**Interview Questions**

**SQL**

1. **Types of commands and their examples.**

* DDL (Data Definition Language): Used for defining and managing the structure of the database.

Example: CREATE TABLE, ALTER TABLE, DROP TABLE

* DML (Data Manipulation Language): Used for managing and manipulating data within the database.

Example: SELECT, INSERT, UPDATE, DELETE

* DCL (Data Control Language): Used for controlling access to data within the database.

Example: GRANT, REVOKE

1. **What is Normalization and denormalization?**

* **Normalization** is the process of organizing data in a database to reduce data redundancy and improve data integrity. It involves breaking down a large table into smaller, related tables and defining relationships between them.
* **Denormalization**, on the other hand, is the process of intentionally adding redundancy to a database to improve query performance. It involves combining tables or adding duplicate data to reduce the need for complex joins.

1. **Explain 1NF, 2NF, 3NF.**

* 1NF (First Normal Form): Ensures that each column in a table contains only atomic (indivisible) values and that there are no repeating groups or arrays.
* 2NF (Second Normal Form): Extends 1NF and ensures that non-key attributes are fully functionally dependent on the primary key.
* 3NF (Third Normal Form): Extends 2NF and ensures that non-key attributes are transitively dependent on the primary key.

1. **Share use case where you had to do denormalization in database..**

Denormalization can be useful in scenarios where read-heavy operations, such as reporting or analytics, require faster query performance. For example, in an e-commerce system, you might denormalize product data into a single table to simplify and speed up product search queries.

1. **What is primary key and foreign key?**

**Primary Key:** A primary key is a column or a set of columns in a table that uniquely identifies eachrow in the table. It enforces data integrity and ensures that there are no duplicate rows.

**Foreign Key:** A foreign key is a column or a set of columns in one table that refers to the primary key of another table. It establishes a relationship between tables and enforces referential integrity.

1. **What is alternate and candidate key?**

**Alternate Key:** An alternate key is a candidate key that is not selected as the primary key. It could have been chosen as the primary key but wasn't.

**Candidate Key:** A candidate key is a set of one or more columns that can uniquely identify each row in a table. From the candidate keys, one is selected as the primary key.

1. **What are window functions?**

Window functions are SQL functions that perform a calculation across a set of table rows related to the current row. They are often used for tasks such as ranking, aggregation, and moving averages.

1. **Explain Ranking Functions?**

Ranking functions are window functions that assign a rank to each row based on the values in one or more columns. Common ranking functions include

**RANK():**

SELECT employee\_id, salary, RANK() OVER (ORDER BY salary DESC) AS salary\_rank

FROM employees;

**Output:**

employee\_id | salary | salary\_rank

---------------------------------

1 | 55000 | 1

2 | 55000 | 1

3 | 52000 | 3

4 | 50000 | 4

5 | 48000 | 5

**DENSE\_RANK():**

SELECT employee\_id, salary, DENSE\_RANK() OVER (ORDER BY salary DESC) AS salary\_rank

FROM employees;

Output:

employee\_id | salary | salary\_rank

---------------------------------

1 | 55000 | 1

2 | 55000 | 1

3 | 52000 | 2

4 | 50000 | 3

5 | 48000 | 4

**ROW\_NUMBER():**

SELECT employee\_id, salary, ROW\_NUMBER() OVER (ORDER BY salary DESC) AS salary\_rank

FROM employees;

**Output:**

employee\_id | salary | salary\_rank

---------------------------------

1 | 55000 | 1

2 | 55000 | 2

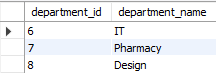
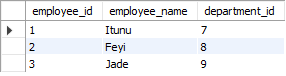
3 | 52000 | 3

4 | 50000 | 4

5 | 48000 | 5

1. **Types of Joins?**

**Employees Departments**

** **

**INNER JOIN:** Returns **only** the rows with matching values in both tables.

SELECT

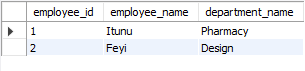
employees.employee\_id,

employee\_name,

department\_name

FROM employees

INNER JOIN departments ON employees.department\_id = departments.department\_id;



**LEFT JOIN (or LEFT OUTER JOIN):** Returns all rows from the left table and the matched rows from the right table.

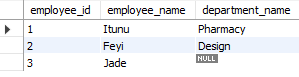
SELECT employees.employee\_id,

employee\_name,

department\_name

FROM employees

LEFT JOIN departments ON employees.department\_id = departments.department\_id;

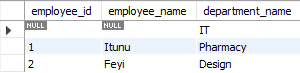


**RIGHT JOIN (or RIGHT OUTER JOIN**): Returns all rows from the right table and the matched rows from the left table.

SELECT employees.employee\_id, employee\_name, department\_name

FROM employees

RIGHT JOIN departments ON employees.department\_id = departments.department\_id;



**FULL JOIN (or FULL OUTER JOIN):** Returns all rows when there is a match in either the left or the right table.

1. **Use case when self join is required.**

A self-join is used when you need to join a table with itself, typically to find relationships between rows within the same table. For example, in an employee table, you might use a self-join to find managers and their subordinates.

1. **What is subquery?**

A subquery is a query that is nested inside another SQL query. It can be used to retrieve data that will be used in the main query's condition, filtering, or comparison. E.g. Retrieve the names of employees in the "employees" table who earn a salary greater than the average salary. **Bit in bold is the subquery**

SELECT employee\_name

FROM employees

WHERE salary > **(SELECT AVG(salary) FROM employees);**

1. **What is correlated subquery?**

A correlated subquery is a subquery that refers to a column from the outer query. It is executed for each row processed by the outer query and depends on values from the outer query. E.g. Retrieve the names of employees in the "employees" table who earn more than their respective department's average salary. **Bit in bold is the correlated**

SELECT employee\_name

FROM employees e

WHERE salary > (SELECT AVG(salary) FROM employees **WHERE department\_id = e.department\_id);**

1. **What is CTE (Common Table Expression)?**

A CTE is a named temporary result set that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement. CTEs must always start with the keyword WITH. CTEs are often used to simplify complex queries and make them more readable.

E.g. Use a CTE to find the average salary for each department and then select employees earning more than their department's average.

WITH DepartmentAvgSalaries AS (

SELECT department\_id, AVG(salary) AS avg\_salary

FROM employees

GROUP BY department\_id

)

SELECT e.employee\_name

FROM employees e

JOIN DepartmentAvgSalaries d ON e.department\_id = d.department\_id

WHERE e.salary > d.avg\_salary;

1. **What is derived table?**

A derived table is a subquery that appears in the FROM clause of a SELECT statement. It's used to create a temporary table for the main query.

Example: Use a derived table to find the total number of employees in each department.

SELECT department\_id, department\_name, employee\_count

FROM (

SELECT department\_id, COUNT(\*) AS employee\_count

FROM employees

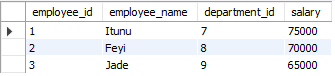
GROUP BY department\_id

) AS DepartmentCounts

JOIN departments ON DepartmentCounts.department\_id = departments.department\_id;

1. **Find third highest employee based on salary?**

I updated the employees table with employees salaries.



SELECT DISTINCT salary

FROM employees

ORDER BY salary DESC

LIMIT 2,1;



1. **Find third highest employee based on salary per department?**

To find the third-highest employee based on salary within each department, you can use a window function like DENSE\_RANK() partitioned by department and then filter for rows where the rank is 3.

1. **How to find duplicate values in a single column?**

SELECT column\_name, COUNT(column\_name)

FROM table\_name

GROUP BY column\_name

HAVING COUNT(column\_name) > 1;

1. **How to find duplicate values in a multiple column?**

To find duplicate values across multiple columns, you can concatenate the values of those columns and then apply the same technique as for a single column.

1. **What are ACID properties?**

These properties ensure that database transactions are reliable and maintain data integrity:

**Atomicity**: A transaction is treated as a single, indivisible unit of work.

**Consistency**: A transaction brings the database from one consistent state to another.

**Isolation**: Transactions are executed in isolation from each other to prevent interference.

**Durability**: Once a transaction is committed, its effects are permanent and survive system failures.

1. **Diff between union and union all**

**UNION** - returns distinct rows by combining the results of two or more SELECT queries, removing duplicate rows.

**UNION ALL** - returns all rows by combining the results of two or more SELECT queries, including duplicates.

1. **Diff between primary key and unique key**

**Primary Key**: A primary key is a column or a set of columns in a database table that uniquely identifies each row. It implies that the column(s) cannot contain NULL values. A table can have only one primary key.

**Unique Key:** A unique key is also a column or a set of columns that enforces uniqueness among values in those columns. However, unlike a primary key, a unique key allows NULL values in the column(s). A table can have multiple unique keys.

1. **Diff between truncate and delete**

**Truncate**: Truncate is a Data Definition Language (DDL) command that quickly and efficiently **removes all rows from a table**, effectively resetting the table to an empty state. It is a minimally logged operation and cannot be rolled back. Truncate does not generate individual delete statements for each row.

**Delete**: Delete is a Data Manipulation Language (DML) command used to **remove specific rows** from a table based on a condition. It allows for more selective removal of data and generates individual delete statements for each row. Deletes can be rolled back if issued within a transaction.

1. **SQL query execution order.**

In SQL, the sequence of execution of a query is as follows:

**FROM**: The tables specified in the FROM clause are identified and combined, forming a virtual table.

**WHERE**: Rows from the virtual table are filtered based on the conditions specified in the WHERE clause.

**GROUP** **BY**: Rows from the filtered result can be grouped into sets based on the GROUP BY clause.

**HAVING**: The groups can be further filtered using the HAVING clause.

**SELECT**: Columns specified in the SELECT clause are computed based on the filtered and grouped rows.

**ORDER** **BY**: The result set can be sorted using the ORDER BY clause.

**LIMIT/OFFSET**: If specified, LIMIT and OFFSET are applied to restrict the number of rows returned.

1. **What are indexes? Types of Indexes and their differences.**

Indexes in SQL are database objects that improve the speed of data retrieval operations on database tables. They work like a data structure, such as a B-tree, that allows for faster lookup of rows based on the indexed columns. Indexes can significantly enhance query performance.

Common types of indexes include:

* **Primary Key Index**: Automatically created when a primary key constraint is defined. Ensures uniqueness and fast lookups.
* **Unique Index**: Enforces uniqueness but allows NULL values.
* **Clustered Index**: Determines the physical order of data rows in a table. There can be only one clustered index per table.
* **Non-Clustered Index**: Provides a separate structure for fast data retrieval, including a pointer to the actual data row.
* **Composite Index**: An index that includes multiple columns, useful for queries involving multiple columns.
* **Full-Text Index**: Used for efficient text-based searching.
* **Bitmap Index**: Used in data warehousing for columns with a limited number of distinct values.

Differences between index types include their purpose, performance characteristics, and use cases.