured and semi-structured data
Data is stored in tables with columns and rows	Data is stored in collections or documents
Follows ACID properties (Atomicity, Consistency, Isolation, Durability) for transaction management	Does not necessarily follow ACID properties
Supports JOIN and complex queries	Does not support JOIN and complex queries
Uses normalized data structure	Uses denormalized data structure
Requires vertical scaling to handle large volumes of data	Horizontal scaling is possible to handle large volumes of data
Examples: MySQL, PostgreSQL, Oracle, SQL Server, Microsoft SQL Server	Examples: MongoDB, Cassandra, Couchbase, Amazon DynamoDB, Redis

What are Relational Databases?

Relational databases use Structured Query Language (SQL) to store and retrieve data.

- Relational databases (also called relational database management systems or RDBMSs) store data in rows and tables. These systems connect information from various tables with keys — unique identifiers that the database assigns to rows of data in tables. Primary keys and foreign keys facilitate this process.

What are Non-Relational Databases (NoSQL)?

Non-relational, or NoSQL databases are more flexible and don't necessarily require the same rigid structure as SQL.

- Non-relational databases store data just like relational databases. However, they don't contain any rows, tables, or keys. This type of database utilizes a storage model based on the type of data it stores.

Database Scaling

Another difference between SQL vs NoSQL databases is scaling. SQL databases are vertically scalable in most situations. That means you can increase the load on a single server by adding more CPU, RAM, or SSD capacity.

NoSQL databases are horizontally scalable. You can handle higher traffic via a process called sharding, which adds more servers to your NoSQL database. Horizontal scaling has a greater overall capacity than vertical scaling, making NoSQL databases the preferred choice for large and frequently changing data sets. For example, you might use a NoSQL database if you have large data objects like images and videos. An SQL database wouldn't be able to handle these objects as effectively, making it difficult to fulfill your data requirements.

Data Structure

SQL databases are table-based, where each field in a data record has the same name as a table column. This proves beneficial when performing multiple data transformations.

NoSQL databases are document, key-value, graph, or wide-column stores. These flexible data models make NoSQL databases easier for some developers to use.

Use Cases

SQL databases are better for multi-row transactions, while NoSQL is better for unstructured data like documents or JSON. SQL databases are also commonly used for legacy systems built around a relational structure.

You might use an SQL database for user-oriented applications with several join operations. SQL schema will help you establish ACID properties and improve data compatibility. These databases are also useful when quickly finding the data you need to complete a task.

You might use a NoSQL database for applications with dynamic data without join operations. NoSQL is also better suited for applications with missing data sets that won't impact business efficiency.

Some examples of SQL databases include MySQL, Oracle, PostgreSQL, and Microsoft SQL Server. NoSQL database examples include MongoDB, BigTable, Redis, Cassandra, HBase, Neo4j, and CouchDB.

SQL Database Systems

Here are some of the most popular SQL database systems:

MySQL

- Free and open-source
- An extremely established database with a huge community, extensive testing, and lots of stability
- Supports all major platforms
- Replication and sharding are available
- Covers a wide range of use cases

Oracle

- Commercial database with frequent updates, professional management, and excellent customer support
- Procedural Language/SQL or PL/SQL is the SQL dialect used
- One of the most expensive database solutions
- Works with huge databases
- Simple upgrades
- Transaction control
- Compatible with all operating systems
- Suitable for enterprises and organizations with demanding workloads

NoSQL Database Systems

Here are a couple of the most popular NoSQL database systems:

MongoDB

- By far the most popular NoSQL database, and for good reason
- Free to use
- Dynamic schema
- Horizontally scalable
- Excellent performance with simple queries
- Add new columns and fields without impacting your existing rows or application performance
- Works best for companies going through rapid growth stages or those with a lot of unstructured data
- Lesser-known alternatives to MongoDB include Apache Cassandra, Google Cloud BigTable, and Apache HBase

SQL QUERIES FOR INTERVIEWS

Q1. Write a query to fetch the EmpFname from the EmployeeInfo table in upper case and use the ALIAS name as EmpName.

```
SELECT UPPER(EmpFname) AS EmpName FROM EmployeeInfo;
```

Q2. Write a query to fetch the number of employees working in the department 'HR'.

```
SELECT COUNT(*) FROM EmployeeInfo WHERE Department = 'HR';
```

Q3. Write a query to get the current date.

You can write a query as follows in SQL Server:

```
SELECT GETDATE();
```

```
SELECT SYSTDATE();
```

Q4. Write a query to retrieve the first four characters of EmpLname from the EmployeeInfo table.

```
SELECT SUBSTRING(EmpLname, 1, 4) FROM EmployeeInfo;
```

Q5. Write a query to fetch only the place name(string before brackets) from the Address column of EmployeeInfo table.

Using the MID function in MySQL

```
SELECT MID(Address, 0, LOCATE('(',Address)) FROM EmployeeInfo;
```

Using SUBSTRING

```
SELECT SUBSTRING(Address, 1, CHARINDEX('(',Address)) FROM EmployeeInfo;
```

SQL Interview Questions

1. What is Database?

A database is an organized collection of structured information/data, stored and retrieved digitally from a remote or local computer system. A database is usually controlled by a database management system (DBMS).

2. What is DBMS?

A database management system (DBMS) is system software for creating and managing databases.
DBMS is a system software responsible for the creation, retrieval, updation, and management of the database.
A DBMS makes it possible for end users to create, protect, read, update and delete data in a database.
It ensures that our data is consistent, organized, and is easily accessible by **serving as an interface between the database and its end-users or application software**

3. What is RDBMS? How is it different from DBMS?

DBMS stands for Database Management System, and RDBMS is the acronym for the Relational Database Management system. In DBMS, the data is stored as a file, whereas in RDBMS, data is stored in the form of tables.
RDBMS stands for Relational Database Management System. The key difference here, compared to DBMS, is that RDBMS stores data in the form of a collection of tables, and relations can be defined between the common fields of these tables. Most modern database management systems like MySQL, Microsoft SQL Server, Oracle, IBM DB2, and Amazon Redshift are based on RDBMS.

4. What is SQL?

Structured query language (SQL) is a programming language for storing and processing information in a relational database. A relational database stores information in tabular form, with rows and columns representing different data attributes and the various relationships between the data values.

5. What is the difference between SQL and MySQL?

SQL is a query programming language for managing RDBMS. In contrast,

MySQL is an RDBMS (Relational Database Management System) that employs SQL. So, the major difference between the two is that MySQL is software, but SQL is a database language.

6. What are Tables and Fields?

A table is an organized collection of data stored in the form of rows and columns. Columns can be categorized as vertical and rows as horizontal. The columns in a table are called fields while the rows can be referred to as records.

7. What are Constraints in SQL?

SQL constraints are used to specify rules for the data in a table. Constraints are used to limit the type of data that can go into a table.

NOT NULL - Restricts NULL value from being inserted into a column.
CHECK - Verifies that all values in a field satisfy a condition.
DEFAULT - Automatically assigns a default value if no value has been specified for the field.
UNIQUE - Ensures unique values to be inserted into the field.
INDEX - Indexes a field providing faster retrieval of records.
PRIMARY KEY - Uniquely identifies each record in a table.

FOREIGN KEY - Ensures referential integrity for a record in another table.

8. What is a Primary Key?

designated to uniquely identify each table record

The PRIMARY KEY constraint **uniquely identifies each row** in a table. It must contain **UNIQUE** values and has an **implicit NOT NULL** constraint.

A table in SQL is strictly restricted to have one and only one primary key, which is composed of single or multiple fields (columns).

Primary key - Implicit NOT NULL - Cannot contain Null values

SQL Query - Primary Key - Unique + NOT NULL

Primary key constraints

The primary key does not accept any duplicate and NULL values

```
CREATE TABLE Students (   /* Create table with a single field as primary key */
  ID INT NOT NULL
  Name VARCHAR(255)
  PRIMARY KEY (ID)
);

CREATE TABLE Students (   /* Create table with multiple fields as primary key */
  ID INT NOT NULL
  LastName VARCHAR(255)
  FirstName VARCHAR(255) NOT NULL,
  CONSTRAINT PK_Student
  PRIMARY KEY (ID, FirstName)
);
```

```
ALTER TABLE Students   /* Set a column as primary key */
ADD PRIMARY KEY (ID);

ALTER TABLE Students   /* Set multiple columns as primary key */
ADD CONSTRAINT PK_Student   /*Naming a Primary Key*/
PRIMARY KEY (ID, FirstName);
```

The PRIMARY KEY constraint specifies that the constrained columns' values must uniquely identify each row.

9. What is a UNIQUE constraint?

The UNIQUE constraint ensures that all values in a column are different. Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns. A PRIMARY KEY constraint automatically has a UNIQUE constraint.

```
CREATE TABLE Students (   /* Create table with a single field as unique */
  ID INT NOT NULL UNIQUE
  Name VARCHAR(255)
);

CREATE TABLE Students (   /* Create table with multiple fields as unique */
  ID INT NOT NULL
  LastName VARCHAR(255)
  FirstName VARCHAR(255) NOT NULL
  CONSTRAINT PK_Student
  UNIQUE (ID, FirstName)
);

ALTER TABLE Students   /* Set a column as unique */
ADD UNIQUE (ID);

ALTER TABLE Students   /* Set multiple columns as unique */
ADD CONSTRAINT PK_Student   /* Naming a unique constraint */
UNIQUE (ID, FirstName);
```

10. What is a Foreign Key?

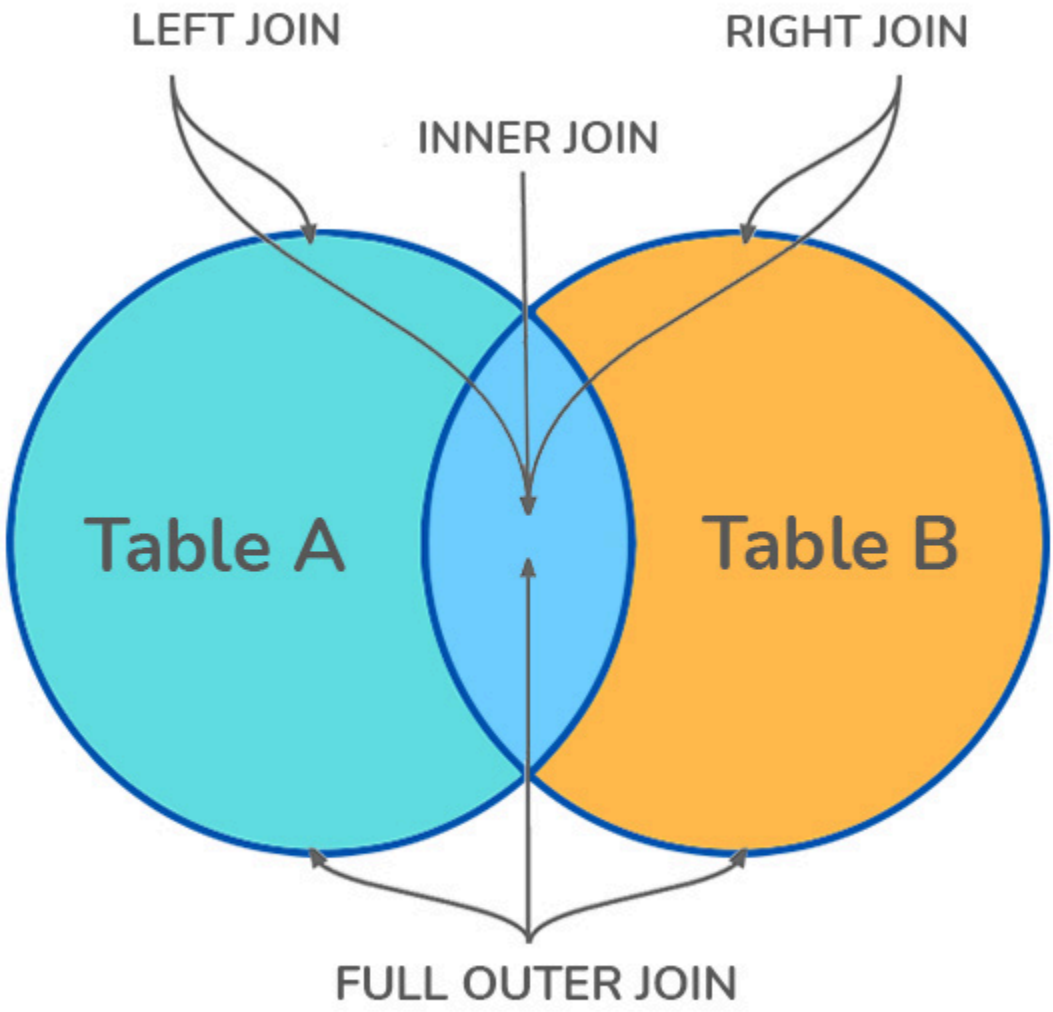
A foreign key is a column or group of columns in a relational database table that provides a **link between data in two tables**. It acts as a cross-reference between tables because it references the primary key of another table, thereby establishing a link between them.

A **FOREIGN KEY** comprises a single or collection of fields in a table that essentially **refers to the PRIMARY KEY in another table.**
Foreign key constraint ensures **referential integrity** in the relation between two tables.
The table with the foreign key constraint is labeled as the child table, and the table containing the candidate key is labeled as the referenced or parent table.

```
CREATE TABLE Students (   /* Create table with foreign key - Way 1 */
  ID INT NOT NULL
  Name VARCHAR(255)
  LibraryID INT
  PRIMARY KEY (ID)
  FOREIGN KEY (Library_ID) REFERENCES Library(LibraryID)
);

CREATE TABLE Students (   /* Create table with foreign key - Way 2 */
  ID INT NOT NULL PRIMARY KEY
  Name VARCHAR(255)
  LibraryID INT FOREIGN KEY (Library_ID) REFERENCES Library(LibraryID)
);

ALTER TABLE Students   /* Add a new foreign key */
ADD FOREIGN KEY (LibraryID)
REFERENCES Library (LibraryID);
```

11. What is a Join? List its different types.

The SQL Join clause is used to combine records (rows) from two or more tables in a SQL database based on a related column between the two.



There are four different types of JOINs in SQL:

- (INNER) JOIN: Retrieves records that have matching values in both tables involved in the join. This is the widely used join for queries.
-

```
SELECT *
FROM Table_A
JOIN Table_B;
SELECT *
FROM Table_A
INNER JOIN Table_B;
```

- LEFT (OUTER) JOIN: Retrieves all the records/rows from the left and the matched records/rows from the right table.

```
SELECT *
FROM Table_A A
LEFT JOIN Table_B B
ON A.col = B.col;
```

- RIGHT (OUTER) JOIN: Retrieves all the records/rows from the right and the matched records/rows from the left table.

```
SELECT *
FROM Table_A A
RIGHT JOIN Table_B B
ON A.col = B.col;
```

- FULL (OUTER) JOIN: Retrieves all the records where there is a match in either the left or right table.

```
SELECT *
FROM Table_A A
FULL JOIN Table_B B
ON A.col = B.col;
```

12. What is a Self-Join?

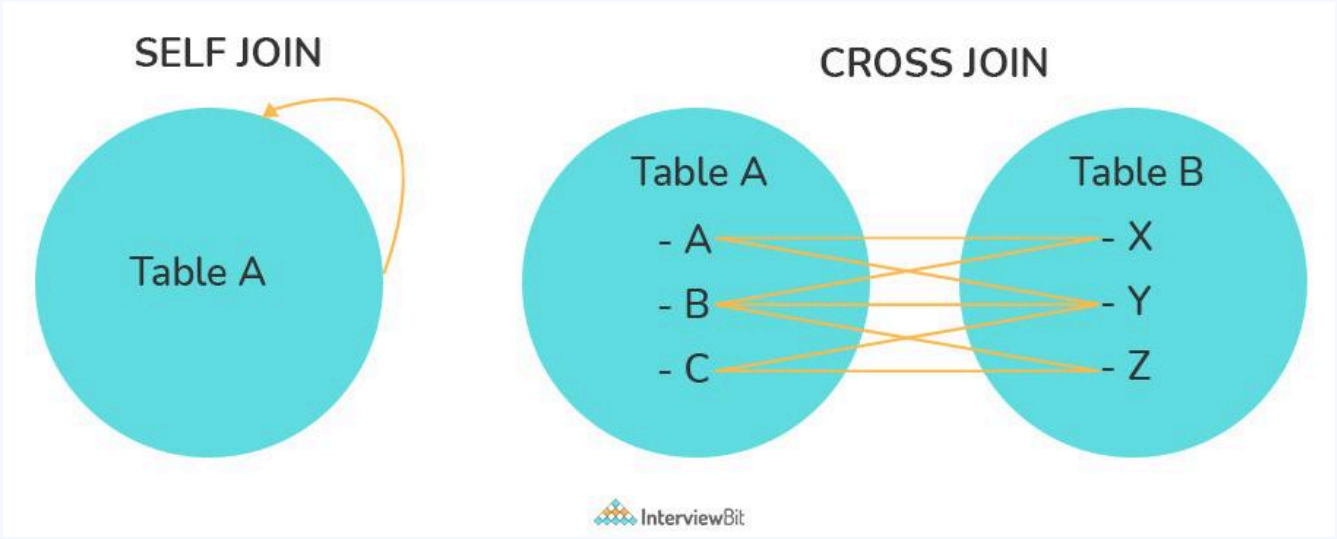
A self JOIN is a case of regular join where a table is joined to itself based on some relation between its own column(s). Self-join uses the INNER JOIN or LEFT JOIN clause and a table alias is used to assign different names to the table within the query.

```
SELECT A.emp_id AS "Emp_ID",A.emp_name AS "Employee",
B.emp_id AS "Sup_ID",B.emp_name AS "Supervisor"
FROM employee A, employee B
WHERE A.emp_sup = B.emp_id;
```

13. What is a Cross-Join?

Cross join can be defined as a cartesian product of the two tables included in the join. The table after join contains the same number of rows as in the cross-product of the number of rows in the two tables. If a WHERE clause is used in cross join then the query will work like an INNER JOIN.

```
SELECT stu.name, sub.subject
FROM students AS stu
CROSS JOIN subjects AS sub;
```



14. What is an Index? Explain its different types.

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more this costs.

A database index is a data structure that provides a quick lookup of data in a column or columns of a table. It enhances the speed of operations accessing data from a database table at the cost of additional writes and memory to maintain the index data structure.

CREATE INDEX index_name ON table_name (column_1, column_2);

DROP INDEX index_name;

14,15 → Later

17. What is a Query?

A query is a request for data or information from a database table or combination of tables. A database query can be either a select query or an action query.

```
SELECT fname, lname    /* select query */
FROM myDb.students
WHERE student_id = 1;

UPDATE myDB.students    /* action query */
SET fname = 'Captain', lname = 'America'
WHERE student_id = 1;
```

18. What is a Subquery? What are its types?

A subquery is a query within another query, also known as a nested query or inner query. It is used to restrict or enhance the data to be queried by the main query, thus restricting or enhancing the output of the main query respectively. **For example, here we fetch the contact information for students who have enrolled for the maths subject:**

```
SELECT name, email, mob, address
FROM myDb.contacts
WHERE roll_no IN (
 SELECT roll_no
 FROM myDb.students
 WHERE subject = 'Maths');
```

There are two types of subquery - Correlated and Non-Correlated.

- A correlated subquery cannot be considered as an independent query, but it can refer to the column in a table listed in the FROM of the main query.
- A non-correlated subquery can be considered as an independent query and the output of the subquery is substituted in the main query.

A non-correlated subquery is executed only once and its result can be swapped back for a query, on the other hand, a correlated subquery is executed multiple times, precisely once for each row returned by the outer query

non-correlated subquery:

SELECT MAX(Salary) FROM Employee WHERE Salary NOT IN (SELECT MAX(Salary) FROM Employee)

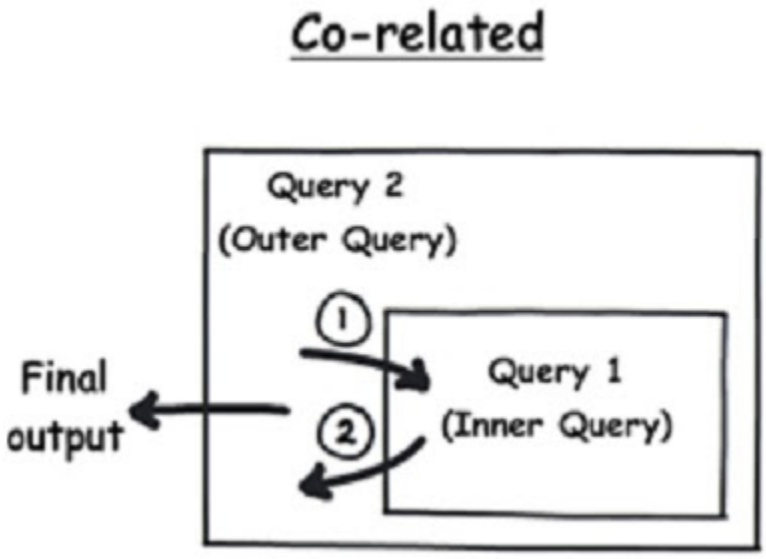
correlated subquery:

SELECT e.Name, e.Salary FROM Employee e WHERE 2 = (SELECT COUNT(Salary) FROM Employee p WHERE p.salary >= e.salary)

Dependency

A correlated subquery depends upon the outer query and cannot execute in isolation, but a regular or non-correlated subquery doesn't depend on the outer query and can execute in isolation.

From the above example, you can see that a correlated subquery like **SELECT COUNT(Salary) FROM Employee p WHERE p.salary >= e.salary** depends upon outer query because it needs the value of e.salary, which comes from the table listed on the outer query. On the other hand, a regular subquery, **SELECT MAX(Salary) FROM Employee**, doesn't depend upon the outer query and can be executed in isolation or independently of the outer query.



3.Speed and Performance

A correlated subquery is much slower than a non-correlated subquery because in the former, the inner query executes for each row of the outer query. This means if your table has n rows then whole processing will take the n * n = n^2 time, as compared to 2n times taken by a

non-correlated subquery. This happens because to execute a non-correlated subquery you need to examine just n rows of the table and similarly to execute the outer query you need to examine n rows, so in total n + n = 2n rows.

This is the reason you should be very careful using a correlated subquery with large tables e.g. tables with millions of rows because that can take a long time and could potentially block other jobs and queries from accessing the table.

In many cases, you can replace correlated subquery with inner join which would result in better performance.

For example, to find all employees whose salary is greater than the average salary of the department you can write the following correlated subquery:

```
SELECT e.id, e.name FROM Employee e WHERE salary > ( SELECT AVG(salary) FROM Employee p WHERE p.department = e.department)
```

Now, you can **convert this correlated subquery to a JOIN based query for better performance**.

```
SELECT e.id, e.name FROM Employee INNER JOIN (SELECT department, AVG(salary) AS department_average FROM Employee GROUP BY department) AS t ON e.department = t.department WHERE e.salary > t.department_average;
```

20. What are some common clauses used with SELECT query in SQL?

- WHERE clause in SQL is used to filter records that are necessary, based on specific conditions.
- ORDER BY clause in SQL is used to sort the records based on some field(s) in ascending (ASC) or descending order (DESC).

```
SELECT *
FROM myDB.students
WHERE graduation_year = 2019
ORDER BY studentID DESC;
```

- GROUP BY clause in SQL is used to group records with identical data and can be used in conjunction with some aggregation functions to produce summarized results from the database.
- HAVING clause in SQL is used to filter records in combination with the GROUP BY clause. It is different from WHERE, since the WHERE clause cannot filter aggregated records.

```
SELECT COUNT(studentId), country
FROM myDB.students
WHERE country != "INDIA"
GROUP BY country
HAVING COUNT(studentID) > 5;
```

21. What are UNION, MINUS and INTERSECT commands?

The UNION operator is used to combining the results of two tables, and it eliminates duplicate rows from the tables.
The MINUS operator is used to returning rows from the first query but not from the second query.
The INTERSECT operator in SQL is used to retrieve the records that are identical/common between the result sets of two SELECT (tables) statements.

The UNION operator combines and returns the result-set retrieved by two or more SELECT statements.

The MINUS operator in SQL is used to remove duplicates from the result-set obtained by the second SELECT query from the result-set obtained by the first SELECT query and then return the filtered results from the first.

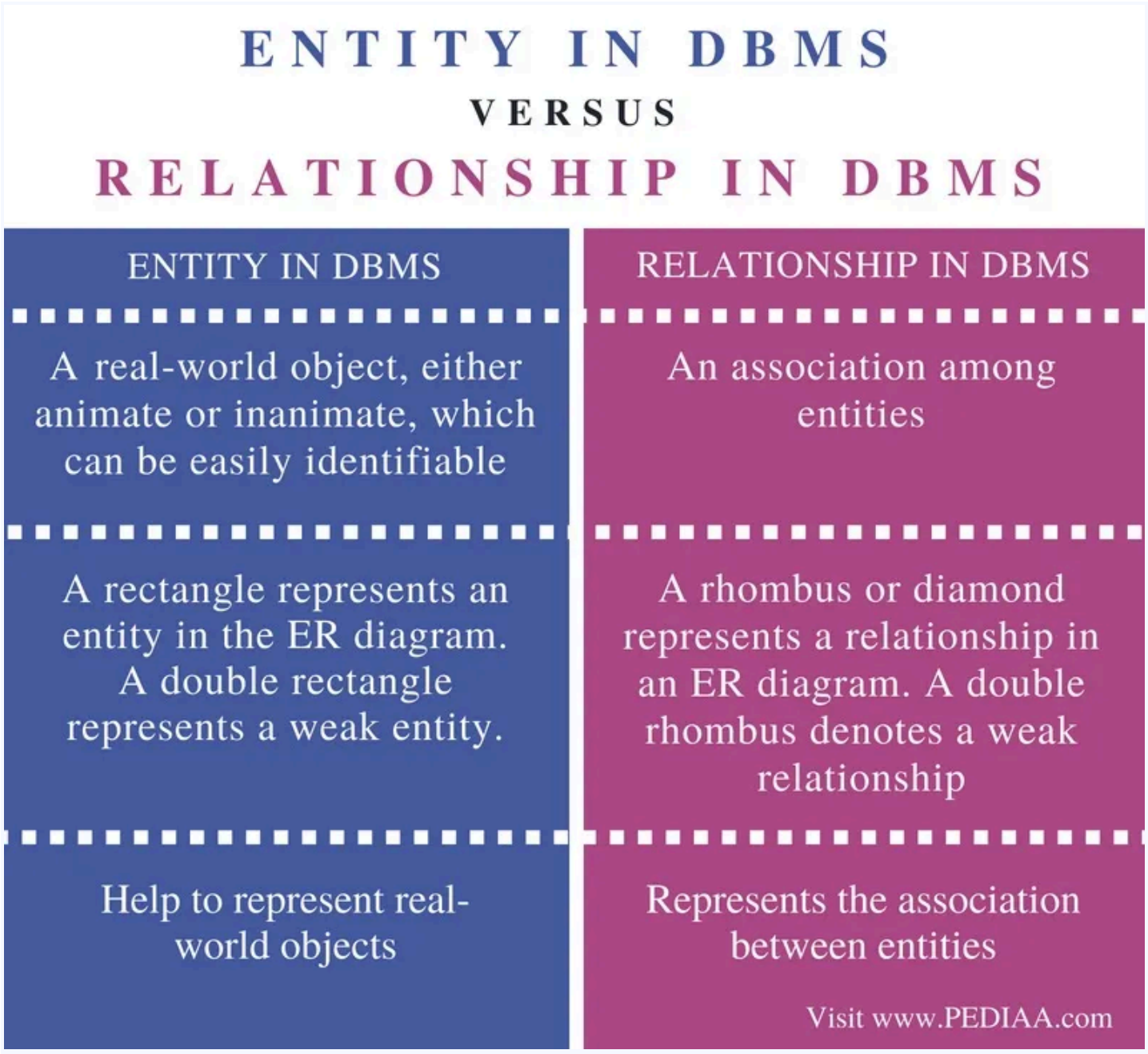
The INTERSECT clause in SQL combines the result-set fetched by the two SELECT statements where records from one match the other and then returns this intersection of result-sets.

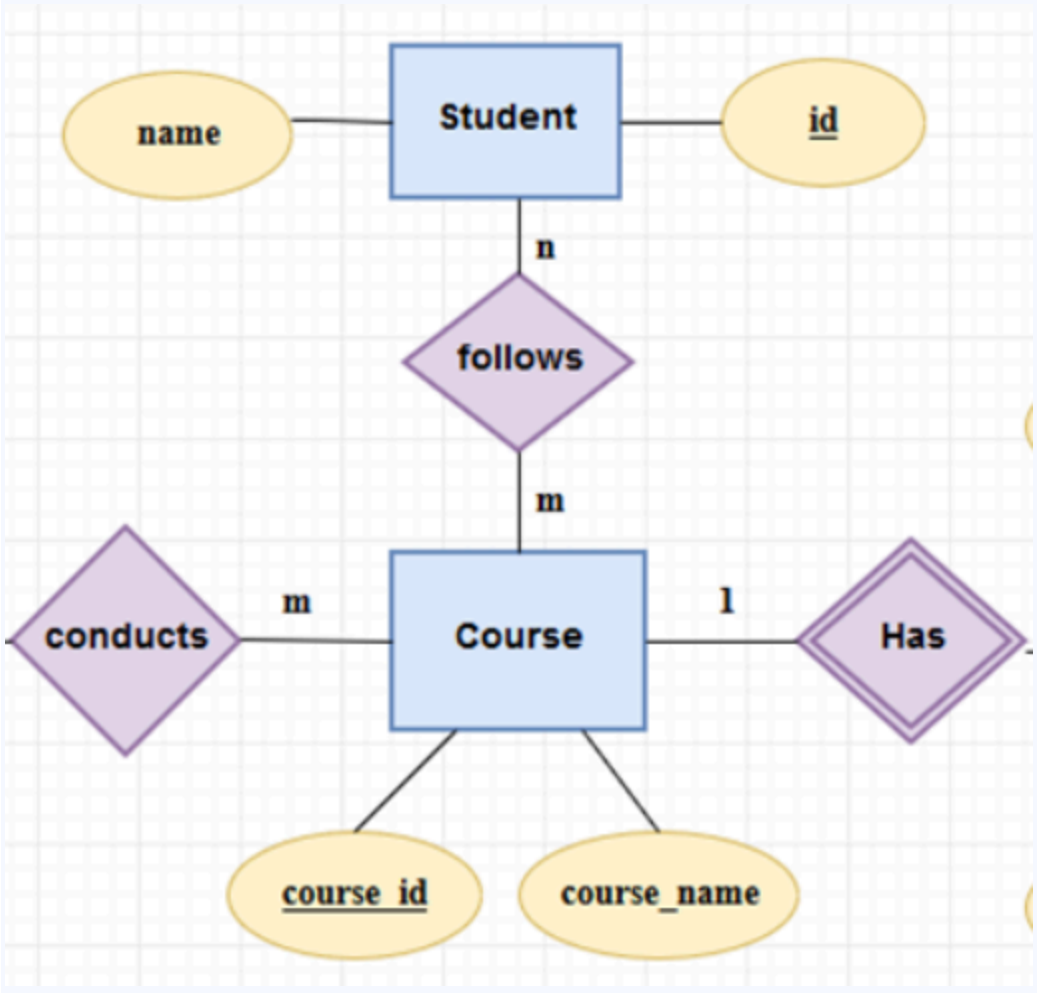
- **Each SELECT statement within the clause must have the same number of columns**
- **The columns must also have similar data types**
- **The columns in each SELECT statement should necessarily have the same order**

```
/* Fetch the union of queries */

SELECT name FROM Students UNION SELECT name FROM Contacts;

 /* Fetch the union of queries with duplicates*/

SELECT name FROM Students UNION ALL SELECT name FROM Contacts;

 /* Fetch names from students */  /* that aren't present in contacts */
SELECT name FROM Students
MINUS
 SELECT name FROM Contacts;
 /* Fetch names from students */  /* that also present in contacts */
SELECT name FROM Students
INTERSECT
 SELECT name FROM Contacts;
```

Entity, Relationship, DBMS

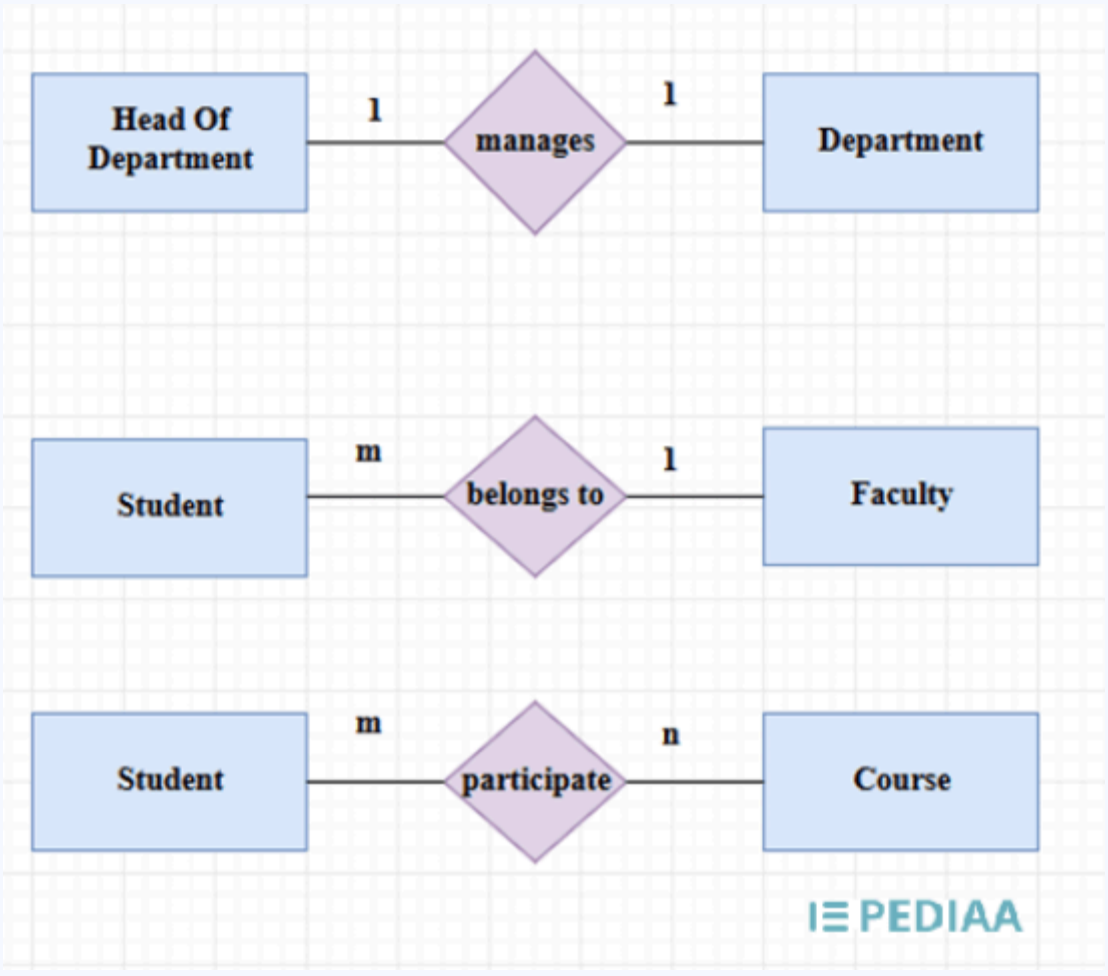
Difference Between Entity and Relationship in DBMS - Comparison Summary





What is a Relationship in DBMS

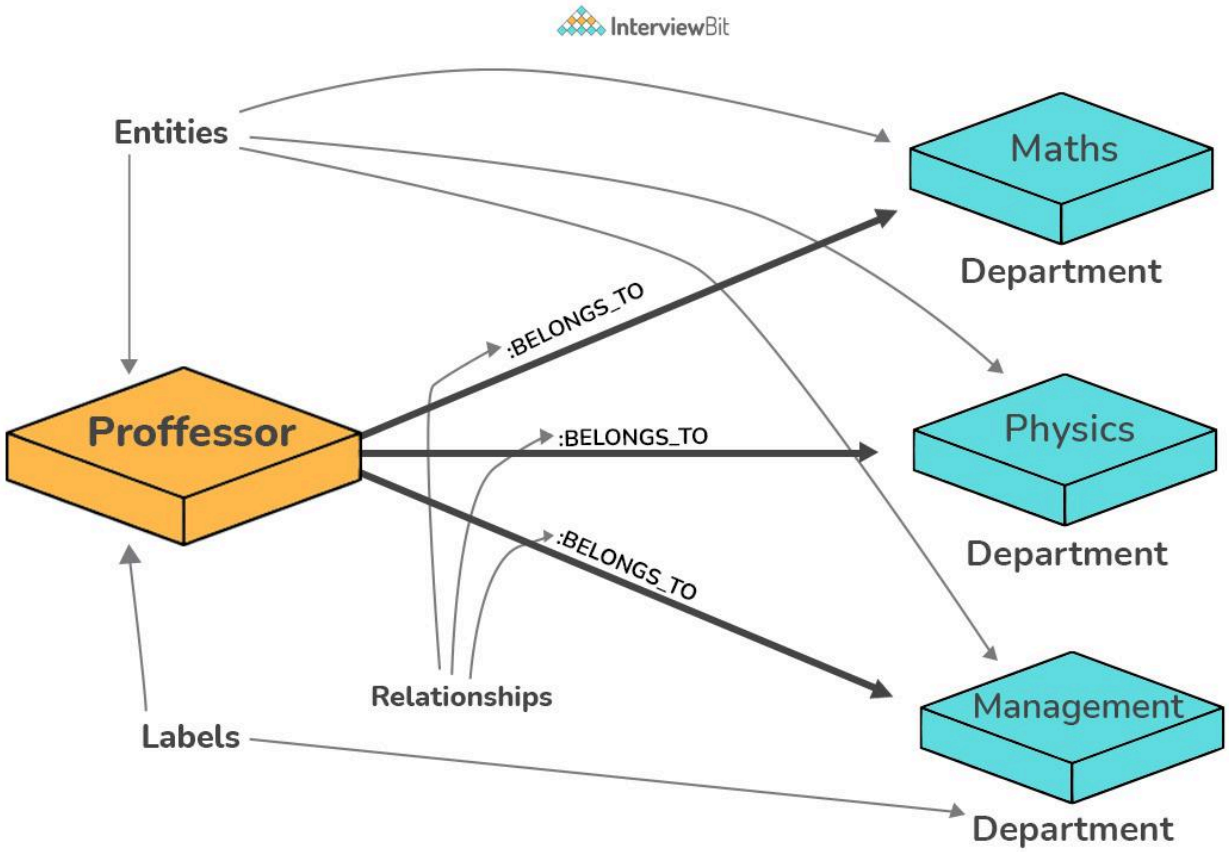
A relationship signifies an association among the entities. There are three types of relationships that can exist between the entities. They are the binary, recursive and the ternary relationship.



What are Entities and Relationships?

Entity: An entity can be a **real-world object**, either tangible or intangible, that can be easily identifiable. For example, in a college database, students, professors, workers, departments, and projects can be referred to as entities. Each entity has some associated properties that provide it an identity.

Relationships: Relations or **links between entities** that have something to do with each other. For example - The employee's table in a company's database can be associated with the salary table in the same database.



24. List the different types of relationships in SQL.

- One-to-One - This can be defined as the relationship between two tables where each record in one table is associated with the maximum of one record in the other table.
- One-to-Many & Many-to-One - This is the most commonly used relationship where a record in a table is associated with multiple records in the other table.
- Many-to-Many - This is used in cases when multiple instances on both sides are needed for defining a relationship.
- Self-Referencing Relationships - This is used when a table needs to define a relationship with itself.

25. What is an Alias in SQL?

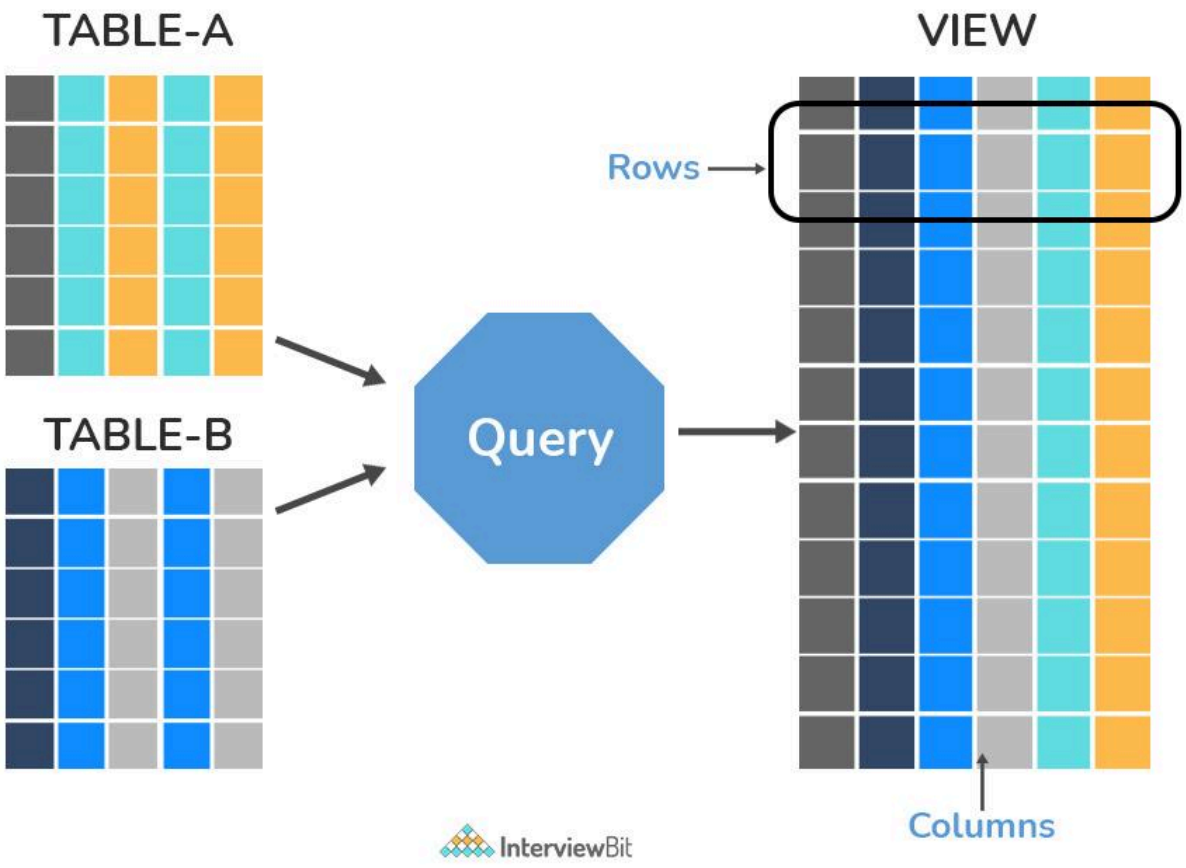
It is a temporary name assigned to the table or table column for the purpose of a particular SQL query. In addition, aliasing can be employed as an obfuscation technique to secure the real names of database fields. A table alias is also called a correlation name.

An alias is represented explicitly by the **AS keyword** but in some cases, the same can be performed without it as well. Nevertheless, using the AS keyword is always a good practice.

```
SELECT A.emp_name AS "Employee"  /* Alias using AS keyword */
B.emp_name AS "Supervisor"
FROM employee A, employee B   /* Alias without AS keyword */
WHERE A.emp_sup = B.emp_id;
```

What is a View?

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.



What is Normalization?

Normalization is the process to eliminate data redundancy and anomalies and enhance data integrity in the table.
Normalization also helps to organize the data in the database. It is a multi-step process that sets the data into tabular form and removes the duplicated data from the relational tables.
Normalization represents the way of organizing structured data in the database efficiently. It includes the creation of tables, establishing relationships between them, and defining rules for those relationships. Inconsistency and redundancy can be kept in check based on these rules, hence, adding flexibility to the database.

What is Denormalization?

Denormalization is the inverse process of normalization, where the normalized schema is converted into a schema that has redundant information. The performance is improved by using redundancy and keeping the redundant data consistent. The reason for performing denormalization is the overheads produced in the query processor by an over-normalized structure.

What are the various forms of Normalization?

Normal Forms are used to eliminate or reduce redundancy in database tables.

- ### First Normal Form:

  A relation is in first normal form if every attribute in that relation is a single-valued attribute.
- If a relation contains a composite or multi-valued attribute, it violates the first normal form.
- Let's consider the following students table. Each student in the table, has a name, his/her address, and the books they issued from the public library -

Students Table

Student	Address	Books Issued	Salutation
Sara	Amanora Park Town 94	Until the Day I Die (Emily Carpenter), Inception (Christopher Nolan)	Ms.
Ansh	62nd Sector A-10	The Alchemist (Paulo Coelho), Inferno (Dan Brown)	Mr.
Sara	24th Street Park Avenue	Beautiful Bad (Annie Ward), Woman 99 (Greer Macallister)	Mrs.
Ansh	Windsor Street 777	Dracula (Bram Stoker)	Mr.

Students Table (1st Normal Form)

Student	Address	Books Issued	Salutation
Sara	Amanora Park Town 94	Until the Day I Die (Emily Carpenter)	Ms.
Sara	Amanora Park Town 94	Inception (Christopher Nolan)	Ms.
Ansh	62nd Sector A-10	The Alchemist (Paulo Coelho)	Mr.
Ansh	62nd Sector A-10	Inferno (Dan Brown)	Mr.
Sara	24th Street Park Avenue	Beautiful Bad (Annie Ward)	Mrs.
Sara	24th Street Park Avenue	Woman 99 (Greer Macallister)	Mrs.
Ansh	Windsor Street 777	Dracula (Bram Stoker)	Mr.

- **Second Normal Form:**

A relation is in second normal form if it satisfies the conditions for the first normal form and does not contain any partial dependency.

A relation in 2NF has no partial dependency, i.e., it has no non-prime attribute that depends on any proper subset of any candidate key of the table. Often, specifying a single column Primary Key is the solution to the problem.

Second Normal Form (2NF)

- In the 2NF, relation must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

- ## Third Normal Form

A relation is said to be in the third normal form, if it satisfies the conditions for the second normal form and **there is no transitive dependency between the non-prime attributes**, i.e., all non-prime attributes are determined only by the candidate keys of the relation and not by any other non-prime attribute.

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

Third Normal Form (3NF):
A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

A relation is in 3NF if at least one of the following condition holds in every non-trivial functional dependency X –> Y:

1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

In other words,

A relation that is in First and Second Normal Form and in which no non-primary-key attribute is transitively dependent on the primary key, then it is in Third Normal Form (3NF).

Note – If A->B and B->C are two FDs then A->C is called transitive dependency.

The normalization of 2NF relations to 3NF involves the removal of transitive dependencies. If a transitive dependency exists, we remove the transitively dependent attribute(s) from the relation by placing the attribute(s) in a new relation along with a copy of the determinant.

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Table 4

FD set:
{STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, **STUD_STATE -> STUD_COUNTRY**, STUD_NO -> STUD_AGE}

Candidate Key:
{STUD_NO}

For this relation in table 4,

STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form.

To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as:

3NF can be achieved by partitioning into tables

```
STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
STATE_COUNTRY (STATE, COUNTRY)
```

- ## Boyce-Codd Normal Form

A relation is in Boyce-Codd Normal Form if satisfies the conditions for third normal form and for every functional dependency, Left-Hand-Side is super key. In other words, a relation in BCNF has non-trivial functional dependencies in form X –> Y, such that X is always a super key.

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.
-

What are the TRUNCATE, DELETE and DROP statements?

- DELETE statement is used to delete rows from a table.

```
DELETE FROM Candidates
WHERE CandidateId > 1000;
```

- TRUNCATE command is used to delete all the rows from the table and
- free the space containing the table.

```
TRUNCATE TABLE Candidates;
```

- DROP command is used to remove an object from the database.
- If you drop a table, all the rows in the table are deleted and
- the table structure is removed from the database.

```
DROP TABLE Candidates;
```

What is the difference between DROP and TRUNCATE statements?

- If a table is dropped, all things associated with the tables are dropped as well.
- This includes - the relationships defined on the table with other tables, the integrity checks and constraints, access privileges and other grants that the table has.
- To create and use the table again in its original form, all these relations, checks, constraints, privileges and relationships need to be redefined.
- **However, if a table is truncated**, none of the above problems exist and the table retains its original structure.

What is the difference between DELETE and TRUNCATE statements?

- The TRUNCATE command is used to delete all the rows from the table and free the space containing the table.

- The DELETE command deletes only the rows from the table based on the condition given in the where clause or deletes all the rows from the table if no condition is specified.
- But it does not free the space containing the table.

What are Aggregate and Scalar functions?

An aggregate function performs operations on a collection of values to return a single scalar value.

Aggregate functions are often used with the GROUP BY and HAVING clauses of the SELECT statement.

- **AVG()** - Calculates the mean of a collection of values.
- **COUNT()** - Counts the total number of records in a specific table or view.
- **MIN()** - Calculates the minimum of a collection of values.
- **MAX()** - Calculates the maximum of a collection of values.
- **SUM()** - Calculates the sum of a collection of values.
- **FIRST()** - Fetches the first element in a collection of values.
- **LAST()** - Fetches the last element in a collection of values.

Note: All aggregate functions described above ignore NULL values except for the COUNT function.

A scalar function returns a single value based on the input value.

- **LEN()** - Calculates the total length of the given field (column).
- **UCASE()** - Converts a collection of string values to uppercase characters.
- **LCASE()** - Converts a collection of string values to lowercase characters.
- **MID()** - Extracts substrings from a collection of string values in a table.
- **CONCAT()** - Concatenates two or more strings.
- **RAND()** - Generates a random collection of numbers of a given length.
- **ROUND()** - Calculates the round-off integer value for a numeric field (or decimal point values).
- **NOW()** - Returns the current date & time.
- **FORMAT()** - Sets the format to display a collection of values.

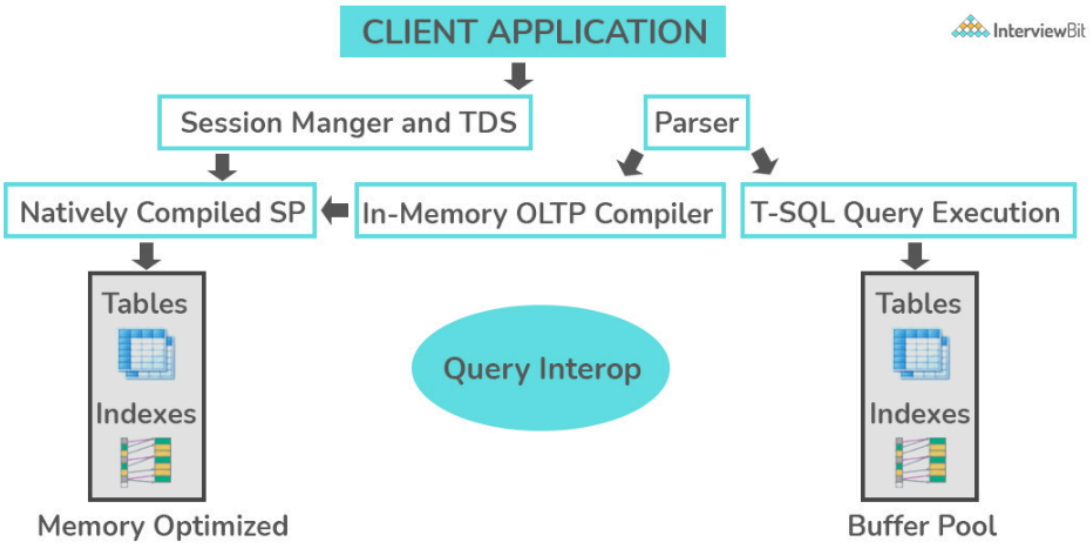
OLTP

OLTP, or online transactional processing, enables the real-time execution of large numbers of database transactions by large numbers of people, typically over the internet.

A database transaction is a change, insertion, deletion, or query of data in a database.

OLTP stands for Online Transaction Processing, is a class of software applications capable of supporting transaction-oriented programs. An essential attribute of an OLTP system is its ability to maintain concurrency.

These systems are usually designed for a large number of users who conduct short transactions. Database queries are usually simple, require sub-second response times, and return relatively few records.



What are the differences between OLTP and OLAP?

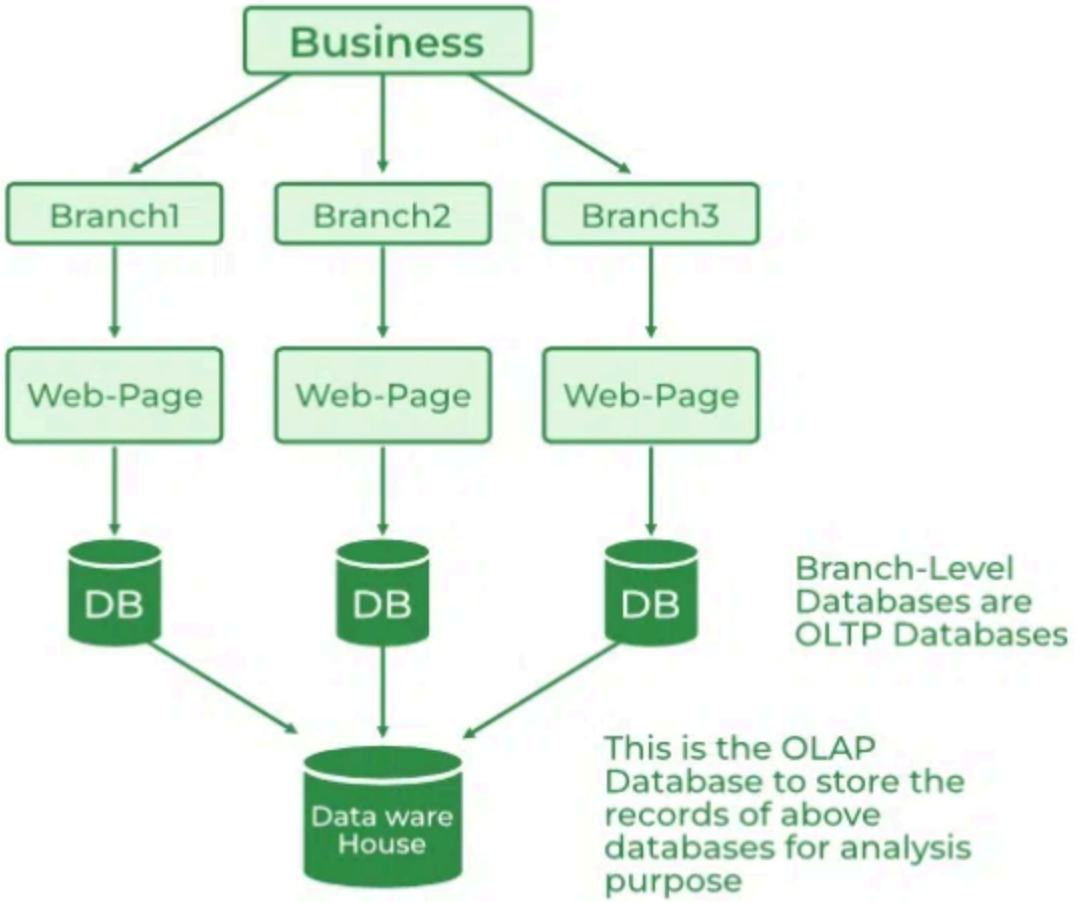
Online Analytical Processing (OLAP)

Online Analytical Processing (OLAP) consists of a type of software tool that is used for data analysis for business decisions. OLAP provides an environment to get insights from the database retrieved from multiple database systems at one time.

OLAP Examples

Any type of Data Warehouse System is an OLAP system.

- Spotify analyzed songs by users to come up with a personalized homepage of their songs and playlist.
- Netflix movie recommendation system.



OLAP

Benefits of OLAP Services

- **OLAP services help in keeping consistency and calculation.**
- **We can store planning, analysis, and budgeting for business analytics within one platform.**
- **OLAP services help in handling large volumes of data, which helps in enterprise-level business applications.**
- **OLAP services help in applying security restrictions for data protection.**

- **OLAP services provide a multidimensional view of data, which helps in applying operations on data in various ways.**

Drawbacks of OLAP Services

- **OLAP Services requires professionals to handle the data because of its complex modeling procedure.**
- **OLAP services are expensive to implement and maintain in cases when datasets are large.**
- **We can perform an analysis of data only after extraction and transformation of data in the case of OLAP which delays the system.**
- **OLAP services are not efficient for decision-making, as it is updated on a periodic basis.**

Online Transaction Processing (OLTP)

Online transaction processing provides transaction-oriented applications in a 3-tier architecture. OLTP administers the day-to-day transactions of an organization.

OLTP Examples

An example considered for OLTP System is ATM Center a person who authenticates first will receive the amount first and the condition is that the amount to be withdrawn must be present in the ATM. The uses of the OLTP System are described below.

- ATM center is an OLTP application.
- OLTP handles the ACID properties during data transactions via the application.

- It’s also used for Online banking, Online airline ticket booking, sending a text message, add a book to the shopping cart.

OLTP vs OLAP

Benefits of OLTP Services

- OLTP services allow users to read, write and delete data operations quickly.
- OLTP services help in increasing users and transactions which helps in real-time access to data.
- OLTP services help to provide better security by applying multiple security features.
- OLTP services help in making better decision making by providing accurate data or current data.
- OLTP Services provide Data Integrity, Consistency, and High Availability to the data.

Drawbacks of OLTP Services

- OLTP has limited analysis capability as they are not capable of intending complex analysis or reporting.
- OLTP has high maintenance costs because of frequent maintenance, backups, and recovery.

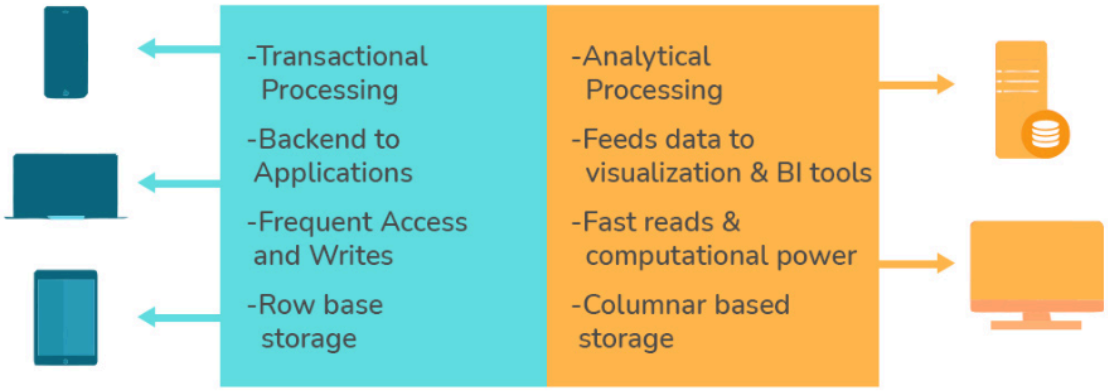
- OLTP Services get hampered in the case whenever there is a hardware failure which leads to the failure of online transactions.
- OLTP Services many times experience issues such as duplicate or inconsistent data.

Difference between OLAP and OLTP

Category	OLAP (Online Analytical Processing)	OLTP (Online Transaction Processing)
Definition	It is well-known as an online database query management system.	It is well-known as an online database modifying system.
Data source	Consists of historical data from various Databases.	Consists of only operational current data.
Method used	It makes use of a data warehouse.	It makes use of a standard database management system (DBMS).
Application	It is subject-oriented. Used for Data Mining, Analytics, Decisions making, etc.	It is application-oriented. Used for business tasks.
Normalized	In an OLAP database, tables are not normalized.	In an OLTP database, tables are normalized (3NF).

Usage of data	The data is used in planning, problem-solving, and decision-making.	The data is used to perform day-to-day fundamental operations.
Task	It provides a multi-dimensional view of different business tasks.	It reveals a snapshot of present business tasks.
Purpose	It serves the purpose to extract information for analysis and decision-making.	It serves the purpose to Insert, Update, and Delete information from the database.
Update	The OLAP database is not often updated. As a result, data integrity is unaffected.	The data integrity constraint must be maintained in an OLTP database.
Backup and Recovery	It only needs backup from time to time as compared to OLTP.	The backup and recovery process is maintained rigorously
Processing time	The processing of complex queries can take a lengthy time.	It is comparatively fast in processing because of simple and straightforward queries.
Types of users	This data is generally managed by CEO, MD, and GM.	This data is managed by clerksForex and managers.
Operations	Only read and rarely write operations.	Both read and write operations.
Updates	With lengthy, scheduled batch operations, data is refreshed on a regular basis.	The user initiates data updates, which are brief and quick.
Nature of audience	The process is focused on the customer.	The process is focused on the market.

Updates	With lengthy, scheduled batch operations, data is refreshed on a regular basis.	The user initiates data updates, which are brief and quick.
Nature of audience	The process is focused on the customer.	The process is focused on the market.
Database Design	Design with a focus on the subject.	Design that is focused on the application.
Productivity	Improves the efficiency of business analysts.	Enhances the user's productivity.

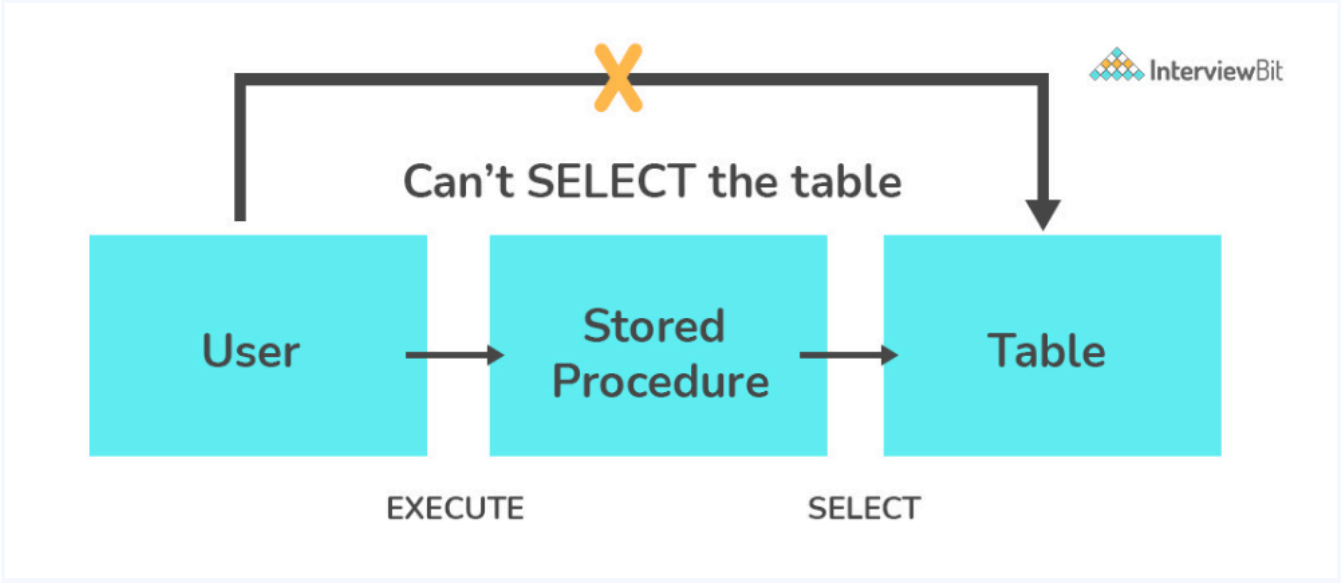


What is a Stored Procedure?

A stored procedure is a set of Structured Query Language (SQL) statements with an assigned name, which are stored in a relational database management system (RDBMS) as a group, so it can be reused and shared by multiple programs.

Such procedures are stored in the database data dictionary. The sole disadvantage of stored procedure is that it can be executed nowhere except in the database and occupies more memory in the database server. It also provides a sense of security and functionality as users who can't access the data directly can be granted access via stored procedures.

```
DELIMITER $$
CREATE PROCEDURE FetchAllStudents()
BEGIN SELECT * FROM myDB.students;
END
$$ DELIMITER ;
```



How to create empty tables with the same structure as another table?

Creating empty tables with the same structure can be done smartly by fetching the records of one table into a new table using the INTO operator while fixing a WHERE clause to be false for all records.

Hence, SQL prepares the new table with a duplicate structure to accept the fetched records but since no records get fetched due to the WHERE clause in action, nothing is inserted into the new table.

```
SELECT * INTO Students_copy
FROM Students WHERE 1 = 2;
```

What is Pattern Matching in SQL?

SQL pattern matching provides for pattern search in data if you have no clue as to what that word should be. This kind of SQL query uses wildcards to match a string pattern, rather than writing the exact word. The LIKE operator is used in conjunction with SQL Wildcards to fetch the required information.

- **Using the % wildcard to perform a simple search**

The % wildcard matches zero or more characters of any type and can be used to define wildcards both before and after the pattern. Search a student in your database with first name beginning with the letter K:

```
SELECT * FROM students WHERE first_name LIKE 'K%'
```

- Omitting the patterns using the NOT keyword

Use the NOT keyword to select records that don't match the pattern. This query returns all students whose first name does not begin with K.

```
SELECT  * FROM students WHERE first_name NOT LIKE 'K%'
```

- Matching a pattern anywhere using the % wildcard twice

Search for a student in the database where he/she has a K in his/her first name.

```
SELECT *  FROM students

WHERE first_name LIKE '%Q%'
```

- Using the _ wildcard to match pattern at a specific position

The _ wildcard matches exactly one character of any type. It can be used in conjunction with % wildcard. This query fetches all students with letter K at the third position in their first name.

```
SELECT * FROM students

WHERE first_name LIKE '__K%'
```

- Matching patterns for a specific length

The _ wildcard plays an important role as a limitation when it matches exactly one character. It limits the length and position of the matched results. For example -

```
SELECT * FROM students

WHERE first_name LIKE '___%'

SELECT * FROM students

WHERE first_name LIKE '____'
```

SQL 50 Questions - Leetcode

To Improve Run Time
Single Query with **OR** ←→ Multiple queries with **UNION**

```sql
#OR
SELECT name, population, area
FROM World
WHERE area > 3000000 OR population > 25000000
```

And Faster Union

```sql
#Union
SELECT name, population, area
FROM World
WHERE area > 3000000

UNION

SELECT name, population, area
FROM World
WHERE population > 25000000
```

LENGTH() returns the length of the string measured in bytes.
CHAR_LENGTH() returns the length of the string measured in characters.

```mysql
select tweet_id from Tweets
where length(content)>15;

select tweet_id from Tweets
where char_length(content)>15;
```

MySQL

```mysql
select tweet_id from Tweets where length(content)>15
```

```mysql
select tweet_id from Tweets where char_length(content)>15
```

LEFT JOIN

Employees table:

id	name
1	Alice
7	Bob
11	Meir
90	Winston
3	Jonathan

EmployeeUNI table:

id	unique_id
3	1
11	2
90	3

Output:

unique_id	name
null	Alice
null	Bob
2	Meir
3	Winston
1	Jonathan

```sql
select e2.unique_id,e1.name from Employees e1
LEFT JOIN EmployeeUNI e2
on e1.id=e2.id;



select e2.unique_id,e1.name from Employees e1
LEFT JOIN EmployeeUNI e2
using(id);
```

INNER JOIN

Sales table:

sale_id	product_id	year	quantity	price
1	100	2008	10	5000
2	100	2009	12	5000
7	200	2011	15	9000

Product table:

product_id	product_name
100	Nokia
200	Apple
300	Samsung

Output:

product_name	year	price
Nokia	2008	5000
Nokia	2009	5000
Apple	2011	9000

```sql
select b.product_name,a.year,a.price
from Sales a
Inner Join Product b
on a.product_id=b.product_id;

select b.product_name,a.year,a.price
from Sales a
Inner Join Product b
using (product_id);
```

1581. Customer Who Visited but Did Not Make Any Transactions

Write a solution to find the IDs of the users who visited without making any transactions and the number of times they made these types of visits.

Input:

Visits

visit_id	customer_id
1	23
2	9
4	30
5	54
6	96
7	54
8	54

Transactions

transaction_id	visit_id	amount
2	5	310
3	5	300
9	5	200
12	1	910
13	2	970

Output:

customer_id	count_no_trans
54	2
30	1
96	1

```sql
select customer_id,count(visit_id) as count_no_trans from
Visits
LEFT JOIN Transactions
using(visit_id)
where transaction_id is NULL
group by customer_id;
```

datediff(date1,date2)

LEFT JOIN

Input:

Employee table:

empId	name	supervisor	salary
3	Brad	null	4000
1	John	3	1000
2	Dan	3	2000
4	Thomas	3	4000

Bonus table:

empId	bonus
2	500
4	2000

Output:

name	bonus
Brad	null
John	null
Dan	500

```sql
select name ,bonus from Employee
left join
Bonus
using(empId)
where bonus<1000 or bonus is NULL;
```

570. Managers with at Least 5 Direct Reports

Input:

Employee table:

id	name	department	managerId
101	John	A	None
102	Dan	A	101
103	James	A	101
104	Amy	A	101
105	Anne	A	101
106	Ron	B	101

Output:

name
John

```sql
select name from Employee
where id in (select managerId from Employee group by managerID having count(managerId)>=5);


select m.name from employee as e
inner join employee as m
on e.managerId=m.id
group by e.managerId
having count(e.id)>=5;
```

AGGREGATIONS

Write a solution to report the movies with an odd-numbered ID and a description that is not `"boring"`.

Return the result table ordered by `rating` **in descending order**.

The result format is in the following example.

Example 1:

Input:

Cinema table:

id	movie	description	rating
1	War	great 3D	8.9
2	Science	fiction	8.5
3	irish	boring	6.2
4	Ice song	Fantacy	8.6
5	House card	Interesting	9.1

Output:

id	movie	description	rating
5	House card	Interesting	9.1
1	War	great 3D	8.9

Explanation:

We have three movies with odd-numbered IDs: 1, 3, and 5. The movie with ID = 3 is boring so we do not include it in the answer.

MOD(ID,2)=1

```sql
select * from Cinema
where mod(id,2)=1 and description!="boring"
order by rating desc;
```

```sql
select * from Cinema
where (id%2=1) and description<>"boring"
order by rating desc;
```

Write an SQL query to find the average selling price for each product. `average_price` should be **rounded to 2 decimal places**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Prices table:

product_id	start_date	end_date	price
1	2019-02-17	2019-02-28	5
1	2019-03-01	2019-03-22	20
2	2019-02-01	2019-02-20	15
2	2019-02-21	2019-03-31	30

UnitsSold table:

product_id	purchase_date	units
1	2019-02-25	100
1	2019-03-01	15
2	2019-02-10	200
2	2019-03-22	30

Output:

product_id	average_price
1	6.96
2	16.96

Explanation:

Average selling price = Total Price of Product / Number of products sold.
Average selling price for product 1 = ((100 * 5) + (15 * 20)) / 115 = 6.96
Average selling price for product 2 = ((200 * 15) + (30 * 30)) / 230 = 16.96

```sql
select p.product_id,round(sum(p.price*u.units)/sum(u.units),2)  as
average_price
from prices p
inner join unitssold u
on p.product_id=u.product_id
and u.purchase_date between p.start_date and p.end_date
group by p.product_id;
```

Write an SQL query that reports the **average** experience years of all the employees for each project, **rounded to 2 digits**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Project table:

project_id	employee_id
1	1
1	2
1	3
2	1
2	4

Employee table:

employee_id	name	experience_years
1	Khaled	3
2	Ali	2
3	John	1
4	Doe	2

Output:

project_id	average_years
1	2.00
2	2.50

Explanation: The average experience years for the first project is (3 + 2 + 1) / 3 = 2.00 and for the second project is (3 + 2) / 2 = 2.50

```sql
select p.project_id,round(sum(e.experience_years)/count(e.employee_id),2) as average_years
from Project p
Inner Join Employee e
on p.employee_id=e.employee_id
group by p.project_id;
```

Write a solution to find the percentage of the users registered in each contest rounded to **two decimals**.

Return the result table ordered by `percentage` in **descending order**. In case of a tie, order it by `contest_id` in **ascending order**.

The result format is in the following example.

Example 1:

Input:

Users table:

user_id	user_name
6	Alice
2	Bob
7	Alex

Register table:

contest_id	user_id
215	6
209	2
208	2
210	6
208	6
209	7
209	6
215	7
208	7
210	2
207	2
210	7

Output:

contest_id	percentage
208	100.0
209	100.0
210	100.0
215	66.67
207	33.33

Explanation:
All the users registered in contests 208, 209, and 210. The percentage is 100% and we sort them in the answer table by contest_id in ascending order.
Alice and Alex registered in contest 215 and the percentage is ((2/3) * 100) = 66.67%
Bob registered in contest 207 and the percentage is ((1/3) * 100) = 33.33%

```sql
select p.project_id,round(sum(e.experience_years)/count(e.employee_id),2) as average_years
from Project p
Inner Join Employee e
on p.employee_id=e.employee_id
group by p.project_id;
```

1211. Queries Quality and Percentage

Write an SQL query to find each `query_name`, the `quality` and `poor_query_percentage`.

Both `quality` and `poor_query_percentage` should be **rounded to 2 decimal places**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:
Queries table:

query_name	result	position	rating
Dog	Golden Retriever	1	5
Dog	German Shepherd	2	5
Dog	Mule	200	1
Cat	Shirazi	5	2
Cat	Siamese	3	3
Cat	Sphynx	7	4

Input:

Queries table:

query_name	result	position	rating
Dog	Golden Retriever	1	5
Dog	German Shepherd	2	5
Dog	Mule	200	1
Cat	Shirazi	5	2
Cat	Siamese	3	3
Cat	Sphynx	7	4

Output:

query_name	quality	poor_query_percentage
Dog	2.50	33.33
Cat	0.66	33.33

Explanation:

Dog queries quality is ((5 / 1) + (5 / 2) + (1 / 200)) / 3 = 2.50
Dog queries poor_ query_percentage is (1 / 3) * 100 = 33.33

Cat queries quality equals ((2 / 5) + (3 / 3) + (4 / 7)) / 3 = 0.66
Cat queries poor_ query_percentage is (1 / 3) * 100 = 33.33

```sql
select query_name,
ROUND(AVG(rating/position),2) as quality,
ROUND(AVG(rating<3)*100,2) as poor_query_percentage
from Queries
group by query_name;
```

CASE WHEN THEN END

Report for every three line segments whether they can form a triangle.

Return the result table in **any order**.

The result format is in the following example.

Example 1:

> **Input:**
> Triangle table:
> +----+----+----+
> | x | y | z |
> +----+----+----+
> | 13 | 15 | 30 |
> | 10 | 20 | 15 |
> +----+----+----+
> **Output:**
> +----+----+----+----------+
> | x | y | z | triangle |
> +----+----+----+----------+
> | 13 | 15 | 30 | No |
> | 10 | 20 | 15 | Yes |
> +----+----+----+----------+

```sql
select x,y,z,
case when x+y>z and y+z>x and z+x>y then 'Yes'
else 'No'
end as triangle
from Triangle;
```

DISTINCT COUNT GROUP BY

Write a solution to calculate the number of unique subjects each teacher teaches in the university.

Input:

Teacher table:

teacher_id	subject_id	dept_id
1	2	3
1	2	4
1	3	3
2	1	1
2	2	1
2	3	1
2	4	1

Output:

teacher_id	cnt
1	2
2	4

Explanation:

Teacher 1:
- They teach subject 2 in departments 3 and 4.
- They teach subject 3 in department 3.

Teacher 2:
- They teach subject 1 in department 1.
- They teach subject 2 in department 1.
- They teach subject 3 in department 1.
- They teach subject 4 in department 1.

```sql
select teacher_id,count(distinct subject_id) as cnt from Teacher
group by teacher_id;
```

BETWEEN, GROUP BY

Write a solution to find the daily active user count for a period of `30` days ending `2019-07-27` inclusively. A user was active on someday if they made at least one activity on that day.

Return the result table in **any order**.

The result format is in the following example.

Example 1:

Input:

Activity table:

user_id	session_id	activity_date	activity_type
1	1	2019-07-20	open_session
1	1	2019-07-20	scroll_down
1	1	2019-07-20	end_session
2	4	2019-07-20	open_session
2	4	2019-07-21	send_message
2	4	2019-07-21	end_session
3	2	2019-07-21	open_session
3	2	2019-07-21	send_message
3	2	2019-07-21	end_session
4	3	2019-06-25	open_session
4	3	2019-06-25	end_session

Output:

day	active_users
2019-07-20	2
2019-07-21	2

```
select activity_date as day,count(distinct user_id) as active_users
from Activity
where activity_date>'2019-06-27' AND activity_date<='2019-07-27'
group by activity_date;
```

GROUP BY & HAVING

Write a solution to find all the classes that have at least five students.

Input:

Courses table:

student	class
A	Math
B	English
C	Math
D	Biology
E	Math
F	Computer
G	Math
H	Math
I	Math

Output:

class
Math

Explanation:

- Math has 6 students, so we include it.
- English has 1 student, so we do not include it.
- Biology has 1 student, so we do not include it.
- Computer has 1 student, so we do not include it.

```sql
select class from Courses
group by class
having count(*)>=5;
```

Write a solution that will, for each user, return the number of followers. Return the result table ordered by user_id in ascending order.

Input:

Followers table:

user_id	follower_id
0	1
1	0
2	0
2	1

Output:

user_id	followers_count
0	1
1	1
2	2

Explanation:

The followers of 0 are {1}
The followers of 1 are {0}
The followers of 2 are {0,1}

```sql
select user_id,count(*) as followers_count from Followers
group by user_id
order by user_id ASC;
```

A single number is a number that appeared only once in the MyNumbers table. Find the largest single number. If there is no single number, report null.

Input:

MyNumbers table:

num
8
8
3
3
1
4
5
6

Output:

num
6

Explanation: The single numbers are 1, 4, 5, and 6. Since 6 is the largest single number, we return it.

```sql
select max(num) as num from
(select num from MyNumbers group by num having count(*)=1)
as unique_numbers;
```

Write a solution to report the customer ids from the Customer table that bought all the products in the Product table.

Input:

Customer table:

customer_id	product_key
1	5
2	6
3	5
3	6
1	6

Product table:

product_key
5
6

Output:

customer_id
1
3

Explanation:

The customers who bought all the products (5 and 6) are customers with IDs 1 and 3.

To find all the customers who bought all the products from the product table , since the customer table it may have duplicate values so
Let's group the customers by customer_id and filter with having condition → having count(distinct product_id) = count od products in products table

```sql
select customer_id from Customer
group by customer_id
having count(distinct product_key)=(select count(*) from Product);
```

Write a solution to select the product id, year, quantity, and price for the first year of every product sold.

Input:

Sales table:

sale_id	product_id	year	quantity	price
1	100	2008	10	5000
2	100	2009	12	5000
7	200	2011	15	9000

Product table:

product_id	product_name
100	Nokia
200	Apple
300	Samsung

Output:

product_id	first_year	quantity	price
100	2008	10	5000
200	2011	15	9000

```sql
select product_id,year as first_year,quantity,price
from Sales
where (product_id,year) in (select product_id,min(year) from Sales group by product_id);
```

SELF JOIN WITH GROUP BY

Write an SQL query to report the ids and the names of all **managers**, the number of employees who report **directly** to them, and the average age of the reports rounded to the nearest integer.

Return the result table ordered by `employee_id`.

The query result format is in the following example.

Example 1:

Input:

Employees table:

employee_id	name	reports_to	age
9	Hercy	null	43
6	Alice	9	41
4	Bob	9	36
2	Winston	null	37

Output:

employee_id	name	reports_count	average_age
9	Hercy	2	39

Explanation: Hercy has 2 people report directly to him, Alice and Bob. Their average age is (41+36)/2 = 38.5, which is 39 after rounding it to the nearest integer.

```sql
select a.employee_id,a.name,count(b.employee_id) as reports_count,round(avg(b.age)) as
average_age
from Employees a
join
Employees b
on a.employee_id=b.reports_to
group by employee_id
order by employee_id;
```

UNION

Employees can belong to multiple departments. When the employee joins other departments, they need to decide which department is their primary department. Note that when an employee belongs to only one department, their primary column is `'N'`.

Write a solution to report all the employees with their primary department. For employees who belong to one department, report their only department.

Return the result table in **any order**.

The result format is in the following example.

Input:

Employee table:

employee_id	department_id	primary_flag
1	1	N
2	1	Y
2	2	N
3	3	N
4	2	N
4	3	Y
4	4	N

Output:

employee_id	department_id
1	1
2	1
3	3
4	3

Explanation:

- The Primary department for employee 1 is 1.
- The Primary department for employee 2 is 1.
- The Primary department for employee 3 is 3.
- The Primary department for employee 4 is 3.

```sql
select employee_id,department_id from Employee
where primary_flag='Y'

UNION

select employee_id,department_id from Employee
group by employee_id
having count(employee_id)=1;
```

CASE WHEN THEN ELSE END AS

Report for every three line segments whether they can form a triangle.

Return the result table in **any order**.

The result format is in the following example.

Input:

Triangle table:

x	y	z
13	15	30
10	20	15

Output:

x	y	z	triangle
13	15	30	No
10	20	15	Yes

```sql
select x,y,z,
case when x+y>z and y+z>x and z+x>y then 'Yes'
else 'No'
end as triangle
from Triangle;
```

SUB-QUERY - EMPLOYEES WHOSE MANAGER LEFT COMPANY

Find the IDs of the employees whose salary is strictly less than `$30000` and whose manager left the company. When a manager leaves the company, their information is deleted from the `Employees` table, but the reports still have their `manager_id` set to the manager that left.

Return the result table ordered by `employee_id`.

The result format is in the following example.

Input:

Employees table:

employee_id	name	manager_id	salary
3	Mila	9	60301
12	Antonella	null	31000
13	Emery	null	67084
1	Kalel	11	21241
9	Mikaela	null	50937
11	Joziah	6	28485

Output:

employee_id
11

Explanation:
The employees with a salary less than $30000 are 1 (Kalel) and 11 (Joziah).
Kalel's manager is employee 11, who is still in the company (Joziah).
Joziah's manager is employee 6, who left the company because there is no row for employee 6 as it was deleted.

```sql
select employee_id from Employees
where salary<30000
and
manager_id is not null
and
manager_id not in (select employee_id from Employees)
order by employee_id;
```

184. Department Highest Salary

Input:

Employee table:

id	name	salary	departmentId
1	Joe	70000	1
2	Jim	90000	1
3	Henry	80000	2
4	Sam	60000	2
5	Max	90000	1

Department table:

id	name
1	IT
2	Sales

Output:

Department	Employee	Salary
IT	Jim	90000
Sales	Henry	80000
IT	Max	90000

Explanation: Max and Jim both have the highest salary in the IT department and Henry has the highest salary in the Sales department.

```
select d.name as Department,e.name as Employee,e.salary
from Employee e,Department d
where e.departmentId=d.id and
(e.departmentId,e.salary) in (select departmentId,max(salary) from Employee group by
departmentId);
```

182. Duplicate Emails

```sql
select email as Email from Person
group by email
having count(*)>1;
```

TOP 50 SQL QUESTIONS - GFG

SECOND HIGHEST SALARY

```sql
select max(salary) as SecondHighestSalary from Employee
where salary not in (select max(salary) from Employee);



select max(salary) as SecondHighestSalary from Employee
where salary < (select max(salary) from Employee);



select IFNULL(NULL,
(select distinct(salary) from Employee
order by salary DESC
limit 1 offset 1)
)  as SecondHighestSalary;
```

MAX salary in each department

```sql
select max(sal),deptno
from emp
group by deptno
```

MIN salary in each department

```
1  select min(sal),deptno
2  from emp
3  group by deptno
```

COUNT(*) from each department

```
1  select count(*) ,deptno
2  from emp
3  group by deptno;
```

COUNT(*)	DEPTNO
1	30
2	10
4	20

DISPLAY ALTERNATE RECORDS IN THE TABLE

```
1
2  select * from
3    (select empno, ename, sal, rownum rn
4      from emp
5     order by rn)
6    where  mod (rn, 2) != 0;
```

EMPNO	ENAME	SAL	RN
7839	KING	5000	1
7782	CLARK	2450	3
7788	SCOTT	3000	5
7369	SMITH	800	7

DISPLAY alternate records in the table

```
1
2  select * from
3    (select empno, ename, sal, rownum rn
4      from emp
5     order by rn)
6    where  mod (rn, 2) = 0;
```

EMPNO	ENAME	SAL	RN
7698	BLAKE	2850	2
7566	JONES	2975	4
7902	FORD	3000	6

DISPLAY EMP NAME AND IT'S FREQUENCY

```
1  select ename,count(*)
2  from emp
3  group by ename;
```

ENAME	COUNT(*)
TRISH	3
RISHI	2
SMITH	1
FORD	1

DISPLAY ALL THE EMPLOYEES WHO ARE DUPLICATED AND ITS ACTUAL FREQUENCY

```
1  select ename,count(*)
2  from emp
3  group by ename
4  having count(*)>1;
```

ENAME	COUNT(*)
TRISH	3
RISHI	2

HAVING → Used to filter the columns after group by condition

NAME STARTS WITH M

```
1  select ename from emp
2  where ename like 'M%';
```

ENAME
MARTIN
MILLER

NAME ENDS WITH N

```
1  select ename from emp
2  where ename like '%N';
```

ENAME
ALLEN
MARTIN

NAME CONTAINS M

```
1  select ename from emp
2  where ename like '%M%';
```

ENAME
SMITH
MARTIN
ADAMS
JAMES

NAME NOT CONTAINS M

```
1  select ename from emp
2  where ename NOT like '%M%';
```

JONES
SCOTT
FORD
ALLEN

2.Display the names of employees whose name contains the (i)second letter as 'L'(ii)fourth Letter as 'M'

```
1  select ename from emp
2  where ename like '_L%';
```

ENAME
BLAKE
CLARK

```
1  select ename from emp
2  where ename like '___M%';
```

ENAME
ADAMS

3.Display the employee names and hire dates for the employees joined in the month of December

Select employees who joined in the month of DECEMBER using LIKE OPERATOR

```
1  select hiredate ,ename from emp
2  where hiredate like '%DEC%';
```

HIREDATE	ENAME
03-DEC-81	FORD
17-DEC-80	SMITH
03-DEC-81	JAMES

NAME CONTAINS EXACTLY TWO L's

For Q4: Display the names of employees whose name contains exactly 2 'L' s
Correction:-

```
SELECT ename from emp
WHERE ename LIKE '%L%L%'
        AND
       ename not LIKE '%L%L%L%'
```

5.Display the names of employees whose name starts with 'J' and ends with 'S'

```
1  select ename from emp
2  where ename like 'J%S';
```

ENAME
JONES
JAMES

- Display 2nd row of emp table
- Display 4th row of emp table

ROWNUM will NOT work for '>' and '=' Operator

TO DISPLAY FIRST FOUR RECORDS OF THE EMPLOYEE TABLE

```
1  select * from emp
2  where rownum<=4
3
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
7839	KING	PRESIDENT	-	17-NOV-81	5000	10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	10
7566	JONES	MANAGER	7839	02-APR-81	2975	20

Display Nth row from the table → Using ROWNUM

```
1  select * from emp
2  where rownum<=4
3  minus
4  select * from emp
5  where rownum<=3;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
7566	JONES	MANAGER	7839	02-APR-81	2975	20

WITHOUT USING MINUS

```
1  select * from
2  (select rownum r,ename ,sal from emp)
3  where r=4;
```

R	ENAME	SAL
4	JONES	2975

DISPLAY NTH ROW OF A TABLE

```
1  select * from (select rownum r,emp.* from emp)
2  where r=4;
```

R	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
4	7566	JONES	MANAGER	7839	02-APR-81	2975	20

Applying union between single column

```
1  select city from sample1;
2  union
3  select city from sample2;
4
5
```

CITY
hyderabad
london
texas
california

Duplicate values are eliminated from both the columns → Union

The columns used in all the select statements must have the following

- ✓the ***same number of columns***
- ✓Similar or ***compatible data types***
- ✓***same logical order***

Selecting more than one column in the select query

- ✓Whenever ***more than one column is specified in the select clause*** then the combination of all the columns ***considered that*** ***is if both the values in the row are same then only this considered as a duplicate value***

```
1  select city,country from sample1;
2  union
3  select city,country from sample2;
4
```

```
3 rows selected.
```

```
Invalid statement
```

CITY	COUNTRY
hyderabad	india
london	uk
texas	usa

Both columns should have same data type,same order

UNION ALL

✓UNION removes duplicate values while comparing where UNION ALL allows duplicate values while combining

```
1  select city from sample1
2  union all
3  select city from sample2
4
```

CITY
hyderabad
hyderabad
bhutan
hyderabad
london
texas

INNER JOIN

What it does

- ✓ Based on the equality condition data is retrieved from multiple tables
- ✓ We must have a common column in both the tables with the same data type
- ✓ Right table row is Joined with Left table row only if there is a matching for the left table row in Right table

```
1  select ename ,sal,dept.deptno,dname,loc
2  from emp,dept
3  where emp.deptno=dept.deptno;
```

ENAME	SAL	DEPTNO	DNAME	LOC
KING	5000	10	ACCOUNTING	NEW YORK
BLAKE	2850	30	SALES	CHICAGO
CLARK	2450	10	ACCOUNTING	NEW YORK
JONES	2975	20	RESEARCH	DALLAS
SCOTT	3000	20	RESEARCH	DALLAS
FORD	3000	20	RESEARCH	DALLAS
SMITH	800	20	RESEARCH	DALLAS

Example Queries

- ✓Display employees who are working in Location Chicago from emp and dept. table
- ✓Display the department name and total salaries from each department

```
1  select ename,sal,d.deptno,dname,loc
2  from emp e,dept d
3  where e.deptno=d.deptno and  LOC='CHICAGO'
```

ENAME	SAL	DEPTNO	DNAME	LOC
BLAKE	2850	30	SALES	CHICAGO

```
1  select dname,sum(sal) from emp e, dept d
2  where e.deptno=d.deptno
3  group by dname;
```

DNAME	SUM(SAL)
RESEARCH	9775
SALES	2850
ACCOUNTING	7450

SELF JOIN

What it does

- ✓ Joining a Table with itself is called self join
- ✓ Comparing values of a values with the values of same column itself or different column values of the same table

Example Queries

- ✓Display employee details who are getting more salary than their manager salary
- ✓Display the employee details who joined before their manager

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7788	SCOTT	ANALYST	7566	19-APR-87	3000	-	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20
7369	SMITH	CLERK	7902	17-DEC-80	800	-	20

EMPLOYEE NAME AND HIS MANAGER

```
1  select e1.ename "employes", e2.ename "manager"
2  from emp e1,emp e2
3  where e2.empno=e1.mgr
```

employes	manager
BLAKE	KING
CLARK	KING
JONES	KING
SCOTT	JONES
FORD	JONES
SMITH	FORD

DETAILS OF EMPLOYEE WHO IS GETTING MORE SALARY THAN HIS MANAGER

```
1  select e1.ename "employes", e2.ename "manager"
2  from emp e1,emp e2
3  where  e1.mgr=e2.empno and e1.sal>e2.sal ;
```

employes	manager
SCOTT	JONES
FORD	JONES

DETAILS OF THE EMPLOYEE WHO JOINED BEFORE HIS MANAGER

```
1  select e1.ename "employes", e2.ename "manager",e2.sal,e2.hiredate
2  from emp e1,emp e2
3  where  e1.mgr=e2.empno and e1.hiredate<e2.hiredate ;
```

employes	manager	SAL	HIREDATE
BLAKE	KING	5000	17-NOV-81
CLARK	KING	5000	17-NOV-81
JONES	KING	5000	17-NOV-81
SMITH	FORD	3000	03-DEC-81

Left Join

- ✓ All rows from left side table
- ✓ Matching value rows from right side table
- ✓ Null values in place of Non matching rows in other table

```
1  select rownum,Empno,Ename,Emp.Deptno,Dname,loc,job
2  from
3  Emp
4  LEFT JOIN
5  Dept
6  on Emp.Deptno=Dept.deptno ;
```

ROWNUM	EMPNO	ENAME	DEPTNO	DNAME	LOC	JOB
1	7839	KING	10	ACCOUNTING	NEW YORK	PRESIDENT
2	7782	CLARK	10	ACCOUNTING	NEW YORK	MANAGER
3	7934	MILLER	10	ACCOUNTING	NEW YORK	CLERK
4	7566	JONES	20	RESEARCH	DALLAS	MANAGER
5	7788	SCOTT	20	RESEARCH	DALLAS	ANALYST

```
1  select rownum,Empno,Ename,Emp.Deptno,Dname,loc,job
2  from
3  Emp
4  LEFT JOIN
5  Dept
6  on Emp.Deptno=Dept.deptno and dname='SALES';
```

ROWNUM	EMPNO	ENAME	DEPTNO	DNAME	LOC	JOB
1	7698	BLAKE	30	SALES	CHICAGO	MANAGER
2	7499	ALLEN	30	SALES	CHICAGO	SALESMAN
3	7521	WARD	30	SALES	CHICAGO	SALESMAN
4	7654	MARTIN	30	SALES	CHICAGO	SALESMAN
5	7844	TURNER	30	SALES	CHICAGO	SALESMAN

Right Join

- ✓ All rows from right side table are displayed
- ✓ Matching value rows from left side table
- ✓ Null values in place of Non matching rows in other table

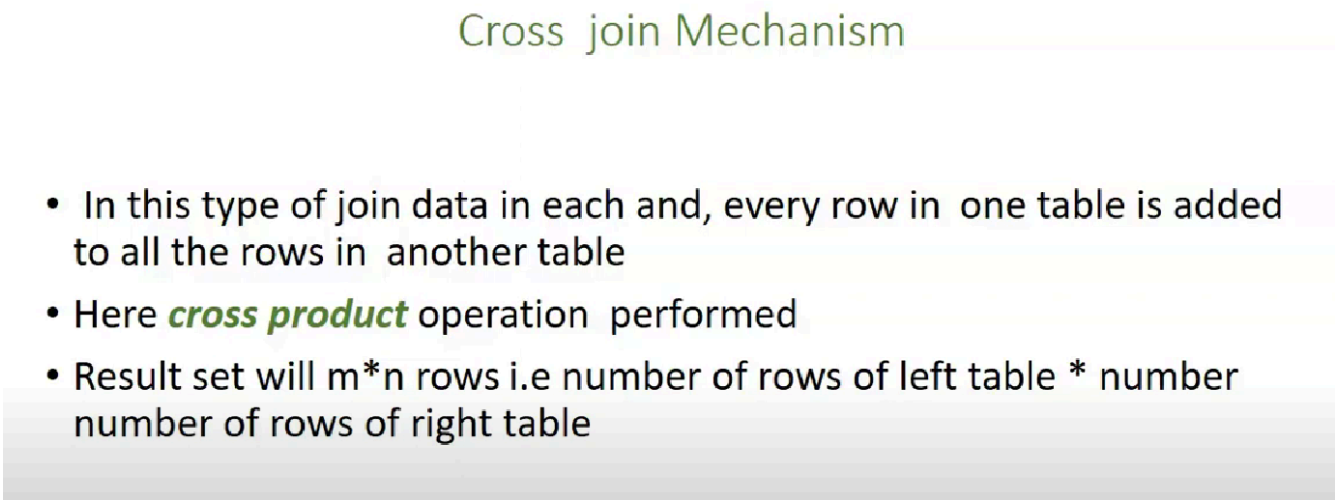
```sql
select ename,job,sal,loc,dname ,dept.deptno
from
emp
RIGHT JOIN
dept
on dept.deptno=emp.deptno;
```

ENAME	JOB	SAL	LOC	DNAME	DEPTNO
KING	PRESIDENT	5000	NEW YORK	ACCOUNTING	10
BLAKE	MANAGER	2850	CHICAGO	SALES	30
CLARK	MANAGER	2450	NEW YORK	ACCOUNTING	10
JONES	MANAGER	2975	DALLAS	RESEARCH	20
SCOTT	ANALYST	3000	DALLAS	RESEARCH	20
FORD	ANALYST	3000	DALLAS	RESEARCH	20
MARTIN	SALESMAN	1250	CHICAGO	SALES	30
TURNER	SALESMAN	1500	CHICAGO	SALES	30
ADAMS	CLERK	1100	DALLAS	RESEARCH	20
JAMES	CLERK	950	CHICAGO	SALES	30
MILLER	CLERK	1300	NEW YORK	ACCOUNTING	10
-	-	-	BOSTON	OPERATIONS	40

```
1  select ename,job,sal,loc,dname ,dept.deptno
2  from
3  emp
4  RIGHT JOIN
5  dept
6  on dept.deptno=emp.deptno and dept.deptno=20;
7
```

Full join Mechanism

- ✓A full join is viewed as a result of union operation of an inner join, left join and right join
- ✓It Returns all matched records i.e. where the join condition is satisfied on both tables
- ✓It Returns all rows from right side table (unmatched right side rows)
- ✓It Returns all rows from the left side table(unmatched left side rows)
- ✓This join returns null values in place of nonmatching tuples in another table



SQL is a standard language for storing, manipulating and retrieving data in databases.

SQL is a standard language for accessing and manipulating databases.

- **SQL stands for Structured Query Language**
- **SQL lets you access and manipulate databases**

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

Using SQL in Your Web Site

To build a web site that shows data from a database, you will need:

- An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
- To use a server-side scripting language, like PHP or ASP
- To use SQL to get the data you want
- To use HTML / CSS to style the page

RDBMS

RDBMS stands for Relational Database Management System.

The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

Every table is broken up into smaller entities called fields.

The fields in the Customers table consist of CustomerID, CustomerName, ContactName, Address, City, PostalCode and Country.

A field is a column in a table that is designed to maintain specific information about every record in the table.

A record, also called a row, is each individual entry that exists in a table.

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

A database most often contains one or more tables.

Most of the actions you need to perform on a database are done with SQL statements.