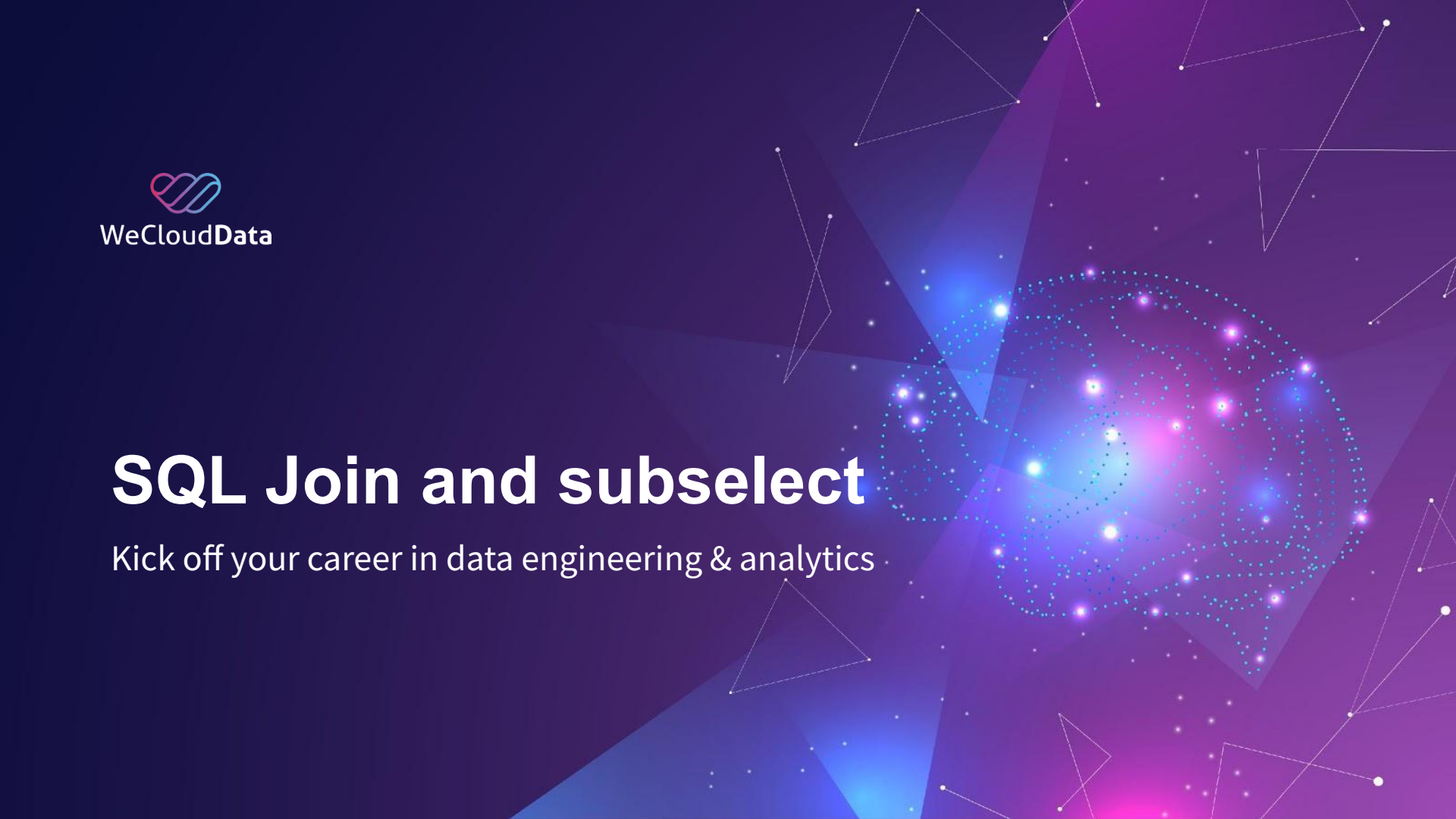


SQL Join and subselect

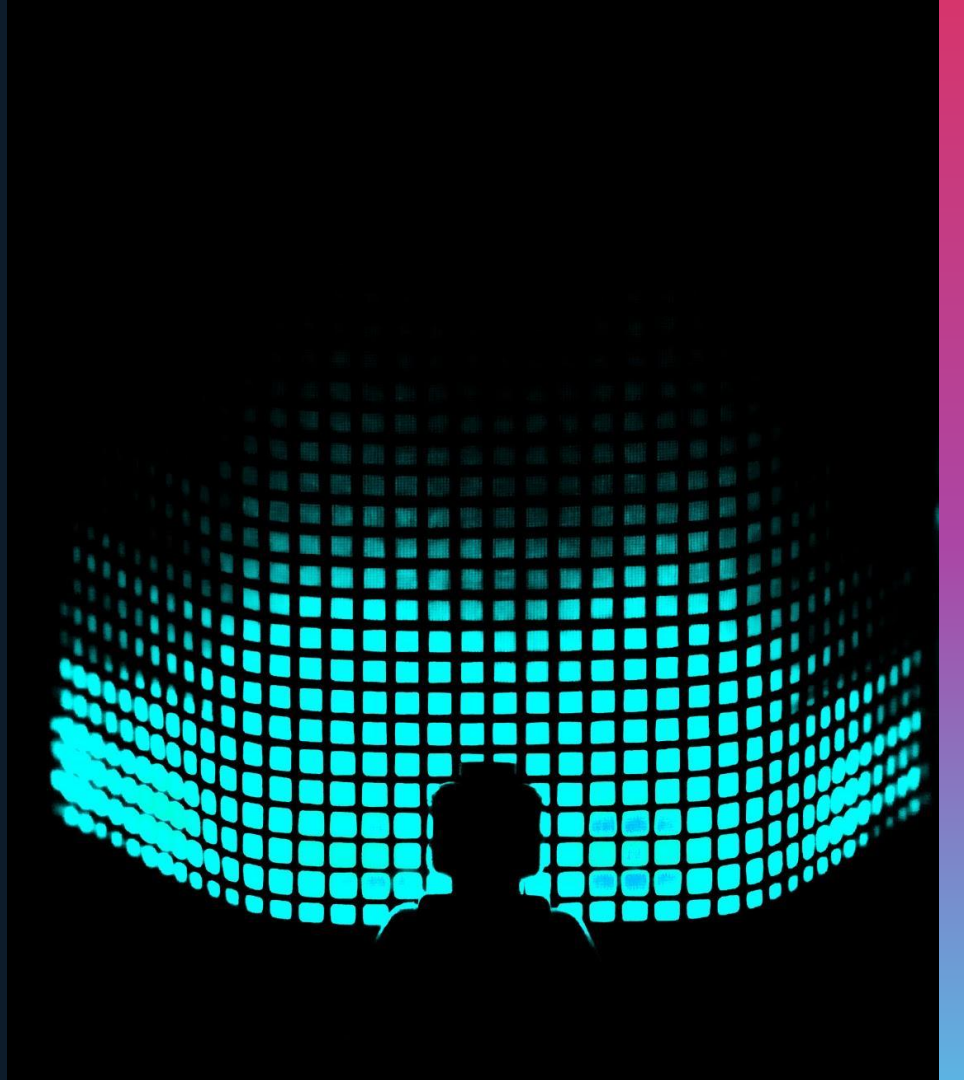
Kick off your career in data engineering & analytics

Aggregation and Subqueries

Learning Objectives

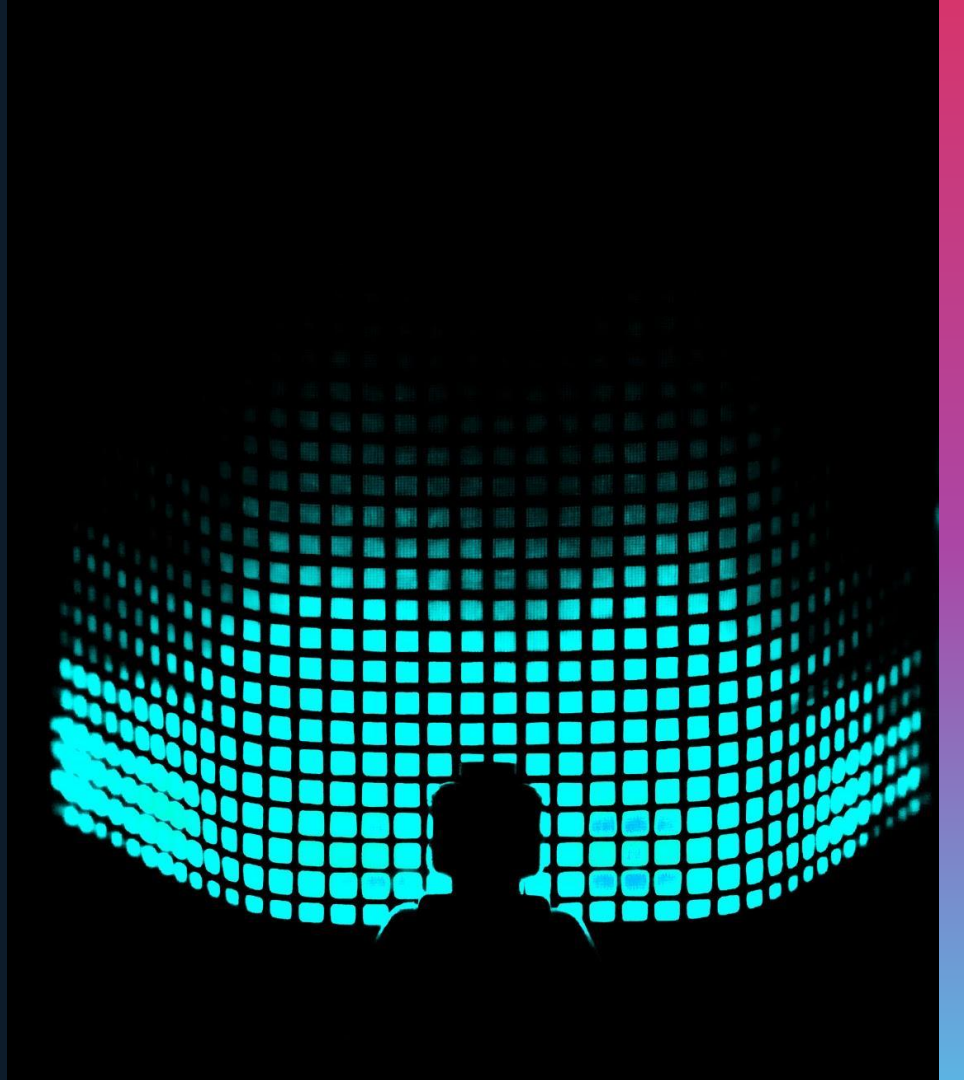
In this module, we will introduce aggregation functions and subqueries in SQL. Specifically, we will share with you:

- SQL aggregate functions
- Group By
- Having
- Case When
- SQL subqueries



Best Practices

- When you're learning SQL, DON'T COPY & PASTE
- When you're using SQL at work, YES, COPY & PASTE
- Always select the first N rows for sanity check
- Always know the rough total number of records in your table
- Know the primary key of a table



Aggregation Functions

Group By

Having

Case When

Subqueries

Agenda.

SQL Terminology

Aggregation and Subqueries

Here are the **aggregation functions** in SQL:

- Count
- Sum
- Min/Max
- Avg

SQL Aggregations: Count, Sum, Max/Min, Avg

Aggregation and Subqueries

Aggregation Function	Description
count()	Counts how many rows are in the specified column
sum()	Sums/Adds all the values in the specified column together
min() / max()	Returns the lowest and highest values in a specified column, respectively
avg()	Returns the average/mean of a group of selected values

Syntax

```
SELECT column1, column2,
       [agg_function(column5) as col_agg]
FROM db.table_name;
```

SQL Aggregations: Count, Sum, Max/Min, Avg (Cont'd)

Aggregation and Subqueries

```
-- Example #1: COUNT()
# What is the biggest customer segment?

select CustomerSegment,
       count(distinct CustomerID) as cnt
from superstore.customer
group by CustomerSegment
order by count(distinct CustomerID) desc;
```

count() was used here!

CustomerSegme...	cnt
Corporate	4
Small Business	3
Home Office	1

Syntax

```
SELECT column1, column2,
       [agg_function(column5) as col_agg]
FROM db.table_name;
```

How Does SQL Aggregation Work?

Aggregation and Subqueries

	OrderYear	OrderID	CustomerID	ProductID	Sales
2009	2009	8710	40732966	657768	151.35000
	2009	17024	40732966	778385	1401.75000
	2009	36647	63834266	114426	55.44000
	2009	47173	82335880	560855	277.07450
	2009	34017	58046234	293693	103.39000
2010	2010	16326	68464052	657768	147.46000
	2010	15808	68464052	518917	58.14000
	2010	12452	68464052	955858	112.86000
	2010	15808	68464052	700324	126.88000
	2010	16326	68464052	851117	1538.33000
2011	2011	5251	40732966	497741	3821.03900
	2011	40961	68464052	189993	23.84000
	2011	36356	63834266	816179	117.77000
	2011	58470	38512011	657768	49.08000
	2011	35239	38512011	460438	2257.88000
2012	2012	34083	38512011	367450	1415.14800
	2012	39972	75726086	656848	224.58000
	2012	39972	75726086	481924	138.59000
	2012	45125	94530777	344239	4834.80000
	2012	5860	85111166	293693	56.73000

Syntax

```
SELECT OrderYear,
       COUNT(distinct OrderID) as cnt_o,
       SUM(Sales) as tot_s,
       AVG(Sales) as avg_s
FROM superstore.orders
GROUP BY OrderYear
ORDER BY OrderYear;
```

2009 → [151.35, 1401.75, 55.44, 277.07, 103.39] ☐ 151.35
2010 → [147.46, 58.14, 112.86, 126.88, 1538.33] ☐ 396.73
2011 → [3821.03, 23.84, 117.77, 49.08, 2257.88] ☐ 1253.92
2012 → [1415.15, 224.58, 138.59, 4834.8, 56.73] ☐ 1333.97

AVG

Aggregation Functions

Group By

Having

Case When

Subqueries

Agenda.

SQL Aggregations: Group By

Aggregation and Subqueries

```
-- Example #1: COUNT()
# What is the biggest customer segment?

select CustomerSegment,
      count(distinct CustomerID) as cnt
from superstore.customer
group by CustomerSegment
order by count(distinct CustomerID) desc;
```

CustomerSegme...	cnt
Corporate	4
Small Business	3
Home Office	1

Syntax

```
SELECT column1, column2,
        [agg_function(column5) as col_agg]
FROM db.table_name
GROUP BY column1, column2;
```

GROUP BY Usage:

- **GROUP BY** is used to group rows that have the same values
  - I.e. It helps summarize data from the db
  - Grouped queries only return a single row for every grouped item
- It's used with the **SELECT** statement
  - I.e. The column you Group By with should be included in the Select clause

SQL Aggregations: More GROUP BY Examples

Aggregation and Subqueries

```
-- Example #2 SUM()
# Which year generate the largest total sales?

select year(OrderDate) as OrderYear,
       sum(Sales) as SalesTotal
from superstore.orders
group by year(OrderDate)
order by SalesTotal desc;

-- Example #3 MIN/MAX
# Smallest and largest order quantity and sales by year and month

select year(OrderDate) as OrderYear,
       month(OrderDate) as OrderMonth,
       min(OrderQuantity) as MinOrderQuantity,
       max(OrderQuantity) as MaxOrderQuantity,
       min(Sales) as MinSales,
       max(Sales) as maxSales
from superstore.orders
group by year(OrderDate), month(OrderDate)
order by year(OrderDate), month(OrderDate);

-- Example #4: AVG()
# Which Shipping Mode has the highest average order shipping cost?

select ShipMode, avg(ShippingCost)
from superstore.orders
group by ShipMode;
```


SQL Aggregations: More GROUP BY Examples (Cont'd)

Aggregation and Subqueries

```
-- Lab #1
# Which Product Sub Category has the highest average base margin?

select ProductCategory,
       ProductSubCategory,
       avg(ProductBaseMargin)
from superstore.product
group by ProductCategory, ProductSubCategory
order by avg(ProductBaseMargin) desc;

-- Lab #2
# Which province has most number of customers in the "Corporate" customer segment?
select Province, count(distinct CustomerID)
from superstore.customer
where CustomerSegment = 'Corporate'
group by Province
order by count(distinct CustomerID) desc;


-- Lab #3
# Which customers purchsed most number of products  in a given year?

select CustomerID,
       year(OrderDate),
       count(distinct ProductID) as num_products
from superstore.orders
group by CustomerID, year(OrderDate)
order by count(distinct ProductID) desc;
```


Aggregation Functions

Group By

Having

Case When

Subqueries

Agenda.

SQL Aggregations: Having

Aggregation and Subqueries

```
# Find all orders that were placed with multiple
# products in a basket?

select OrderID, count(ProductID) as cnt
from superstore.orders
group by OrderID
having count(ProductID) > 1
order by count(ProductID) desc;
```

OrderID	cnt
69	2

```
select *
from superstore.orders
where OrderID = 69;
```

Syntax

```
SELECT column1, column2,
       sum(column5)
FROM db.table_name
WHERE [conditions]
GROUP BY column1, column2
HAVING [conditions]
ORDER BY column1, column2;
```

GROUP BY - HAVING Usage:

- The **HAVING** clause must follow the **GROUP BY** clause in a query
- If an **ORDER BY** clause is used, the **HAVING** clause must precede it

OrderID	ProductID	CustomerID	OrderDate	OrderPriority	OrderQuantity	Sales
69	213268	58189342	2009-06-03	Not Specified	42	1186.06000
69	115501	58189342	2009-06-03	Not Specified	28	51.53000

SQL Aggregations: Having (Cont'd)

Aggregation and Subqueries

HAVING:
Filter based on
aggregated columns

```
# Find all orders that were placed with multiple
# products in a basket?
select OrderID, count(ProductID) as cnt
from superstore.orders
group by OrderID
having count(ProductID) > 1
order by count(ProductID) desc;
```

Aggregated column

Having Clause

OrderID	cnt
69	2

WHERE:
Filter based on
existing columns

```
select *
from superstore.orders
where OrderID = 69;
```

OrderID	ProductID	CustomerID	OrderDate	OrderPriority	OrderQuantity	Sales
69	213268	58189342	2009-06-03	Not Specified	42	1186.06000
69	115501	58189342	2009-06-03	Not Specified	28	51.53000

Aggregation Functions

Group By

Having

Case When

Subqueries

Agenda.

IF Condition: Case When

Aggregation and Subqueries

```
-- Example
# Total number of distinct orders year over year
# Expecting one column for each year

select sum(case when year(OrderDate)=2009 then 1 else 0 end) as orders_2009,
       sum(case when year(OrderDate)=2010 then 1 else 0 end) as orders_2010,
       sum(case when year(OrderDate)=2011 then 1 else 0 end) as orders_2011,
       sum(case when year(OrderDate)=2012 then 1 else 0 end) as orders_2012
from (select distinct OrderID, OrderDate from superstore.orders) t
;
```

orders_2009	orders_2010	orders_2011	orders_2012
4	1	1	1

```
# [discussion] we can of course solve the question using group by
select year(OrderDate), count(distinct OrderID) as orders
from superstore.orders
group by year(OrderDate)
order by year(OrderDate);
```

year(OrderDate)	orders
2009	4
2010	1
2011	1
2012	1

Syntax

```
SELECT column1,
        CASE WHEN condition1 THEN value1
        WHEN condition2 THEN value2
        ...
        WHEN conditionN THEN valueN
        END AS derived_column
FROM db.table_name;
```

CASE WHEN Usage:

- Evaluates a list of conditions and returns one of multiple possible result expressions
- CASE WHEN can be used in any statement or clause that allows a valid expression including aggregation functions

Agenda.

Aggregation Functions

Group By

Having

Case When

Subqueries

Subqueries: Nested Queries

Aggregation and Subqueries

A **subquery** is a **SELECT** statement within another statement (a.k.a **inner query**).
The statement containing the subquery is called an **outer query**.

The inner query executes before the outer query so the inner query result can be passed to the outer query.

Note:

- A subquery must always be enclosed in parentheses
- You can use comparison operators in a subquery, such as: >, <, or =
- The comparison operator can also be a multiple-row operator, such as: **IN, ANY,** or **ALL**

Subquery #1: Inner Query with IN Clause

Aggregation and Subqueries

```
-- Example: Subquery with IN clause
# How many orders got returned and what is the
#  total sales revenue loss due to product return?
select count(distinct OrderID), sum(Sales)
from superstore.orders
where OrderID in (select distinct OrderID
                  from superstore.returns)
;
```

superstore.orders

OrderID	Sales
17024	1401.75000
15808	882.96000
29537	318.56000
38118	138.91000
69	1186.06000
16768	101.47000
359	3659.66000
69	51.53000

superstore.returns

OrderID	Status
69	Returned
359	Returned

count(distinct Order...	sum(Sales)
2	4897.25000

Syntax

```
SELECT column1, ..., columnN
FROM db.table_A
WHERE column1 IN ( SELECT columnK
                   FROM db.table_B
                   WHERE [condition1] );
```

Subquery Advantages:

- It's easy to isolate subqueries in a statement
- It's an alternative to JOINS/UNIONS
- Better readability because it is more "structured"

Subquery #2: Inner Query as Temp Table

Aggregation and Subqueries

```
-- Example: Subquery - SELECT FROM a temp table
-- with inner query
# What is the highest single day sales number?

select max(TotSalesByDay)
from (select OrderDate, sum(sales) as TotSalesByDay
      from superstore.orders
      group by OrderDate
      order by sum(sales) asc) tmp
;
```

Temp Table

superstore.orders

OrderDate	TotSalesByDay
2009-06-03	1237.59000
2009-07-25	101.47000
2009-10-13	1401.75000
2009-12-18	3659.66000
2010-12-14	882.96000
2011-04-18	138.91000
2012-09-19	318.56000

max(TotSalesByDay)
3659.66000

Syntax

```
SELECT tmp.column1, tmp.columnK
FROM ( SELECT column1, columnK
       FROM db.table_B
       WHERE [condition] ) tmp
WHERE tmp.column1 [conditions] );
```

Subquery Advantages:

- It's easy to isolate subqueries in a statement
- It's an alternative to JOINS/UNIONS
- Better readability because it is more "structured"

Lab #1: Subqueries

Aggregation and Subqueries

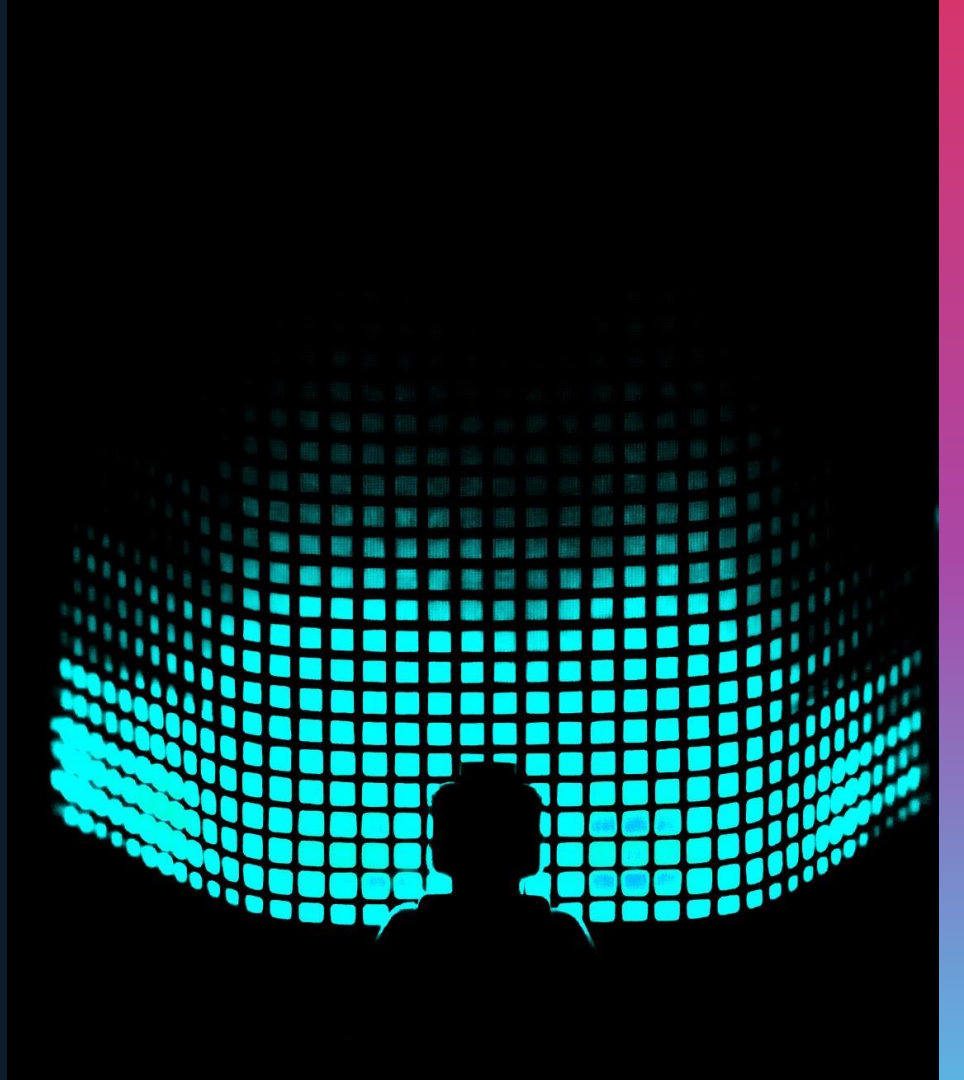
1. Create a summary table that has one column each year that calculates total spending for each customer
2. Create a pivot table that calculates number of orders in each order priority (rows) by different ship mode (columns)
3. Generate a sales report that has total sales by year and month for products in the ProductCategory "Office Supplies"
4. How many orders were placed with more than 3 products purchased?

Joins and Unions

Learning Objectives

In this module, we will introduce joins and unions in SQL. Specifically, we will share with you:

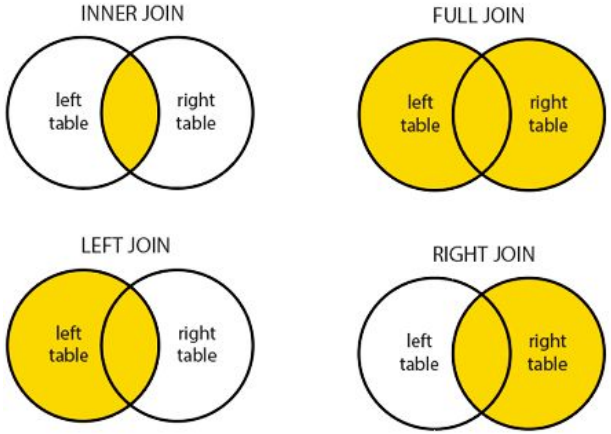
- Inner Join
- Left Join
- Right Join
- Full Join



Subqueries: Nested Queries

Joins and Unions

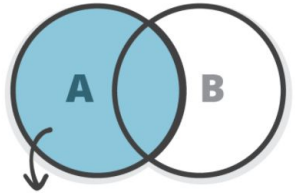
An **SQL JOIN** combines records from two or more tables.
A **JOIN** locates related column values in the two tables.
A **query** can contain zero, one, or multiple JOIN operations.



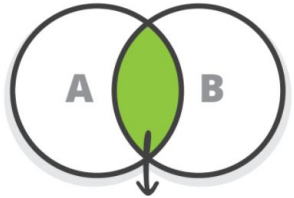
Subqueries: Nested Queries

Joins and Unions

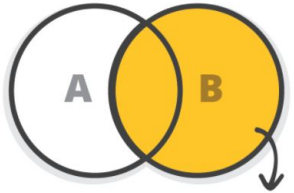
Types of Join	Explanation
INNER JOIN	• Select records that have matching values in both tables
LEFT OUTER JOIN	• Select records from the first (left-most) table with matching right table records
RIGHT OUTER JOIN	• Select records from the second (right-most) table with matching left table records
FULL OUTER JOIN	• Select all records that match either left or right table records



***LEFT OUTER JOIN** - all rows from table A, even if they do not exist in table B*



***INNER JOIN** - fetch the results that exist in both tables*



***RIGHT OUTER JOIN** - all rows from table B, even if they do not exist in table A*

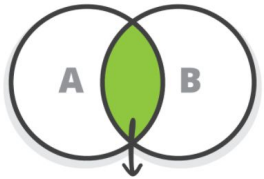
Inner Join

Joins and Unions

```
-- Example
# How many orders got returned
# and what is the total sales
# revenue loss due to product return?
# (HINT: some returned orders cannot be
#  found in orders table)

select count(distinct a.OrderID), sum(a.Sales)
from superstore.orders as a
    inner join
    superstore.returns as b
   on a.OrderID = b.OrderID
;
```

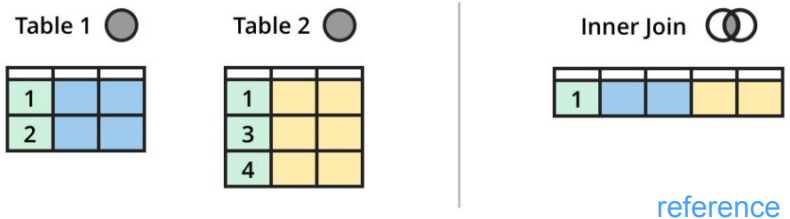
count(distinct a.Order...	sum(a.Sales)
2	4897.25000



INNER JOIN - fetch the results that exist in both tables

Syntax

```
SELECT a.column1, ..., b.columnK, ...
FROM db.table_A  as a
INNER JOIN db.table_B as b
ON a.column1 = b.column1 [AND ...];
```



reference

Left | Right Outer Join

Joins and Unions

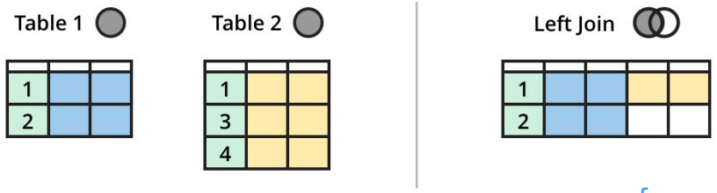
```
-- Example
# Add a Return column to the orders table to indicate if the
# order has been returned

create table superstore.orders_1 as
select a.*,
        case when b.Status is not null then 'Returned'
             else 'Not Returned'
        end as ReturnStatus
from superstore.orders as a
   left join superstore.returns as b
   on a.OrderID = b.OrderID
;
```

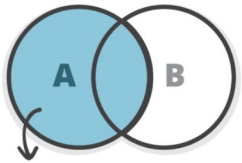
OrderID	ProductID	CustomerID	OrderDate	ReturnStatus
69	213268	58189342	2009-06-03	Returned
69	115501	58189342	2009-06-03	Returned
16768	681809	28395632	2009-07-25	Not Returned
17024	778385	40732966	2009-10-13	Not Returned
359	491105	7210830	2009-12-18	Returned
15808	284312	68464052	2010-12-14	Not Returned
38118	284312	61555104	2011-04-18	Not Returned
29537	681809	33980383	2012-09-19	Not Returned

Syntax

```
SELECT a.column1, ..., b.column1, ...
FROM db.table_A  as a
LEFT | RIGHT JOIN db.table_B as b
ON a.column1 = b.column1 [AND ...];
```



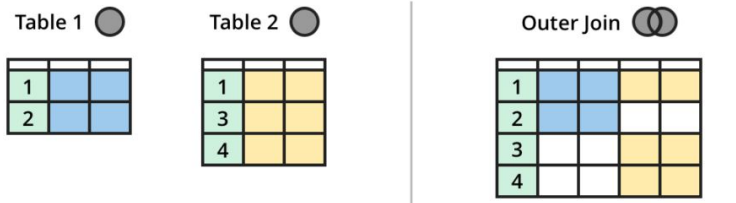
reference



LEFT OUTER JOIN *- all rows from table A, even if they do not exist in table B*

Full Outer Join

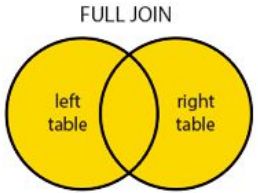
Joins and Unions



reference

Syntax

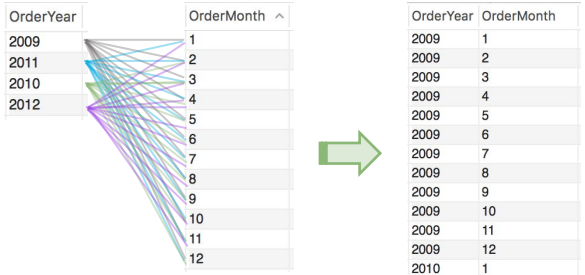
```
SELECT a.column1, ..., b.column1, ...
FROM db.table_A  as a
FULL JOIN db.table_B as b
ON a.column1 = b.column1 [AND ...];
```



Cross Join

Joins and Unions

```
# Cross join
select yr.OrderYear, mn.OrderMonth
from (select distinct year(OrderDate) as OrderYear
        from superstore.orders) yr
    cross join
      (select distinct month(OrderDate) as OrderMonth
        from superstore.orders) mn
order by OrderYear, OrderMonth;
```

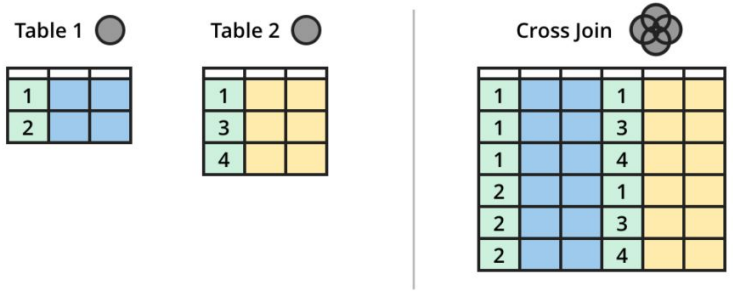


OrderYear	OrderMonth
2009	1
2009	2
2009	3
2009	4
2009	5
2009	6
2009	7
2009	8
2009	9
2009	10
2009	11
2009	12
2010	1
2010	2
2010	3
2010	4
2010	5
2010	6
2010	7
2010	8
2010	9
2010	10
2010	11
2010	12
2011	1
2011	2

...

Syntax

```
SELECT a.column1, ..., b.column1, ...
FROM db.table_A  as a
CROSS JOIN db.table_B as b;
```



reference

CROSS JOIN Usage:

- Creates a Cartesian product of the two tables without specifying any field
- It is an expensive operation!!
- Unless there's a specific purpose, **don't** use it!

Lab #2: SQL Joins

Joins and Unions

1. Create a temporary table that has order detail as well as product details such as product category and sub-category
2. For all products sold, what is the total sales and number of orders without a discount?
3. Of all products sold in 2012, which products had total sales greater than $1000 in both 'Ontario' and 'West' regions?
4. Calculate RFM (Recency, Frequency, Monetary) attributes for each user. Build a customer attributes table that contains the following columns
5. Find customers who has made orders from two consecutive days
6. Find customers who purchased the same product more than once within a week and enjoyed a better discount on the second purchase
7. Calculating year-over-year (YoY) revenue growth trend using self join

Union | Intersect

Joins and Unions

Union | Intersect	Explanation
UNION	• Row combines the results of several SELECT statements
INTERSECT	• Selects the distinct common rows of several results of SELECT query; similar format to UNION
EXCEPT	• Select the distinct rows in the left query results but not in the right one; similar format to UNION

Union

Joins and Unions

```
-- Example
# Find customers who purchased products in December in either 2009 and 2010

select CustomerID, OrderDate
    from superstore.orders
    where month(OrderDate) = 12 and year(OrderDate) = 2009
union
select CustomerID, OrderDate
    from superstore.orders
    where month(OrderDate) = 12 and year(OrderDate) = 2010;
```

OrderID	OrderDate
17024	2009-10-13
15808	2010-12-14
29537	2012-09-19
38118	2011-04-18
69	2009-06-03
16768	2009-07-25
359	2009-12-18
69	2009-06-03

CustomerID	OrderDate
7210830	2009-12-18
68464052	2010-12-14

Syntax

```
SELECT * FROM db.table_A
UNION [ALL]
SELECT * FROM db.table_B;
```

Note:

- **UNION**: Will remove the duplicates after union
- **UNION ALL**: Will keep the duplicated rows making it faster to run that union

Lab #3: SQL Unions

Joins and Unions

1. Find customers who purchased products on New Years Eve in either Low priority or Critical Priority
2. How many customers purchased products on New Years Eve in either Low priority or Critical Priority
3. Try the same query above with UNION ALL and see if you get same number of records back