Module 3
Data Tier

Advanced SQL

Presented by
Domenico Mossali

IBM Client Innovation Center - Bari

Bari, 27/11/2023

IBM Client Innovation Center
Italy

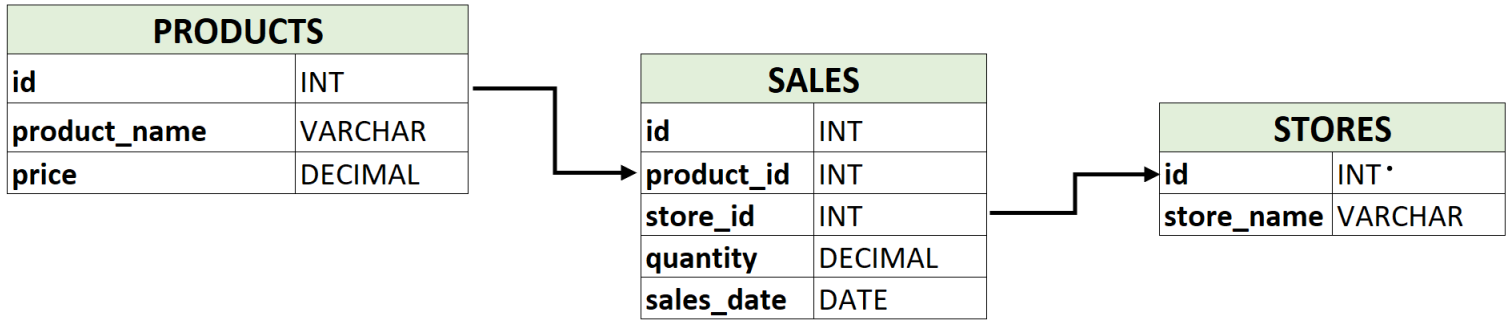
Agenda

1. **SQL JOINS**
2. SQL keywords and constructs

IBM Client Innovation Center
Italy

DB we will work with

In the following session we will use the DB Vendite



What JOIN is

In the SQL language, the **Join** operation combines the rows of two or more tables based on the values contained in the columns.

Implicit Join

We can perform the Join operation implicitly, by indicating the Join condition in the WHERE clause of the SELECT.

```
SELECT * FROM tab1, tab2
WHERE tab1.column = tab2.column
```

What JOIN is #2

Explicit join

Alternatively, we can perform the Join operation explicitly, called **INNER JOIN**, indicating the Join condition in the FROM clause with the **JOIN** and **ON** operators. The word INNER is optional.

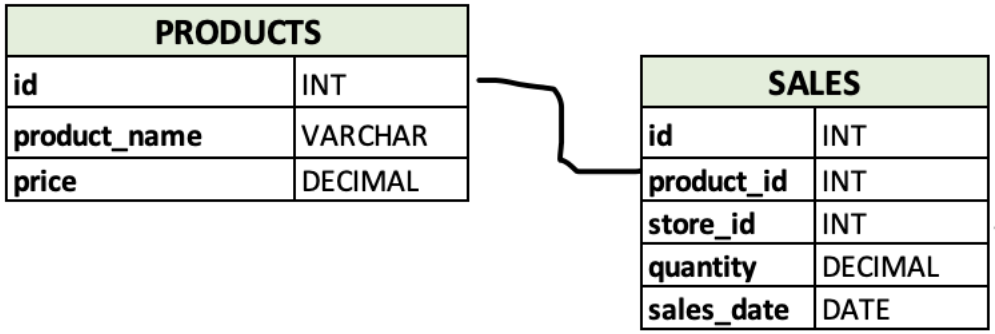
```
SELECT *
FROM tab1 JOIN tab2
ON tab1.column= tab2.column
```

The final result is always the same.

A query with a join produces a table with that rows of the tables that meet the join condition.

An example

We consider 2 tables



I need for a table that shows for each **product** the **quantity** has been sold, in the **day** 2022-01-04

We can solve this task either with an implicit or an explicit Join

An example #2

Implicit Join

```
SELECT product_name, quantity, sales_date
FROM products T1, sales T2
WHERE T1.id = T2.product_id
AND sales_date = '2022-01-04';
```

Rows 7; SELECT product_name, quantity, sales_date FROM products T1, sales T2 W

Results | Meta data | Info | Overview / Charts | Rotated table | Results as text

product_name	quantity	sales_date
iPhone 12	25.00	2022-01-04
iPhone	40.00	2022-01-04
iPad	20.00	2022-01-04
iPhone 11	20.00	2022-01-04
Macbook Air	20.00	2022-01-04
Apple Watch	15.00	2022-01-04
Macbook Pro	25.00	2022-01-04

An example #3

Explicit Join

```
SELECT product_name, quantity, sales_date
FROM products JOIN sales
ON products.id = sales.product_id
AND sales_date = '2022-01-04';
```

Rows 7; SELECT product_name, quantity, sales_date FROM products T1, sales T2 W

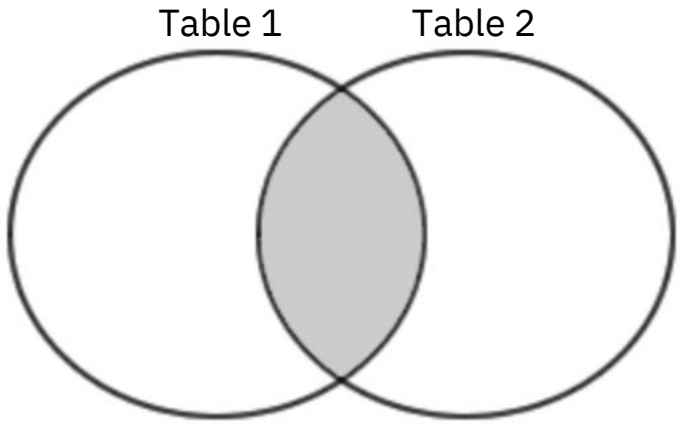
Results | Meta data | Info | Overview / Charts | Rotated table | Results as text

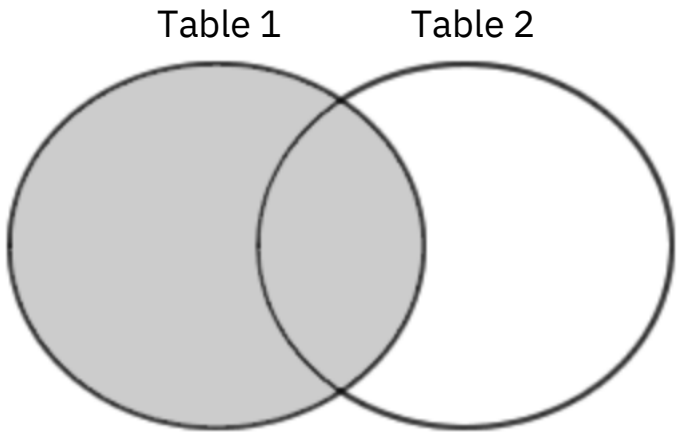
product_name	quantity	sales_date
iPhone 12	25.00	2022-01-04
iPhone	40.00	2022-01-04
iPad	20.00	2022-01-04
iPhone 11	20.00	2022-01-04
Macbook Air	20.00	2022-01-04
Apple Watch	15.00	2022-01-04
Macbook Pro	25.00	2022-01-04

Types of JOIN

INNER JOIN

Combines the rows from the tables that satisfy the join condition. The word INNER is optional. This is the default Join operation already seen in the previous example.





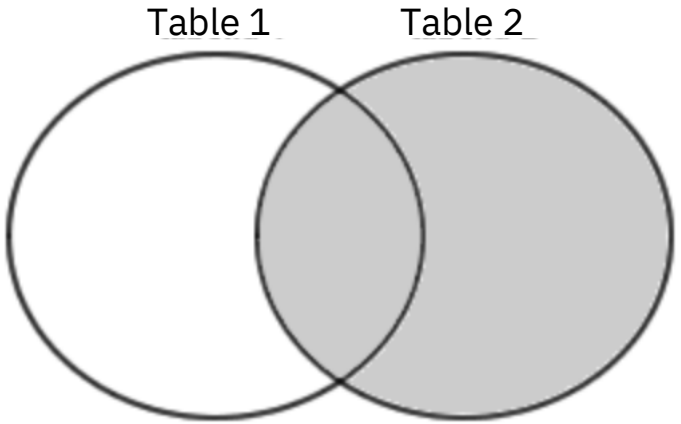
LEFT OUTER JOIN

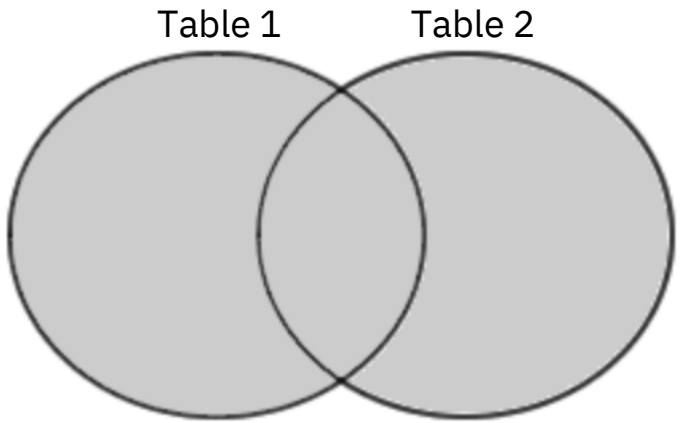
It is an outer join that completes all the rows of the first table with the rows of the second table that satisfy the join condition. Rows of the first unmatched table in the second are spanned with null values. The word OUTER is optional.

Types of JOIN #2

RIGHT OUTER JOIN

It is an outer join that completes all the rows of the second table with the rows of the first table that satisfy the join condition. The rows of the second table with no match in the first are extended with null values. The word OUTER is optional.





FULL OUTER JOIN

It is an external join that includes all the rows of the first and second table with the Right Join and the Left Join. The word OUTER is optional.

OUTER JOIN - a deep dive

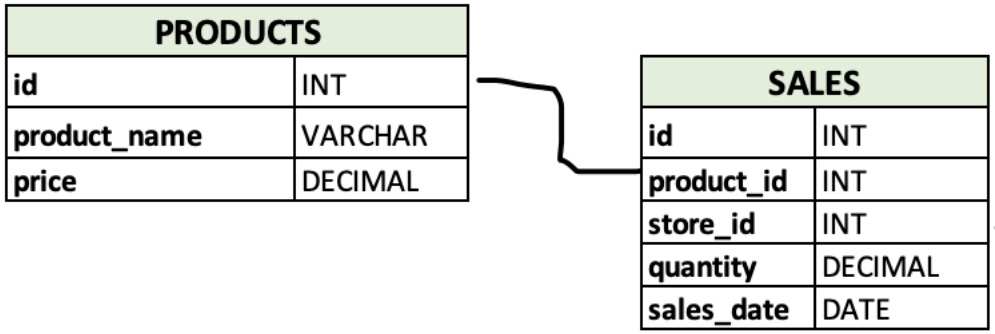
OUTER JOIN allows to keep all rows of a table or both tables.

There are three different types of OUTER JOIN

- LEFT OUTER JOIN: It keeps all the rows from the first table and it adds the rows from the second that satisfy the join condition.
- RIGHT OUTER JOIN keeps all the rows of the second table to which it adds the rows of the first that satisfy the join condition.
- FULL OUTER JOIN: It is the combination of the LEFT and RIGHT OUTER JOIN. Keeps all tuples from both tables, extending them with null values if necessary.

LEFT OUTER JOIN - an example

We consider 2 tables



I want to shown the **product name**, the **quantity** of sold products in the **day** 2022-01-04 and, in order to avoid to neglect rows of the first table without a corresponding in the second (i.e. products not sold in that day) , I use the LEFT OUTER JOIN

LEFT OUTER JOIN - an example #2

```
SELECT product_name, quantity, sales_date
FROM products LEFT JOIN sales
ON products.id = sales.product_id
AND sales_date = '2022-01-04';
```

Rows 9; SELECT product_name, quantity, sales_date FROM products LEFT JOIN sa

Results | Meta data | Info | Overview / Charts | Rotated table | Results as text

product_name	quantity	sales_date
iPhone	40.00	2022-01-04
iPhone 11	20.00	2022-01-04
iPhone 12	25.00	2022-01-04
iPhone 13	<null>	<null>
iPad	20.00	2022-01-04
iPad Pro	<null>	<null>
Apple Watch	15.00	2022-01-04
Macbook Air	20.00	2022-01-04
Macbook Pro	25.00	2022-01-04

The OUTER qualifier is optional. For this reason, there is only LEFT JOIN in the query.

The OUTER JOIN outputs a table with all the records from the first table.

Information about the sold quantity is added only if present.

Agenda

1. SQL JOINS
2. **SQL keywords and constructs**

IBM Client Innovation Center
Italy

What is subquery in SQL?

A subquery is a SQL query nested inside a larger query.

- A subquery may occur in :
  - A SELECT clause
  - A FROM clause
  - A WHERE clause
- You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.
- A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.
- The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

What is subquery in SQL? #2

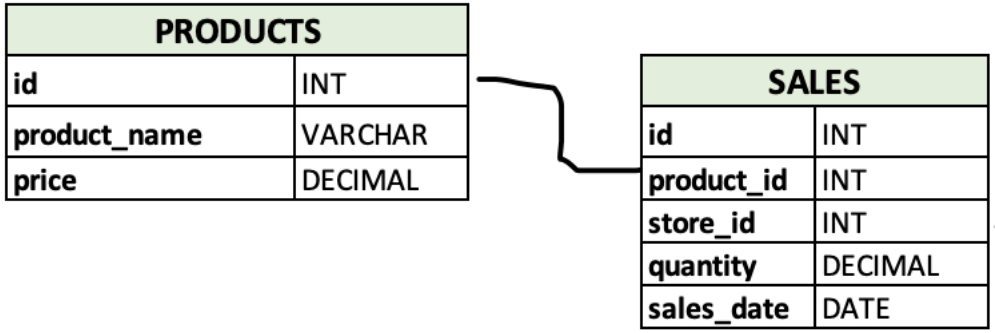
Syntax:

```
SELECT    select_list
FROM      table
WHERE     expr operator
                (SELECT    select_list
                 FROM      table);
```

- The subquery (inner query) executes once before the main query (outer query) executes.
- The main query (outer query) use the subquery result.

Subquery example

We consider 2 tables



I want to know in which days have been sold more iPads than the iPads sold the 2022-01-12

Subquery example #2

I could break the problem in two steps:

```
SELECT product_name, quantity, sales_date
FROM products T1, sales T2
WHERE T1.id = T2.product_id
AND T1.product_name = 'iPad'
AND sales_date = '2022-01-12';
```

Rows 1; SELECT product_name, quantity, sales_date FROM pr

Results | Meta data | Info | Overview / Charts | Rotated table

product_name	quantity	sales_date
iPad	12.00	2022-01-12

```
SELECT product_name, quantity, sales_date
FROM products T1, sales T2
WHERE T1.id = T2.product_id
AND T1.product_name = 'iPad'
AND quantity > 12;
```

Rows 2; SELECT product_name, quantity, sales_date FROM products T1, sales

Results | Meta data | Info | Overview / Charts | Rotated table | Results as text

product_name	quantity	sales_date
iPad	27.00	2022-01-09
iPad	20.00	2022-01-04

Subquery example #3

OR I can solve the problem in a single step:

```
SELECT product_name, quantity, sales_date
FROM products T1, sales T2
WHERE T1.id = T2.product_id
AND T1.product_name = 'iPad'
AND quantity > (
  SELECT quantity
  FROM products T1, sales T2
  WHERE T1.id = T2.product_id
  AND T1.product_name = 'iPad’
  AND sales_date = '2022-01-12'
);
```

Rows 2; SELECT product_name, quantity, sales_date FROM products T1, sales

Results | Meta data | Info | Overview / Charts | Rotated table | Results as text

product_name	quantity	sales_date
iPad	27.00	2022-01-09
iPad	20.00	2022-01-04

Aggregate functions

An aggregate function performs a calculation one or more values and returns a single value.

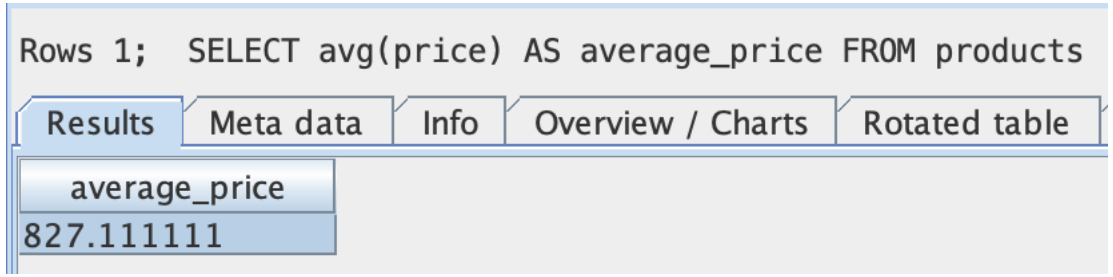
The aggregate function is often used with the GROUP BY clause and HAVING clause of the SELECT statement.

- **COUNT** – counts the number of elements in the group defined
- **SUM** – calculates the sum of the given attribute/expression in the group defined
- **AVG** – calculates the average value of the given attribute/expression in the group defined
- **MIN** – finds the minimum in the group defined
- **MAX** – finds the maximum in the group defined

Aggregate function: AVG

What is the average price of the product we sell?

```
SELECT avg(price) AS average_price
FROM products;
```



Aggregate function: COUNT

How many types of products have been sold in the day 2022-01-04?

```
SELECT count(product_name)
FROM products T1, sales T2
WHERE T1.id = T2.product_id
AND sales_date = '2022-01-04';
```

Rows 1; SELECT count(product_name)

Results | Meta data | Info | Overview

count(product_name)
7

In fact, the types of products sold in the day 2022-01-04 are:

```
SELECT product_name
FROM products T1, sales T2
WHERE T1.id = T2.product_id
AND sales_date = '2022-01-04';
```

Rows 7; SELECT product_name FROM products T1, sale

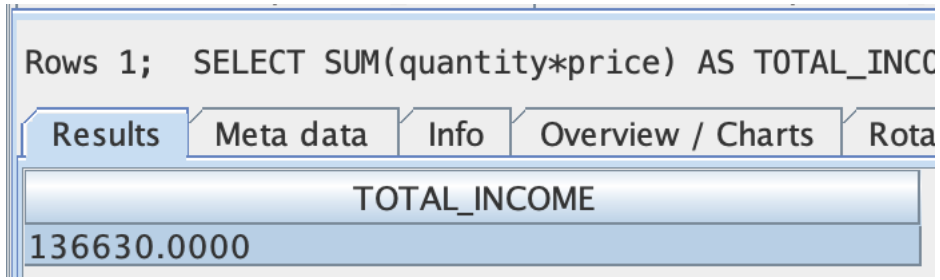
Results | Meta data | Info | Overview / Charts | Rotated

product_name
iPhone 12
iPhone
iPad
iPhone 11
Macbook Air
Apple Watch
Macbook Pro

Aggregate function: SUM

What the total income of the day 2022-01-04?

```
SELECT SUM(quantity*price) AS TOTAL_INCOME
FROM products T1, sales T2
WHERE T1.id = T2.product_id
AND sales_date = '2022-01-04';
```

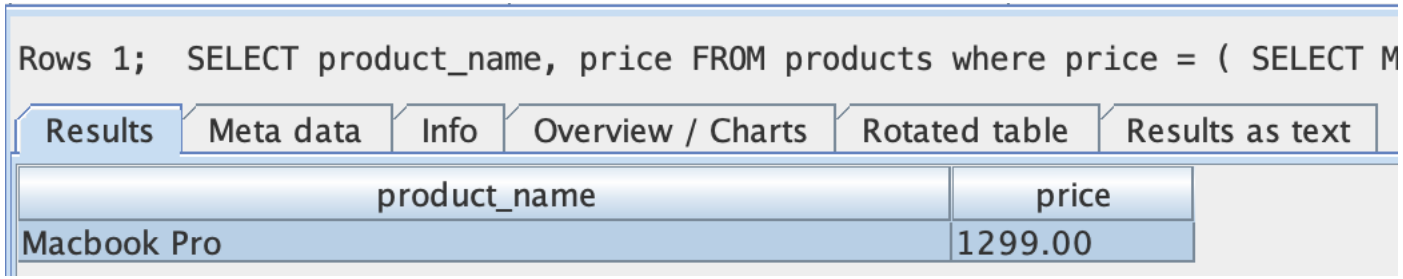


IBM Client Innovation Center
Italy

Aggregate function: MAX

What is the most expensive product we sell?

```
SELECT product_name, price
FROM products
where price = (
  SELECT MAX(price)
  FROM products
);
```



SQL Keyword: ORDER BY

What is the ordered list (by name) of products we sell?

```
SELECT product_name
FROM products
ORDER BY product_name;
```

Rows 9; SELECT product_name FROM products ORDER BY product_name

Results | Meta data | Info | Overview / Charts | Rotated table | Results

product_name
Apple Watch
iPad
iPad Pro
iPhone
iPhone 11
iPhone 12
iPhone 13
Macbook Air
Macbook Pro

SQL Keyword: GROUP BY

How many devices have been sold in each store the day 2022-01-04?

```
SELECT SUM(quantity), store_name
FROM sales SA, stores ST
WHERE SA.store_id = ST.id
AND SA.sales_date = '2022-01-04'
GROUP BY store_name;
```

Rows 2; SELECT SUM(quantity), store_name FROM sales SA, stores ST WHERE

Results | Meta data | Info | Overview / Charts | Rotated table | Results as text

SUM(quantity)	store_name
50.00	North
115.00	West

In fact:

```
SELECT product_name, quantity, store_name
FROM products P, sales SA, stores ST
WHERE P.id = SA.product_id
AND SA.store_id = ST.id
AND SA.sales_date = '2022-01-04';
```

Rows 7; SELECT product_name, quantity, store_name FROM products P, sales SA, store

Results | Meta data | Info | Overview / Charts | Rotated table | Results as text

product_name	quantity	store_name
iPhone 12	25.00	North
iPhone	40.00	West
iPad	20.00	West
iPhone 11	20.00	West
Macbook Air	20.00	West
Apple Watch	15.00	West
Macbook Pro	25.00	North

SQL clause: HAVING

The **HAVING** clause was added to SQL because the **WHERE** keyword cannot be used with aggregate functions.

What are the sale locations where we sold more than 100 devices?

```
SELECT SUM(quantity) item_sold, store_name
FROM sales SA, stores ST
WHERE SA.store_id = ST.id
GROUP BY store_name
HAVING item_sold > 100;
```

Rows 4; SELECT SUM(quantity) item_sold, store_name FROM sales SA, stores

Results | Meta data | Info | Overview / Charts | Rotated table | Results as text

item_sold	store_name
147.00	North
130.00	North East
160.00	North West
115.00	West

SQL Operator: EXISTS

The **EXISTS** operator is used to test for the existence of any record in a subquery.

The **EXISTS** operator returns TRUE if the subquery returns one or more records.

EXISTS Syntax

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

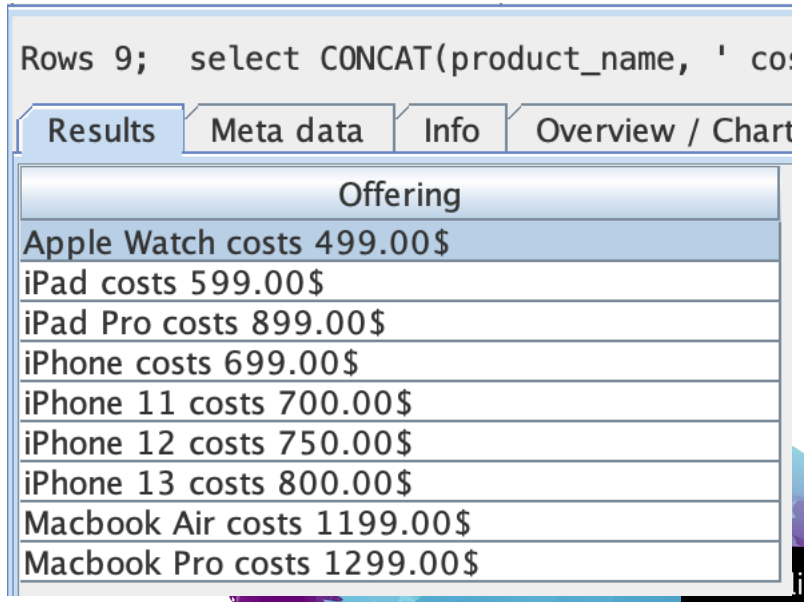
SQL function: CONCAT

The CONCAT() function adds two or more strings together.

Concat Syntax

```
CONCAT(string1, string2, ...., string_n)
```

```
select CONCAT(product_name, ' costs ', price, '$') AS Offering
from products
ORDER BY product_name;
```



SQL function: UNION

The **UNION** operator is used to combine the result-set of two or more SELECT statements

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types

UNION Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

The **UNION** operator selects only distinct values by default. To allow duplicate values, use **UNION ALL**

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

SQL expression: CASE #1

The **CASE** expression goes through conditions and returns a value when the first condition is met

CASE is like an if-then-else statement.

Once a condition is true, it will stop reading and return the result.

If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

CASE Syntax

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

SQL expression: CASE #2

Select for each shop and for each day:

- the sum of the total pieces sold
- the sum of the pieces of products with ID between 1 and 5
- the sum of the pieces of the products with ID between 6 and 9

```
select store_id, sales_date,
sum(quantity) qtaTotale,
sum(case when (product_id between 1 and 5) then quantity else 0 end) qtaProdDaId1AId5,
sum(case when product_id between 6 and 9 then quantity else 0 end) qtaProdDaId6AId9
from sales
group by store_id,sales_date
order by store_id, sales_date
```

select store_id

Rows 16; select store_id, sales_date, sum(quantity) qtaTotale, sum(cas

Results | Meta data | Info | Overview / Charts | Rotated table | Results as text

store_id	sales_date	qtaTotale	qtaProdDaId1AId5	qtaProdDaId6AId9
1	2022-01-02	40	20	20
1	2022-01-04	50	25	25
1	2022-01-05	30	15	15
1	2022-01-09	27	27	0
1	2022-02-23	20	20	0

What views are?

A view is a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or selected rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are a type of virtual tables allow users to do the following:

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

The basic CREATE VIEW syntax is as follows

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

View: Example

We refer to the database *Vendite*

We search for all sales performed in the store = ‘North’

```
SELECT sales.*
FROM sales, stores
WHERE sales.store_id = stores.id
AND stores.store_name = 'North';
```

Rows 7; SELECT sales.* FROM sales, stores WHERE sales.sto

Results | Meta data | Info | Overview / Charts | Rotated table

id	product_id	store_id	quantity	sales_date
1	1	1	20.00	2022-01...
2	2	1	15.00	2022-01...
3	3	1	25.00	2022-01...
10	5	1	27.00	2022-01...
12	8	1	20.00	2022-01...
13	7	1	15.00	2022-01...
19	9	1	25.00	2022-01...

View: Example #2

We can create a *View* that shows only sales performed in the store ‘North’ in such a way

```
CREATE VIEW north_sales AS
SELECT sales.*
FROM sales, stores
WHERE sales.store_id = stores.id
AND stores.store_name = 'North';
```

View: Example #3

We can perform queries against the view *north_sales*

```
SELECT *
FROM north_sales;
```

Rows 7; SELECT * FROM north_sales

Results | Meta data | Info | Overview / Charts | Rotated table

id	product_id	store_id	quantity	sales_date
1	1	1	20.00	2022-01...
2	2	1	15.00	2022-01...
3	3	1	25.00	2022-01...
10	5	1	27.00	2022-01...
12	8	1	20.00	2022-01...
13	7	1	15.00	2022-01...
19	9	1	25.00	2022-01...

View: Example #4

We can perform queries against the view *north_sales*

```
SELECT *
FROM north_sales
WHERE quantity >=20;
```

Rows 5; SELECT * FROM north_sales WHERE quantity >=20

Results | Meta data | Info | Overview / Charts | Rotated table

id	product_id	store_id	quantity	sales_date
1	1	1	20.00	2022-01...
3	3	1	25.00	2022-01...
10	5	1	27.00	2022-01...
12	8	1	20.00	2022-01...
19	9	1	25.00	2022-01...

SQL Suggested References

https://www.w3schools.com/sql/