BOSSCODER ACADEMY

ADVANCED SQL CONCEPTS

Most Asked Interview Questions

Disclaimer

Everyone learns uniquely.

What matters is your preparation and consistency.

With this doc understand the major SQL concepts and ace your next interview.

01 QUESTION

How do you find the last ID in a SQL table?

In SQL, you can find the last ID in a table using the MAX function along with the column that represents the ID. Assuming you have an "id" column in your table, you can use the following query:

```SQL
SELECT MAX(id) AS last_id FROM your_table_name;
```

This query selects the maximum (largest) value in the "id" column and aliases it as "last_id." The result will be a single row with the highest ID value in the specified table.

Otherwise, in many SQL versions, we can use the following syntax:

```SQL
SELECT id
FROM your_table_name
ORDER BY id DESC
LIMIT 1;
```

This query selects the maximum (largest) value in the "id" column and aliases it as "last_id." The result will be a single row with the highest ID value in the specified table.

02 QUESTION

How do you remove duplicates from a table?

Using `DISTINCT`:

```SQL
SELECT DISTINCT * FROM your_table;
```

This will retrieve distinct rows from the table based on all columns. Keep in mind that this doesn't actually remove duplicates from the table; it just returns a result set with distinct values.

This query selects the maximum (largest) value in the "id" column and aliases it as "last_id." The result will be a single row with the highest ID value in the specified table.

Otherwise, in many SQL versions, we can use the following syntax:

Using `GROUP BY`:

```SQL
SELECT col1, col2, ..., colN, COUNT(*)
FROM your_table
GROUP BY col1, col2, ..., colN
HAVING COUNT(*) > 1;
```

This will group the rows by specified columns and count the occurrences. Rows with a count greater than 1 are duplicates.

Using `ROW_NUMBER()` with Common Table Expressions (CTE):

```sql
WITH CTE AS (
    SELECT *,
           ROW_NUMBER() OVER (PARTITION BY col1, col2, ..., colN ORDER BY (SELECT 0)) AS rn
    FROM your_table
)
DELETE FROM CTE WHERE rn > 1;
```

This method uses the ROW_NUMBER() window function to assign a unique number to each row within a partition. Rows with rn > 1 are duplicates.

Using `INNER JOIN`:

```sql
DELETE t1
FROM your_table t1
INNER JOIN your_table t2
WHERE t1.id > t2.id
AND t1.col1 = t2.col1
```

```
  AND t1.col2 = t2.col2
  AND ...;
```

This query deletes duplicates based on specified columns, keeping the row with the lowest ID.

03 QUESTION

Give the resulting tables arising from applying Joins on the following tables in SQL

Employees Table:

id	name	department_id
1	Alice	101
2	Bob	102
3	Charlie	101
4	David	103

Departments Table:

id	department_id
101	HR
102	IT
103	Marketing
104	Sales

Inner Join:

- Returns only the rows with matching values in both tables.
- Filters out rows with no match.

SQL Query:

```SQL
SELECT employees.name, departments.department_name
FROM employees
INNER JOIN departments ON employees.department_id =
departments.id;
```

Output:

name	department_name
Alice	HR
Bob	IT
Charlie	HR
David	Marketing

Left Join (Left Outer Join):

- Returns all rows from the left table and the matched rows from the right table.
- If there is no match in the right table, NULL values are returned.

SQL Query:

```SQL
SELECT employees.name, departments.department_name
FROM employees
LEFT JOIN departments ON employees.department_id = departments.id;
```

Output:

name	department_name
Alice	HR
Bob	IT
Charlie	HR
David	Marketing

Right Join (Right Outer Join):

- Returns all rows from the right table and the matched rows from the left table.
- If there is no match in the left table, NULL values are returned.

SQL Query:

```SQL
SELECT employees.name, departments.department_name
FROM employees
RIGHT JOIN departments ON employees.department_id = departments.id;
```

Output:

name	department_name
Alice	HR
Bob	IT
Charlie	HR
David	Marketing
NULL	Sales

Full Outer Join:

- Returns all rows when there is a match in either the left or right table.
- Includes rows with no match in either table with NULL values.

SQL Query:

```SQL
SELECT employees.name, departments.department_name
FROM employees
FULL OUTER JOIN departments ON
employees.department_id = departments.id;
```

Output:

name	department_name
Alice	HR
Bob	IT
Charlie	HR
David	Marketing
NULL	Sales

Self Join:

- Combines rows from a single table, treating it as two separate tables.
- Often used for hierarchical data.

SQL Query:

```SQL
SELECT e1.name, e2.name AS manager
FROM employees e1
LEFT JOIN employees e2 ON e1.manager_id = e2.id;
```

Output:

name	manager
Alice	NULL
Bob	NULL
Charlie	Alice
David	NULL

04 QUESTION

What is difference between HAVING and WHERE in SQL?

GROUP BY Clause:

The GROUP BY clause is used to group rows that have the same values in specified columns into summary rows, like categories.

It is often used in conjunction with aggregate functions (e.g., COUNT, SUM, AVG, MAX, MIN) to perform calculations on each group of rows.

It helps to create a result set where rows are grouped based on the values in specified columns.

Example:

```SQL
SELECT department, AVG(salary)
FROM employees
GROUP BY department;
```

In this example, the result set will have one row for each distinct department, and the average salary for each department will be calculated.

HAVING Clause:

- The HAVING clause is used to filter the results of a GROUP BY query based on a specified condition.
- It is similar to the WHERE clause but is applied after the GROUP BY operation to filter groups of rows.
- It allows you to filter the result set based on aggregate functions.

Example:

```SQL
SELECT department, AVG(salary)
FROM employees
GROUP BY department
HAVING AVG(salary) > 50000;
```

05 QUESTION

Explain about Auto Increment in SQL?

In SQL, the auto-increment feature is often used to generate unique, sequential numeric values automatically for a column. This is typically used for primary key columns to ensure each row has a unique identifier. The specific implementation of auto-increment may vary slightly between different database management systems (DBMS). Here are examples for some popular SQL database systems:

MySQL:

In MySQL, the auto-increment feature is achieved using the `AUTO_INCREMENT` attribute.

```SQL
CREATE TABLE example_table (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50)
);
INSERT INTO example_table (name) VALUES ('John');
INSERT INTO example_table (name) VALUES ('Jane');
-- The 'id' column will be automatically incremented
for each new record.
```

PostgreSQL:

In PostgreSQL, the SERIAL data type can be used to create an auto-increment column.

```SQL
CREATE TABLE example_table (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50)
);
INSERT INTO example_table (name) VALUES ('John');
INSERT INTO example_table (name) VALUES ('Jane');
-- The 'id' column will be automatically incremented for each new record.
```

Example:

In SQL Server, the IDENTITY property is used for auto-incrementing columns.

```SQL
CREATE TABLE example_table (
    id INT PRIMARY KEY IDENTITY(1,1),
    name VARCHAR(50)
);
```

```
INSERT INTO example_table (name) VALUES ('John');

INSERT INTO example_table (name) VALUES ('Jane');

-- The 'id' column will be automatically incremented
for each new record.
```

06 QUESTION

Explain about set operators in SQL

SQL provides several set operators that allow you to combine the results of multiple queries or tables. The main set operators include `UNION`, `UNION ALL`, `INTERSECT`, and `MINUS` (or `EXCEPT` in some database systems). Here's an explanation of each along with examples:

UNION:

The `UNION` operator is used to combine the result sets of two or more SELECT statements. It removes duplicate rows from the result set.

```SQL
SELECT column1, column2 FROM table1
UNION
SELECT column1, column2 FROM table2;
```

Example:

```SQL
-- Assuming two tables with similar structure
SELECT employee_id, employee_name FROM employees
UNION
SELECT vendor_id, vendor_name FROM vendors;
```

UNION ALL:

Similar to `UNION`, the `UNION ALL` operator combines the result sets of two or more SELECT statements. However, it does not remove duplicate rows, including all rows from each query.

```SQL
SELECT column1, column2 FROM table1
UNION
SELECT column1, column2 FROM table2;
```

Example:

```SQL
-- Assuming two tables with similar structure
SELECT employee_id, employee_name FROM employees
UNION
SELECT vendor_id, vendor_name FROM vendors;
```

INTERSECT:

The INTERSECT operator is used to retrieve the common rows that appear in both result sets of two SELECT statements.

```sql
SELECT column1, column2 FROM table1
INTERSECT
SELECT column1, column2 FROM table2;
```

Example:

```sql
-- Assuming two tables with similar structure
SELECT employee_id, employee_name FROM employees
INTERSECT
SELECT manager_id, manager_name FROM managers;
```

MINUS (or EXCEPT):

The MINUS or EXCEPT operator is used to retrieve the rows that are unique to the first SELECT statement and not present in the result set of the second SELECT statement.

```sql
SELECT column1, column2 FROM table1
MINUS
SELECT column1, column2 FROM table2;
```

Example:

```SQL
-- Assuming two tables with similar structure
SELECT customer_id, customer_name FROM customers
MINUS
SELECT order_customer_id, order_customer_name FROM
orders;
```

07 QUESTION

Give SQL query to retrieve nth record from an employee table

Method 1: Using LIMIT and OFFSET

```SQL
SELECT * FROM Employee

ORDER BY <column_name> -- Here, <column_name> is the column according to which the rows are ordered (e.g., ID).

OFFSET 5 ROWS            -- N - 1 = 6 - 1 = 5, so we skip the first 5 rows

FETCH NEXT 1 ROWS ONLY; -- Retrieve the next 1 row
```

Explanation:

- **ORDER BY:** It specifies the column based on which the ordering of rows is done.
- **OFFSET 5 ROWS:** It skips the first 5 rows, so the starting point is the 6th row.
- **FETCH NEXT 1 ROWS ONLY:** It retrieves the next 1 row after the offset, giving you the 6th row.

Method 2: Using ROWNUM

```SQL
SELECT *
FROM (
    SELECT *, ROW_NUMBER() OVER (ORDER BY
Employee_ID) AS RowNum
    FROM Employees
) AS sub
WHERE RowNum = 8;
```

Explanation:

- **ROW_NUMBER():** It assigns a unique number to each row based on the specified ordering. In this case, it's ordered by `Employee_ID`.
- **SELECT , ROW_NUMBER() OVER (ORDER BY Employee_ID) AS RowNum:* It selects all columns and also calculates the row number using `ROW_NUMBER()` function, creating a subquery.
- **WHERE RowNum = 8:** It filters out only the row with the desired row number, in this case, the 8th row.

08 QUESTION

Explain how to get unique records without using the DISTINCT keyword.

Let's consider an example with a hypothetical table called `employee` that contains information about employees, including their `employee_id` and `employee_name`. In this example, we want to retrieve distinct rows without using the DISTINCT clause, and we'll demonstrate four different approaches.

Assuming the `employee` table looks like this:

```SQL
CREATE TABLE employee (
    employee_id INT,
    employee_name VARCHAR(255)
);

INSERT INTO employee VALUES (1, 'John Doe');
INSERT INTO employee VALUES (2, 'Jane Doe');
INSERT INTO employee VALUES (3, 'John Doe');
INSERT INTO employee VALUES (4, 'Bob Smith');
INSERT INTO employee VALUES (5, 'Jane Doe');
```

GROUP BY Clause:

```SQL
SELECT employee_id, employee_name
FROM employee
GROUP BY employee_id, employee_name;
```

This query groups the rows by `employee_id` and `employee_name`, effectively giving us distinct combinations.

UNION Operator:

```SQL
SELECT employee_id, employee_name FROM employee
UNION
SELECT employee_id, employee_name FROM employee;
```

The UNION operator combines the results of two SELECT statements, and it automatically eliminates duplicate rows.

INTERSECT Operator:

```SQL
SELECT employee_id, employee_name FROM employee
INTERSECT
SELECT employee_id, employee_name FROM employee;
```

The INTERSECT operator returns only the rows that are common to both SELECT statements, effectively giving us distinct rows.

CTE & row_number() Function:

```SQL
WITH cte AS (
    SELECT employee_id, employee_name,
           ROW_NUMBER() OVER (PARTITION BY
employee_id, employee_name ORDER BY employee_id) AS
row_num
    FROM employee
)
SELECT employee_id, employee_name
FROM cte
WHERE row_num = 1;v
```

The INTERSECT operator returns only the rows that are common to both SELECT statements, effectively giving us distinct rows.

09 QUESTION

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

To select all even or all odd records from a table in SQL, you can use the modulo operator (%). The modulo operator calculates the remainder of a division operation. If you divide a number by 2 and the remainder is 0, the number is even; otherwise, it's odd. Here is an example:

Assuming you have a table named `your_table` with a primary key column named `id`, you can use the following queries:

Select All Even Records:

SQL

```
SELECT *
FROM your_table
WHERE id % 2 = 0;
```

This query selects all records where the value in the id column is even.

Select All Odd Records:

```SQL
SELECT *
FROM your_table
WHERE id % 2 <> 0;
```

This query selects all records where the value in the id column is odd. The <> operator means "not equal to."

10 QUESTION

Explain about clustered index?

A clustered index is established under two specific conditions:

1. The data or file being moved into secondary memory must be in sequential or sorted order.
2. A unique key value is required, meaning there should be no repeated values.

When implementing clustered indexing in a table, it organizes the data within that table. Similar to a primary key, only one clustered index can be created for a table. A clustered index can be likened to a dictionary, arranging data in alphabetical order.

In a clustered index, the index contains a pointer to a block, not directly to the data.

Example of Clustered Index:

If a primary key is applied to any column, it automatically becomes a clustered index.

```SQL
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DepartmentID INT
);
INSERT INTO Employee VALUES (101, 'John', 'Doe', 1);
INSERT INTO Employee VALUES (102, 'Jane', 'Smith',
2);
INSERT INTO Employee VALUES (103, 'Bob', 'Johnson',
1);
```

In this example, the `EmployeeID` column, which is a primary key, serves as a clustered index. The output of queries on this table will display results sorted by `EmployeeID` in ascending order.

You can have only one clustered index in a table, but a composite index can be created on multiple columns, allowing for a more complex indexing structure.

11 QUESTION

Explain about non clustered index?

The concept of a non-clustered index can be likened to the index of a book. Similar to how a book's index consists of chapter names and page numbers, allowing quick access to specific topics, a non-clustered index in a database provides a way to swiftly locate data without the need to traverse every record.

In contrast to a clustered index where data and index are stored together, a non-clustered index keeps the data and index in separate locations. This distinction allows for the existence of multiple non-clustered indexes within a single table.

In a non-clustered index, the index contains pointers to the data.

Example of Non-Clustered Index:

Consider the following code, which creates a table named "Employee" with columns such as "EmployeeID," "FirstName," "LastName," and "DepartmentID." The primary key is set on the "EmployeeID" column. Three records are inserted into the table, and a non-clustered index named "NIX_LastName" is established on the "LastName" column in ascending order.

```SQL
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DepartmentID INT
);

INSERT INTO Employee VALUES (101, 'John', 'Doe', 1);
INSERT INTO Employee VALUES (102, 'Jane', 'Smith',
2);
INSERT INTO Employee VALUES (103, 'Bob', 'Johnson',
1);

CREATE NONCLUSTERED INDEX NIX_LastName
ON Employee (LastName ASC);
```

In this example, the "EmployeeID" column acts as the primary key, automatically creating a clustered index. If there is a need for a non-clustered index on the "LastName" column in ascending order, a separate index structure is created for efficient data retrieval.

Output:

Employees Table:

EmployeeID	FirstName	LastName	DepartmentID
101	John	Doe	1
102	Jane	Smith	2
103	Bob	Johnson	1

In this output, the "Employee" table is displayed, showcasing the records with columns for "EmployeeID," "FirstName," "LastName," and "DepartmentID." The non-clustered index "NIX_LastName" allows for efficient searching based on the "LastName" column, providing quick access to the relevant row addresses.

12 QUESTION

Explain about Recursive CTE with an example.

A recursive Common Table Expression (CTE) in SQL is used when you need to query hierarchical or tree-structured data, such as an organizational chart or a file system. Recursive CTEs allow you to traverse a hierarchical structure by repeatedly executing a SELECT statement until a specified condition is met.

Let's consider an example of an employee table with columns `EmployeeID` and `ManagerID`, where `ManagerID` refers to the `EmployeeID` of the manager:

SQL

```
CREATE TABLE Employee (
    EmployeeID INT,
    ManagerID INT,
    EmployeeName VARCHAR(50)
);
INSERT INTO Employee VALUES (1, NULL, 'CEO');
INSERT INTO Employee VALUES (2, 1, 'Manager1');
INSERT INTO Employee VALUES (3, 1, 'Manager2');
INSERT INTO Employee VALUES (4, 2, 'Employee1');
INSERT INTO Employee VALUES (5, 2, 'Employee2');
```

```
INSERT INTO Employee VALUES (6, 3, 'Employee3');
INSERT INTO Employee VALUES (7, 3, 'Employee4');
```

Now, let's use a recursive CTE to retrieve the hierarchy for a specific manager, starting from the CEO. Suppose we want to find all employees under Manager2:

SQL

```
WITH RecursiveCTE AS (
    SELECT EmployeeID, ManagerID, EmployeeName
    FROM Employee
    WHERE EmployeeID = 3 -- Starting from Manager2
    UNION ALL
    SELECT e.EmployeeID, e.ManagerID, e.EmployeeName
    FROM Employee e
    INNER JOIN RecursiveCTE r ON e.ManagerID =
r.EmployeeID
)
SELECT * FROM RecursiveCTE;
```

The base case of the recursive CTE is the SELECT statement before the UNION ALL. It selects the initial row where EmployeeID is 3 (Manager2).

The recursive part comes after the UNION ALL, where we reference the CTE (RecursiveCTE) in the SELECT statement and join it with the Employee table based on the ManagerID and EmployeeID relationship.

The recursive CTE continues to expand until no more rows are selected, meaning that there are no more employees managed by the current set of results.

The final SELECT statement outside the CTE retrieves all rows from the recursive CTE.

In this example, the query will return the hierarchy under Manager2, including Manager2, Employee3, and Employee4.

13 QUESTION

Explain difference between RANK and DENSE_RANK in SQL

In SQL, the `RANK()` and `DENSE_RANK()` are window functions that are used to assign a rank to each row within a result set based on the values in one or more columns. Both functions are commonly used in scenarios where you want to analyze and compare the relative positions of rows based on certain criteria.

Here's the key difference between `RANK()` and `DENSE_RANK()`:

RANK():

The `RANK()` function assigns a unique rank to each distinct row within the result set.

If two or more rows have the same values, they receive the same rank, and the next rank is skipped.

After assigning ranks, there may be gaps in the ranking sequence if there are tied values.

Example:

```SQL
SELECT
    column1,
    column2,
```

```
      RANK() OVER (ORDER BY column1) AS ranking
FROM
    your_table;
```

DENSE_RANK():

- The `DENSE_RANK()` function also assigns a unique rank to each distinct row within the result set.
- If two or more rows have the same values, they receive the same rank, but the next rank is not skipped. There are no gaps in the ranking sequence.
- This function is useful when you want to avoid gaps in the ranking and want a continuous sequence.

Example:

SQL

```
SELECT
    column1,
    column2,
    DENSE_RANK() OVER (ORDER BY column1) AS
dense_ranking
FROM
    your_table;
```

In both examples, `column1` is the column based on which the ranking is calculated. The `ORDER BY` clause specifies the column(s) by which the ranking should be determined. You can adjust the `ORDER BY` clause to rank based on different columns or criteria.

Assuming you have a table like this:

column1	column2
10	A
20	B
10	C
30	D

RANK() Example Output:

column1	column2	column3
10	A	1
10	C	1
20	B	3
30	D	4

DENSE_RANK() Example Output:

column1	column2	column3
10	A	1
10	C	1
20	B	2
30	D	3

In the `RANK()` example, there is a gap in the ranking sequence between 3 and 4 because two rows share the same rank. In the `DENSE_RANK()` example, there are no gaps in the ranking sequence, and the ranks are assigned continuously.

14 QUESTION

What is a correlated query?

A correlated subquery is a type of SQL subquery where the inner query depends on the outer query. In other words, the inner query references one or more columns from the outer query. The result of the inner query is then used to determine the result of the outer query.

Let's look at an example to illustrate a correlated subquery. Consider a database with two tables: "employees" and "salaries." The "employees" table contains information about employees, including their ID and name, while the "salaries" table contains information about their salaries.

Here are the sample tables:

SQL

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(50)
);
CREATE TABLE salaries (
    employee_id INT,
    salary INT,
    FOREIGN KEY (employee_id) REFERENCES
employees(employee_id)
);
```

```
INSERT INTO employees VALUES (1, 'John');
INSERT INTO employees VALUES (2, 'Jane');
INSERT INTO salaries VALUES (1, 50000);
INSERT INTO salaries VALUES (2, 60000);
```

Now, suppose you want to find employees whose salary is greater than the average salary for their department. You can use a correlated subquery for this:

SQL

```
SELECT
    e.employee_id,
    e.employee_name,
    s.salary
FROM
    employees e
JOIN
    salaries s ON e.employee_id = s.employee_id
WHERE
    s.salary > (
        SELECT AVG(salary)
        FROM salaries
        WHERE employee_id = e.employee_id
    );
```

In this example: The outer query selects employee information from the "employees" and "salaries" tables.

The inner query calculates the average salary for a specific employee's department. Note that it references the employee_id from the outer query (correlated).

The WHERE clause of the outer query compares each employee's salary with the average salary for their department, and only returns rows where the salary is greater.

15 QUESTION

Explain the difference between OLAP (Online Analytical Processing) and OLTP (Online Transaction Processing) databases.

	OLTP System Online Transaction Processing (Operational System)	OLAP System Online Analytical Processing (Data Warehouse)
Source Of data	Operational data; OLTPs are the original source of the data.	Consolidation data; OLAP data comes from the various OLTP Databases
Purpose of data	To control and run fundamental business tasks	To help with planning, problem solving, and decision support
What the data	Reveals a snapshot of ongoing business processes	Multi-dimensional views of various kinds of business activities
Insets and Updates	Short and fast inserts and updates initiated by end Users	Periodic long-running batch jobs refresh the data
Queries	Relatively standardized and simple queries Returning relatively few records	Often complex queries involving aggregations

Processing Speed	Typically very fast	Depends on the amount of data involved; batch data refreshes and complex queries may take many hours; query speed can be improved by creating indexes
Space Requirements	Can be relatively small if historical data is archived	Larger due to the existence of aggregation structures and history data; requires more indexes than OLTP
Database Design	Highly normalized with many tables	Typically de-normalized with fewer tables; use of star and/or snowflake schemas
Backup and Recovery	Backup religiously; operational data is critical to run the business, data loss is likely to entail significant monetary loss and legal liability	Instead of regular backups, some environments may consider simply reloading the OLTP data as a recovery method

16 QUESTION

What are MAGIC TABLES in SQL?

In SQL Server, "magic tables" is a term commonly used to refer to two special tables, `INSERTED` and `DELETED`, that are automatically created and populated during the execution of triggers. These tables are used in the context of DML (Data Manipulation Language) triggers, which are activated in response to changes in data, such as INSERT, UPDATE, or DELETE operations on a table.

1. INSERTED Table:

- The `INSERTED` table is used in the context of an `AFTER INSERT` or `INSTEAD OF INSERT` trigger.
- It contains the rows that were inserted or affected by the `INSERT` statement that triggered the trigger.
- The columns in the `INSERTED` table correspond to the columns in the table on which the trigger is defined.

2. DELETED Table:

- The `DELETED` table is used in the context of `AFTER UPDATE`, `AFTER DELETE`, or `INSTEAD OF DELETE` triggers.
- It contains the rows that were either updated or deleted by the UPDATE or `DELETE` statement that triggered the trigger.
- The columns in the `DELETED` table also correspond to the columns in the table on which the trigger is defined.

These tables allow you to access the old and new values of rows affected by the triggering operation within the body of the trigger. You can use them to perform actions based on the changes made to the data, compare values before and after the operation, or log changes for auditing purposes.

17 QUESTION

What are the different ways to fetch the first 5 characters of the string in SQL?

SUBSTRING() Function:

You can use the `SUBSTRING()` function to extract a portion of the string. For example:

```SQL
SELECT SUBSTRING(column_name, 1, 5) AS
first_five_chars

FROM your_table;
```

You can use the SUBSTRING() function to extract a portion of the string. For example:

LEFT() Function:

The `LEFT()` function can also be used to retrieve the leftmost characters from a string:

```SQL
SELECT LEFT(column_name, 5) AS first_five_chars

FROM your_table;
```

This query achieves the same result as the `SUBSTRING()` approach.

18 QUESTION

Given three tables - Orders, Customers, and Products - write a query to find the names of customers who have never placed an order.

To find the names of customers who have never placed an order, you can use a query with a LEFT JOIN between the Customers table and the Orders table, and then filter out the rows where there is no matching order. Here's an example query:

```SQL
SELECT Customers.CustomerName

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID =
Orders.CustomerID

WHERE Orders.CustomerID IS NULL;
```

In this query:

- `Customers` is the table containing customer information.
- `Orders` is the table containing order information.
- `Customers.CustomerID` and `Orders.CustomerID` are assumed to be the columns linking the two tables.

The `LEFT JOIN` is used to retrieve all records from the `Customers` table and the matching records from the `Orders` table. The `WHERE` clause filters out the rows where there is no matching order, meaning customers who have never placed an order.

19 QUESTION

What is conditional aggregation in SQL?

Conditional aggregation in SQL involves applying aggregate functions like SUM, COUNT, AVG, etc., based on certain conditions specified in the query. This allows you to perform aggregate calculations selectively on rows that meet specific criteria.

Let's consider an example to illustrate conditional aggregation. Suppose you have a table named "sales" with columns "product_id," "quantity_sold," and "sale_date." You want to calculate the total quantity of products sold and the average quantity sold per day, but only for sales that occurred after a certain date.

Here's an example query using conditional aggregation:

SQL

```
SELECT
    SUM(CASE WHEN sale_date > '2023-01-01' THEN
quantity_sold ELSE 0 END) AS
total_quantity_sold_after_date,
    AVG(CASE WHEN sale_date > '2023-01-01' THEN
quantity_sold ELSE NULL END) AS
avg_quantity_sold_after_date
FROM
    sales;
```

In this example:

- The `SUM` function is used with a `CASE` statement to calculate the total quantity sold after the specified date. The `CASE` statement checks if the sale_date is greater than '2023-01-01', and if true, it includes the corresponding quantity_sold; otherwise, it includes 0.
- The `AVG` function is used with a `CASE` statement to calculate the average quantity sold after the specified date. Again, the `CASE` statement checks if the sale_date is greater than '2023-01-01', and if true, it includes the corresponding quantity_sold; otherwise, it includes NULL. The `AVG` function automatically ignores NULL values when calculating the average.

20 QUESTION

Write a query that delivers the names of all employees who work in the same department as the employee with the highest salary.

SQL

```
-- Selecting the employee names who work in the same department as the employee with the highest salary

SELECT e2.employee_name

-- Joining the employees table with itself using aliases e1 and e2

FROM employees e1

JOIN employees e2 ON e1.department_id = e2.department_id

-- Filtering the result to include only those employees whose salary is equal to the maximum salary in the employees table

WHERE e1.salary = (SELECT MAX(salary) FROM employees);
```

- `SELECT e2.employee_name:` This part of the query is selecting the employee_name from the table aliased as e2.
- `FROM employees e1:` This specifies the main table as employees and aliases it as e1. This is used to identify the employee with the highest salary.
- `JOIN employees e2 ON e1.department_id = e2.department_id:` This is a self-join, where the table is joined with itself using different aliases (e1 and e2). It is joining based on the department_id to find employees who work in the same department.
- `WHERE e1.salary = (SELECT MAX(salary) FROM employees):` This is the condition that filters the results. It ensures that only employees from the same department as the one with the maximum salary are selected.

Did you find it Useful?

Leave a comment!

Alamin CodePapa

@CodePapa360

FOLLOW FOR MORE

Like Comment Repost