SQL NOTES & Interview(The below link has a huge collection of SQl Q&A):
https://drive.google.com/file/d/1uph1pC2IONe-vgVCesXAglyAYyBRbNTg/view?usp=sharing

Retrieve all records from a table.

```
SELECT * FROM table_name;
```

Count the number of records in a table.

```
SELECT COUNT(*) FROM table_name;
```

Select distinct values from a column.

```
SELECT DISTINCT column_name FROM table_name;
```

Filter records based on a specific condition.

```
SELECT * FROM table_name WHERE condition;
```

Sort records in ascending order.

```
SELECT * FROM table_name ORDER BY column_name ASC;
```

Sort records in descending order.

```
SELECT * FROM table_name ORDER BY column_name DESC;
```

Select records from multiple tables using JOIN.

```
SELECT * FROM table1 JOIN table2 ON table1.column = table2.column;
```

Perform inner JOIN to retrieve common records.

```
SELECT * FROM table1 INNER JOIN table2 ON table1.column =
table2.column;
```

Perform left JOIN to retrieve records from the left table.

```
SELECT * FROM table1 LEFT JOIN table2 ON table1.column =
table2.column;
```

Perform right JOIN to retrieve records from the right table.

```
SELECT * FROM table1 RIGHT JOIN table2 ON table1.column =
table2.column;
```

Calculate the sum of values in a column.

```
SELECT SUM(column_name) FROM table_name;
```

Find the average value of a column.

```
SELECT AVG(column_name) FROM table_name;
```

Retrieve the maximum value from a column.

```
SELECT MAX(column_name) FROM table_name;
```

Retrieve the minimum value from a column.

```
SELECT MIN(column_name) FROM table_name;
```

Group records based on a specific column.

```
SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name;
```

Filter records using the HAVING clause.

```
SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name
HAVING COUNT(*) > 1;
```

Use the LIKE operator to search for patterns in text.

```
SELECT * FROM table_name WHERE column_name LIKE 'pattern%';
```

Use the BETWEEN operator to filter records within a range.

```
SELECT * FROM table_name WHERE column_name BETWEEN value1 AND
value2;
```

Use the IN operator to filter records from a list of values.

```
SELECT * FROM table_name WHERE column_name IN (value1, value2,
value3);
```

Use the EXISTS operator for subquery comparisons.

```
SELECT column_name FROM table_name WHERE EXISTS (SELECT * FROM
another_table WHERE another_table.column = table_name.column);
```

Count the number of records in a group using GROUP BY.

```
SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name;
```

Retrieve records that match a pattern using regular expressions.

```
SELECT * FROM table_name WHERE column_name ~ 'pattern';
```

Use the UNION operator to combine results from multiple SELECT statements.

```
SELECT column_name FROM table1 UNION SELECT column_name FROM table2;
```

Update records in a table.

```
UPDATE table_name SET column_name = new_value WHERE condition;
```

Delete records from a table.

```
DELETE FROM table_name WHERE condition;
```

Insert new records into a table.

```
INSERT INTO table_name (column1, column2, column3) VALUES (value1,
value2, value3);
```

Create a new table.

```
CREATE TABLE new_table (column1 datatype, column2 datatype, column3
datatype);
```

Modify table structure using ALTER TABLE.

```
ALTER TABLE table_name ADD column_name datatype;
```

Retrieve records from multiple tables using subqueries.

```
SELECT column_name FROM table1 WHERE column_name IN (SELECT
column_name FROM table2);
```

Use aggregate functions (SUM, AVG, COUNT, MAX, MIN) for data analysis.

```
SELECT SUM(column_name), AVG(column_name), COUNT(column_name),
MAX(column_name), MIN(column_name) FROM table_name;
```

Qus: find the nth highest salary in table using sql wuery

Ans: To find the nth highest salary in a SQL table, you can use the following SQL query. Replace `n` with the desired position of the highest salary you want to retrieve:

```
SELECT DISTINCT Salary

FROM Employee

ORDER BY Salary DESC

LIMIT 1 OFFSET n-1;
```

In this query, `Employee` is the name of the table, and `Salary` is the name of the column containing the salary information. The `ORDER BY Salary DESC` sorts the salaries in descending order (highest to lowest), and the `LIMIT 1 OFFSET n-1` part retrieves the nth highest salary. Make sure to replace `Employee` and `Salary` with the actual table and column names in your database.

1. Write An SQL Query To Print Details Of The EMPLOYEEs Whose FIRST_NAME Ends With ‘A’.

```
Select * from EMPLOYEE where FIRST_NAME like '%a';
```

2. Write An SQL Query To Print Details Of The EMPLOYEEs Whose FIRST_NAME Ends With ‘H’ And Contains Six Alphabets.

```
Select * from EMPLOYEE where FIRST_NAME like '_____h';
```

3. Write An SQL Query To Print Details Of The EMPLOYEEs Whose SALARY Lies Between 100000 And 500000.

```
Select * from EMPLOYEE where SALARY between 100000 and 500000;
```

4. Write An SQL Query To Print Details Of The EMPLOYEEs Who Have Joined In Feb'2014.

```
Select * from EMPLOYEE where year(JOINING_DATE) = 2014 and
month(JOINING_DATE) = 2
```

5. Write An SQL Query To Fetch EMPLOYEE Names With Salaries >= 50000 And <= 100000.

```
Select FIRST_NAME, SALARY from EMPLOYEE where SALARY between 50000 And
100000.
```

6. Write An SQL Query To Fetch The No. Of EMPLOYEEs For Each Department In The Descending Order.

```
SELECT DEPARTMENT, count(EMPLOYEE_ID) No_Of_EMPLOYEEs
FROM EMPLOYEE
GROUP BY DEPARTMENT
ORDER BY No_Of_EMPLOYEEs DESC;
```

7. Write An SQL Query To Print Details Of The EMPLOYEEs Who Are Also Managers.

```
SELECT EMPLOYEE.EMPLOYEE_ID,EMPLOYEE.DEPARTMENT,EMPLOYEE.FIRST_NAME
,Title.EMPLOYEE_TITLE Title from EMPLOYEE Inner Join Title
```

```
on EMPLOYEE.EMPLOYEE_ID=Title.EMPLOYEE_Ref_ID  where
Title.EMPLOYEE_TITLE='Manager' order by EMPLOYEE_ID
```

8. Write An SQL Query To Fetch Duplicate Records Having Matching Data In Some Fields Of A Table.

```
SELECT EMPLOYEE_TITLE, AFFECTED_FROM, COUNT(*)
FROM Title
GROUP BY EMPLOYEE_TITLE, AFFECTED_FROM
HAVING COUNT(*) > 1;
```

9. Write An SQL Query To Show Only Odd Rows From A Table.

```
SELECT * FROM Emplyee WHERE MOD (EMPLOYEE_ID, 2) <> 0;
```

10. Write An SQL Query To Show Only Even Rows From A Table.

```
SELECT * FROM EMPLOYEE WHERE MOD (EMPLOYEE_ID, 2) = 0;
```

.

QUS: Drop vs TRuncate vs Delete:

In SQL, "DROP," "TRUNCATE," and "DELETE" are three different commands used for different purposes related to managing data and database objects. Here's a brief explanation of each:

1. DROP:
    - Purpose: The DROP command is used to delete database objects, such as tables, indexes, or views. When you drop a table, all the data within the table is permanently removed, and the table itself no longer exists in the database.
    - Effect: Irreversibly deletes the entire table structure along with its data. It cannot be rolled back.
2. Example:
   I

```
DROP TABLE employees;
```

TRUNCATE:

- Purpose: The TRUNCATE command is used to quickly remove all rows from a table without logging individual row deletions. It's a faster operation compared to DELETE because it doesn't generate as much transaction log data.
- Effect: Removes all rows from a table, but the table structure remains intact.

Example:

```
TRUNCATE TABLE sales_data;
```

DELETE:

- Purpose: The DELETE command is used to remove specific rows from a table based on a condition or criteria. It is more flexible than TRUNCATE because you can specify which rows to delete.
- Effect: Removes selected rows from a table, and you can use a WHERE clause to specify the conditions for deletion. It generates transaction log entries for each deleted row.

Example:

```
DELETE FROM customers WHERE last_purchase_date < '2023-01-01';
```

In summary, the main differences are:

- DROP deletes the entire table structure and data, and it's used for removing entire database objects.
- TRUNCATE removes all rows from a table but leaves the table structure intact.
- DELETE selectively removes rows based on specified conditions and generates individual transaction log entries for each deleted row.

SCENARIO BASED REAL TIME

Question: Explain the different types of joins in SQL and provide an example of when you would use each type.
Answer: Joins like INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN are used to combine rows from different tables. For example, an INNER JOIN is used when you want to retrieve rows that have matching values in both tables.

```
SELECT employees.employee_id, employees.employee_name,
departments.department_name
FROM employees
INNER JOIN departments ON employees.department_id = departments.department_id;
```

Question: What is a subquery, and how would you use it to retrieve specific information from a database?
Answer: A subquery is a query nested inside another query. It can be used to retrieve data that will be used by the main query for further processing.

```
SELECT employee_name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

Question: Explain the concept of window functions in SQL and provide an example of when you would use them.
Answer: Window functions perform calculations across a specified range of rows related to the current row. They are used for tasks like ranking, aggregation, and cumulative totals.

```
SELECT department_name, employee_name, salary,
    RANK() OVER (PARTITION BY department_name ORDER BY salary DESC) AS salary_rank
FROM employees;
```

Question: How does indexing work in a database, and how can it be used to optimize query performance?
Answer: Indexing is a database feature that enhances the speed of data retrieval operations. It works by creating a data structure that provides faster access to rows in a table.

```
CREATE INDEX idx_employee_name ON employees(employee_name);
```

Question: Describe the steps you would take to migrate data from one database to another, ensuring data integrity.
Answer: Data migration involves exporting data from the source, transforming it as needed, and importing it into the destination while ensuring data consistency and accuracy.

```
-- Example of exporting data
SELECT * INTO new_database.dbo.new_table FROM old_database.dbo.old_table;
```

Question: What are stored procedures and triggers, and in what situations would you use them?
Answer: Stored procedures are precompiled SQL code that can be reused, and triggers are special types of stored procedures that are automatically executed in response to certain events.

```
-- Example of a stored procedure
CREATE PROCEDURE sp_GetEmployeeCount
AS
BEGIN
```

```
    SELECT COUNT(*) FROM employees;
END;
```

Question: Explain how NULL values are handled in SQL, and how you would retrieve or handle them in query results.
Answer: NULL is used to represent missing or undefined data. Functions like COALESCE or IS NULL can be used to handle NULL values in queries.

```
SELECT employee_name, COALESCE(salary, 0) AS adjusted_salary
FROM employees;
```

Question: Discuss the importance of transactions in a database and how you would ensure data consistency.
Answer: Transactions are sequences of one or more SQL statements that are executed as a single unit. They ensure data consistency by either committing all changes or rolling back if an error occurs.

```
BEGIN TRANSACTION;
UPDATE accounts SET balance = balance - 100 WHERE account_id = 123;
COMMIT;
```

Question: Explain the concept of data encryption in a database and why it is important for security.
Answer: Data encryption involves converting data into a code to prevent unauthorized access. It is crucial for protecting sensitive information stored in databases.

```
CREATE TABLE secured_data (
    sensitive_info VARBINARY(256) ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY =
CEK1, ENCRYPTION_TYPE = AEAD_AES_256_CBC_HMAC_SHA_256)
);
```

Question: What is dynamic SQL, and under what circumstances would you use it?

Answer: Dynamic SQL involves constructing and executing SQL statements at runtime. It is used when the structure of a query is not known until the application is running.

```
DECLARE @sql_query NVARCHAR(MAX);

SET @sql_query = 'SELECT * FROM ' + @table
```