Class - 04

Unnormalized Data

Order ID	Customer Name	Customer Phone	Product	Category	Price	Quantity	Supplier	Supplier Phone
101	John Doe	1234567890	Laptop	Electronics	1000	1	Dell	9876543210
102	Alice Smith	0987654321	Mouse	Accessories	50	2	Logitech	9123456789
103	John Doe	1234567890	Keyboard	Accessories	80	1	Logitech	9123456789

Why Do We Need to Normalize?

1. Avoid Data Redundancy → "John Doe" appears multiple times.
2. Improve Data Consistency → Supplier details are repeated and might cause inconsistency.
3. Better Data Integrity → Helps maintain relationships without duplication.
4. Efficient Storage → Reducing redundancy saves space.
5. Easier Updates → Changing a supplier's phone number requires updates in multiple rows in an unnormalized table.

Normalized Data

Orders Table

OrderID	CustomerID	ProductID	Quantity
101	C001	P001	1
102	C002	P002	2
103	C001	P003	1

Customers Table

CustomerID	CustomerName	CustomerPhone
C001	John Doe	1234567890
C002	Alice Smith	0987654321

Products Table

ProductID	ProductName	CategoryID	Price	SupplierID
P001	Laptop	CAT1	1000	S001
P002	Mouse	CAT2	50	S002
P003	Keyboard	CAT2	80	S002

Categories Table

CategoryID	CategoryName
CAT1	Electronics
CAT2	Accessories

Suppliers Table

SupplierID	SupplierName	SupplierPhone
S001	Dell	9876543210
S002	Logitech	9123456789

Types of Keys in a Database

Primary Key

A Primary Key is a unique identifier for each record in a table. It cannot be **NULL** and must be **unique**.

CustomerID (PK)	CustomerName	Email	PhoneNumber
1	Alice	alice@email.com	555-1234
2	Bob	bob@email.com	555-5678

Foreign Key

A Foreign Key establishes a relationship between two tables. It refers to a **Primary Key** in another table.

OrderID (PK)	CustomerID (FK)	OrderDate
101	1	2024-02-28
102	2	2024-02-29

Types of Keys in a Database

Unique Key

A Unique Key ensures that values in a column remain unique, but unlike a Primary Key, it **can** contain **NULL** values.

EmployeeID (PK)	EmployeeName	Email
1	John Doe	john@email.com
2	Jane Doe	jane@email.com

Composite Key

A Composite Key consists of two or more columns combined to uniquely identify a row.

StudentID	CourseID	EnrollmentDate
1	101	2024-02-28
1	102	2024-02-29

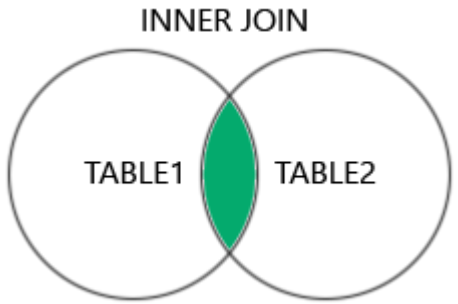
Joins in PostgreSQL

PostgreSQL supports:

1. Inner join
2. Left join
3. Right join
4. Full outer join
5. Cross join
6. Natural join
7. Self-join (special kind of join)

Inner join

Returns only the rows where there is a match in both tables.



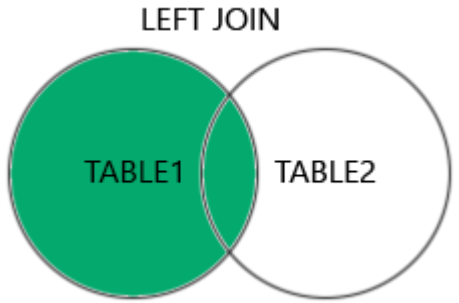
Joins in PostgreSQL

PostgreSQL supports:

1. Inner join
2. Left join
3. Right join
4. Full outer join
5. Cross join
6. Natural join
7. Self-join (special kind of join)

Left join

Get all from the left, even if no match.



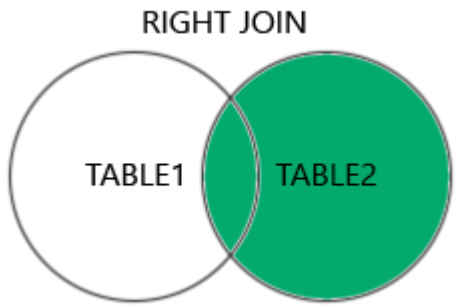
Joins in PostgreSQL

PostgreSQL supports:

1. Inner join
2. Left join
3. Right join
4. Full outer join
5. Cross join
6. Natural join
7. Self-join (special kind of join)

Right join

Get all from the right, even if no match.



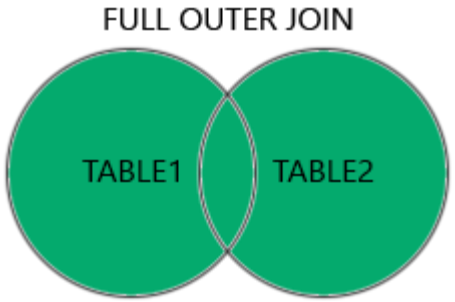
Joins in PostgreSQL

PostgreSQL supports:

1. Inner join
2. Left join
3. Right join
4. Full outer join
5. Cross join
6. Natural join
7. Self-join (special kind of join)

Full outer join

Get everything!



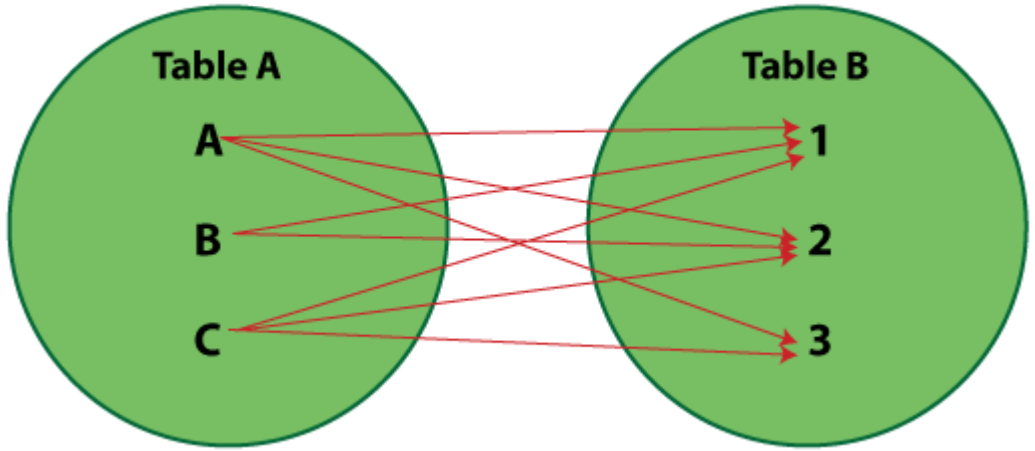
Joins in PostgreSQL

PostgreSQL supports:

1. Inner join
2. Left join
3. Right join
4. Full outer join
5. Cross join
6. Natural join
7. Self-join (special kind of join)

Cross join

Every combination of rows.



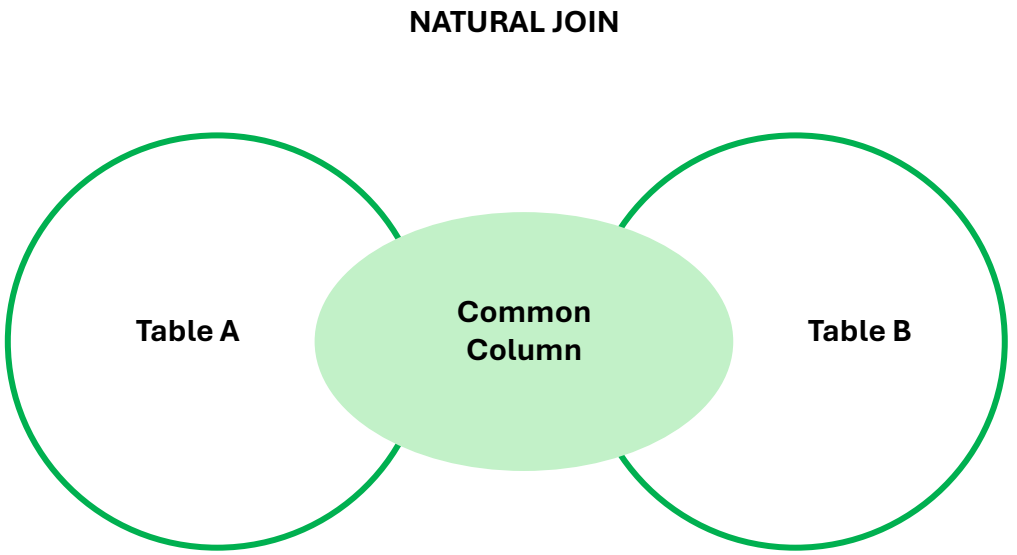
Joins in PostgreSQL

PostgreSQL supports:

1. Inner join
2. Left join
3. Right join
4. Full outer join
5. Cross join
6. Natural join
7. Self-join (special kind of join)

Natural join

Automatically joins tables on columns with the same name.



Joins in PostgreSQL

PostgreSQL supports:

1. Inner join
2. Left join
3. Right join
4. Full outer join
5. Cross join
6. Natural join
7. Self-join (special kind of join)

Self join

Joining a Table with Itself