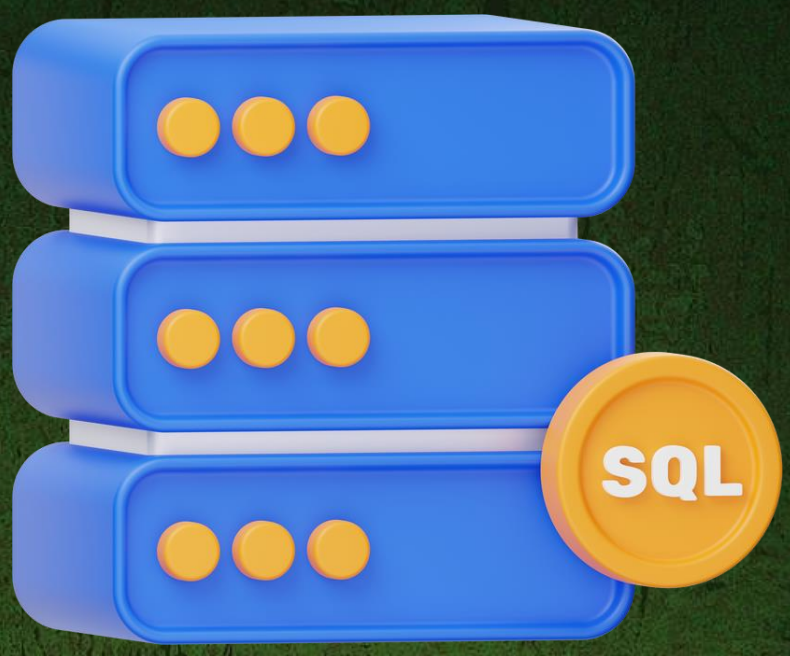
***Database Using SQL Server & Raw Data available over my GitHub Profile***




MySQL

PostgreSQL

# SET Operators With 12 Most Used Queries

***AnshLibrary***

# What is a JOINs

Set operators allow you to **combine the results** of two or more SELECT statements into a single result set — similar to how sets work in mathematics.

## Types Of Set Operators:

| Operator | Description |
|---|---|
| UNION | Combines results and removes duplicates |
| UNION ALL | Combines results including duplicates |
| INTERSECT | Returns rows common to both queries |
| EXCEPT | Returns rows from first query not in second |

## Syntax :

```
SELECT
   FirstName
   LastName
FROM Customers
JOIN Clause
WHERE Clause
GROUP BY Clause
```

1st SELECT Statement

SET Operator

```
UNION
```

```
SELECT
   FirstName
   LastName
FROM Employees
JOIN Clause
WHERE Clause
GROUP BY Clause
```

2nd SELECT Statement

ORDER BY
can be used only at the
end to sort the final Result

```
ORDER BY FirstName
```

# SET RULES

1. SET operators can be used in any clause.
2. ORDER BY is allowed only once—at the end of the query.
3. Each query must have the same number of columns.
4. Column data types must be compatible across queries.
5. The result set takes column names from the first query.

# 01

## UNION for Customer Order Tracking (with source tag)

```
SELECT TOP 10 OrderID, CustomerID, 'Current' AS Source
FROM Orders
UNION
SELECT OrderID, CustomerID, 'Archive' AS Source
FROM OrdersArchive;
```

| | OrderID | CustomerID | Source |
|---|---|---|---|
| 1 | 1 | 2 | Current |
| 2 | 2 | 3 | Current |
| 3 | 3 | 1 | Current |
| 4 | 4 | 1 | Current |
| 5 | 5 | 2 | Current |
| 6 | 6 | 3 | Current |
| 7 | 7 | 1 | Current |
| 8 | 8 | 4 | Current |
| 9 | 9 | 2 | Current |
| 10 | 10 | 3 | Current |
| 11 | 1 | 2 | Archive |
| 12 | 2 | 3 | Archive |
| 13 | 3 | 1 | Archive |
| 14 | 4 | 1 | Archive |
| 15 | 5 | 2 | Archive |
| 16 | 6 | 3 | Archive |
| 17 | 7 | 3 | Archive |
| 18 | 8 | 4 | Archive |
| 19 | 9 | 2 | Archive |
| 20 | 10 | 1 | Archive |
| 21 | 11 | 5 | Archive |
| 22 | 12 | 6 | Archive |

# 02

## Total number of orders across all time

```
SELECT

  'AllTimeOrders' AS Label,
  COUNT(*) AS TotalOrders
FROM (

  SELECT OrderID FROM Orders
  UNION ALL
  SELECT OrderID FROM OrdersArchive

) AllOrders;
```

Results | Messages

| | Label | TotalOrders |
|---|---|---|
| 1 | AllTimeOrders | 50 |

**Use by:** UNION ALL + Aggregation

# 03

## Customers who never placed archived orders

```
SELECT Customer_id FROM Customers

EXCEPT

SELECT DISTINCT CustomerID FROM
OrdersArchive;
```

| | Customer_id |
|---|---|
| 1 | 10 |
| 2 | 11 |
| 3 | 12 |
| 4 | 13 |
| 5 | 14 |
| 6 | 15 |

AnshLibrary

# 04

## Compare recent vs old order volume

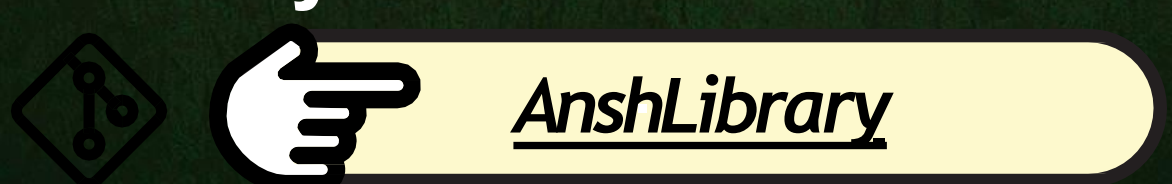
```
SELECT
  '2025' AS YearTag,
  COUNT(*) AS Total
FROM Orders
WHERE YEAR(OrderDate) = 2025
UNION ALL

SELECT
  '2024' AS YearTag,
  COUNT(*)
FROM OrdersArchive
WHERE YEAR(OrderDate) = 2024;
```

Results | Messages

| | YearTag | Total |
|---|---|---|
| 1 | 2025 | 30 |
| 2 | 2024 | 20 |

**Use by:** UNION ALL with Filtering



# 05

## INTERSECT on Employees involved in both current and archived sales

```
SELECT SalesPersonID
FROM Orders

INTERSECT

SELECT SalesPersonID
FROM OrdersArchive;
```

Results | Messages

| | SalesPersonID |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 4 |
| 4 | 5 |

# 06

## High-spending customers only in current data

```
SELECT CustomerID FROM (
  SELECT CustomerID, SUM(Sales)
AS TotalSpent
  FROM Orders
  GROUP BY CustomerID
  HAVING SUM(Sales) > 100
) curr
EXCEPT
SELECT CustomerID FROM
OrdersArchive
Where CustomerID NOT IN (2,4,6);
```

Results | Messages

| | CustomerID |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |

```
SELECT CustomerID FROM (
  SELECT CustomerID, SUM(Sales)
AS TotalSpent
  FROM Orders
  GROUP BY CustomerID
  HAVING SUM(Sales) > 100
) curr
EXCEPT
SELECT CustomerID FROM
OrdersArchive
Where CustomerID IN (2,4,6);
```

| | CustomerID |
|---|---|
| 1 | 1 |
| 2 | 3 |
| 3 | 5 |
| 4 | 9 |

**Use by:** EXCEPT with Aggregation and IN , NOT IN Operators

# 07

## Top 3 ordered products from each dataset

```
WITH TopCurr AS (
  SELECT TOP 3 ProductID, COUNT(*) AS Cnt
  FROM Orders
  GROUP BY ProductID
  ORDER BY COUNT(*) DESC
),
TopArch AS (
  SELECT TOP 3 ProductID, COUNT(*) AS Cnt
  FROM OrdersArchive
  GROUP BY ProductID
  ORDER BY COUNT(*) DESC
)

SELECT * FROM TopCurr
UNION
SELECT * FROM TopArch;
```

Results | Messages

| | ProductID | Cnt |
|---|---|---|
| 1 | 101 | 4 |
| 2 | 101 | 7 |
| 3 | 102 | 6 |
| 4 | 104 | 5 |
| 5 | 105 | 4 |

Use by: UNION with Multiple CTE

AnshLibrary

# 08

## Common product-customer pairs with sales over threshold

```
SELECT ProductID, CustomerID
FROM (
  SELECT ProductID, CustomerID, SUM(Sales) AS Total
FROM Orders
GROUP BY ProductID, CustomerID
 HAVING SUM(Sales) > 50
) a

INTERSECT

SELECT ProductID, CustomerID
FROM (
  SELECT ProductID, CustomerID, SUM(Sales) AS Total
FROM OrdersArchive
GROUP BY ProductID, CustomerID
HAVING SUM(Sales) > 50
) b;
```

| | ProductID | CustomerID |
|---|---|---|
| 1 | 102 | 3 |
| 2 | 103 | 4 |
| 3 | 104 | 3 |
| 4 | 105 | 1 |
| 5 | 107 | 6 |

**Using:** INTERSECT with Window

# 09

## Rank order volumes by year

```
SELECT

    OrderID,
    OrderDate,
    Sales,
    YEAR(OrderDate) AS Yr,
     RANK() OVER(PARTITION BY YEAR(OrderDate) ORDER
BY Sales DESC) AS RankByYear

FROM (
  SELECT OrderID, OrderDate, Sales FROM Orders
  UNION ALL
  SELECT OrderID, OrderDate, Sales FROM OrdersArchive
) all_data;
```

Results | Messages

| | OrderID | OrderDate | Sales | Yr | RankByYear |
|---|---|---|---|---|---|
| 1 | 14 | 2024-07-04 | 150.00 | 2024 | 1 |
| 2 | 15 | 2024-07-05 | 66.00 | 2024 | 2 |
| 3 | 13 | 2024-07-03 | 60.00 | 2024 | 3 |
| 4 | 4 | 2024-04-20 | 60.00 | 2024 | 3 |
| 5 | 4 | 2024-04-20 | 60.00 | 2024 | 3 |
| 6 | 7 | 2024-06-15 | 60.00 | 2024 | 3 |
| 7 | 12 | 2024-07-01 | 55.00 | 2024 | 7 |
| 8 | 6 | 2024-05-05 | 50.00 | 2024 | 8 |
| 9 | 6 | 2024-05-05 | 50.00 | 2024 | 8 |
| 10 | 6 | 2024-05-05 | 50.00 | 2024 | 8 |
| 11 | 8 | 2024-06-18 | 45.00 | 2024 | 11 |
| 12 | 16 | 2024-07-06 | 44.95 | 2024 | 12 |
| 13 | 11 | 2024-06-25 | 40.00 | 2024 | 13 |
| 14 | 10 | 2024-06-21 | 35.00 | 2024 | 14 |
| 15 | 9 | 2024-06-20 | 25.00 | 2024 | 15 |
| 16 | 5 | 2024-05-01 | 25.00 | 2024 | 15 |
| 17 | 3 | 2024-04-10 | 20.00 | 2024 | 17 |
| 18 | 17 | 2024-07-07 | 18.50 | 2024 | 18 |
| 19 | 2 | 2024-04-05 | 15.00 | 2024 | 19 |
| 20 | 1 | 2024-04-01 | 10.00 | 2024 | 20 |

**Use by:** UNION ALL with Partitioned Window

# 10

## Validate backup data using multiple fields

```
SELECT
  OrderID,
  CustomerID,
  ProductID,
 Quantity
FROM Orders
```

```
INTERSECT

SELECT
  OrderID,
  CustomerID,
  ProductID,
  Quantity
FROM OrdersArchive;
```

| | OrderID | CustomerID | ProductID | Quantity |
|---|---|---|---|---|
| 1 | 1 | 2 | 101 | 1 |
| 2 | 2 | 3 | 102 | 1 |
| 3 | 3 | 1 | 101 | 2 |
| 4 | 4 | 1 | 105 | 2 |
| 5 | 5 | 2 | 104 | 1 |
| 6 | 6 | 3 | 104 | 2 |

**Use by:** INTERSECT on Composite Keys

AnshLibrary

# 11

## Employees who were managers and also submitted orders to same customers

```
SELECT e.Employee_id, o.CustomerID
FROM Employees e
JOIN Orders o
ON e.Employee_id = o.SalesPersonID
WHERE EXISTS (
  SELECT 1 FROM Employees e2
WHERE e2.ManagerID = e.Employee_id
)

INTERSECT

SELECT e.Employee_id, oa.CustomerID
FROM Employees e
JOIN OrdersArchive oa
ON e.Employee_id = oa.SalesPersonID;
```

Results | Messages

| | Employee_id | CustomerID |
|---|---|---|
| 1 | 2 | 4 |
| 2 | 2 | 6 |
| 3 | 3 | 1 |
| 4 | 3 | 2 |
| 5 | 3 | 3 |
| 6 | 4 | 5 |
| 7 | 4 | 9 |

**Use Case**: Mixed-role activity analysis in legacy migration.

# 12

## Orders with mismatched ShipDate due to timezone conversion issue

```
SELECT
   OrderID,
   FORMAT(ShipDate, 'yyyy-MM-dd') AS ShipDate
FROM Orders

EXCEPT

SELECT
   OrderID,
   FORMAT(ShipDate, 'yyyy-MM-dd') AS ShipDate
FROM OrdersArchive;
```

Results | Messages

| | OrderID | ShipDate |
|---|---|---|
| 1 | 1 | 2025-01-05 |
| 2 | 2 | 2025-01-10 |
| 3 | 3 | 2025-01-25 |
| 4 | 4 | 2025-01-25 |
| 5 | 5 | 2025-02-05 |
| 6 | 6 | 2025-02-10 |
| 7 | 7 | 2025-02-27 |
| 8 | 8 | 2025-02-27 |
| 9 | 9 | 2025-03-15 |
| 10 | 10 | 2025-03-20 |
| 11 | 11 | 2025-07-05 |
| 12 | 12 | 2025-07-06 |
| 13 | 13 | 2025-07-07 |
| 14 | 14 | 2025-07-08 |
| 15 | 15 | 2025-07-09 |
| 16 | 16 | 2025-07-10 |
| 17 | 17 | 2025-07-11 |
| 18 | 18 | 2025-07-12 |

**Use Case:** Catch data loss during date/time format conversions in ETL.