The Ultimate Guide of SQL

Here's what you'll find in this document:-

- Complete SQL Syllabus with Resources
- SQL Practice Websites Categorized by Levels
- Types of Most Frequently Asked SQL Interview Questions & Answers

---

- Complete SQL Syllabus with Resources

1. SQL Data Definition Language (DDL)

- **CREATE**: Create a new database object (e.g., table).
- **ALTER**: Modify an existing database object.
- **DROP**: Remove an existing database object.
- **TRUNCATE**: Remove all records from a table (but keep the structure).

DDL is used to define and manage database structures like tables and schemas.

---

2. SQL Data Manipulation Language (DML)

- **INSERT**: Insert new data into a table.
- **UPDATE**: Modify existing data in a table.
- **DELETE**: Remove data from a table.

DML is used to manipulate data within database tables.

---

3. SQL Data Types

- Numeric Data Types: `INT`, `FLOAT`, `DECIMAL`, etc.
- String Data Types: `VARCHAR`, `CHAR`, `TEXT`, etc.
- Date/Time Data Types: `DATE`, `DATETIME`, `TIMESTAMP`, etc.
- Boolean Data Type: `BOOLEAN`.

Data types define the nature of data that can be stored in a column (numbers, text, dates, etc.).

---

4. SQL Querying Data

- **SELECT**: Retrieve data from the database.
- **DISTINCT**: Remove duplicate rows in a result set.
- **WHERE**: Filter records based on a condition.
- **LIKE**: Search for a specified pattern.
- **ORDER BY**: Sort the result set.
- **LIMIT**: Limit the number of returned rows (MySQL/PostgreSQL).
- **TOP**: Limit the number of returned rows (SQL Server).
- **AND, OR, NOT**: Logical operators for filtering data.
- **IN**: Filter records based on a list of values.
- **BETWEEN**: Filter records based on a range.

These clauses and operators are used to filter and organize query results.

---

5. Aggregate Functions

- **SUM()**: Calculate the total sum of a numeric column.
- **MAX()**: Find the maximum value.
- **MIN()**: Find the minimum value.
- **COUNT()**: Count the number of rows.
- **AVG()**: Calculate the average value.

Aggregate functions are used to calculate summary statistics on sets of data.

---

6. Grouping and Filtering Data

- **GROUP BY**: Group rows that have the same values into summary rows.
- **HAVING**: Filter groups based on a condition.

These clauses are used to group data and filter aggregated results.

---

7. SQL Joins

- **INNER JOIN**: Return rows with matching values in both tables.

- **LEFT JOIN (LEFT OUTER JOIN)**: Return all rows from the left table, and the matched rows from the right table.
- **RIGHT JOIN (RIGHT OUTER JOIN)**: Return all rows from the right table, and the matched rows from the left table.
- **FULL OUTER JOIN**: Return all rows when there is a match in either left or right table.
- **SELF JOIN**: Join a table to itself.

Joins are used to combine rows from two or more tables based on related columns.

---

8. Subqueries and CTEs (Common Table Expressions)

- **CTE**: Temporary result set defined within the execution scope of a SELECT, INSERT, UPDATE, or DELETE statement.
- **SUBQUERIES**: A query nested inside another query.

Subqueries and CTEs are used to simplify complex queries and improve readability.

---

9. Set Operators

- **UNION**: Combine the result sets of two or more SELECT statements (without duplicates).
- **UNION ALL**: Combine the result sets of two or more SELECT statements (with duplicates).

Set operators are used to combine the results of multiple queries.

---

10. Existential and Conditional Queries

- **EXISTS**: Test for the existence of any records in a subquery.
- **CASE WHEN**: Perform conditional logic in a SQL statement.

Existential queries check for the existence of data, while `CASE WHEN` is used for conditional logic within queries.

---

11. Window Functions

- **ROW_NUMBER() OVER**: Assigns a unique sequential integer to rows within a partition of a result set.
- **RANK() OVER**: Provides a ranking of rows within a partition, allowing for ties.

- **DENSE_RANK() OVER**: Similar to `RANK()`, but without gaps in ranking.
- **LEAD() OVER**: Access data from the following row in a partition.
- **LAG() OVER**: Access data from the preceding row in a partition.
- **NTILE() OVER**: Distribute rows into a specified number of equal-sized groups.
- **FIRST_VALUE() OVER**: Get the first value in an ordered set of values.
- **LAST_VALUE() OVER**: Get the last value in an ordered set of values.

Window functions allow for calculations across a set of rows related to the current row, without collapsing data into groups.

---

12. Aggregate Functions as Window Functions

- **SUM() OVER**: Calculate the sum of values across a window of rows.
- **MAX() OVER**: Find the maximum value across a window of rows.
- **MIN() OVER**: Find the minimum value across a window of rows.
- **COUNT() OVER**: Count the number of rows across a window.
- **AVG() OVER**: Calculate the average value across a window.

By using the `OVER()` clause, aggregate functions become window functions, allowing for aggregate calculations over a defined set of rows (the window) without collapsing the data like a traditional `GROUP BY` would.

---

13. SQL Date and Time Functions

- Functions to manipulate date and time values (e.g., `NOW()`, `CURRENT_DATE`, `DATEADD()`, `DATEDIFF()`, etc.).

These functions allow for the manipulation and comparison of date and time values in queries.

RESOURCES:

Websites:

1. https://www.w3schools.com/sql/
2. https://sqlbolt.com/

Youtube Playlist:

This below playlist contains the complete tutorial video of SQL with all the required topics in English.

https://youtube.com/playlist?list=PLavw5C92dz9Ef4E-1Zi9KfCTXS_IN8gXZ&si=XCwpStf9zZ0YISN8

And if you want to learn in Hindi, then you can follow this below playlist:

https://youtube.com/playlist?list=PLdOKnrf8EcP17p05q13WXbHO5Z_JfXNpw&si=8m4E9IGf-2MR9ZKA

Note - Below mentioned are some top playlists of SQL interview Q&A which I also use to prepare before any SQL interview.

Top SQL Interview Q&A Playlists:-

https://youtube.com/playlist?list=PLavw5C92dz9Hxz0YhttDniNgKejQlPoAn&si=NgKECJfJ8gYCMzxS

https://youtube.com/playlist?list=PLBTZqjSKn0IfuIqbMIqzS-waofsPHMS0E&si=kurTh9-krlyBTZSc

https://youtube.com/playlist?list=PLBTZqjSKn0IeKBQDjLmzisazhqQy4iGkb&si=HFvZN7s3pPAQlpYL

- ## SQL Practice Websites Categorized by Levels

Easy

These websites are perfect for beginners who want to start with the basics and build a solid foundation:

1. **W3Schools SQL Tutorial**
   https://www.w3schools.com/sql/
2. **SQLZoo**
   https://sqlzoo.net/wiki/SQL_Tutorial

3. **SQLBolt**
   https://sqlbolt.com/

Medium

These platforms are great for those with some SQL knowledge who want to practice real-world problems and prepare for interviews:

1. **Leetcode (Top 50 SQL Study Plan)**
   https://leetcode.com/studyplan/top-sql-50/
2. **Leetcode Database Problems**
   https://leetcode.com/problemset/database/
3. **Datalemur**
   https://datalemur.com/
4. **HackerRank SQL**
   https://www.hackerrank.com/domains/sql
5. **StrataScratch**
   https://platform.stratascratch.com/coding?code_type=1

Expert

For advanced practice, you can solve the hard-difficulty questions available on the medium-level websites mentioned above, such as:

- Leetcode Database Problems (Hard)
  https://leetcode.com/problemset/database/
- HackerRank Advanced SQL Challenges
  https://www.hackerrank.com/domains/sql
- StrataScratch Expert-Level Problems
  https://platform.stratascratch.com/coding?code_type=1

- Types of Most Frequently Asked SQL Interview Questions & Answers

Overview

For any data-related job role, SQL is a key skill, and your interview will mostly revolve around it. In any SQL round, you can face two types of questions:

- Query writing-based questions
- Verbally asked conceptual questions

First, we will cover query-based questions, followed by verbal questions.

Query-Based Questions

Types of Most Frequently Asked SQL Interview Questions & Answers

Overview

For any data-related job role, SQL is a key skill, and your interview will mostly revolve around it. In any SQL round, you can face two types of questions:

- Query writing-based questions
- Verbally asked conceptual questions

First, we will cover query-based questions, followed by verbal questions.

Query-Based Questions

Question 1: Write a SQL query to find the second highest salary from the table emp.

- **Table:** emp

- **Columns:** id, salary

Answer (Using DENSE_RANK):

```
WITH RankedSalaries AS (
   SELECT salary, DENSE_RANK() OVER (ORDER BY salary DESC) AS rank
   FROM emp
)
SELECT salary AS SecondHighestSalary
FROM RankedSalaries
WHERE rank = 2;
```

Question 2: Write a SQL query to find the numbers which consecutively occur 3 times.

- **Table:** table_name
- **Columns:** id, numbers

Answer:

```
SELECT numbers
FROM (
   SELECT numbers,
        LEAD(numbers, 1) OVER (ORDER BY id) AS next_num,
        LEAD(numbers, 2) OVER (ORDER BY id) AS next_next_num
   FROM table_name
) t
WHERE numbers = next_num AND numbers = next_next_num;
```

Question 3: Write a SQL query to find the days when temperature was higher than its previous dates.

- **Table:** table_name

- **Columns:** Days, Temp

Answer (Using CTE):

```
WITH TempWithLag AS (
   SELECT Days, Temp, LAG(Temp) OVER (ORDER BY Days) AS prev_temp
   FROM table_name
)
SELECT Days
FROM TempWithLag
WHERE Temp > prev_temp;
```

Question 4: Write a SQL query to delete duplicate rows in a table.

- **Table:** table_name
- **Columns:** column1, column2, ..., columnN

Answer:

```
DELETE FROM table_name
WHERE id NOT IN (
   SELECT MIN(id)
   FROM table_name
   GROUP BY column1, column2, ..., columnN
);
```

Question 5: Write a SQL query for the cumulative sum of salary of each employee from January to July.

- **Table:** table_name
- **Columns:** Emp_id, Month, Salary

Answer:

```
SELECT Emp_id, Month, SUM(Salary) OVER (
   PARTITION BY Emp_id ORDER BY Month ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW
) AS CumulativeSalary
FROM table_name;
```

Question 6: Write a SQL query to display year-on-year growth for each product.

- **Table:** table_name
- **Columns:** transaction_id, Product_id, transaction_date, spend

Answer (Using CTE):

```
WITH YearlySpend AS (
   SELECT
      Product_id,
      YEAR(transaction_date) AS year,
      SUM(spend) AS total_spend
   FROM table_name
   GROUP BY Product_id, YEAR(transaction_date)
),
Growth AS (
   SELECT
      year,
      Product_id,
      total_spend,
      LAG(total_spend) OVER (PARTITION BY Product_id ORDER BY year) AS
prev_year_spend
   FROM YearlySpend
)
SELECT year, Product_id,
      (total_spend - prev_year_spend) / prev_year_spend AS yoy_growth
FROM Growth
WHERE prev_year_spend IS NOT NULL;
```

Question 7: Write a SQL query to find the rolling average of posts on a daily basis for each user_id. Round up the average to two decimal places.

- **Table:** table_name
- **Columns:** user_id, date, post_count

Answer:

```
SELECT user_id, date,
    ROUND(AVG(post_count) OVER (
        PARTITION BY user_id ORDER BY date ROWS BETWEEN 6
PRECEDING AND CURRENT ROW
    ), 2) AS RollingAvg
FROM table_name;
```

Question 8: Write a SQL query to get the emp_id and department for each department where the most recently joined employee is still working.

- **Table:** table_name
- **Columns:** emp_id, first_name, last_name, date_of_join, date_of_exit, department

Answer:

```
SELECT emp_id, department
FROM table_name
WHERE date_of_exit IS NULL
ORDER BY date_of_join DESC;
```

Question 9: How many rows will come in the outputs of Left, Right, Inner, and Outer Join from two tables having duplicate rows?

- **Left Table A:**

Column
1
1
1
2
2
2
3
4
5

- **Right Table B:**

Column
1
1
2
2
2
3
3
3
4

Answer:

- Left Join: 17 rows
- Right Join: 16 rows
- Inner Join: 16 rows
- Outer Join: 17 rows

Explanation:

- **Left Join:** The left join combines all rows from Table A with matching rows in Table B. For values like 1 and 2, multiple matches occur, leading to repeated rows in the output. Unique values in A without matches in B (5) are included with NULL values.

  Method for calculating the rows -

  3 rows of 1 from left table * 2 rows of 1 from right table = 6 Rows of 1
  2 rows of 2 from left table * 3 rows of 2 from right table = 6 Rows of 2

  1 rows of 3 from left table * 3 rows of 3 from right table = 3 Rows of 3

  1 rows of 4 from left table * 1 rows of 4 from right table = 1 Rows of 4

  1 rows of 5 from left table will come with Null in corresponding row as there is no value of 5 in right and we are doing left join so it is mandatory to take all values from left table -

  So, Total output of left join will be 17 rows

  Note - Please use above method and try to understand other joins output too

- **Right Join:** The right join behaves symmetrically, including all rows from Table B with matches in Table A. Unique values in B without matches in A (None in this case) would appear with NULL values, but no such rows exist here.

- **Inner Join:** The inner join only includes rows with matching values in both tables. Duplicates amplify the matches, yielding 16 rows.

- **Outer Join:** The full outer join includes all rows from both tables, combining matched rows and appending unmatched rows with NULL values. Here, only 5 from Table A contributes an unmatched row, leading to 17 total rows.

Question 10: Write a query to get mean, median, and mode for earnings.

- **Table:** table_name
- **Columns:** Emp_id, salary

Answer:

```
-- Mean
SELECT AVG(salary) AS MeanSalary FROM table_name;

-- Median
SELECT AVG(salary) AS MedianSalary
FROM (
    SELECT salary
    FROM table_name
    ORDER BY salary
    LIMIT 2 - (SELECT COUNT(*) FROM table_name) % 2 OFFSET (SELECT
(COUNT(*) - 1) / 2 FROM table_name)
) t;

-- Mode
SELECT salary AS ModeSalary
FROM table_name
GROUP BY salary
ORDER BY COUNT(*) DESC
LIMIT 1;
```

Question 11: Determine the count of rows in the output of the following queries for Table X and Table Y.

- **Table X:**

ids

1
1
1
1

- **Table Y:**

ids
1
1
1
1
1
1
1
1

Queries:

1. SELECT * FROM X JOIN Y ON X.ids != Y.ids
2. SELECT * FROM X LEFT JOIN Y ON X.ids != Y.ids
3. SELECT * FROM X RIGHT JOIN Y ON X.ids != Y.ids
4. SELECT * FROM X FULL OUTER JOIN Y ON X.ids != Y.ids

Answer:

Since the join condition X.ids != Y.ids cannot be satisfied (as all ids in both tables are 1), the output for all queries will be:

- Query 1: 0 rows
- Query 2: 0 rows
- Query 3: 0 rows
- Query 4: 0 rows

Explanation:

- The condition X.ids != Y.ids checks for inequality between the columns, which is not possible as every row in both tables has the same value for ids.
- Hence, no rows are returned for any join type.

Question 12: Write a SQL query to calculate the percentage of total sales contributed by each product category in a given year.

- **Table:** sales
- **Columns:** product_category, sale_year, revenue

Answer:

```
WITH TotalSales AS (
    SELECT sale_year, SUM(revenue) AS total_revenue
    FROM sales
    GROUP BY sale_year
)
SELECT s.product_category, s.sale_year,
       (SUM(s.revenue) / t.total_revenue) * 100 AS percentage_contribution
FROM sales s
JOIN TotalSales t ON s.sale_year = t.sale_year
GROUP BY s.product_category, s.sale_year, t.total_revenue;
```

Question 13: Write a SQL query to find the longest streak of consecutive days an employee worked.

- **Table:** attendance
- **Columns:** emp_id, work_date

Answer:

```
WITH ConsecutiveDays AS (
    SELECT emp_id, work_date,
        ROW_NUMBER() OVER (PARTITION BY emp_id ORDER BY work_date) -
        DENSE_RANK() OVER (PARTITION BY emp_id, DATE_ADD(work_date, -ROW_NUMBER() OVER (PARTITION BY emp_id ORDER BY work_date))) AS streak_group
    FROM attendance
)
SELECT emp_id, COUNT(*) AS longest_streak
FROM ConsecutiveDays
GROUP BY emp_id, streak_group
ORDER BY longest_streak DESC
LIMIT 1;
```

Question 14: Write a query to identify customers who made purchases in all quarters of a year.

- **Table:** transactions
- **Columns:** customer_id, transaction_date

Answer:

```
WITH QuarterlyData AS (
    SELECT customer_id,
        CONCAT(YEAR(transaction_date), '-Q', QUARTER(transaction_date)) AS
quarter
    FROM transactions
    GROUP BY customer_id, YEAR(transaction_date),
QUARTER(transaction_date)
)
SELECT customer_id
FROM QuarterlyData
GROUP BY customer_id
HAVING COUNT(DISTINCT quarter) = 4;
```

Question 15: Write a query to find the first and last purchase dates for each customer, along with their total spending.

- **Table:** transactions
- **Columns:** customer_id, transaction_date, amount

Answer:

```
SELECT customer_id,
    MIN(transaction_date) AS first_purchase,
    MAX(transaction_date) AS last_purchase,
    SUM(amount) AS total_spending
```

```
FROM transactions

GROUP BY customer_id;
```

Question 16: Write a query to find the top 3 employees who generated the highest revenue in the last year.

- **Table:** employee_sales
- **Columns:** emp_id, sale_date, revenue

Answer:

```
SELECT emp_id, SUM(revenue) AS total_revenue

FROM employee_sales

WHERE YEAR(sale_date) = YEAR(CURDATE()) - 1

GROUP BY emp_id

ORDER BY total_revenue DESC

LIMIT 3;
```

Question 17: Write a query to calculate the monthly retention rate for a subscription-based service.

- **Table:** subscriptions
- **Columns:** user_id, start_date, end_date

Answer:

```
WITH MonthlyRetention AS (
```

```
    SELECT DATE_FORMAT(start_date, '%Y-%m') AS subscription_month,
        COUNT(DISTINCT user_id) AS new_users,
        COUNT(DISTINCT CASE WHEN end_date >=
LAST_DAY(DATE_ADD(start_date, INTERVAL 1 MONTH)) THEN user_id
END) AS retained_users
    FROM subscriptions
    GROUP BY subscription_month
)
SELECT subscription_month,
    (retained_users / new_users) * 100 AS retention_rate
FROM MonthlyRetention;
```

Question 18: Write a query to identify products with declining sales for 3 consecutive months.

- **Table:** monthly_sales
- **Columns:** product_id, month, sales

Answer:

```
WITH DeclineCheck AS (
    SELECT product_id, month,
        LAG(sales) OVER (PARTITION BY product_id ORDER BY month) AS
prev_month_sales,
        LAG(sales, 2) OVER (PARTITION BY product_id ORDER BY month) AS
prev_2_months_sales
```

```
    FROM monthly_sales
)
SELECT product_id
FROM DeclineCheck
WHERE sales < prev_month_sales AND prev_month_sales <
prev_2_months_sales
GROUP BY product_id;
```

Question 19: Write a query to find the average order value (AOV) for customers who placed at least 5 orders in the last year.

- **Table:** orders
- **Columns:** customer_id, order_date, order_amount

Answer:

```
WITH OrderCounts AS (
    SELECT customer_id, COUNT(*) AS total_orders, SUM(order_amount) AS
total_spent
    FROM orders
    WHERE YEAR(order_date) = YEAR(CURDATE()) - 1
    GROUP BY customer_id
)
SELECT customer_id, (total_spent / total_orders) AS avg_order_value
FROM OrderCounts
```

```
WHERE total_orders >= 5;
```

Verbally Asked Conceptual Questions

Question 1: Explain the order of execution of SQL.

Answer:

1. FROM: Specifies the source table or tables and establishes any joins between them.
2. WHERE: Filters rows based on specified conditions before grouping or aggregations.
3. GROUP BY: Groups rows into summary rows based on specified columns.
4. HAVING: Filters aggregated groups, often used with aggregate functions.
5. SELECT: Specifies the columns or expressions to include in the final output.
6. ORDER BY: Sorts the result set in ascending or descending order.
7. LIMIT: Restricts the number of rows returned in the final output.

Question 2: What is the difference between WHERE and HAVING?

Answer:

- WHERE: Filters rows before any grouping takes place. It works on individual rows.
- HAVING: Filters aggregated data after grouping. It works on grouped rows.

Example: Use WHERE to filter employees with a salary above 50,000, and HAVING to filter departments with an average salary above 60,000.

Question 3: What is the use of GROUP BY?

Answer: GROUP BY is used to aggregate data into groups based on one or more columns. It is often used with aggregate functions like SUM, COUNT, AVG, MAX, and MIN.

Example: To calculate the total salary for each department:

```
SELECT department_id, SUM(salary) AS total_salary
FROM employees
GROUP BY department_id;
```

Question 4: Explain all types of joins in SQL.

Answer:

1. **INNER JOIN**: Returns rows where there is a match in both tables.
    - Example: Find employees with matching departments.
2. **LEFT JOIN**: Returns all rows from the left table, and matching rows from the right table. Non-matches are filled with NULL.
    - Example: List all employees with their departments, even if they are not assigned.
3. **RIGHT JOIN**: Returns all rows from the right table, and matching rows from the left table. Non-matches are filled with NULL.
    - Example: List all departments with their employees, even if they have none.
4. **FULL OUTER JOIN**: Returns all rows from both tables, with NULL in places where no match exists.
    - Example: Combine all employees and departments, regardless of matches.
5. **CROSS JOIN**: Produces the Cartesian product of both tables.
    - Example: Pair every employee with every department.

Question 5: What are triggers in SQL?

Answer: Triggers are automated actions executed in response to specific database events like INSERT, UPDATE, or DELETE. They are used to enforce rules, log changes, or cascade updates.

Example: Automatically update a log table whenever a row is inserted into the orders table.

Question 6: What is a stored procedure in SQL?

Answer: A stored procedure is a precompiled set of SQL statements stored in the database. It allows reusability, simplifies complex operations, and improves performance by reducing query execution time.

Example: A stored procedure to calculate monthly sales and store the result in a report table.

Question 7: Explain all types of window functions (Mainly RANK, ROW_NUMBER, DENSE_RANK, LEAD, and LAG).

Answer:

- **RANK**: Assigns a rank to rows within a partition, skipping ranks for ties.
- **ROW_NUMBER**: Assigns a unique sequential number to rows within a partition, without skipping.
- **DENSE_RANK**: Similar to RANK, but does not skip ranks for ties.
- **LEAD**: Accesses data from the following row in the same partition.
- **LAG**: Accesses data from the preceding row in the same partition.

Example: Use ROW_NUMBER to assign unique IDs to duplicate records in a dataset.

Question 8: What is the difference between DELETE and TRUNCATE?

Answer:

- **DELETE**: Removes specific rows based on a WHERE clause. It logs each row deletion, can be slower, and maintains table structure.
- **TRUNCATE**: Removes all rows from a table without logging individual deletions. It is faster but cannot filter rows or trigger cascades.

Question 9: What is the difference between DML, DDL, and DCL?

Answer:

- **DML (Data Manipulation Language)**: Deals with data manipulation.
  - Commands: INSERT, UPDATE, DELETE, SELECT.
- **DDL (Data Definition Language)**: Manages table structure.
  - Commands: CREATE, ALTER, DROP, TRUNCATE.
- **DCL (Data Control Language)**: Controls access and permissions.
  - Commands: GRANT, REVOKE.

Question 10: What are aggregate functions, and when do we use them? Explain with examples.

Answer: Aggregate functions perform calculations on a set of values. Examples:

- SUM: Adds values. Example: SELECT SUM(salary) FROM employees;
- AVG: Calculates average. Example: SELECT AVG(salary) FROM employees;
- COUNT: Counts rows. Example: SELECT COUNT(*) FROM employees;
- MAX/MIN: Finds maximum or minimum values.

Question 11: Which is faster between CTE and subquery?

Answer: CTEs are often faster and more readable for complex queries, especially when reused multiple times within a query. Subqueries can sometimes be less efficient due to re-evaluation.

Question 12: What are constraints and their types?

Answer: Constraints enforce data integrity and rules on tables. Types include:

- **NOT NULL**: Ensures a column cannot have NULL values.
- **UNIQUE**: Ensures all values in a column are unique.
- **PRIMARY KEY**: A unique identifier for a row, combining NOT NULL and UNIQUE.
- **FOREIGN KEY**: Ensures referential integrity by linking to another table.
- **CHECK**: Ensures values satisfy a condition.
- **DEFAULT**: Assigns a default value if none is provided.

Question 13: What are keys, and what are their types?

Answer:

- **Primary Key**: Uniquely identifies a row. Example: emp_id in an employee table.
- **Foreign Key**: Links two tables. Example: department_id in an employee table referencing id in the department table.
- **Candidate Key**: Potential column(s) for the primary key.
- **Composite Key**: Combines multiple columns to uniquely identify a row.

Question 14: Differentiate between UNION and UNION ALL.

Answer:

- **UNION**: Combines results from two queries and removes duplicates.

- **UNION ALL**: Combines results from two queries without removing duplicates. Faster than UNION.

Question 15: What are indexes, and what are their types?

Answer: Indexes improve query performance by providing faster data access.

- **Clustered Index**: Determines the physical order of rows in a table.
- **Non-Clustered Index**: Contains pointers to the actual data in a table.
- **Unique Index**: Ensures all values in a column are distinct.

Question 16: What are views, and what are their limitations?

Answer: Views are virtual tables based on SQL queries. They do not store data but simplify query reuse. **Limitations:**

- Cannot be indexed.
- Performance depends on the underlying base tables.
- Cannot directly include ORDER BY.

Question 17: What is the difference between VARCHAR and NVARCHAR? Similarly, CHAR and NCHAR?

Answer:

- **VARCHAR**: Variable-length, stores ASCII characters.
- **NVARCHAR**: Variable-length, supports Unicode for multilingual data.
- **CHAR**: Fixed-length ASCII.
- **NCHAR**: Fixed-length Unicode.

Question 18: List the different types of relationships in SQL.

Answer:

1. **One-to-One**: Each row in Table A links to exactly one row in Table B.
2. **One-to-Many**: Each row in Table A links to multiple rows in Table B.
3. **Many-to-Many**: Rows in Table A link to multiple rows in Table B and vice versa.

Question 19: Write retention query in SQL.

Answer:

```
WITH Retention AS (
    SELECT customer_id, COUNT(*) AS total_orders,
        COUNT(CASE WHEN order_date >= DATE_ADD(first_order_date,
INTERVAL 1 MONTH) THEN 1 END) AS retained_orders
    FROM orders
    GROUP BY customer_id
)
SELECT customer_id, (retained_orders / total_orders) * 100 AS retention_rate
FROM Retention;
```