ADVANCED SQL OPERATIONS

Advanced SQL Operations

- Window Functions
- Common Table Expressions (CTEs)
- PIVOT and UNPIVOT Operations
- Analytical Functions
- Partitioning
- SET Operations

Window Functions

Aggregation Window Functions

Ranking Window Functions

Analytical Window Functions

Cumulative Distribution Functions

Window Functions

Aggregation Window Functions

- Perform calculations across rows related to the current row and return a single value for each row in the result set.

- They can be any of the standard aggregation functions like SUM(), AVG(), COUNT(), MIN(), and MAX(), but used over a specific "window" of rows.

Window Functions

Ranking Window Functions

Ranking window functions in SQL are a subset of window functions specifically designed to assign ranks to each row within a partition of the result set.

ROW_NUMBER()

RANK()

DENSE_RANK()

NTILE(n)

Window Functions

Ranking Window Functions

ROW_NUMBER() :

- Assigns a unique sequential integer to rows within a partition of the result set, starting from 1.

```
 SELECT sale_date, amount, ROW_NUMBER() OVER (PARTITION BY sale_date ORDER BY amount DESC) AS rank
FROM sales;
```

Window Functions

Ranking Window Functions

RANK() :

- Assigns a rank to each row within a partition, with the same rank assigned to rows that have identical values as defined by the ORDER BY clause.
- Gaps are introduced in the ranking sequence for tied ranks.

```
SELECT student_id, score, RANK() OVER (ORDER BY score DESC) AS rank
FROM exams;
```

Window Functions

Ranking Window Functions

DENSE_RANK() :

- Similar to RANK(), but DENSE_RANK() does not introduce gaps in the ranking sequence for tied ranks.
- Each consecutive rank is incremented by 1, regardless of ties.

```
SELECT product_id, sales, DENSE_RANK() OVER (ORDER BY sales DESC) AS rank
FROM product_sales;
```

Window Functions

Ranking Window Functions

NTILE(n):

- Divides the rows in an ordered partition into a specified number of approximately equal groups, n, and assigns a group number to each row.

```
SELECT student_id, score, NTILE(4) OVER (ORDER BY score DESC) AS quartile
FROM exams;
```

Window Functions

Analytical Window Functions

Analytical window functions in SQL extend the capabilities of standard SQL queries by allowing you to perform complex calculations across a set of rows that are related to the current row, much like ranking window functions.

LAG()

LEAD()

FIRST_VALUE()

LAST_VALUE()

PERCENT_RANK()

CUME_DIST()

Window Functions

Analytical Window Functions

LAG():

- Accesses data from a previous row in the partition without the need for a self-join. It's useful for comparing current row values with those of a preceding row.

```
SELECT sale_date, amount, LAG(amount, 1) OVER (ORDER BY sale_date) AS previous_day_sales
FROM sales;
```

Window Functions

Analytical Window Functions

LEAD():

- Accesses data from a following row in the partition, similar to LAG but looks ahead instead of behind. This function is handy for forecasting or planning scenarios.

```
SELECT sale_date, amount, LEAD(amount, 1) OVER (ORDER BY sale_date) AS next_day_sales
FROM sales;
```

Window Functions

Analytical Window Functions

FIRST_VALUE():

- These function allow you to fetch the first value in a specified partition. They're useful for comparing all rows in a partition against a common value.

```
SELECT sale_date, amount,
     FIRST_VALUE(amount) OVER (PARTITION BY MONTH(sale_date) ORDER BY sale_date) AS
first_sale_of_month,
FROM sales;
```

Window Functions

Analytical Window Functions

LAST_VALUE():

- These function allow you to fetch the last value in a specified partition. They're useful for comparing all rows in a partition against a common value.

```
SELECT sale_date, amount,
    LAST_VALUE(amount) OVER (PARTITION BY MONTH(sale_date) ORDER BY sale_date ROWS BETWEEN
UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS last_sale_of_month
FROM sales;
```

Window Functions

Analytical Window Functions

PERCENT_RANK() :

- This function is used for statistical analysis. PERCENT_RANK() calculates the relative rank of a row within a partition as a percentage.

```
SELECT sale_date, amount,
      PERCENT_RANK() OVER (PARTITION BY MONTH(sale_date) ORDER BY amount) AS percent_rank,
FROM sales;
```

Window Functions

Analytical Window Functions

CUME_DIST() :

- This function is used for statistical analysis. CUME_DIST() calculates the cumulative distribution of a value within a partition.

```
SELECT sale_date, amount,
    CUME_DIST() OVER (PARTITION BY MONTH(sale_date) ORDER BY amount) AS cumulative_distribution
FROM sales;
```

Common Table Expressions (CTEs) :

- Common Table Expressions (CTEs) in MySQL are temporary result sets that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.
- CTEs provide a way to create more readable and modular queries by encapsulating complex subqueries, making it easier to understand and maintain complex SQL queries.
- They were introduced in MySQL 8.0, aligning MySQL with other SQL databases that already supported this feature.

```
WITH AvgSalary AS (
   SELECT AVG(salary) AS average FROM employees
)
SELECT name, salary
FROM employees
WHERE salary > (SELECT average FROM AvgSalary);
```

Pivot & UnPivot :

Pivoting Data in MySQL:

- Pivoting transforms rows into columns, effectively turning unique values from one column into multiple columns in the output, with another column's values as the cells under these new columns.

- This is commonly used in reporting and data analysis to transform data into a more readable or useful format.

```
SELECT year,
    SUM(CASE WHEN product = 'Product A' THEN amount ELSE 0 END) AS `Product A`,
    SUM(CASE WHEN product = 'Product B' THEN amount ELSE 0 END) AS `Product B`,
    SUM(CASE WHEN product = 'Product C' THEN amount ELSE 0 END) AS `Product C`
FROM sales
GROUP BY year;
```

Pivot & UnPivot :

Unpivoting Data in MySQL:

- Unpivoting does the opposite of pivoting; it transforms columns into rows, often to normalize the data structure or prepare it for further operations that require a long format.

```
SELECT year, 'Product A' AS product, `Product A` AS amount FROM sales_summary
UNION ALL
SELECT year, 'Product B', `Product B` FROM sales_summary
UNION ALL
SELECT year, 'Product C', `Product C` FROM sales_summary;
```

Partitioning :

- Partitioning in MySQL is a database design technique that divides tables into smaller, more manageable pieces while still treating them as a single table.
- This approach can significantly improve performance for large tables by enabling more efficient data access patterns, especially for operations involving large datasets.
- Partitioning can help with faster data retrieval (queries), more efficient data maintenance operations (like backups, restores, and deletes), and improved overall database performance.

```
SELECT year, 'Product A' AS product, `Product A` AS amount FROM sales_summary
UNION ALL
SELECT year, 'Product B', `Product B` FROM sales_summary
UNION ALL
SELECT year, 'Product C', `Product C` FROM sales_summary;
```

Set Operations:

- SET operations in SQL are used to combine the results of two or more SELECT statements. The most common set operations are UNION, INTERSECT, and EXCEPT (or MINUS).

- INTERSECT and EXCEPT are not available in MySQL, but you can achieve similar results with INNER JOIN, LEFT JOIN, and WHERE NOT EXISTS or NOT IN.

UNION

UNION ALL

INTERSECT

EXCEPT

Set Operations:

UNION

UNION :

- Combines the results of two or more SELECT statements into a single result set.

```
(SELECT name FROM departments)
UNION
(SELECT title FROM positions);
```

Set Operations:

UNION ALL

UNION ALL :

- Similar to UNION, but includes duplicates.

```
(SELECT location AS name_or_location FROM departments)
UNION ALL
(SELECT name FROM employees);
```

Set Operations:

INTERSECT

INTERSECT :

- Returns the rows that two SELECT statements have in common.

```
SELECT name FROM employees
WHERE EXISTS (
   SELECT 1 FROM departments WHERE departments.name = employees.name
);
```

Set Operations:

EXCEPT

EXCEPT :

- Returns the rows from the first SELECT statement that are not present in the second SELECT statement.

```
SELECT name FROM departments d
WHERE NOT EXISTS (
   SELECT 1 FROM employees e WHERE d.name = e.name
);
```