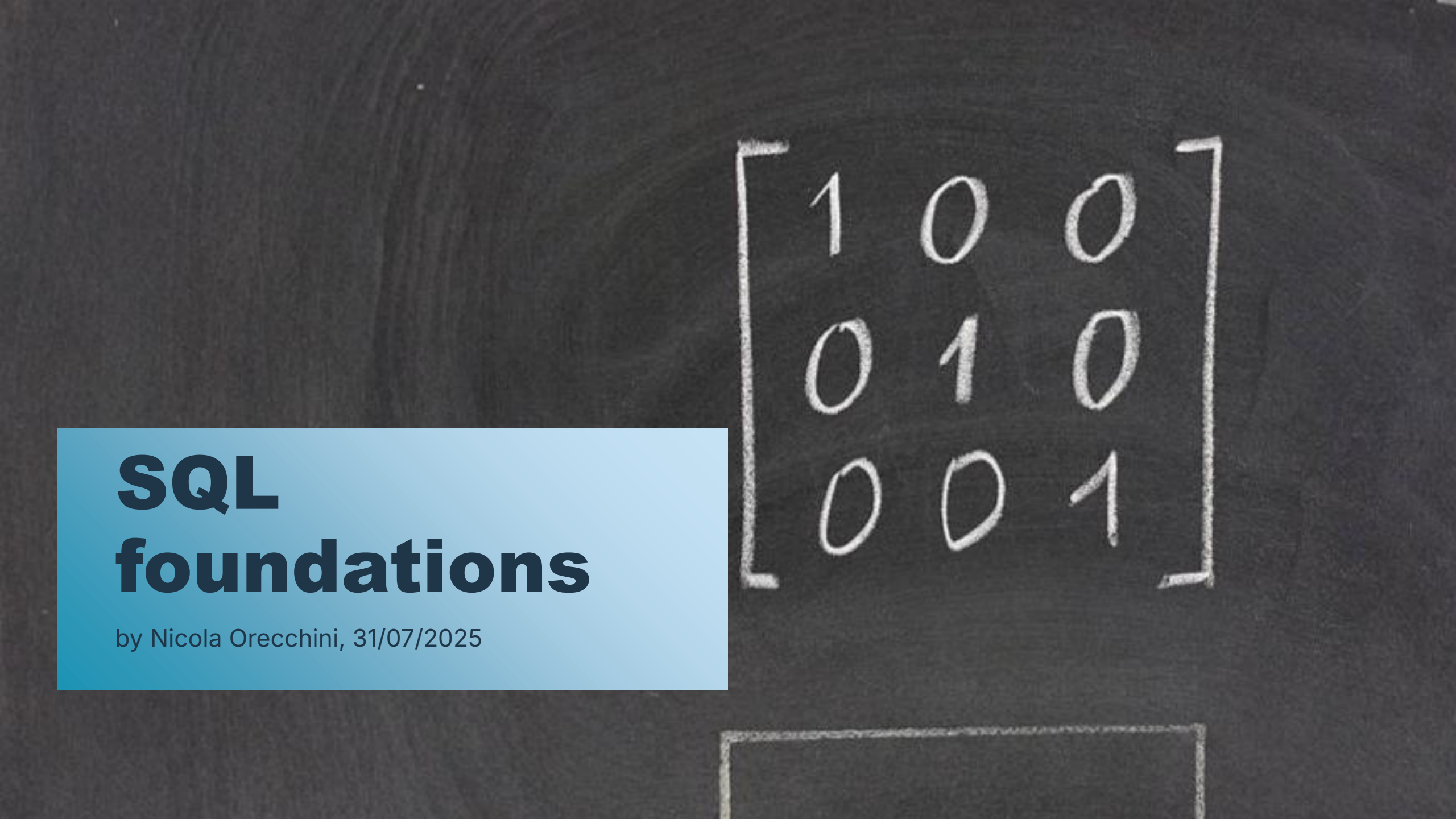



# SQL foundations

by Nicola Orecchini, 31/07/2025

# Agenda

1. Introduction to relational data model
2. SQL basic operations

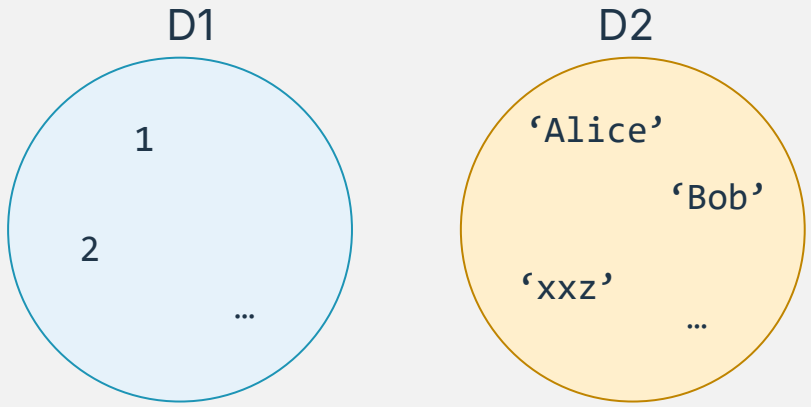
Deep dive on JOIN

3. Other SQL operations

# Agenda

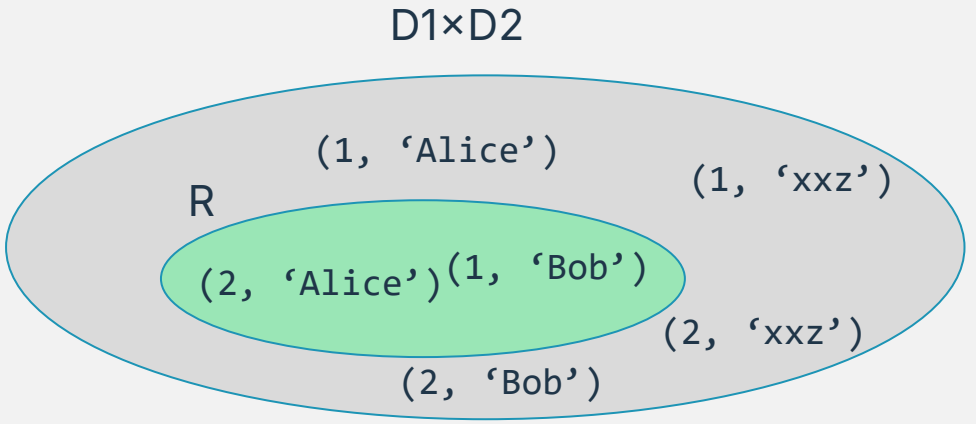
1. Introduction to relational data model
2. SQL basic operations
   Deep dive on JOIN
3. Other SQL operations

# A relational database is a database that stores data in tables (called *relations*)




$D_1 \times D_2 \times \cdots \times Dn$ are domains: sets of possible values for a variable. Each domain corresponds to a database column type, e.g.:

- $D_1 = \mathbb{Z}$ (integers for id)
- $D_2$=strings (for name)



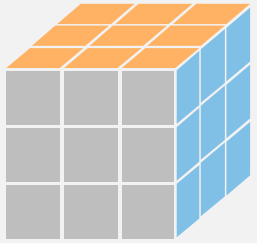

A relation R is a subset of the cartesian product $D_1 \times D_2 \times \cdots \times D_n$ :
$R \subseteq D_1 \times D_2 \times \cdots \times D_n$

Each element of R is a n-tuple $(d_1, d_2, \ldots, d_n)$ where $d_i \in D_i$.

So, a relation corresponds to the concept of table, in a database. A table is thus a set of tuples

The word «relational» in relational databases refers to the fact that data is stored in structured tables (relations), and not to the fact that tables have relationships between them (e.g., keys)

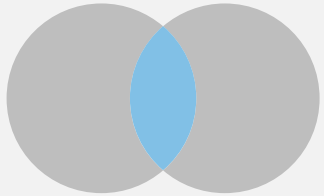
# Users can manipulate tables through Structured Query Language, a language that allows multiple operations



Filters

σ Selection

π Projection



Set operations

∪ Union

∩ Intersection

\ Exclusion

Relational operations

× Cartesian Product

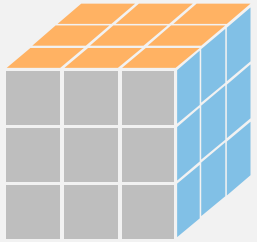
⋈ Join

Source: https://blog.jooq.org/say-no-to-venn-diagrams-when-explaining-joins/; https://www.geeksforgeeks.org/dbms/difference-between-selection-and-projection-in-dbms/

# Agenda

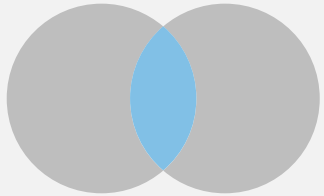
1. Introduction to relational data model
2. SQL basic operations
   Deep dive on JOIN
3. Other SQL operations

# Filters allow to take only specific rows or columns of a table



| | σ Selection | π Projection |
|---|---|---|
| What it does | Considers only rows of a table that meet a specified condition | Considers only columns of a table as specified by a list of names |
| Example | `SELECT *`<br>`FROM customer`<br>`WHERE name='bob'` | `SELECT name, surname`<br>`FROM customer` |

# Set operations combine or compare the results of multiple queries that return the same column structure



| | ∪ Union | ∩ Intersection | \ Exclusion |
|---|---|---|---|
| What it does | Starting from 2 (or more) tables, creates a new table containing all records of the 1st and all those of the 2nd. The 2 tables must be made of the same type of tuples | Starting from 2 (or more) tables, creates a new table containing only records that are present in both tables. The 2 tables must be made of the same type of tuples | Starting from 2 (or more) tables, creates a new table containing only records that are in the first but not in the second. The 2 tables must be made of the same type of tuples |
| Example | `SELECT name, surname`<br>`FROM customer`<br>`UNION`<br>`SELECT name, surname`<br>`FROM staff` | `SELECT name, surname`<br>`FROM customer`<br>`INTERSECT`<br>`SELECT name, surname`<br>`FROM staff` | `SELECT name, surname`<br>`FROM customer`<br>`EXCEPT`<br>`SELECT name, surname`<br>`FROM staff` |

# Relational operations combine information from two or more tables by matching rows based on a condition

| | Cartesian product | Join |
|---|---|---|
| What it does | Given 2 (or more) tables, creates a new table whose records are tuples obtained by combining a record of the first table with one of the second, until all possible pairs have been generated | Given 2 (or more) tables, calculates the Cartesian product between them, and then filters only tuples where a specific condition (specified by the user) is met. Then, depending on the join type, special extra records can be returned |
| Example | `SELECT * FROM customer, staff` | `SELECT * FROM customer JOIN staff ON customer.cust_id=staff.st_id` |

# Agenda

1. Introduction to relational data model
2. SQL basic operations
   - Deep dive on JOIN
3. Other SQL operations

# Joins are of 2 types: inner & outer

| | Inner join | Outer join |
|---|---|---|
| What it does | A Cartesian product (also called “Cross join”) between 2 (or more) tables in which only combinations that fulfil a given predicate are retained | An inner join with extra records. These extra records are rows from either the LEFT, the RIGHT, or both (FULL) tables, for which no rows satisfying the predicate were found in the inner join results |

Examples (equivalent variants)

```sql
-- "Classic" ANSI JOIN syntax
SELECT *
FROM customer c
JOIN staff s ON c.cust_id=s.staff_id
```

```sql
-- "Old" syntax using a "CROSS JOIN"
SELECT *
FROM customer c, staff s
WHERE c.cust_id=s.staff_id
```

```sql
SELECT *
FROM customer
LEFT JOIN staff ON
customer.cust_id=staff.st_id
```

LEFT/RIGHT/FULL

```sql
SELECT *
FROM customer c
JOIN staff s ON c.cust_id=s.st_id

UNION

SELECT rows_not_matched.*, NULL, ..., NULL
FROM (
  SELECT rows_not_matched.*
  FROM customer c

  EXCEPT

  SELECT c.*
  FROM customer c
  JOIN staff s ON c.cust_id=s.st_id
) rows_not_matched
```

Source: https://blog.jooq.org/say-no-to-venn-diagrams-when-explaining-joins/

Click here for interactive example

# **Inner Join** is a filtered Cartesian product

orders

| Order_id | Customer_id | Order_date |
|---|---|---|
| 1003 | AZ501 | 2025-07-03 |
| 1004 | BB223 | 2025-07-03 |
| 1005 | CX987 | 2025-07-03 |

orders-products

| Order_id | product_id | quantity |
|---|---|---|
| 1003 | Xxx705 | 2 |
| 1003 | Xxx102 | 1 |
| 1003 | Xxx258 | 1 |
| 1004 | Xxx258 | 3 |

1

| Order_id | Customer_id | Order_date | Order_id | product_id | quantity |
|---|---|---|---|---|---|
| 1003 | AZ501 | 2025-07-03 | 1003 | Xxx705 | 2 |
| 1003 | AZ501 | 2025-07-03 | 1003 | Xxx102 | 1 |
| 1003 | AZ501 | 2025-07-03 | 1003 | Xxx258 | 1 |
| 1003 | AZ501 | 2025-07-03 | 1004 | Xxx258 | 3 |
| 1004 | BB223 | 2025-07-03 | 1003 | Xxx705 | 2 |
| 1004 | BB223 | 2025-07-03 | 1003 | Xxx102 | 1 |
| 1004 | BB223 | 2025-07-03 | 1003 | Xxx258 | 1 |
| 1004 | BB223 | 2025-07-03 | 1004 | Xxx258 | 3 |
| 1005 | CX987 | 2025-07-03 | 1003 | Xxx705 | 2 |
| 1005 | CX987 | 2025-07-03 | 1003 | Xxx102 | 1 |
| 1005 | CX987 | 2025-07-03 | 1003 | Xxx258 | 1 |
| 1005 | CX987 | 2025-07-03 | 1004 | Xxx258 | 3 |

σ `WHERE orders.order_id=orders-products.order_id`

2

| Order_id | Customer_id | Order_date | Order_id | product_id | quantity |
|---|---|---|---|---|---|
| 1003 | AZ501 | 2025-07-03 | 1003 | Xxx705 | 2 |
| 1003 | AZ501 | 2025-07-03 | 1003 | Xxx102 | 1 |
| 1003 | AZ501 | 2025-07-03 | 1003 | Xxx258 | 1 |
| 1004 | BB223 | 2025-07-03 | 1004 | Xxx258 | 3 |

```
SELECT *
FROM orders o, orders-products p
WHERE o.order_id=p.order_id
```

1 2

× Cross product

σ Selection

Click here for interactive example

U Union  \ Exclusion

# **Left Join** is an Inner Join unioned with unmatched records from the left table padded with NULLs

```
SELECT *
FROM order o
JOIN order-product p ON o.order_id=p._id

UNION

SELECT rows_not_matched.*, NULL, ..., NULL
FROM (
  SELECT o.*
  FROM order o

  EXCEPT

  SELECT o.*
  FROM order o
  JOIN order-prod p ON o._id=p._id
) rows_not_matched
```

5

| Order_id | Customer_id | Order_date | Order_id | product_id | quantity |
|---|---|---|---|---|---|
| 1003 | AZ501 | 2025-07-03 | 1003 | Xxx705 | 2 |
| 1003 | AZ501 | 2025-07-03 | 1003 | Xxx102 | 1 |
| 1003 | AZ501 | 2025-07-03 | 1003 | Xxx258 | 1 |
| 1004 | BB223 | 2025-07-03 | 1004 | Xxx258 | 3 |
| 1005 | CX987 | 2025-07-03 | NULL | NULL | NULL |

4

| Order_id | Customer_id | Order_date | Order_id | product_id | quantity |
|---|---|---|---|---|---|
| 1005 | CX987 | 2025-07-03 | NULL | NULL | NULL |

U

3

| Order_id | Customer_id | Order_date |
|---|---|---|
| 1005 | CX987 | 2025-07-03 |

1

| Order_id | Customer_id | Order_date | Order_id | product_id | quantity |
|---|---|---|---|---|---|
| 1003 | AZ501 | 2025-07-03 | 1003 | Xxx705 | 2 |
| 1003 | AZ501 | 2025-07-03 | 1003 | Xxx102 | 1 |
| 1003 | AZ501 | 2025-07-03 | 1003 | Xxx258 | 1 |
| 1004 | BB223 | 2025-07-03 | 1004 | Xxx258 | 3 |

\

| Order_id | Customer_id | Order_date |
|---|---|---|
| 1003 | AZ501 | 2025-07-03 |
| 1004 | BB223 | 2025-07-03 |
| 1005 | CX987 | 2025-07-03 |

| Order_id | Customer_id | Order_date |
|---|---|---|
| 1003 | AZ501 | 2025-07-03 |
| 1003 | AZ501 | 2025-07-03 |
| 1003 | AZ501 | 2025-07-03 |
| 1004 | BB223 | 2025-07-03 |

2

Click here for interactive example

# What's the difference between putting the pairing condition in the ON clause vs in the WHERE?

```
INSERT INTO existing
SELECT * FROM incoming
WHERE tech_date > '2025-
07-01'
```

Source: https://stackoverflow.com/questions/354070/sql-join-what-is-the-difference-between-where-clause-and-on-clause

# Agenda

1. Introduction to relational data model
2. SQL basic operations

   Deep dive on JOIN

3. Other SQL operations

# Null-safe equality

```
INSERT INTO existing
SELECT * FROM incoming
WHERE tech_date > '2025-
07-01'
```

# Window over partition by

```
INSERT INTO existing
SELECT * FROM incoming
WHERE tech_date > '2025-
07-01'
```

# Row number, Rank, Dense Rank

```
INSERT INTO existing
SELECT * FROM incoming
WHERE tech_date > '2025-07-01'
```


# Nicola Orecchini

nicolaorecchini@gmail.com