SQL Queries for Interview -KirkYagami - Nikhil Sharma👨‍💻🕵️

1. What is the difference between renaming a column and giving an alias to it?

Answer:
Renaming a column in a database table permanently changes the column's name in the table's structure. This operation affects how the column is referenced in all subsequent SQL queries until the name is changed again.

Giving an alias to a column, on the other hand, provides a temporary name for the column during a specific SQL query execution. The alias only exists for the duration of that query and is useful for improving readability or shortening complex column names.

Examples:

```sql
-- Renaming a Column
ALTER TABLE departments
RENAME COLUMN department_name TO dept_name;

-- Giving an Alias to a Column
SELECT department_name AS dept_name
FROM departments
WHERE location = 'New York';
```

Real-World Scenario:
Consider a scenario where you have a `products` table with a long column name `product_description`. If you want to rename it to `description` for easier access, you would use the `ALTER TABLE` command. However, if you only want to shorten the name for a specific report or query, you can use an alias:

```sql
-- Renaming a column for permanent use
ALTER TABLE products
RENAME COLUMN product_description TO description;

-- Using alias for temporary reference
SELECT description AS short_desc
FROM products;
```

2. What is the `CASE()` function?

Answer:
The `CASE()` function in SQL is a way to implement conditional logic similar to an `if-then-else` statement in programming. It evaluates a list of conditions and returns a corresponding value when a condition is met. If none of the conditions are met, it can return a default value specified in the `ELSE` clause.

Syntax:

```
CASE
    WHEN condition_1 THEN value_1
    WHEN condition_2 THEN value_2
    ...
    ELSE value
END;
```

Example:

```
SELECT employee_id,
       employee_name,
       CASE
           WHEN salary < 30000 THEN 'Low'
           WHEN salary BETWEEN 30000 AND 60000 THEN 'Medium'
           WHEN salary > 60000 THEN 'High'
           ELSE 'Not Specified'
       END AS salary_category
FROM employees;
```

Real-World Scenario:
You might use the `CASE()` function to categorize employee salaries into different tiers. This categorization can then be used for reporting or analytics to understand salary distributions within an organization.

3. What is the difference between the `DELETE` and `TRUNCATE` statements?

Answer:

- **DELETE:** This is a DML (Data Manipulation Language) command that removes rows from a table based on a specified condition in the `WHERE` clause. It is reversible if transactions are used (with ROLLBACK), and triggers can be activated by the deletion.
- **TRUNCATE:** This is a DDL (Data Definition Language) command that removes all rows from a table without logging individual row deletions. It is irreversible, does not activate triggers, and resets any identity columns back to their seed value.

Example:

```
-- Using DELETE
DELETE FROM employees
WHERE employee_id = 5;

-- Using TRUNCATE
TRUNCATE TABLE employees;
```

Real-World Scenario:
If you want to remove specific employees based on criteria (e.g., termination), you'd use `DELETE`.

However, if you want to quickly clear out a staging table before reloading it with new data, you'd use `TRUNCATE` for efficiency.

4. What is the difference between the `DROP` and `TRUNCATE` statements?

Answer:

- **DROP:** This command permanently removes a table from the database, including all data, constraints, and associated privileges. It is irreversible and means the table structure will no longer exist.
- **TRUNCATE:** This command deletes all rows from a table while preserving the table structure and its constraints. It is faster than `DELETE` and does not log individual row deletions.

Example:

```sql
-- Using DROP
DROP TABLE employees;

-- Using TRUNCATE
TRUNCATE TABLE employees;
```

Real-World Scenario:
If you need to completely remove an outdated table from your database, use `DROP`. If you just want to clear out data from a temporary table that will be reused later, use `TRUNCATE`.

5. What is the difference between the `HAVING` and `WHERE` statements?

Answer:

- **WHERE:** This clause filters records before any groupings are made. It operates on individual rows and is used to specify conditions on columns in the original dataset.
- **HAVING:** This clause filters records after the aggregation has taken place. It is used in conjunction with `GROUP BY` to filter groups based on aggregate functions.

Example:

```sql
-- Using WHERE
SELECT employee_id, employee_name
FROM employees
WHERE salary > 50000;

-- Using HAVING
SELECT department_id, COUNT(*) AS employee_count
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;
```

Real-World Scenario:
If you want to find employees with a salary greater than $50,000, you would use `WHERE`. If you want to find departments with more than 5 employees, you would use `HAVING` to filter the result of an aggregation.

Here are well-explained answers with multiple ways of performing each action, along with real-world examples for the SQL interview questions you provided:

6. How do you add a record to a table?

Answer:
To add a record to a table in SQL, you primarily use the `INSERT INTO` statement. There are a couple of variations to consider:

1. **Basic INSERT:**
   You can insert values directly into a table without specifying the column names, provided that you provide values for all columns in the order they are defined in the table.

   ```
   INSERT INTO employees
   VALUES (1, 'John Doe', 'Software Engineer', 60000);
   ```

2. **INSERT with Column Names:**
   It's a good practice to specify the column names, especially when you do not want to provide values for all columns, or if the order of columns is different.

   ```
   INSERT INTO employees (employee_id, employee_name, job_title, salary)
   VALUES (2, 'Jane Smith', 'Data Analyst', 55000);
   ```

3. INSERT using SELECT:
You can also insert records by selecting from another table. This is useful for copying data.

```
INSERT INTO employees (employee_id, employee_name, job_title, salary)
SELECT employee_id, employee_name, job_title, salary
FROM temp_employees;
```

Real-World Example:
Suppose you have a table `employees` and you want to add a new employee record. You can use any of the above methods based on your requirements.

7. How to delete a record from a table?

Answer:
To delete a record from a table, you use the `DELETE` statement. Be cautious, as this action is irreversible.

1. **Basic DELETE:**
   Delete a specific record using a condition in the `WHERE` clause.

```sql
DELETE FROM employees
WHERE employee_id = 1;
```

2. **DELETE with Multiple Conditions:**
   You can specify multiple conditions using logical operators like `AND` or `OR`.

   ```sql
   DELETE FROM employees
   WHERE job_title = 'Intern' AND salary < 30000;
   ```

3. **DELETE all records:**
   If you want to delete all records from a table without deleting the table itself (be cautious), you can omit the `WHERE` clause.

   ```sql
   DELETE FROM employees;
   ```

Real-World Example:
If you need to remove an employee record based on their ID or job title, you would use the appropriate `DELETE` command as shown.

8. How to add a column to a table?

Answer:
To add a column to an existing table, you use the `ALTER TABLE` statement combined with `ADD`.

1. **Basic ADD:**
   To add a new column with a specified datatype.

   ```sql
   ALTER TABLE employees
   ADD date_of_birth DATE;
   ```

2. **Adding Multiple Columns:**
   You can also add multiple columns in a single command.

   ```sql
   ALTER TABLE employees
   ADD (
       phone_number VARCHAR(15),
       address VARCHAR(255)
   );
   ```

Real-World Example:
If you decide to include employees' phone numbers and addresses in your `employees` table, you can do so with the commands above.

9. How to rename a column of a table?

Answer:
To rename a column in a table, you use the `ALTER TABLE` statement combined with `RENAME COLUMN`.

1. **Basic RENAME:**
   The syntax may vary slightly depending on the SQL database system, but a general way is as follows:

   ```
   ALTER TABLE employees
   RENAME COLUMN job_title TO position;
   ```

2. **Renaming with SQL Server:**
   In SQL Server, you might need to use the `sp_rename` stored procedure.

   ```
   EXEC sp_rename 'employees.job_title', 'position', 'COLUMN';
   ```

Real-World Example:
If the column `job_title` is considered outdated and you want to rename it to `position`, you can use either of the commands depending on your database system.

10. How to delete a column from a table?

Answer:
To delete a column from a table, you use the `ALTER TABLE` statement combined with `DROP COLUMN`.

1. **Basic DROP:**
   To delete a single column.

   ```
   ALTER TABLE employees
   DROP COLUMN phone_number;
   ```

2. **Dropping Multiple Columns:**
   Some database systems allow dropping multiple columns in one command.

   ```
   ALTER TABLE employees
   DROP COLUMN phone_number, address;
   ```

Real-World Example:
If the `phone_number` column is no longer needed in the `employees` table, you would use the `DROP COLUMN` command to remove it.

11. How to select all even or all odd records in a table?

Answer:
To select all even or odd records from a table, you can use the modulo operator to check the remainder of a column divided by 2. The specific function or operator used may depend on the SQL database system.

1. **Using MOD Function (e.g., MySQL, PostgreSQL):**

   ```
   SELECT * FROM table_name
   WHERE MOD(ID_column, 2) = 0;  -- For even records
   ```

2. **Using Modulo Operator (%):**

   ```
   SELECT * FROM table_name
   WHERE ID_column % 2 = 0;  -- For even records

   SELECT * FROM table_name
   WHERE ID_column % 2 = 1;  -- For odd records
   ```

3. **Using CTE (Common Table Expressions):**
   If you have more complex conditions or need to perform further analysis on even or odd records.

   ```
   WITH EvenRecords AS (
       SELECT * FROM table_name
       WHERE ID_column % 2 = 0
   )
   SELECT * FROM EvenRecords;
   ```

Real-World Scenario:
Assuming you have a `students` table with an `ID` column, you might want to select students with even IDs for a specific operation or analysis.

```
-- Select all students with even IDs
SELECT * FROM students
WHERE ID % 2 = 0;
```

When no sequential id column is present

1. Using `ROW_NUMBER()`
   You can use the `ROW_NUMBER()` window function to assign a unique sequential integer to rows in your result set. After that, you can filter based on that row number.

```
WITH NumberedRecords AS (
    SELECT *, ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS RowNum
    FROM table_name
)
SELECT * FROM NumberedRecords
WHERE RowNum % 2 = 0;  -- For even records

-- For odd records:
SELECT * FROM NumberedRecords
WHERE RowNum % 2 = 1;  -- For odd records
```

Explanation:

- This query creates a Common Table Expression (CTE) that generates a new column (`RowNum`) representing the row number for each record based on an arbitrary order (since no specific order is defined). You can filter this based on the modulo operation.

2. Using `RANK()`

Similar to `ROW_NUMBER()`, you can use the `RANK()` function. This is particularly useful if you want to handle duplicate values differently, but the idea is the same.

Example Query:

```sql
WITH RankedRecords AS (
    SELECT *, RANK() OVER (ORDER BY (SELECT NULL)) AS RankNum
    FROM table_name
)
SELECT * FROM RankedRecords
WHERE RankNum % 2 = 0;   -- For even records

-- For odd records:
SELECT * FROM RankedRecords
WHERE RankNum % 2 = 1;   -- For odd records
```

12. How to prevent duplicate records when making a query?

Answer:
To prevent duplicate records in your query results, you can use the `DISTINCT` keyword in conjunction with the `SELECT` statement. Additionally, to prevent duplicates at the data level, you can create unique constraints.

1. **Using DISTINCT:**

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

2. **Using UNIQUE Constraint:**
   When creating or altering a table, you can enforce uniqueness on a column to prevent duplicates.

    ```sql
    CREATE TABLE employees (
        employee_id INT PRIMARY KEY,
        email VARCHAR(255) UNIQUE
    );
    ```

3. **Using GROUP BY:**
   You can also use `GROUP BY` to ensure unique values based on specific columns.

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

Real-World Scenario:
If you have a `sales` table and want to retrieve unique product IDs sold:

```
SELECT DISTINCT product_id
FROM sales;
```

13. How to insert many rows in a table?

Answer:
You can insert multiple rows into a table in a single `INSERT` statement using the `VALUES` clause.

1. **Basic INSERT for Multiple Rows:**

   ```
   INSERT INTO table_name (column1, column2, column3)
   VALUES
       (value1_1, value1_2, value1_3),
       (value2_1, value2_2, value2_3),
       (value3_1, value3_2, value3_3);
   ```

2. **Using INSERT with SELECT:**
   You can also insert multiple rows by selecting from another table.

   ```
   INSERT INTO table_name (column1, column2)
   SELECT column1, column2
   FROM another_table;
   ```

3. **Using a Stored Procedure or Bulk Insert (for larger datasets):**
   For larger datasets, consider using a bulk insert operation or stored procedure if supported by your DBMS.

   ```
   BULK INSERT table_name
   FROM 'path_to_your_data_file'
   WITH (FIELDTERMINATOR = ',', ROWTERMINATOR = '\n');
   ```

Real-World Scenario:
If you need to insert multiple employee records into an `employees` table, you could use:

```
INSERT INTO employees (employee_id, employee_name, job_title)
VALUES
    (1, 'John Doe', 'Software Engineer'),
    (2, 'Jane Smith', 'Data Analyst'),
    (3, 'Sam Brown', 'Product Manager');
```

14. How to find the nth highest value in a column of a table?

Answer:
To find the nth highest value in a column, you can use different methods depending on the SQL database system.

1. **Using LIMIT and OFFSET (PostgreSQL, MySQL):**

```sql
SELECT DISTINCT column_name
FROM table_name
ORDER BY column_name DESC
LIMIT 1 OFFSET n-1;  -- Replace n with the desired rank
```

2. **Using Common Table Expressions (CTEs):**
   You can also use CTEs to simplify finding the nth highest value.

```sql
WITH RankedValues AS (
    SELECT column_name,
           DENSE_RANK() OVER (ORDER BY column_name DESC) AS rank
    FROM table_name
)
SELECT column_name
FROM RankedValues
WHERE rank = n;  -- Replace n with the desired rank
```

3. **Using Subquery:**
   This method is widely applicable and works in most SQL databases.

```sql
SELECT MAX(column_name)
FROM table_name
WHERE column_name < (
    SELECT DISTINCT column_name
    FROM table_name
    ORDER BY column_name DESC
    LIMIT 1 OFFSET n-1
);
```

Real-World Scenario:
If you have a `salaries` table and want to find the 3rd highest salary:

```sql
SELECT DISTINCT salary
FROM salaries
ORDER BY salary DESC
LIMIT 1 OFFSET 2;  -- 3rd highest salary
```

15. How to find the values in a text column of a table that start with a certain letter?

Answer:
To find values in a text column that start with a specific letter, you can use the `LIKE` operator combined with wildcards.

1. **Using LIKE with Wildcard `%`:**
   To match any string that starts with a specific letter.

   ```
   SELECT * FROM table_name
   WHERE column_name LIKE 'A%';  -- Finds all values starting with 'A'
   ```

2. **Using LIKE with Wildcard `_`:**
   To match a specific pattern where you know the length.

   ```
   SELECT * FROM table_name
   WHERE column_name LIKE 'A__';  -- Finds all values starting with 'A' and 2 more
   characters
   ```

3. **Using LOWER Function (for case-insensitive searches):**
   If you want to ensure that the search is case-insensitive.

   ```
   SELECT * FROM table_name
   WHERE LOWER(column_name) LIKE 'a%';  -- Finds all values starting with 'a' or 'A'
   ```

Real-World Scenario:
Suppose you have a `customers` table and want to find all customers whose last names start with the letter "S":

```
SELECT * FROM customers
WHERE last_name LIKE 'S%';  -- Finds all last names starting with 'S'
```

16. How to find the last id in a table?

Using the `MAX()` function. Otherwise, in many SQL versions, we can use the following syntax:

```
SELECT id
FROM table_name
ORDER BY id DESC
LIMIT 1;
```

or in Microsoft SQL Server:

```
SELECT TOP 1 id FROM table_name ORDER BY id DESC
```

17. How to select random rows from a table?

Using the `RAND()` function in combination with `ORDER BY` and `LIMIT`. In some SQL flavors, such as PostgreSQL, it's called `RANDOM()`. For example, the following code will return five random rows from a table in MySQL:

```
SELECT * FROM table_name
ORDER BY RAND()
LIMIT 5;
```

---

1. Student Table DDL

Here's the SQL code to create the `Student` table with non-null values:

```
CREATE TABLE Student (
    STUDENT_ID INT PRIMARY KEY,
    FIRST_NAME VARCHAR(50) NOT NULL,
    LAST_NAME VARCHAR(50) NOT NULL,
    GPA DECIMAL(3, 2) NOT NULL CHECK (GPA >= 0 AND GPA <= 10),
    ENROLLMENT_DATE DATETIME NOT NULL,
    MAJOR VARCHAR(100) NOT NULL
);

INSERT INTO Student (STUDENT_ID, FIRST_NAME, LAST_NAME, GPA, ENROLLMENT_DATE, MAJOR) VALUES
(201, 'Kommuju', 'Sai kiran', 8.79, '2021-09-01 09:30:00', 'Computer Science'),
(202, 'S', 'Manya dharshini', 8.44, '2021-09-01 08:30:00', 'Mathematics'),
(203, 'Bhavana', 'alekhya saiDaripalli', 5.60, '2021-09-01 10:00:00', 'Biology'),
(204, 'Basava', 'Anil kumar', 9.20, '2021-09-01 12:45:00', 'Chemistry'),
(205, 'Bulasara', 'Babita', 7.85, '2021-09-01 08:30:00', 'Physics'),
(206, 'Jafarvali', 'Shaik', 9.56, '2021-09-01 09:24:00', 'History'),
(207, 'Ummar basha', 'Syed', 9.78, '2021-09-01 02:30:00', 'English'),
(208, 'R', 'Manoj', 7.00, '2021-09-01 06:30:00', 'Mathematics'),
(209, 'Panduri', 'Baby sri karthikeyi', 8.00, '2021-09-01 06:30:00', 'Computer Science'),
(210, 'Bandi', 'Praveen kumar', 8.50, '2021-09-01 06:30:00', 'Engineering'),
(211, 'Pasupuleti', 'Bhargav', 7.25, '2021-09-01 06:30:00', 'Mathematics'),
(212, 'Sangeetha', 'Chennamshetty', 9.10, '2021-09-01 06:30:00', 'Biology'),
(213, 'Vallepu', 'Vivek', 8.60, '2021-09-01 06:30:00', 'Chemistry'),
(214, 'M', 'Nagendran', 7.75, '2021-09-01 06:30:00', 'Physics'),
(215, 'Erige', 'Lokesh', 9.30, '2021-09-01 06:30:00', 'Computer Science'),
(216, 'Balu', 'Pavan teja', 8.45, '2021-09-01 06:30:00', 'Mathematics'),
(217, 'Mamilla', 'Bhavana', 8.90, '2021-09-01 06:30:00', 'Engineering'),
(218, 'Pasupuleti', 'Supriya', 7.80, '2021-09-01 06:30:00', 'Mathematics'),
(219, 'Pamarthi', 'Venkata ram', 8.00, '2021-09-01 06:30:00', 'Biology'),
(220, 'Baskaran', 'Rahul', 9.00, '2021-09-01 06:30:00', 'Chemistry'),
(221, 'Raghavanandu', 'Chenna', 8.30, '2021-09-01 06:30:00', 'Physics'),
(222, 'V', 'Sudharshan', 8.20, '2021-09-01 06:30:00', 'Computer Science'),
(223, 'Ganesh', 'Kotha', 8.40, '2021-09-01 06:30:00', 'Mathematics'),
(224, 'Dharmapuri', 'Chandu', 9.10, '2021-09-01 06:30:00', 'Engineering'),
(225, 'Palireddy', 'Swapna', 7.90, '2021-09-01 06:30:00', 'Biology'),
(226, 'Pathipati', 'Chiranjeevi', 8.60, '2021-09-01 06:30:00', 'Chemistry'),
(227, 'Birru', 'Hanish kumar', 8.55, '2021-09-01 06:30:00', 'Physics'),
(228, 'Balamurugan', 'Mukesh', 9.20, '2021-09-01 06:30:00', 'Computer Science'),
(229, 'B', 'Aravindan', 7.70, '2021-09-01 06:30:00', 'Mathematics'),
(230, 'Yeruva', 'Srilekha', 8.00, '2021-09-01 06:30:00', 'Engineering'),
(231, 'Krishna', 'chaithanyaMamidi', 8.40, '2021-09-01 06:30:00', 'Biology');
```

2. Program Table DDL

Here's the SQL code to create the `Program` table:

```
CREATE TABLE Program (
    STUDENT_REF_ID INT NOT NULL,
    PROGRAM_NAME VARCHAR(100) NOT NULL,
    PROGRAM_START_DATE DATETIME NOT NULL,
    PRIMARY KEY (STUDENT_REF_ID, PROGRAM_NAME),
    FOREIGN KEY (STUDENT_REF_ID) REFERENCES Student(STUDENT_ID)
);

INSERT INTO Program (STUDENT_REF_ID, PROGRAM_NAME, PROGRAM_START_DATE) VALUES
(201, 'Computer Science', '2021-09-01 00:00:00'),
(202, 'Mathematics', '2021-09-01 00:00:00'),
(208, 'Mathematics', '2021-09-01 00:00:00'),
(205, 'Physics', '2021-09-01 00:00:00'),
(204, 'Chemistry', '2021-09-01 00:00:00'),
(207, 'Psychology', '2021-09-01 00:00:00'),
(206, 'History', '2021-09-01 00:00:00'),
(203, 'Biology', '2021-09-01 00:00:00');
```

3. Scholarship Table DDL

Here's the SQL code to create the `Scholarship` table:

```
CREATE TABLE Scholarship (
    STUDENT_REF_ID INT NOT NULL,
    SCHOLARSHIP_AMOUNT DECIMAL(10, 2) NOT NULL CHECK (SCHOLARSHIP_AMOUNT >= 0),
    SCHOLARSHIP_DATE DATETIME NOT NULL,
    PRIMARY KEY (STUDENT_REF_ID, SCHOLARSHIP_DATE),
    FOREIGN KEY (STUDENT_REF_ID) REFERENCES Student(STUDENT_ID)
);

INSERT INTO Scholarship (STUDENT_REF_ID, SCHOLARSHIP_AMOUNT, SCHOLARSHIP_DATE) VALUES
(201, 5000, '2021-10-15 00:00:00'),
(202, 4500, '2022-08-18 00:00:00'),
(203, 3000, '2022-01-25 00:00:00'),
(201, 4000, '2021-10-15 00:00:00');
```

Summary

- **Non-Null Constraints:** Each column in the tables is set to `NOT NULL`, ensuring that no null values will be allowed.
- **Data Integrity:** Foreign key constraints ensure that the relationships between tables are maintained correctly.

- **Checks on GPA and Scholarship Amount:** Constraints on GPA (between 0 and 10) and scholarship amounts (greater than or equal to 0) help maintain data integrity.

---

1. Student Table

STUDENT_ID	FIRST_NAME	LAST_NAME	GPA	ENROLLMENT_DATE	MAJOR
201	Kommuju	Sai kiran	8.79	2021-09-01 09:30:00	Computer Science
202	S	Manya dharshini	8.44	2021-09-01 08:30:00	Mathemati
203	Bhavana	alekhya saiDaripalli	5.60	2021-09-01 10:00:00	Biology
204	Basava	Anil kumar	9.20	2021-09-01 12:45:00	Chemistry
205	Bulasara	Babita	7.85	2021-09-01 08:30:00	Physics
206	Jafarvali	Shaik	9.56	2021-09-01 09:24:00	History
207	Ummar basha	Syed	9.78	2021-09-01 02:30:00	English
208	R	Manoj	7.00	2021-09-01 06:30:00	Mathemati
209	Panduri	Baby sri karthikeyi	8.00	2021-09-01 06:30:00	Computer Science
210	Bandi	Praveen kumar	8.50	2021-09-01 06:30:00	Engineerin
211	Pasupuleti	Bhargav	7.25	2021-09-01 06:30:00	Mathemati
212	Sangeetha	Chennamshetty	9.10	2021-09-01 06:30:00	Biology
213	Vallepu	Vivek	8.60	2021-09-01 06:30:00	Chemistry
214	M	Nagendran	7.75	2021-09-01 06:30:00	Physics
215	Erige	Lokesh	9.30	2021-09-01 06:30:00	Computer Science
216	Balu	Pavan teja	8.45	2021-09-01 06:30:00	Mathemati
217	Mamilla	Bhavana	8.90	2021-09-01 06:30:00	Engineerin
218	Pasupuleti	Supriya	7.80	2021-09-01 06:30:00	Mathemati
219	Pamarthi	Venkata ram	8.00	2021-09-01 06:30:00	Biology
220	Baskaran	Rahul	9.00	2021-09-01 06:30:00	Chemistry
221	Raghavanandu	Chenna	8.30	2021-09-01 06:30:00	Physics
222	V	Sudharshan	8.20	2021-09-01 06:30:00	Computer Science
223	Ganesh	Kotha	8.40	2021-09-01 06:30:00	Mathemati
224	Dharmapuri	Chandu	9.10	2021-09-01 06:30:00	Engineerin
225	Palireddy	Swapna	7.90	2021-09-01 06:30:00	Biology
226	Pathipati	Chiranjeevi	8.60	2021-09-01 06:30:00	Chemistry
227	Birru	Hanish kumar	8.55	2021-09-01 06:30:00	Physics
228	Balamurugan	Mukesh	9.20	2021-09-01 06:30:00	Computer Science

STUDENT_ID	FIRST_NAME	LAST_NAME	GPA	ENROLLMENT_DATE	MAJOR
229	B	Aravindan	7.70	2021-09-01 06:30:00	Mathemati
230	Yeruva	Srilekha	8.00	2021-09-01 06:30:00	Engineerin
231	Krishna	chaithanyaMamidi	8.40	2021-09-01 06:30:00	Biology

2. Program Table

STUDENT_REF_ID	PROGRAM_NAME	PROGRAM_START_DATE
201	Computer Science	2021-09-01 00:00:00
202	Mathematics	2021-09-01 00:00:00
208	Mathematics	2021-09-01 00:00:00
205	Physics	2021-09-01 00:00:00
204	Chemistry	2021-09-01 00:00:00
207	Psychology	2021-09-01 00:00:00
206	History	2021-09-01 00:00:00
203	Biology	2021-09-01 00:00:00

3. Scholarship Table

STUDENT_REF_ID	SCHOLARSHIP_AMOUNT	SCHOLARSHIP_DATE
201	5000	2021-10-15 00:00:00
202	4500	2022-08-18 00:00:00
203	3000	2022-01-25 00:00:00
201	4000	2021-10-15 00:00:00

Scenario-Based Interview Questions

1. **Question 1: Identifying Top Students by Major**
   - **Scenario:** The university wants to reward students with the highest GPA in each major.
   - **Task:** Write an SQL query to find the top student (highest GPA) for each major along with their details (STUDENT_ID, FIRST_NAME, LAST_NAME, GPA, MAJOR).
   - **Follow-Up:** How would you modify your query if two or more students have the same highest GPA in a major?
2. **Question 2: Analyzing Program Enrollment Trends**
   - **Scenario:** The academic department is interested in understanding the enrollment trends in different programs over the years.
   - **Task:** Write a query to get the number of students enrolled in each program.
   - **Follow-Up:** How would you extend your analysis to include only those programs that have

seen an increase in enrollment compared to the previous year?

3. **Question 3: Scholarship Analysis for Financial Aid**
    - **Scenario:** The financial aid office wants to analyze the total scholarship amount awarded to students.
    - **Task:** Write an SQL query to calculate the total scholarship amount awarded to each student along with their names (FIRST_NAME, LAST_NAME) and the number of scholarships they received.
    - **Follow-Up:** How would you handle cases where students may have received multiple scholarships?
4. **Question 4: Students with No Scholarships**
    - **Scenario:** The university is interested in identifying students who have not received any scholarships for outreach purposes.
    - **Task:** Write a query to list all students (including their names and majors) who have not received any scholarships.
    - **Follow-Up:** How would you modify your query to include students who are currently enrolled in a program but have not received a scholarship?
5. **Question 5: GPA Improvement Analysis**
    - **Scenario:** The academic committee wants to track GPA improvements for students who have received scholarships.
    - **Task:** Write a query that identifies students who received a scholarship and had an improvement in their GPA from their enrollment date to the current date (consider the current date to be 2023-09-01).
    - **Follow-Up:** What additional factors would you consider important when analyzing GPA improvement for students?

Example SQL Queries for Each Scenario

Here are example SQL queries that could be used to answer the questions (note that actual implementations may vary based on the SQL dialect being used).

1. **Top Students by Major**

```
SELECT STUDENT_ID, FIRST_NAME, LAST_NAME, GPA, MAJOR
FROM Student
WHERE (MAJOR, GPA) IN (
    SELECT MAJOR, MAX(GPA)
    FROM Student
    GROUP BY MAJOR
);
```

2. **Program Enrollment Trends**

```
SELECT PROGRAM_NAME, COUNT(STUDENT_REF_ID) AS Enrollment_Count
FROM Program
JOIN Student ON Program.STUDENT_REF_ID = Student.STUDENT_ID
GROUP BY PROGRAM_NAME;
```

3. Total Scholarship Amount

```
SELECT s.FIRST_NAME, s.LAST_NAME, COUNT(sch.STUDENT_REF_ID) AS Scholarship_Count,
       SUM(sch.SCHOLARSHIP_AMOUNT) AS Total_Scholarship
FROM Student s
LEFT JOIN Scholarship sch ON s.STUDENT_ID = sch.STUDENT_REF_ID
GROUP BY s.STUDENT_ID;
```

4. Students with No Scholarships

```
SELECT s.FIRST_NAME, s.LAST_NAME, s.MAJOR
FROM Student s
LEFT JOIN Scholarship sch ON s.STUDENT_ID = sch.STUDENT_REF_ID
WHERE sch.STUDENT_REF_ID IS NULL;
```

5. GPA Improvement Analysis

```
SELECT s.FIRST_NAME, s.LAST_NAME, s.GPA
FROM Student s
JOIN Scholarship sch ON s.STUDENT_ID = sch.STUDENT_REF_ID
WHERE s.GPA > (
    SELECT GPA FROM Student
    WHERE STUDENT_ID = s.STUDENT_ID -- Assuming initial GPA is stored during
enrollment
);
```

Question: Calculate the GPA Status of Students

Task: Write a query to categorize students based on their GPA into different status groups. The categories should be as follows:

- "Excellent" for a GPA of 9.0 and above
- "Good" for a GPA between 7.0 and 8.9
- "Average" for a GPA between 5.0 and 6.9
- "Poor" for a GPA below 5.0

In addition to the student details, return the GPA status for each student.

SQL Query Example:

```
SELECT
    STUDENT_ID,
    FIRST_NAME,
    LAST_NAME,
    GPA,
    CASE
        WHEN GPA >= 9.0 THEN 'Excellent'
        WHEN GPA >= 7.0 THEN 'Good'
        WHEN GPA >= 5.0 THEN 'Average'
        ELSE 'Poor'
    END AS GPA_Status
FROM
    Student;
```

Explanation:

- In this query, the `CASE` statement is used to evaluate the `GPA` for each student and categorize them accordingly.
- Each `WHEN` clause checks the GPA range and assigns a corresponding status.
- The `ELSE` clause ensures that any GPA below 5.0 is classified as "Poor."
- This approach provides a clear and concise way to derive multiple categorical outputs based on a single numeric input (the GPA) within a single SQL query.

1. Write a SQL query to fetch the `FIRST_NAME` from the `Student` table in uppercase and use an alias name as `STUDENT_NAME`.

```
SELECT UPPER(FIRST_NAME) AS STUDENT_NAME
FROM Student;
```

2. Write a SQL query to fetch unique values of `MAJOR` subjects from the `Student` table.

```
SELECT DISTINCT MAJOR
FROM Student;
```

3. Write a SQL query to print the first 3 characters of `FIRST_NAME` from the `Student` table.

```
SELECT SUBSTRING(FIRST_NAME, 1, 3) AS FIRST_3_CHARACTERS
FROM Student;
```

4. Write a SQL query to find the position of the alphabet ('a') in the `FIRST_NAME` column for the student named 'Kommuju' from the `Student` table.

```
SELECT POSITION('a' IN FIRST_NAME) AS POSITION_OF_A
FROM Student
WHERE FIRST_NAME = 'Kommuju';
```

5. Write a SQL query that fetches the unique values of `MAJOR` subjects from the `Student` table and prints the length of each unique major.

```sql
SELECT DISTINCT MAJOR, LENGTH(MAJOR) AS MAJOR_LENGTH
FROM Student;
```

6. Write a SQL query to print `FIRST_NAME` from the `Student` table after replacing 'a' with 'A'.

```sql
SELECT REPLACE(FIRST_NAME, 'a', 'A') AS UPDATED_FIRST_NAME
FROM Student;
```

7. Write a SQL query to concatenate `FIRST_NAME` and `LAST_NAME` from the `Student` table into a single column named `COMPLETE_NAME`.

```sql
SELECT CONCAT(FIRST_NAME, ' ', LAST_NAME) AS COMPLETE_NAME
FROM Student;
```

8. Write a SQL query to calculate the average `GPA` for each `MAJOR` in the `Student` table.

```sql
SELECT MAJOR, AVG(GPA) AS AVERAGE_GPA
FROM Student
GROUP BY MAJOR;
```

9. Write a SQL query to retrieve all students who received a scholarship amount greater than 4000 from the `Scholarship` table.

```sql
SELECT s.FIRST_NAME, s.LAST_NAME, sch.SCHOLARSHIP_AMOUNT
FROM Scholarship sch
JOIN Student s ON sch.STUDENT_REF_ID = s.STUDENT_ID
WHERE sch.SCHOLARSHIP_AMOUNT > 4000;
```

10. Write a SQL query to find the students who are enrolled in more than one program from the `Program` table.

```sql
SELECT s.FIRST_NAME, s.LAST_NAME, COUNT(p.PROGRAM_NAME) AS PROGRAM_COUNT
FROM Program p
JOIN Student s ON p.STUDENT_REF_ID = s.STUDENT_ID
GROUP BY s.STUDENT_ID, s.FIRST_NAME, s.LAST_NAME
HAVING COUNT(p.PROGRAM_NAME) > 1;
```

11. Write a SQL query to print details of the students whose `FIRST_NAME` ends with 'a'.

```
SELECT *
FROM Student
WHERE FIRST_NAME LIKE '%a';
```

12. Write an SQL query to print details of the students whose `FIRST_NAME` ends with 'a' and contains exactly six characters.

```
SELECT *
FROM Student
WHERE FIRST_NAME LIKE '%a' AND LENGTH(FIRST_NAME) = 6;
```

13. Write an SQL query to print details of the students whose `GPA` lies between 9.00 and 9.99.

```
SELECT *
FROM Student
WHERE GPA BETWEEN 9.00 AND 9.99;
```

14. Write an SQL query to fetch the count of students having the major subject 'Computer Science'.

```
SELECT COUNT(*) AS COMPUTER_SCIENCE_COUNT
FROM Student
WHERE MAJOR = 'Computer Science';
```

15. Write an SQL query to fetch students' full names with `GPA` >= 8.5 and <= 9.5.

```
SELECT CONCAT(FIRST_NAME, ' ', LAST_NAME) AS FULL_NAME
FROM Student
WHERE GPA >= 8.5 AND GPA <= 9.5;
```

16. Write an SQL query to fetch the number of students for each major subject in descending order.

```
SELECT MAJOR, COUNT(*) AS STUDENT_COUNT
FROM Student
GROUP BY MAJOR
ORDER BY STUDENT_COUNT DESC;
```

17. Display the details of students who have received scholarships, including their names, scholarship amounts, and scholarship dates.

```sql
SELECT s.FIRST_NAME, s.LAST_NAME, sch.SCHOLARSHIP_AMOUNT, sch.SCHOLARSHIP_DATE
FROM Scholarship sch
JOIN Student s ON sch.STUDENT_REF_ID = s.STUDENT_ID;
```

18. Write an SQL query to show only odd rows from the `Student` table.

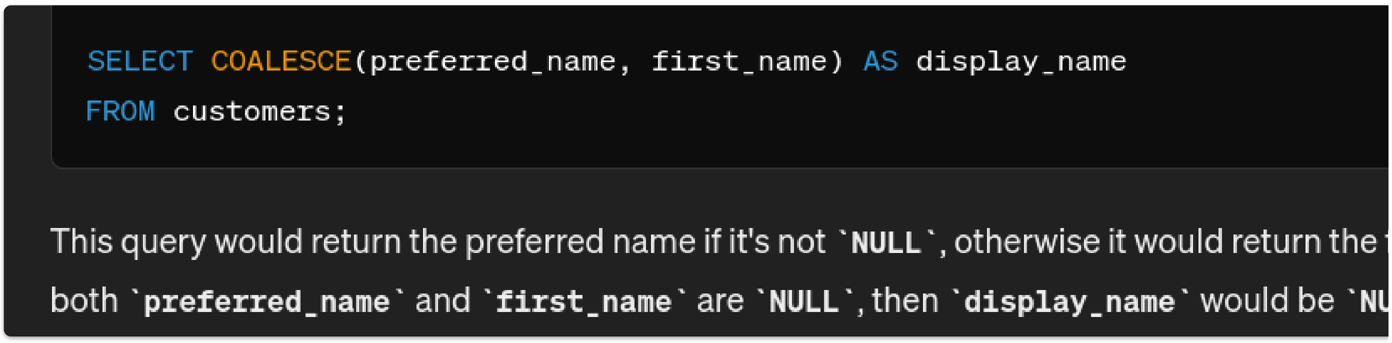
```sql
SELECT *
FROM Student
WHERE MOD(STUDENT_ID, 2) = 1;  -- Assuming STUDENT_ID is sequential and starts from 1
```

19. Write an SQL query to show only even rows from the `Student` table.

```sql
SELECT *
FROM Student
WHERE MOD(STUDENT_ID, 2) = 0;  -- Assuming STUDENT_ID is sequential and starts from 1
```

20. List all students and their scholarship amounts if they have received any. If a student has not received a scholarship, display NULL for the scholarship details.

```sql
SELECT
    Student.FIRST_NAME,
    Student.LAST_NAME,
    COALESCE(Scholarship.SCHOLARSHIP_AMOUNT, NULL) AS SCHOLARSHIP_AMOUNT,
    COALESCE(Scholarship.SCHOLARSHIP_DATE, NULL) AS SCHOLARSHIP_DATE
FROM
    Student
LEFT JOIN
    Scholarship ON Student.STUDENT_ID = Scholarship.STUDENT_REF_ID;
```



```sql
SELECT COALESCE(preferred_name, first_name) AS display_name
FROM customers;
```

This query would return the preferred name if it's not `NULL`, otherwise it would return the
both `preferred_name` and `first_name` are `NULL`, then `display_name` would be `NU

21. Write an SQL query to show the top 5 records from the `Student` table ordered by descending GPA.

```sql
SELECT *
FROM Student
```

```
ORDER BY GPA DESC
LIMIT 5;
```

22. Write an SQL query to determine the nth (say n=5) highest GPA from the `Student` table.

```
SELECT DISTINCT GPA
FROM Student
ORDER BY GPA DESC
OFFSET 4 ROWS FETCH NEXT 1 ROW ONLY;  -- n=5 corresponds to OFFSET 4
```

23. Write an SQL query to determine the 5th highest GPA without using the `LIMIT` keyword.

```
SELECT MIN(GPA) AS Fifth_Highest_GPA
FROM (
    SELECT DISTINCT GPA
    FROM Student
    ORDER BY GPA DESC
    OFFSET 4 ROWS  -- Skip the first 4 highest GPAs
) AS Subquery;
```

24. Write an SQL query to fetch the list of students with the same GPA.

```
SELECT FIRST_NAME, LAST_NAME, GPA
FROM Student
WHERE GPA IN (
    SELECT GPA
    FROM Student
    GROUP BY GPA
    HAVING COUNT(*) > 1
);
```

25. Write an SQL query to show the second highest GPA from the `Student` table using a sub-query.

```
SELECT MAX(GPA) AS Second_Highest_GPA
FROM Student
WHERE GPA < (SELECT MAX(GPA) FROM Student);
```

26. Write an SQL query to show one row twice in the results from the `Student` table.

```sql
SELECT *
FROM Student
WHERE STUDENT_ID = 201  -- Replace with the STUDENT_ID of the row you want to duplicate
UNION ALL
SELECT *
FROM Student
WHERE STUDENT_ID = 201;  -- Same STUDENT_ID to duplicate the row
```

27. Write an SQL query to list `STUDENT_ID` who does not receive a scholarship.

```sql
SELECT STUDENT_ID
FROM Student
WHERE STUDENT_ID NOT IN (
    SELECT STUDENT_REF_ID
    FROM Scholarship
);
```

28. Write an SQL query to fetch the first 50% of records from the `Student` table.

```sql
SELECT *
FROM Student
ORDER BY STUDENT_ID
OFFSET (SELECT COUNT(*) / 2 FROM Student) ROWS;  -- Fetch first 50%
```

29. Write an SQL query to fetch the MAJOR subjects that have less than 4 students in them.

```sql
SELECT MAJOR
FROM Student
GROUP BY MAJOR
HAVING COUNT(*) < 4;
```

30. Write an SQL query to show all MAJOR subjects along with the number of students in each.

```sql
SELECT MAJOR, COUNT(*) AS STUDENT_COUNT
FROM Student
GROUP BY MAJOR;
```

31. Write an SQL query to show the last record from the `Student` table.

```sql
SELECT *
FROM Student
```

```
ORDER BY STUDENT_ID DESC
LIMIT 1;
```

32. Write an SQL query to fetch the first row of the `Student` table.

```
SELECT *
FROM Student
ORDER BY STUDENT_ID
LIMIT 1;
```

33. Write an SQL query to fetch the last five records from the `Student` table.

```
SELECT *
FROM Student
ORDER BY STUDENT_ID DESC
LIMIT 5;
```

34. Write an SQL query to fetch the three maximum GPAs from the `Student` table using a correlated subquery.

```
SELECT DISTINCT GPA
FROM Student AS S1
WHERE (
    SELECT COUNT(DISTINCT GPA)
    FROM Student AS S2
    WHERE S2.GPA > S1.GPA
) < 3;  -- This will return the top 3 maximum GPAs
```

35. Write an SQL query to fetch the three minimum GPAs from the `Student` table using a correlated subquery.

```
SELECT DISTINCT GPA
FROM Student AS S1
WHERE (
    SELECT COUNT(DISTINCT GPA)
    FROM Student AS S2
    WHERE S2.GPA < S1.GPA
) < 3;  -- This will return the bottom 3 minimum GPAs
```

36. Write an SQL query to fetch the nth maximum GPA from the `Student` table.

```
SELECT DISTINCT GPA
FROM Student
```

```
ORDER BY GPA DESC
OFFSET n - 1 ROWS FETCH NEXT 1 ROW ONLY;  -- Replace n with the desired value for nth
maximum
```

37. Write an SQL query to fetch MAJOR subjects along with the maximum GPA in each of these MAJOR subjects.

```
SELECT MAJOR, MAX(GPA) AS Max_GPA
FROM Student
GROUP BY MAJOR;
```

Here are the SQL questions from 38 to 45, rewritten for the provided dataset, along with the corresponding SQL queries:

38. Write an SQL query to fetch the names of students who have the highest GPA.

```
SELECT FIRST_NAME, LAST_NAME
FROM Student
WHERE GPA = (SELECT MAX(GPA) FROM Student);
```

39. Write an SQL query to show the current date and time.

```
SELECT NOW() AS CurrentDateTime;  -- MySQL
```

Note: Depending on the SQL database system, the function to get the current date and time may differ. For example, in SQL Server, you would use `SELECT GETDATE()`.

40. Write a query to create a new table which consists of data and structure copied from the `Student` table (or clone the table named `Student`).

```
CREATE TABLE Student_Clone AS
SELECT *
FROM Student;
```

Note: The syntax for cloning a table might differ slightly between different SQL databases. In SQL Server, you might need to use `SELECT INTO Student_Clone FROM Student`. The above is for databases like PostgreSQL or MySQL.

41. Write an SQL query to update the GPA of all the students in the 'Computer Science' major subject to 7.5.

```
UPDATE Student
SET GPA = 7.5
```

```sql
WHERE MAJOR = 'Computer Science';
```

42. Write an SQL query to find the average GPA for each major.

```sql
SELECT MAJOR, AVG(GPA) AS Average_GPA
FROM Student
GROUP BY MAJOR;
```

43. Write an SQL query to show the top 3 students with the highest GPA.

```sql
SELECT FIRST_NAME, LAST_NAME, GPA
FROM Student
ORDER BY GPA DESC
LIMIT 3;
```

44. Write an SQL query to find the number of students in each major who have a GPA greater than 7.5.

```sql
SELECT MAJOR, COUNT(*) AS NumberOfStudents
FROM Student
WHERE GPA > 7.5
GROUP BY MAJOR;
```

45. Write an SQL query to find the students who have the same GPA as 'Kommuju Sai kiran'.

```sql
SELECT FIRST_NAME, LAST_NAME
FROM Student
WHERE GPA = (SELECT GPA FROM Student WHERE FIRST_NAME = 'Kommuju' AND LAST_NAME = 'Sai kiran');
```