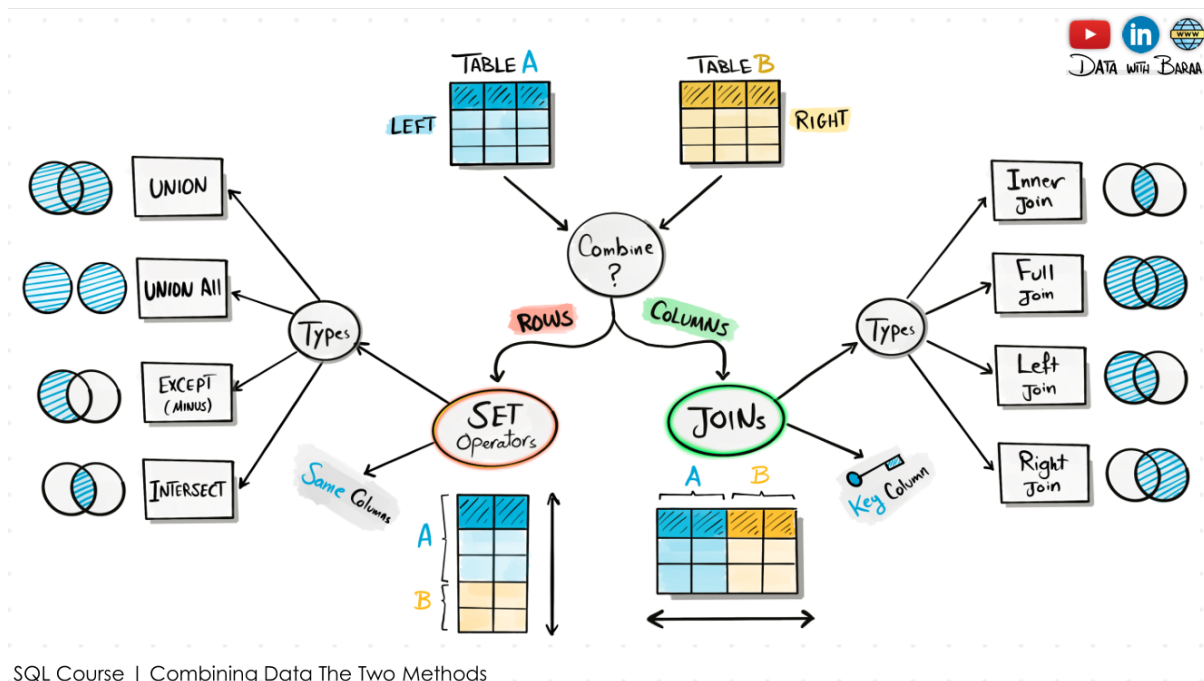
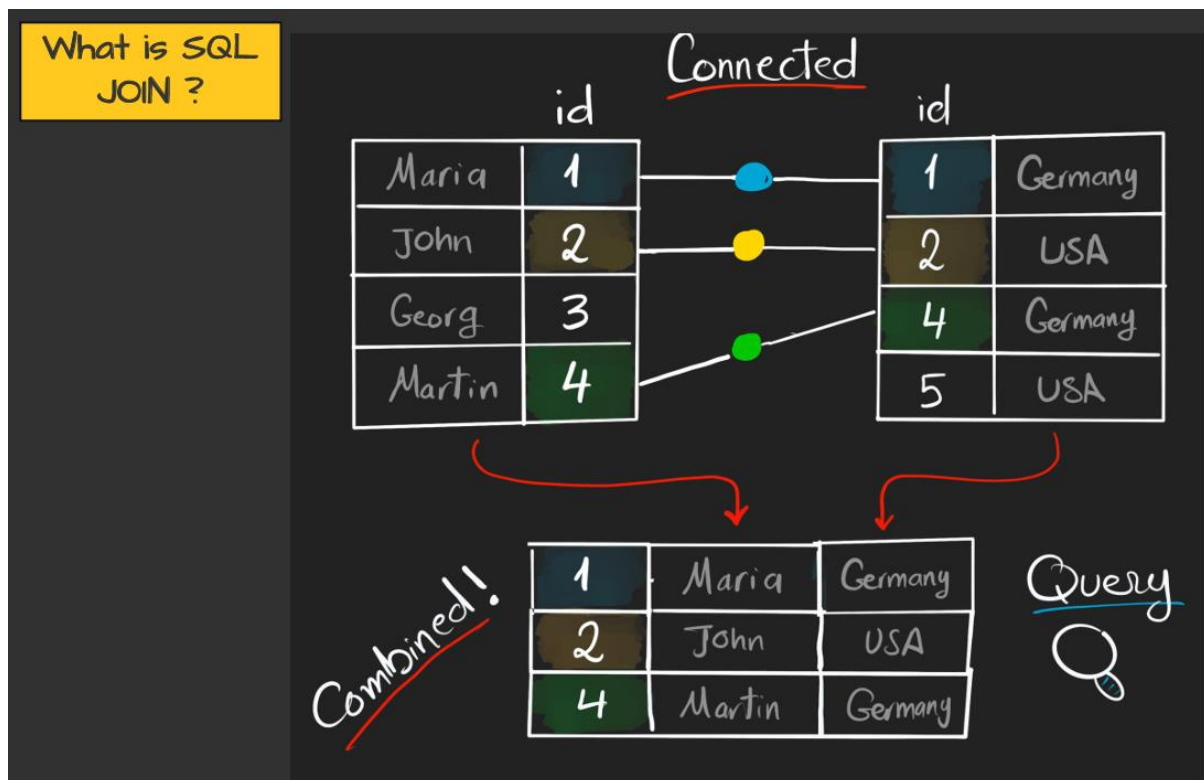


## COMBINING DATA:



## SQL JOINS:



## NO JOIN :

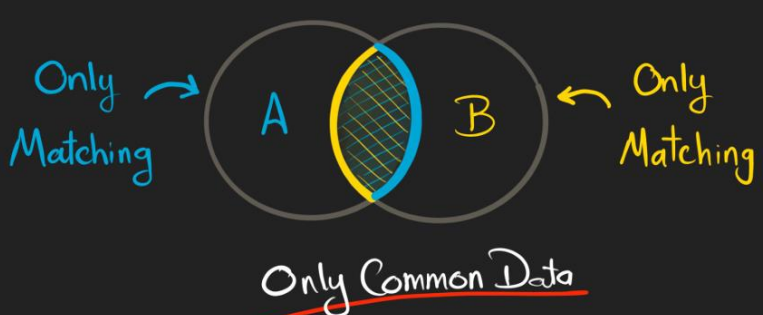
QUERY 1: RETREIVE ALL DATA FROM CUSTOMERS AND ORDERS AS SEPARATE RESULTS.

```
SELECT *  
FROM customers;  
SELECT *  
FROM orders;
```

	id	first_name	country	score
1	1	Maria	Germany	350
2	2	John	USA	900
3	3	Georg	UK	750
4	4	Martin	Germany	500
5	5	Peter	USA	0

	order_id	customer_id	order_date	sales
1	1001	1	2021-01-11	35
2	1002	2	2021-04-05	15
3	1003	3	2021-06-18	20
4	1004	6	2021-08-31	10

## INNER JOIN:



DATA WITH BARAA

### INNER JOIN

Returns Only Matching Rows from both Tables

Only Matching →

← Only Matching

Only Common Data

The Order of Tables Doesn't Matters

```
SELECT *  
FROM A  
INNER JOIN B  
ON A.Key = B.Key
```

How to Match Rows ???

SQL Course | Joins

**QUERY 1: GET ALL CUSTOMERS ALONG WITH THEIR ORDERS, BUT ONLY FOR CUSTOMERS WHO HAVE PLACED AN ORDER.**

```
SELECT *
FROM customers
INNER JOIN orders
ON id=customer_id
```

	id	first_name	country	score	order_id	customer_id	order_date	sales
1	1	Maria	Germany	350	1001	1	2021-01-11	35
2	2	John	USA	900	1002	2	2021-04-05	15
3	3	Georg	UK	750	1003	3	2021-06-18	20

THIS IS HOW IT LOOKS. BUT THIS IS NOT GOOD PRACTICE, WE NEED TO SPECIFY COLUMNS WE NEED FROM THE TWO TABLES.

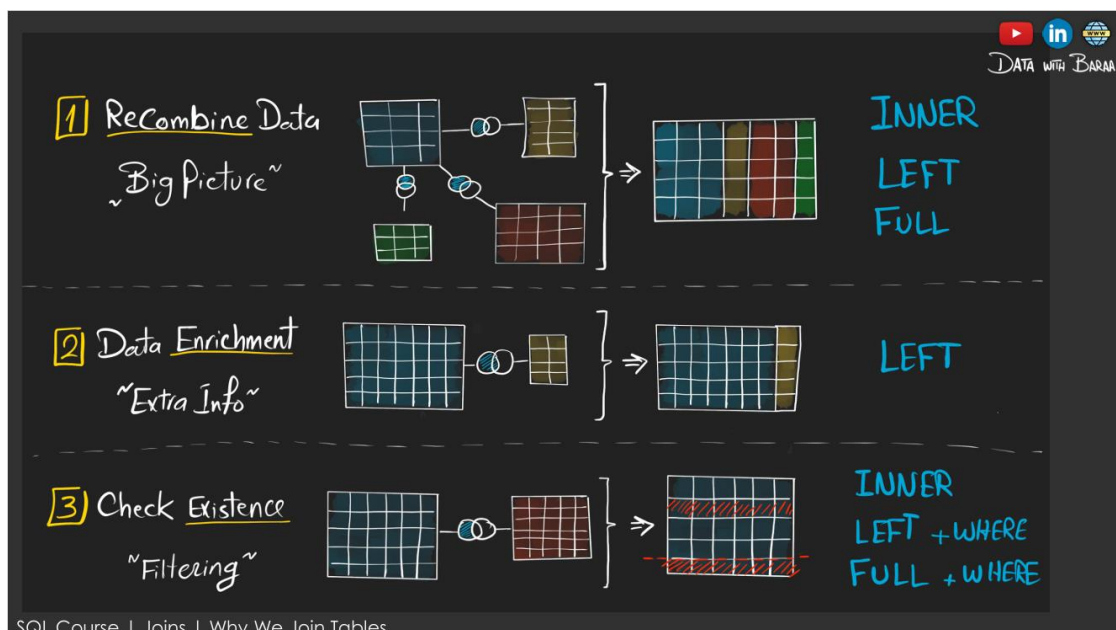
```
SELECT
    c.id,
    c.first_name,
    o.order_id,
    o.order_date,
    o.sales
FROM customers c
INNER JOIN orders o
ON id=customer_id
```

	id	first_name	order_id	order_date	sales
1	1	Maria	1001	2021-01-11	35
2	2	John	1002	2021-04-05	15
3	3	Georg	1003	2021-06-18	20

In INNER JOIN we can join the tables with any order.

And in INNER JOIN, it starts from the left table, and starts matching the data on the right table. Only if there is a match between the data, the result is presented in the output.

We can use INNER JOIN to recombine and filter data.



**LEFT JOIN:** Returns all the rows from the left table and matching rows from right table.

DATA WITH BARAA

## LEFT JOIN

Returns All rows from Left and Only Matching from Right

The Order of Tables is IMPORTANT

```
SELECT *  
FROM A  
LEFT JOIN B  
ON A.Key = B.Key
```

SQL Course | Joins

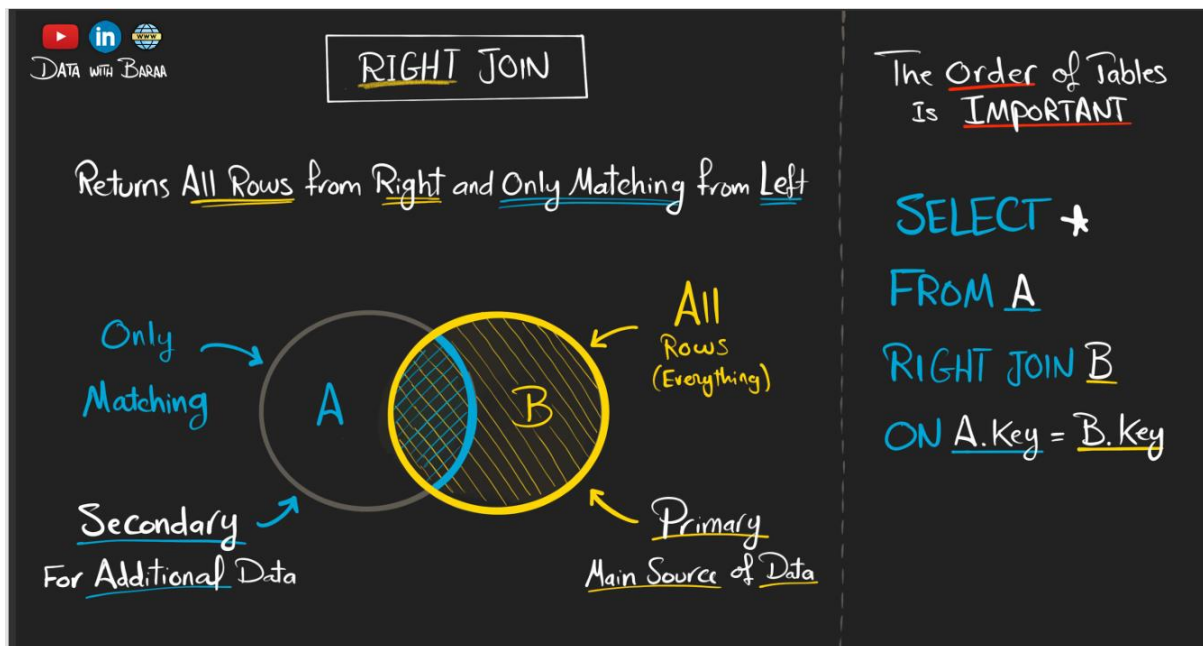
ORDER OF TABLES IS VERY IMPORTANT IN LEFT JOIN.

QUERY 1: GET ALL THE CUSTOMERS ALONG WITH THEIR ORDERS, INCLUDING THOSE WITHOUT ORDERS.

```
SELECT  
    c.id,  
    c.first_name,  
    o.order_id,  
    o.order_date,  
    o.sales  
FROM customers c  
LEFT JOIN orders o  
ON id=customer_id
```

	id	first_name	order_id	order_date	sales
1	1	Maria	1001	2021-01-11	35
2	2	John	1002	2021-04-05	15
3	3	Georg	1003	2021-06-18	20
4	4	Martin	NULL	NULL	NULL
5	5	Peter	NULL	NULL	NULL

**RIGHT JOIN:** Returns all rows from right and matching rows from left.



**ORDER OF TABLE IS VERY IMPORTANT IN RIGHT JOIN.**

**QUERY 1: GET ALL THE CUSTOMERS ALONG WITH THEIR ORDERS, INCLUDING ORDERS WITHOUT CUSTOMERS.**

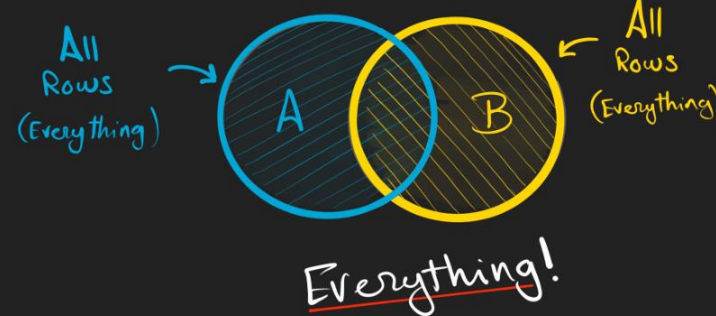
```
SELECT
    c.id,
    c.first_name,
    o.order_id,
    o.order_date,
    o.sales
FROM customers c
RIGHT JOIN orders o
ON id=customer_id
```

	id	first_name	order_id	order_date	sales
1	1	Maria	1001	2021-01-11	35
2	2	John	1002	2021-04-05	15
3	3	Georg	1003	2021-06-18	20
4	NULL	NULL	1004	2021-08-31	10

**SAME QUERY USING LEFT JOIN INSTEAD OF RIGHT JOIN.**

```
SELECT
    c.id,
    c.first_name,
    o.order_id,
    o.order_date,
    o.sales
FROM orders o
LEFT JOIN customers c
ON id=customer_id
```

## FULL JOIN:



A Venn diagram with two overlapping circles, A (blue) and B (yellow). Both circles are filled with diagonal lines. An arrow points from the text 'All Rows (Everything)' to circle A. Another arrow points from the text 'All Rows (Everything)' to circle B. Below the circles, the word 'Everything!' is written in a large, stylized font.

**FULL JOIN**

Returns All Rows from Both Tables

The Order of Tables Doesn't Matter

```
SELECT *  
FROM A  
FULL JOIN B  
ON A.Key = B.Key
```

SQL Course | Joins

### ORDER OF TABLES DOES NOT MATTER.

QUERY 1: GET ALL CUSTOMERS AND ALL ORDERS, EVEN IF THERE IS NO MATCH.

SELECT

c.id,  
c.first\_name,  
o.order\_id,  
o.order\_date,  
o.sales

FROM customers c

FULL JOIN orders o

ON id=customer\_id

	id	first_name	order_id	order_date	sales
1	1	Maria	1001	2021-01-11	35
2	2	John	1002	2021-04-05	15
3	3	Georg	1003	2021-06-18	20
4	4	Martin	NULL	NULL	NULL
5	5	Peter	NULL	NULL	NULL
6	NULL	NULL	1004	2021-08-31	10



## ADVANCED JOINS:

**LEFT ANTI JOIN: RETURNS ROW FROM LEFT THAT HAS NO MATCH IN RIGHT.**

DATA WITH BARAA

### LEFT ANTI JOIN

Returns Row from Left that has NO MATCH in Right

Only UnMatching Rows

Primary Main Source of Data

Nothing

Lookup(Filter) Not for Data, Just a Filter

SQL Course | Joins

The Order of Tables Is IMPORTANT

```
SELECT *  
FROM A  
LEFT JOIN B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```

QUERY 1: GET ALL CUSTOMERS WHO HAS NOT PLACED ANY ORDER.

```
SELECT *  
FROM customers c  
LEFT JOIN orders o  
ON c.id=o.customer_id  
WHERE o.customer_id is NULL
```

	id	first_name	country	score	order_id	customer_id	order_date	sales
1	4	Martin	Germany	500	NULL	NULL	NULL	NULL
2	5	Peter	USA	0	NULL	NULL	NULL	NULL

## RIGHT ANTI JOIN:

DATA WITH BARAA

### RIGHT ANTI JOIN

Returns Rows from Right that has NO MATCH in Left

Nothing

Only UnMatching Rows

Primary Main Source of Data

Lookup(Filter) Not for Data, Just a Filter

SQL Course | Joins

The Order of Tables Is IMPORTANT

```
SELECT *  
FROM A  
RIGHT JOIN B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```

### QUERY 1: GET ALL ORDERS WITHOUT MATCHING CUSTOMERS:

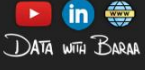
```
SELECT *  
FROM customers c  
RIGHT JOIN orders o  
ON c.id=o.customer_id  
WHERE c.id is NULL
```

	id	first_name	country	score	order_id	customer_id	order_date	sales
1	NULL	NULL	NULL	NULL	1004	6	2021-08-31	10

### SAME USING LEFT ANTI JOIN

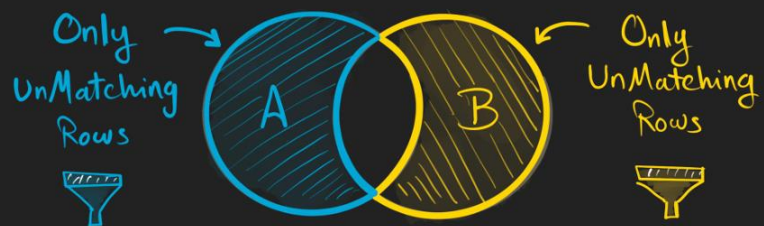
```
SELECT *  
FROM orders o  
LEFT JOIN customers c  
ON c.id=o.customer_id  
WHERE c.id is NULL
```

### FULL ANTI JOIN:



## FULL ANTI JOIN

Returns Only Rows that Don't Match in either Tables



Only Unmatching Data

SQL Course | Joins

The Order of Tables Doesn't Matter

```
SELECT *  
FROM A  
FULL JOIN B  
ON A.Key = B.Key  
WHERE  
    B.Key IS NULL  
OR  
    A.Key IS NULL
```

### QUERY 1: FIND CUSTOMERS WITHOUT ORDERS AND ORDERS WITHOUT CUSTOMERS.

```
SELECT *  
FROM orders o  
FULL JOIN customers c  
ON c.id=o.customer_id  
WHERE c.id is NULL  
OR o.customer_id is NULL
```



	order_id	customer_id	order_date	sales	id	first_name	country	score
1	1004	6	2021-08-31	10	NULL	NULL	NULL	NULL
2	NULL	NULL	NULL	NULL	4	Martin	Germany	500
3	NULL	NULL	NULL	NULL	5	Peter	USA	0

### CHALLENGE TASK:

GET ALL THE CUSTOMERS WITH THEIR ORDERS, BUT ONLY FOR CUSTOMERS WHO HAVE PLACED AN ORDER (WITHOUT USING INNER JOIN).

```
SELECT *
FROM customers c
LEFT JOIN orders o
ON c.id=o.customer_id
WHERE O.customer_id is NOT NULL
```

	id	first_name	country	score	order_id	customer_id	order_date	sales
1	1	Maria	Germany	350	1001	1	2021-01-11	35
2	2	John	USA	900	1002	2	2021-04-05	15
3	3	Georg	UK	750	1003	3	2021-06-18	20

### CROSS JOIN :

**CROSS JOIN**

Combines Every Row from Left with Every Row from Right

All Possible Combinations - Cartesian Join -

$2 \times 3 = 6$  ← Total Rows

The Order of Table Doesn't Matter

```
SELECT *
FROM A
CROSS JOIN B
```

↗

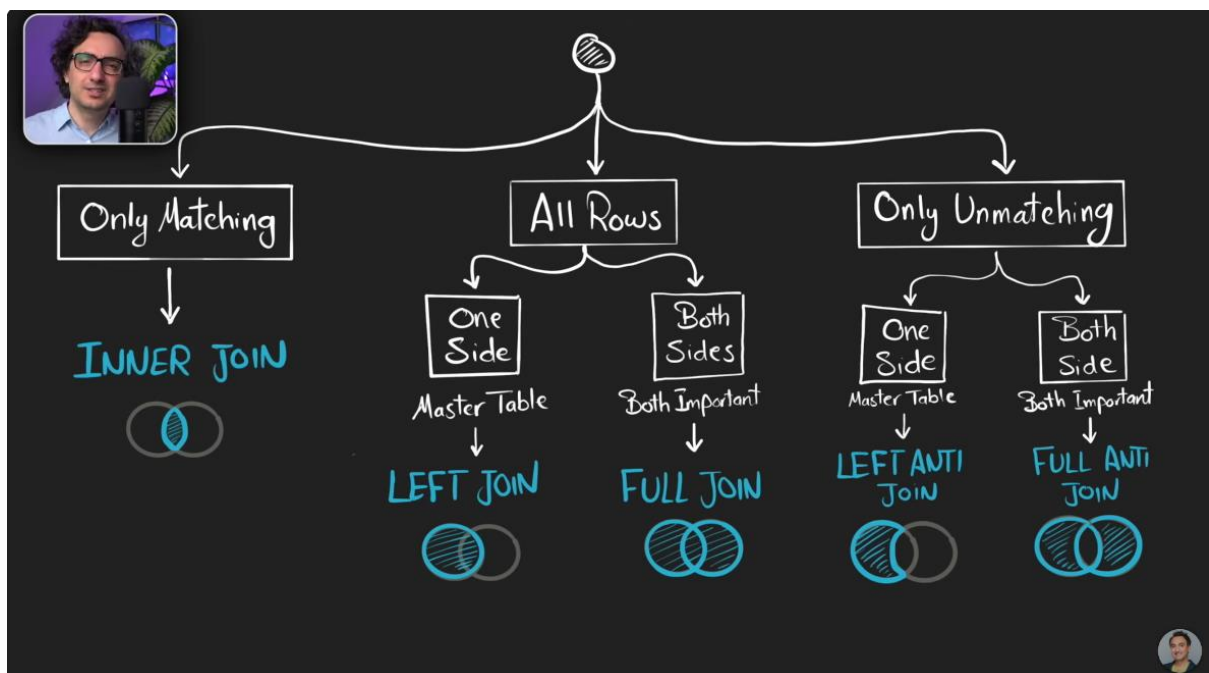
No Condition Needed

SQL Course | Joins

### QUERY 1: GENERATE ALL POSSIBLE COMBINATIONS OF CUSTOMERS AND ORDERS

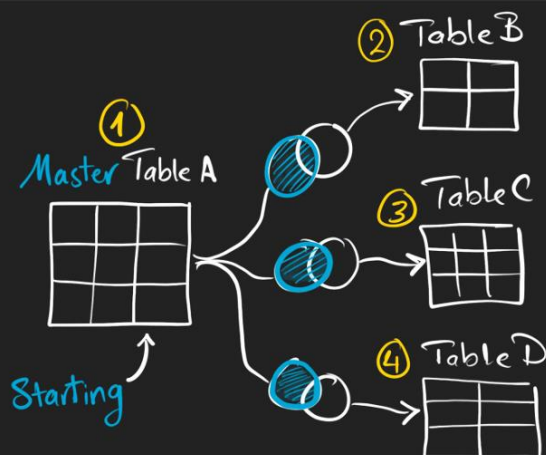
```
SELECT *  
FROM customers  
CROSS JOIN orders
```

	id	first_name	country	score	order_id	customer_id	order_date	sales
1	1	Maria	Germany	350	1001	1	2021-01-11	35
2	2	John	USA	900	1001	1	2021-01-11	35
3	3	Georg	UK	750	1001	1	2021-01-11	35
4	4	Martin	Germany	500	1001	1	2021-01-11	35
5	5	Peter	USA	0	1001	1	2021-01-11	35
6	1	Maria	Germany	350	1002	2	2021-04-05	15
7	2	John	USA	900	1002	2	2021-04-05	15
8	3	Georg	UK	750	1002	2	2021-04-05	15
9	4	Martin	Germany	500	1002	2	2021-04-05	15
10	5	Peter	USA	0	1002	2	2021-04-05	15
11	1	Maria	Germany	350	1003	3	2021-06-18	20
12	2	John	USA	900	1003	3	2021-06-18	20
13	3	Georg	UK	750	1003	3	2021-06-18	20
14	4	Martin	Germany	500	1003	3	2021-06-18	20
15	5	Peter	USA	0	1003	3	2021-06-18	20
16	1	Maria	Germany	350	1004	6	2021-08-31	10
17	2	John	USA	900	1004	6	2021-08-31	10
18	3	Georg	UK	750	1004	6	2021-08-31	10
19	4	Martin	Germany	500	1004	6	2021-08-31	10
20	5	Peter	USA	0	1004	6	2021-08-31	10



```
SELECT *
FROM A
LEFT B ON...
LEFT C ON...
LEFT D ON...
WHERE
```

Control what  
to keep



### TASK:

QUERY 1: USING SALESDB, RETRIEVE A LIST OF ALL ORDERS, ALONG WITH THE RELATED CUSTOMER, PRODUCT, AND EMPLOYEE DETAILS.

**For each order, display:**

- Order ID
- Customer's name
- Product name
- Sales amount
- Product price
- Salesperson's name

```

SELECT
    o.OrderID AS Order_ID,
    o.Sales AS Sales_Amount,
    c.FirstName AS Customer_Firstname,
    c.LastName AS Customer_Lastname,
    p.Product AS Product_Name,
    p.Price AS Product_price,
    e.FirstName AS Salesperson_Firstname,
    e.LastName AS Salesperson_LastName
FROM Sales.Orders o
LEFT JOIN Sales.Customers c
ON o.CustomerID = c.CustomerID
LEFT JOIN Sales.Products p
ON o.ProductID = p.ProductID
LEFT JOIN Sales.Employees e
ON o.SalesPersonID = e.EmployeeID

```

	Order_ID	Sales_Amount	Customer_Firstname	Customer_Lastname	Product_Name	Product_price	Salesperson_Firstname	Salesperson_LastName
1	1	10	Kevin	Brown	Bottle	10	Mary	NULL
2	2	15	Mary	NULL	Tire	15	Mary	NULL
3	3	20	Jossef	Goldberg	Bottle	10	Carol	Baker
4	4	60	Jossef	Goldberg	Gloves	30	Mary	NULL
5	5	25	Kevin	Brown	Caps	25	Carol	Baker
6	6	50	Mary	NULL	Caps	25	Carol	Baker
7	7	30	Jossef	Goldberg	Tire	15	Frank	Lee
8	8	90	Mark	Schwarz	Bottle	10	Mary	NULL
9	9	20	Kevin	Brown	Bottle	10	Mary	NULL
10	10	60	Mary	NULL	Tire	15	Carol	Baker

## SETS:

If we wanted to combine rows, we have to use SET operators.

# RULES OF SET OPERATORS

#1 RULE | ORDER BY can be used only once

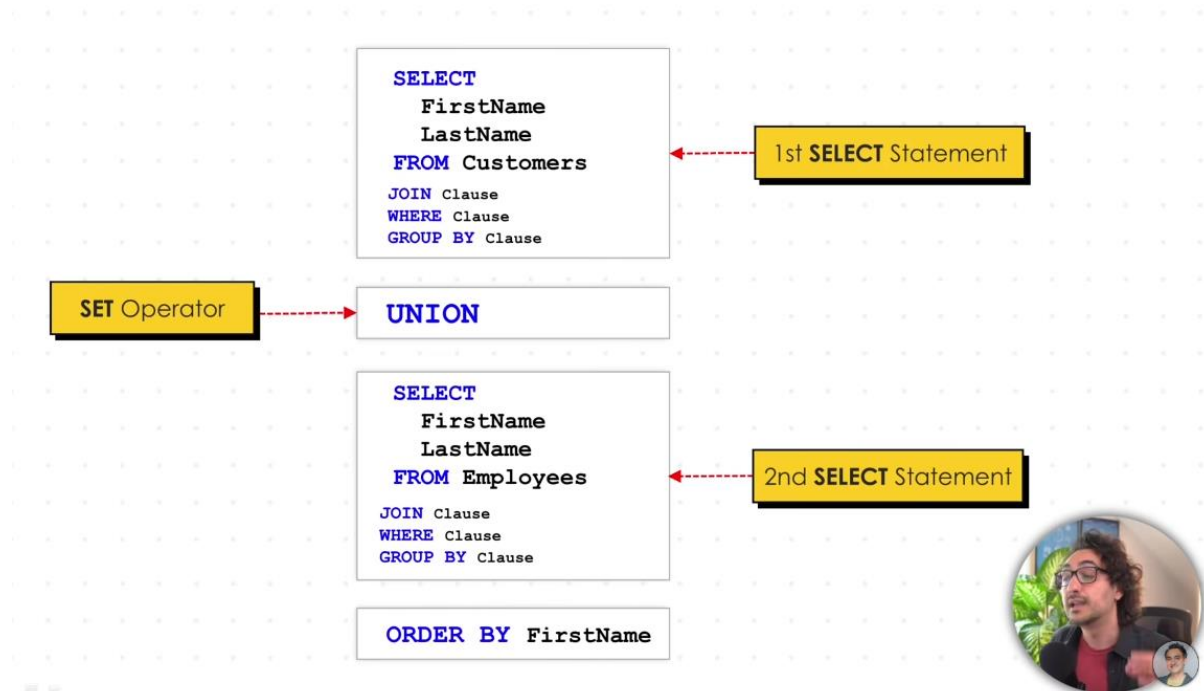
#2 RULE | Same Number of Columns

#3 RULE | Matching Data Types

#4 RULE | Same Order of Columns

#5 RULE | First Query Controls Aliases

#6 RULE | Mapping Correct Columns



## UNION:



- Returns all distinct rows from both queries.
- Removes duplicate rows from the result.

QUERY 1: COMBINE THE DATA FROM EMPLOYEES AND CUSTOMERS IN ONE TABLE.

```
SELECT
    CustomerID AS ID,
    FirstName,
    LastName
FROM Sales.Customers
UNION
SELECT
    EmployeeID,
    FirstName,
    LastName
FROM Sales.Employees
```

	ID	FirstName	LastName
1	1	Frank	Lee
2	1	Jossef	Goldberg
3	2	Kevin	Brown
4	3	Mary	NULL
5	4	Mark	Schwarz
6	4	Michael	Ray
7	5	Anna	Adams
8	5	Carol	Baker



## UNION ALL:



Returns all rows from both queries, including **duplicates**.

UNION ALL IS WAY FASTER THAN UNION, BECAUSE IT WILL NOT CHECK DUPLICATES IN THE DATA LIKE UNION.

## **UNION ALL vs UNION**

If you're confident there are no duplicates, use **UNION ALL**

## **UNION ALL vs UNION**

Use UNION ALL to find duplicates and quality issues

EXCEPT(minus):



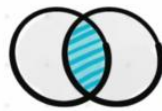
- Returns all distinct rows from the first query that are not found in the second query.
- It is the only one where the order of queries affects the final result.

QUERY 1: FIND THE CUSTOMERS, WHO ARE NOT EMPLOYEES AT THE SAME TIME.

```
SELECT
    CustomerID AS ID,
    FirstName,
    LastName
FROM Sales.Customers
EXCEPT
SELECT
    EmployeeID,
    FirstName,
    LastName
FROM Sales.Employees
```

	ID	FirstName	LastName
1	1	Jossef	Goldberg
2	4	Mark	Schwarz
3	5	Anna	Adams

## INTERSECT:



# INTERSECT

Returns only the rows that are common in both queries

QUERY 1: FIND THE CUSTOMERS, WHO ARE EMPLOYEES.

```
SELECT
    CustomerID AS ID,
    FirstName,
    LastName
FROM Sales.Customers
INTERSECT
SELECT
    EmployeeID,
    FirstName,
    LastName
FROM Sales.Employees
```

	ID	FirstName	LastName
1	2	Kevin	Brown
2	3	Mary	NULL

## SET OPERATORS



### UNION

Returns All rows from both sets, **elimination duplicates**

```
SELECT FirstName, LastName
FROM Customers

UNION

SELECT FirstName, LastName
FROM Employees
```

### UNION ALL

Returns All rows from both sets, **including duplicates**

```
SELECT FirstName, LastName
FROM Customers

UNION ALL

SELECT FirstName, LastName
FROM Employees
```

### EXCEPT/MINUS

Return unique rows in **first set** that are not in second table

```
SELECT FirstName, LastName
FROM Customers

EXCEPT

SELECT FirstName, LastName
FROM Employees
```

### INTERSECT

Return only the **common rows** between two sets

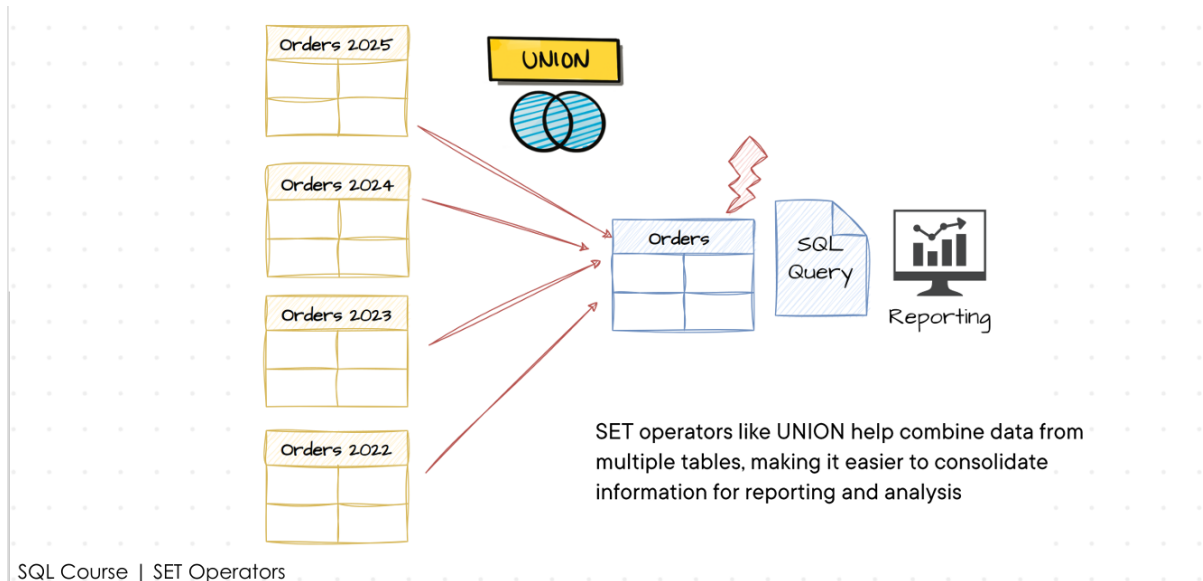
```
SELECT FirstName, LastName
FROM Customers

INTERSECT

SELECT FirstName, LastName
FROM Employees
```

## UNION USE CASES:

- COMBINE INFORMATION.



## SQL TASK:

ORDERS ARE STORED IN SEPARATE TABLES(Orders AND OrdersArchive OF SalesDB). COMINE ALL ORDERS IN TO ONE REPORT WITHOUT DUPLICATES.

```
SELECT *  
FROM Sales.Orders  
UNION  
SELECT *  
FROM Sales.OrdersArchive
```

-----It is a best practice bot to use asterisk(\*) to combine tables, list needed columns instead.

```
SELECT  
    [OrderID]  
    , [ProductID]  
    , [CustomerID]  
    , [SalesPersonID]  
    , [OrderDate]  
    , [ShipDate]  
    , [OrderStatus]  
    , [ShipAddress]  
    , [BillAddress]  
    , [Quantity]  
    , [Sales]  
    , [CreationTime]  
FROM Sales.Orders  
UNION  
SELECT  
    [OrderID]  
    , [ProductID]  
    , [CustomerID]  
    , [SalesPersonID]  
    , [OrderDate]
```

```

    , [ShipDate]
    , [OrderStatus]
    , [ShipAddress]
    , [BillAddress]
    , [Quantity]
    , [Sales]
    , [CreationTime]
FROM Sales.OrdersArchive

```

	OrderID	ProductID	CustomerID	SalesPersonID	OrderDate	ShipDate	OrderStatus	ShipAddress	BillAddress	Quantity	Sales	CreationTime
1	1	101	2	3	2024-04-01	2024-04-05	Shipped	123 Main St	456 Billing St	1	10	2024-04-01 12:34:56.0000000
2	1	101	2	3	2025-01-01	2025-01-05	Delivered	9833 Mt. Dias Blv.	1226 Shoe St	1	10	2025-01-01 12:34:56.0000000
3	2	102	3	3	2024-04-05	2024-04-10	Shipped	456 Elm St	789 Billing St	1	15	2024-04-05 23:22:04.0000000
4	2	102	3	3	2025-01-05	2025-01-10	Shipped	250 Race Court	NULL	1	15	2025-01-05 23:22:04.0000000
5	3	101	1	4	2024-04-10	2024-04-25	Shipped	789 Maple St	789 Maple St	2	20	2024-04-10 18:24:08.0000000
6	3	101	1	5	2025-01-10	2025-01-25	Delivered	8157 W. Book	8157 W. Book	2	20	2025-01-10 18:24:08.0000000
7	4	105	1	3	2024-04-20	2024-04-25	Delivered	987 Victory Lane		2	60	2024-04-20 14:50:33.0000000
8	4	105	1	3	2024-04-20	2024-04-25	Shipped	987 Victory Lane		2	60	2024-04-20 05:50:33.0000000
9	4	105	1	3	2025-01-20	2025-01-25	Shipped	5724 Victory Lane		2	60	2025-01-20 05:50:33.0000000
10	5	104	2	5	2024-05-01	2024-05-05	Shipped	345 Oak St	678 Pine St	1	25	2024-05-01 14:02:41.0000000
11	5	104	2	5	2025-02-01	2025-02-05	Delivered	NULL	NULL	1	25	2025-02-01 14:02:41.0000000
12	6	101	3	5	2024-05-05	2024-05-10	Delivered	543 Belmont Rd.	3768 Door Way	2	50	2024-05-12 20:36:55.0000000
13	6	104	3	5	2024-05-05	2024-05-10	Delivered	543 Belmont Rd.	NULL	2	50	2024-05-06 15:34:57.0000000
14	6	104	3	5	2024-05-05	2024-05-10	Delivered	543 Belmont Rd.	3768 Door Way	2	50	2024-05-07 13:22:05.0000000
15	6	104	3	5	2025-02-05	2025-02-10	Delivered	1792 Belmont Rd.	NULL	2	50	2025-02-06 15:34:57.0000000
16	7	102	1	1	2025-02-15	2025-02-27	Delivered	136 Balboa Court		2	30	2025-02-16 06:22:01.0000000
17	7	102	3	5	2024-06-15	2024-06-20	Shipped	111 Main St	222 Billing St	0	60	2024-06-16 23:25:15.0000000
18	8	101	4	3	2025-02-18	2025-02-27	Shipped	2947 Vine Lane	4311 Clay Rd	3	90	2025-02-18 10:45:22.0000000
19	9	101	2	3	2025-03-10	2025-03-15	Shipped	3768 Door Way		2	20	2025-03-10 12:59:04.0000000
20	10	102	3	5	2025-03-15	2025-03-20	Shipped	NULL	NULL	0	60	2025-03-16 23:25:15.0000000

**\*\* Include additional column to indicate the source of each row.**

```

SELECT
    'Orders' AS SourceTable
    , [OrderID]
    , [ProductID]
    , [CustomerID]
    , [SalesPersonID]
    , [OrderDate]
    , [ShipDate]
    , [OrderStatus]
    , [ShipAddress]
    , [BillAddress]
    , [Quantity]
    , [Sales]
    , [CreationTime]
FROM Sales.Orders
UNION
SELECT
    'OrdersArchive' AS SourceTable
    , [OrderID]
    , [ProductID]
    , [CustomerID]
    , [SalesPersonID]
    , [OrderDate]
    , [ShipDate]
    , [OrderStatus]
    , [ShipAddress]
    , [BillAddress]
    , [Quantity]
    , [Sales]
    , [CreationTime]
FROM Sales.OrdersArchive
ORDER BY OrderID

```

	SourceTable	OrderID	ProductID	CustomerID	SalesPersonID	OrderDate	ShipDate	OrderStatus	ShipAddress	BillAddress	Quantity	Sales	CreationTime
1	Orders	1	101	2	3	2025-01-01	2025-01-05	Delivered	9833 Mt. Dias Blv.	1226 Shoe St.	1	10	2025-01-01 12:34:56.0000000
2	OrdersArchive	1	101	2	3	2024-04-01	2024-04-05	Shipped	123 Main St	456 Billing St	1	10	2024-04-01 12:34:56.0000000
3	Orders	2	102	3	3	2025-01-05	2025-01-10	Shipped	250 Race Court	NULL	1	15	2025-01-05 23:22:04.0000000
4	OrdersArchive	2	102	3	3	2024-04-05	2024-04-10	Shipped	456 Elm St	789 Billing St	1	15	2024-04-05 23:22:04.0000000
5	Orders	3	101	1	5	2025-01-10	2025-01-25	Delivered	8157 W. Book	8157 W. Book	2	20	2025-01-10 18:24:08.0000000
6	OrdersArchive	3	101	1	4	2024-04-10	2024-04-25	Shipped	789 Maple St	789 Maple St	2	20	2024-04-10 18:24:08.0000000
7	Orders	4	105	1	3	2025-01-20	2025-01-25	Shipped	5724 Victory Lane		2	60	2025-01-20 05:50:33.0000000
8	OrdersArchive	4	105	1	3	2024-04-20	2024-04-25	Delivered	987 Victory Lane		2	60	2024-04-20 14:50:33.0000000
9	OrdersArchive	4	105	1	3	2024-04-20	2024-04-25	Shipped	987 Victory Lane		2	60	2024-04-20 05:50:33.0000000
10	Orders	5	104	2	5	2025-02-01	2025-02-05	Delivered	NULL	NULL	1	25	2025-02-01 14:02:41.0000000
11	OrdersArchive	5	104	2	5	2024-05-01	2024-05-05	Shipped	345 Oak St	678 Pine St	1	25	2024-05-01 14:02:41.0000000
12	Orders	6	104	3	5	2025-02-05	2025-02-10	Delivered	1792 Belmont Rd.	NULL	2	50	2025-02-06 15:34:57.0000000
13	OrdersArchive	6	101	3	5	2024-05-05	2024-05-10	Delivered	543 Belmont Rd.	3768 Door Way	2	50	2024-05-12 20:36:55.0000000
14	OrdersArchive	6	104	3	5	2024-05-05	2024-05-10	Delivered	543 Belmont Rd.	NULL	2	50	2024-05-06 15:34:57.0000000
15	OrdersArchive	6	104	3	5	2024-05-05	2024-05-10	Delivered	543 Belmont Rd.	3768 Door Way	2	50	2024-05-07 13:22:05.0000000
16	Orders	7	102	1	1	2025-02-15	2025-02-27	Delivered	136 Balboa Court		2	30	2025-02-16 06:22:01.0000000
17	OrdersArchive	7	102	3	5	2024-06-15	2024-06-20	Shipped	111 Main St	222 Billing St	0	60	2024-06-16 23:25:15.0000000
18	Orders	8	101	4	3	2025-02-18	2025-02-27	Shipped	2947 Vine Lane	4311 Clay Rd	3	90	2025-02-18 10:45:22.0000000
19	Orders	9	101	2	3	2025-03-10	2025-03-15	Shipped	3768 Door Way		2	20	2025-03-10 12:59:04.0000000
20	Orders	10	102	3	5	2025-03-15	2025-03-20	Shipped	NULL	NULL	0	60	2025-03-16 23:25:15.0000000

## EXCEPT USE CASES:

- DELTA DETECTION

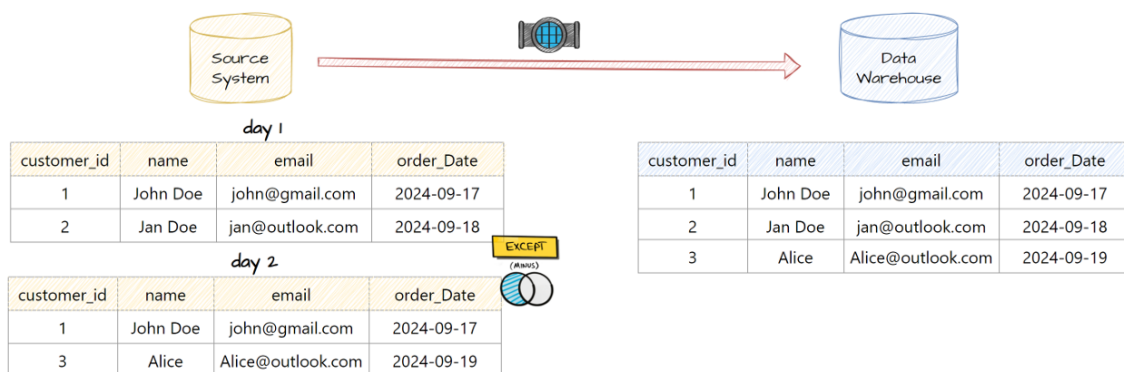
# DELTA DETECTION

Identifying the differences or changes (delta) between two batches of data.

## SET USE CASE Delta Detection



SET operators like EXCEPT help detect changes between datasets, making it easier to identify new, updated, or missing records during data integration.





- DATA COMPLETENESS CHECK:

## DATA COMPLETENESS CHECK

EXCEPT operator can be used to compare tables to detect discrepancies between databases.

SET operators like EXCEPT help verify data completeness by comparing tables across databases, ensuring no records are missing or mismatched.

