SET OPERATORS

Set operators—UNION, UNION ALL, EXCEPT, and INTERSECT—are essential SQL operations that allow combining the results of multiple `SELECT` queries. Each set operator serves a unique purpose, enabling industries to manage and compare datasets effectively. Here's a detailed explanation of each, along with real-life examples from different industries:

1. UNION

1. Definition: Combines the result sets of two or more `SELECT` queries into a single result set, removing any duplicate rows.
2. Use Case: Used when you want to merge distinct data from multiple sources or queries.

Industry-Specific Examples:

a. Finance: Combine customer records from multiple branches, ensuring no duplicate entries.

- Example: Fetch a distinct list of customers from two different branches.

```
SELECT customer_id, customer_name FROM branch_a_customers
UNION
SELECT customer_id, customer_name FROM branch_b_customers;
```

b. Healthcare: Combine patient records from different departments without duplication.

- Example: Retrieve a unique list of all patients from the surgery and cardiology departments.

```
SELECT patient_id, patient_name FROM surgery_patients
UNION
SELECT patient_id, patient_name FROM cardiology_patients;
```

c. Education: Merge student enrollments from different semesters into a distinct list.

- Example: Create a list of all unique students who enrolled in both the Spring and Fall semesters.

```
SELECT student_id, student_name FROM spring_enrollments
UNION
SELECT student_id, student_name FROM fall_enrollments;
```

2. UNION ALL

1. Definition: Combines the result sets of two or more `SELECT` queries, but does not remove duplicate rows.
2. Use Case: Used when you want to combine data, including duplicates, for further analysis or aggregation.

Industry-Specific Examples:

a. Retail: Combine sales transactions from multiple stores, including repeated transactions.

- Example: Fetch all sales data from two stores, keeping track of duplicate sales records for comparison.

```
SELECT transaction_id, store_id FROM store_a_sales

UNION ALL

SELECT transaction_id, store_id FROM store_b_sales;
```

b. Logistics: Combine shipment records from different warehouses, even if the same shipment is processed multiple times.

- Example: Retrieve all shipment data from warehouse A and warehouse B, including duplicated entries.

```
SELECT shipment_id, warehouse_id FROM warehouse_a_shipments

UNION ALL

SELECT shipment_id, warehouse_id FROM warehouse_b_shipments;
```

c. Telecom: Merge customer usage records from different months, allowing duplicates for detailed usage analysis.

- Example: Combine the call records of customers across January and February for further analysis of repeat users.

```
SELECT customer_id, call_duration FROM january_usage

UNION ALL

SELECT customer_id, call_duration FROM february_usage;
```

3. EXCEPT (or `MINUS` in some databases)

- Definition: Returns the rows from the first `SELECT` query that are not present in the second `SELECT` query.

- Use Case: Used to find differences between two datasets, useful for data comparison and auditing.

Industry-Specific Examples:

a. Finance: Identify customers who made payments last year but not this year.

- Example: Find the list of customers who made a payment in 2022 but did not make any in 2023.

```
SELECT customer_id FROM payments_2022

EXCEPT

SELECT customer_id FROM payments_2023;
```

b. Healthcare: Find patients who visited last year but haven't returned this year.

- Example: Retrieve the list of patients who visited the hospital in 2022 but not in 2023.

```
SELECT patient_id FROM visits_2022

EXCEPT

SELECT patient_id FROM visits_2023;
```

c. Education: Identify students who enrolled in the previous semester but are not enrolled in the current semester.

- Example: List of students who enrolled in the Fall semester but did not return for the Spring semester.

```
SELECT student_id FROM fall_enrollments

EXCEPT

SELECT student_id FROM spring_enrollments;
```

4. INTERSECT

1. Definition: Returns the rows that are common to both `SELECT` queries.
2. Use Case: Used when you need to find commonalities between datasets, such as shared records between two systems or departments.

Industry-Specific Examples:

a. Retail: Identify products sold in both store A and store B.

- Example: Fetch the list of products that were sold in both Store A and Store B.

```
SELECT product_id FROM store_a_sales

INTERSECT

SELECT product_id FROM store_b_sales;
```

b. Telecom: Find customers who have subscribed to both internet and mobile services.

- Example: Retrieve the list of customers who have both internet and mobile services.

```
SELECT customer_id FROM internet_customers

INTERSECT

SELECT customer_id FROM mobile_customers;
```

c. Logistics: Identify shipments that were processed by both warehouse A and warehouse B.

- Example: Find the list of shipments handled by both warehouses.

```
SELECT shipment_id FROM warehouse_a_shipments

INTERSECT

SELECT shipment_id FROM warehouse_b_shipments;
```

Conclusion:

Set operators are vital for businesses and industries that manage large datasets. They help in tasks like:

1. Data Integration: Merging records from different systems (e.g., sales from multiple stores).
2. Data Comparison: Identifying differences between time periods, locations, or datasets (e.g., comparing yearly customer activity).
3. Data Auditing: Finding common or unique records across different data sources (e.g., auditing customer overlap between services).

By using these operators, industries can improve decision-making, streamline processes, and extract meaningful insights from complex data.