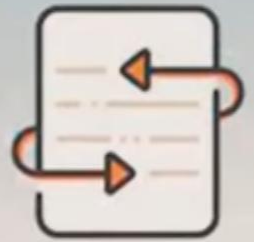
What is SQL ?

Structured Query Language (SQL) is a programming language used to interact with databases

Create

Read



Update

Delete

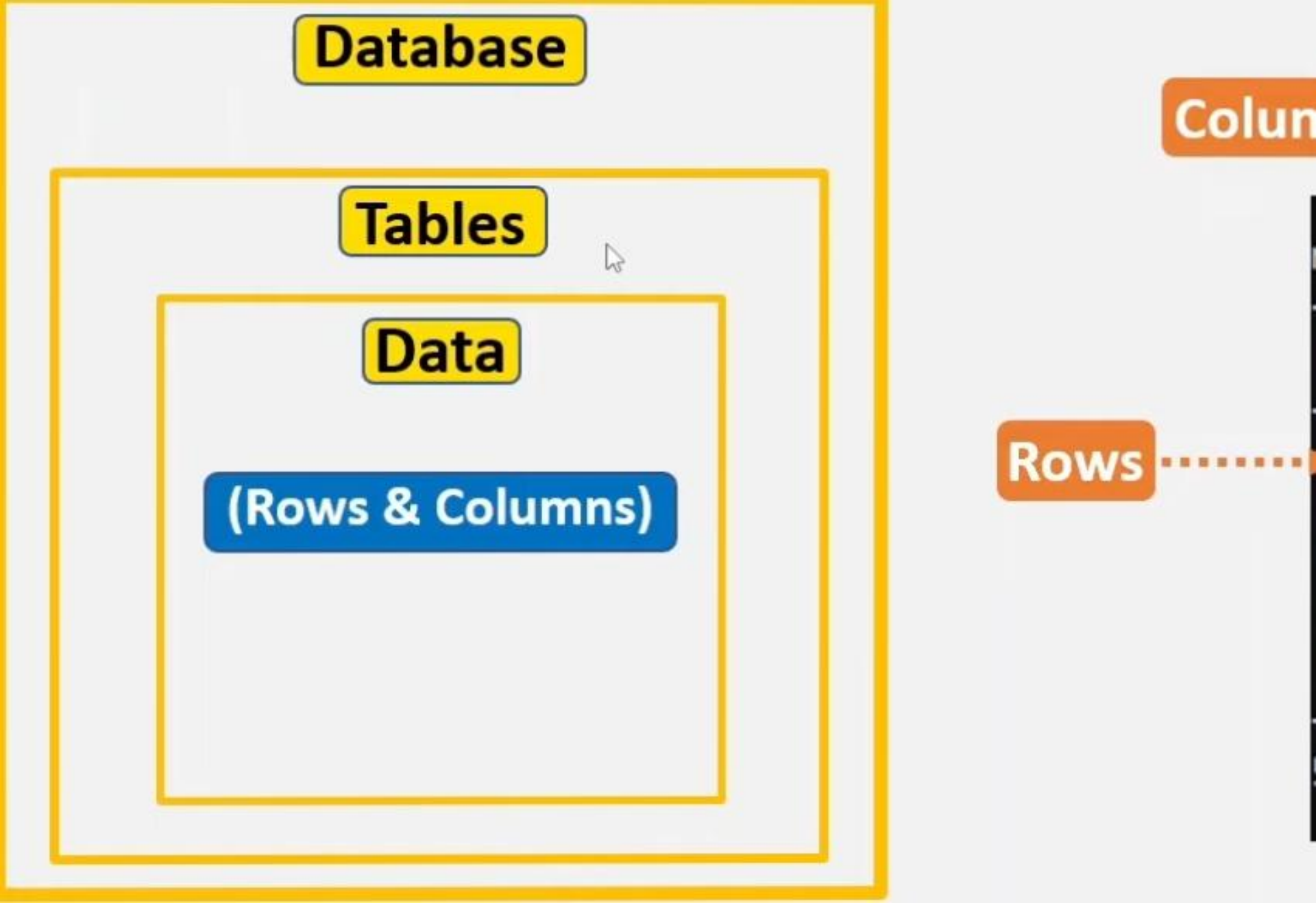
C R U D

What is Database ?

Database is a system that allow users to store and organise data

DATABASE USE

- LARGE AMOUNT OF DATA
- STORE REAL TIME DATA FROM WEBSITES/APPS
- EASILY COMBINE WITH DIFFERENT DATASETS ✔
- AUTOMATE STEPS AND CAN RE-USE
- EASY & SAFE ACCESS
- DATA INTEGRITY
- DEEP SEARCH CAPABILITIES


Database
Tables
Data
(Rows & Columns)
Colun
Rows

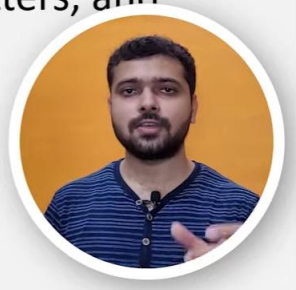
Data Types

- Data type of a column defines what value the column can store in table
- Defined while creating tables in database
- Data types mainly classified into three categories + most used
  - String: char, varchar, etc
  - Numeric: int, float, bool, etc
  - Date and time: date, datetime, etc

Data Types

Commonly Used data types in SQL:

- **int:** used for the integer value
- **float:** used to specify a decimal point number
- **bool:** used to specify Boolean values true and false
- **char:** fixed length string that can contain numbers, letters, and special characters
- **varchar:** variable length string that can contain numbers, letters, and special characters
- **date:** date format YYYY-MM-DD
- **datetime:** date & time combination, format is YYYY-MM-DD



Primary and Foreign Keys:

Primary key (PK):

- A Primary key is a unique column we set in a table to easily identify and locate data in queries
- A table can have only one primary key, which should be unique and NOT NULL

Foreign keys (FK):

- A Foreign key is a column used to link two or more tables toget
- A table can have any number of foreign keys, can contain dup and NULL values

Constraints

- Constraints are used to specify rules for data in a table
- This ensures the accuracy and reliability of the data in the table
- Constraints can be specified when the table is created with the CREATE TABLE statement, or
- after the table is created with the ALTER TABLE statement
- Syntax

```
CREATE TABLE table_name (
  column1 datatype constraint,
  column2 datatype constraint,
  column3 datatype constraint,
  ....
);
```

Constraints

Commonly used constraints in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE
- FOREIGN KEY - Prevents actions that would destroy links between tables (used to link multiple tables together)
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is spec
- CREATE INDEX - Used to create and retrieve data from the da very quickly

Create Table

The CREATE TABLE statement is used to create a new table in a database

- **Syntax**

```
CREATE TABLE table_name
(
   column_name1 datatype constraint,
   column_name2 datatype constraint,
   column_name3 datatype constraint,
);
```

- **Example**

```
CREATE TABLE customer
(
   "ID"  int8  PRIMARY KEY,
   "Name"  varchar(50) NOT NULL,
   "Age"  int  NOT NULL,
   "City"  char(50),
   "Salary"  numeric
);
```

Browser

Properties SQL Statistics Dependencies Dependents postgres/postgres@PostgreSQL 14*

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (2)
      - postgres
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas
        - Subscriptions
      - testdb
    - Login/Group Roles
    - Tablespaces

postgres/postgres@PostgreSQL 14

No limit

Query Query History

Scratch Pad

```
1  CREATE DATABASE testdb
```

Data output Messages Notifications

```
CREATE DATABASE

Query returned successfully in 18 secs 817 msec.
```

Total rows: 0 of 0 Query complete 00:00:18.817 Ln 1, Col 23

pgAdmin File Object Tools Help

Browser

Properties SQL Statistics Dependencies Dependents postgres/post... testdb/postgres@PostgreS

testdb/postgres@PostgreSQL 14

No limit

Execute/Refresh
F5

Query Query History

Scratch Pad

```
1  CREATE TABLE customer
2  (
3      "ID"  int8  PRIMARY KEY,
4      "Name"  varchar(50) NOT NULL,
5      "Age"  int  NOT NULL,
6      "City"  char(50),
7      "Salary"  numeric
8  );
9
```

Data output Messages Notifications

```
CREATE TABLE

Query returned successfully in 543 msec.
```

Total rows: 0 of 0 Query complete 00:00:00.543

Query returned msec.

- Publications
- Schemas (1)
  - public
    - Aggregates
    - Collations
    - Domains
    - FTS Configurations
    - FTS Dictionaries
    - FTS Parsers
    - FTS Templates
    - Foreign Tables
    - Functions
    - Materialized Views
    - Operators
    - Procedures
    - Sequences
    - Tables
    - Trigger Functions
    - Types
    - Views
  - Subscriptions
- testdb1
- Login/Group Roles

pgAdmin File Object Tools Help

Browser

- Publications
- Schemas (1)
  - public
    - Aggregates
    - Collations
    - Domains
    - FTS Configurations
    - FTS Dictionaries
    - FTS Parsers
    - FTS Templates
    - Foreign Tables
    - Functions
    - Materialized Views
    - Operators
    - Procedures
    - Sequences
    - Tables (1)
      - customer
    - Trigger Functions
    - Types
    - Views
- Subscriptions
- testdb1

Properties SQL Statistics Dependencies Dependents postgres/post... testdb/postgres@PostgreS

testdb/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SELECT * FROM customer
```

Scratch Pad

Data output Messages Notifications

ID [PK] bigint	Name character varying (50)	Age integer	City character (50)	Salary numeric

Total rows: 0 of 0 Query complete 00:00:01.505

Successfully run. Total query runtime: 1 secs 5

Insert Values In Table

The INSERT INTO statement is used to insert new records in a table

- **Syntax**

```
INSERT INTO TABLE_NAME
(column1, column2, column3,...columnN)
VALUES
(value1, value2, value3,...valueN);
```

- **Example**

```
INSERT INTO customer
(CustID, CustName, Age, City, Salary)
VALUES
(1, 'Sam', 26, 'Delhi', 9000),
(2, 'Ram', 19, 'Bangalore', 11000),
(3, 'Pam', 31, 'Mumbai', 6000),
(4, 'Jam', 42, 'Pune', 10000);
```

pgAdmin File Object Tools Help

Browser

- FTS Parsers
- FTS Templates
- Foreign Tables
- Functions
- Materialized Views
- Operators
- Procedures
- Sequences
- Tables (2)
  - cust
  - customer
    - Columns (5)
      - custid
      - custname
      - age
      - city
      - salary
    - Constraints
    - Indexes
    - RLS Policies
    - Rules
    - Triggers
- Trigger Functions

Properties SQL Statistics Dependencies Dependents testdb/postgres@PostgreSQL 14*

testdb/postgres@PostgreSQL 14

No limit

Query Query History

```
1  INSERT INTO customer
2  (CustID, CustName, Age, City, Salary)
3  VALUES
4  (1, 'Sam', 26, 'Delhi', 9000),
5  (2, 'Ram', 19, 'Bangalore', 11000),
6  (3, 'Pam', 31, 'Mumbai', 6000),
7  (4, 'Jam', 42, 'Pune', 10000);
8
```

Scratch Pad

Data output Messages Notifications

```
INSERT 0 4

Query returned successfully in 395 msec.
```

Query returned msec.

Total rows: 0 of 0 Query complete 00:00:00.395

Update Values In Table

The UPDATE command is used to update existing rows in a table

- **Syntax**

```
UPDATE TABLE_NAME
SET “Column_name1” = ‘value1’, “Column_name2” = ‘value2’
WHERE “ID” = ‘value’
```

- **Example**

```
UPDATE customer
SET CustName = 'Xam’, Age= 32
WHERE CustID = 4;
```

pgAdmin File Object Tools Help

Browser

Properties SQL Statistics Dependencies Dependents testdb/postgres@PostgreSQL 14*

- FTS Parsers
- FTS Templates
- Foreign Tables
- Functions
- Materialized Views
- Operators
- Procedures
- Sequences
- Tables (2)
  - cust
  - customer
    - Columns (5)
      - custid
      - custname
      - age
      - city
      - salary
    - Constraints
    - Indexes
    - RLS Policies
    - Rules
    - Triggers
- Trigger Functions

testdb/postgres@PostgreSQL 14

No limit

Query Query History

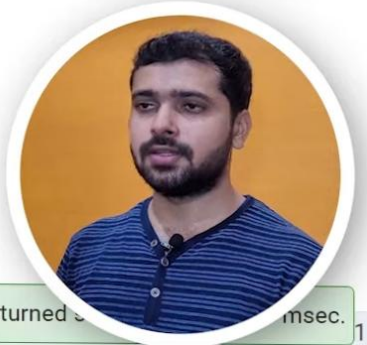
Scratch Pad

```
1  UPDATE customer
2  SET custname = 'Xam', age= 32
3  WHERE custid = 4;
4
```

Data output Messages Notifications

```
UPDATE 1

Query returned successfully in 725 msec.
```



Total rows: 4 of 4 Query complete 00:00:00.725

Query returned msec.

Browser

Properties SQL Statistics Dependencies Dependents testdb/postgres@PostgreSQL 14*

testdb/postgres@PostgreSQL 14

No limit

- FTS Parsers
- FTS Templates
- Foreign Tables
- Functions
- Materialized Views
- Operators
- Procedures
- Sequences
- Tables (2)
  - cust
  - customer
    - Columns (5)
      - custid
      - custname
      - age
      - city
      - salary
    - Constraints
    - Indexes
    - RLS Policies
    - Rules
    - Triggers
- Trigger Functions

Query Query History

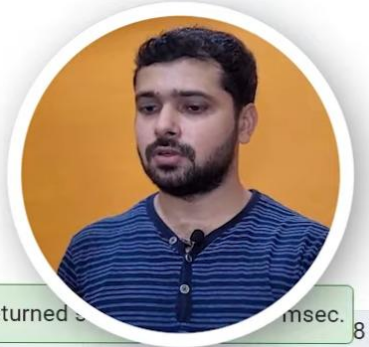
Scratch Pad

```
1  DELETE FROM customer
2  WHERE custid = 3;
3
```

Data output Messages Notifications

```
DELETE 1

Query returned successfully in 288 msec.
```



Total rows: 4 of 4 Query complete 00:00:00.288

Query returned … msec.

ALTER Table

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table

- **ALTER TABLE - ADD Column Syntax**

```
ALTER TABLE table_name
ADD COLUMN column_name ;
```

- **ALTER TABLE - DROP COLUMN Syntax**

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

- **ALTER TABLE - ALTER/MODIFY COLUMN Syntax**

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

Drop & Truncate Table

The DROP TABLE command deletes a table in the database

- **Syntax**

```
DROP TABLE table_name;
```

The TRUNCATE TABLE command deletes the data inside a table, but not the table itself

- **Syntax**

```
TRUNCATE TABLE table_name;
```

SELECT Statement

The SELECT statement is used to select data from a database.

- **Syntax**

```
SELECT column_name FROM table_name;
```

To select all the fields available in the table

- **Syntax**

```
SELECT * FROM table_name;
```

To select distinct/unique fields available in the table

- **Syntax**

```
SELECT DISTINCT Column_name FROM table_name;
```

WHERE Clause

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition

- **Syntax**

```
SELECT column_name FROM table_name
WHERE conditions;
```

- **Example**

```
SELECT name FROM classroom
WHERE grade='A';
```

pgAdmin File Object Tools Help

Browser

Properties SQL Statistics Dependencies Dependents testdb/postgres@PostgreSQL 14*

- Databases (3)
  - postgres
  - testdb
    - Casts
    - Catalogs
    - Event Triggers
    - Extensions
    - Foreign Data Wrappers
    - Languages
    - Publications
    - Schemas (1)
      - public
        - Aggregates
        - Collations
        - Domains
        - FTS Configurations
        - FTS Dictionaries
        - FTS Parsers
        - FTS Templates
        - Foreign Tables
        - Functions
        - Materialized Views
        - Operators

testdb/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SELECT name FROM classroom
2  WHERE grade= 'A'
```

Scratch Pad

Data output Messages Notifications

	rollno [PK] bigint	name character varying (50)	house character (12)	grade character (1)
1	1	Sam	Akash	B
2	2	Ram	Agni	A
3	3	Shyam	Jal	B
4	4	Sundar	Agni	A
5	5	Ram	Yayu	B

Operators In SQL

The SQL reserved words and characters are called operators, which are used with a WHERE clause in a SQL query

Most used operators:

1. **Arithmetic operators :** arithmetic operations on numeric values
   Example: Addition (+), Subtraction (-), Multiplication (*), Division (/), Modulus (%)
2. **Comparison operators:** compare two different data of SQL table
   - Example: Equal (=), Not Equal (!=), Greater Than (>), Greater Than Equals t
3. **Logical operators:** perform the Boolean operations
   - Example: ALL, IN, BETWEEN, LIKE, AND, OR, NOT, ANY
4. **Bitwise operators:** perform the bit operations on the In
   - Example: Bitwise AND (&), Bitwise OR(|)

pgAdmin File Object Tools Help

Browser

Properties SQL Statistics Dependencies Dependents testdb/postgres@PostgreSQL 14*

- Databases (3)
  - postgres
  - testdb
    - Casts
    - Catalogs
    - Event Triggers
    - Extensions
    - Foreign Data Wrappers
    - Languages
    - Publications
    - Schemas (1)
      - public
        - Aggregates
        - Collations
        - Domains
        - FTS Configurations
        - FTS Dictionaries
        - FTS Parsers
        - FTS Templates
        - Foreign Tables
        - Functions
        - Materialized Views
        - Operators

testdb/postgres@PostgreSQL 14

No limit

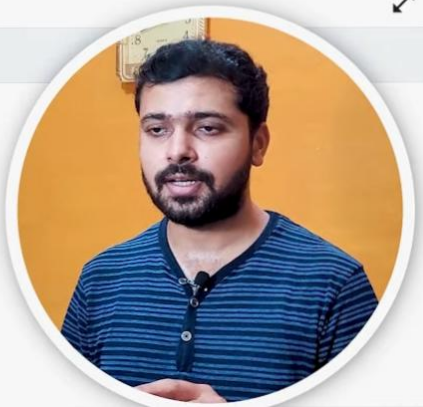
Query Query History

Scratch Pad

```
1  SELECT * FROM classroom
2  WHERE grade= 'A' AND rollno > 3
3
```

Data output Messages Notifications

	rollno [PK] bigint	name character varying (50)	house character (12)	grade character (1)
1	4	Sundar	Agni	A



LIMIT Clause

The LIMIT clause is used to set an upper limit on the number of tuples returned by SQL.

Example: below code will return 5 rows of data

```
SELECT column_name FROM table_name
LIMIT 5;
```

ORDER BY Clause

The ORDER BY is used to sort the result-set in ascending (ASC) or desc order (DESC).

Example: below code will sort the output data by column name in as

```
SELECT column_name FROM table_name
ORDER BY column_name e ASC;
```

pgAdmin File Object Tools Help

Browser

Properties SQL Statistics Dependencies Dependents rnm_db/postgres@PostgreSQL 14*

rnm_db/postgres@PostgreSQL 14

No limit

Query Query History

Scratch Pad

```
1  SELECT * FROM customer
2  ORDER BY first_name ASC
```

Data output Messages Notifications

	customer_id [PK] bigint	first_name character varying (50)	last_name character varying (50)	email character varying (100)
10	9	Margaret	Moore	margaret.moore@sakilacust
11	7	Maria	Miller	maria.miller@sakilacustom
12	1	Mary	Smith	mary.smith@sakilacustome
13	12	Nancy	Thomas	nancy.thomas@sakilacustom
14	2	Patricia	Johnson	patricia.johnson@sakilacustom
15	8	Susan	Wilson	susan.wilson@sakilacustomer.org

Total rows: 15 of 15 Query complete 00:00:00.301 Ln 2, Col 1

- Schemas (1)
  - public
    - Aggregates
    - Collations
    - Domains
    - FTS Configurations
    - FTS Dictionaries
    - FTS Parsers
    - FTS Templates
    - Foreign Tables
    - Functions
    - Materialized Views
    - Operators
    - Procedures
    - Sequences
    - Tables (2)
      - customer
      - payment
    - Trigger Functions
    - Types
    - Views
  - Subscriptions
- Login/Group Roles

Browser

Properties SQL Statistics Dependencies Dependents rnm_db/postgres@PostgreSQL 14*

- Schemas (1)
  - public
    - Aggregates
    - Collations
    - Domains
    - FTS Configurations
    - FTS Dictionaries
    - FTS Parsers
    - FTS Templates
    - Foreign Tables
    - Functions
    - Materialized Views
    - Operators
    - Procedures
    - Sequences
    - Tables (2)
      - customer
      - payment
    - Trigger Functions
    - Types
    - Views
  - Subscriptions
- Login/Group Roles

rnm_db/postgres@PostgreSQL 14

No limit

Query Query History

Scratch Pad

```
1  SELECT * FROM customer
2  LIMIT 5
```

Data output Messages Notifications

	customer_id [PK] bigint	first_name character varying (50)	last_name character varying (50)	email character varying (100)
10	9	Margaret	Moore	margaret.moore@sakilacust
11	7	Maria	Miller	maria.miller@sakilacustom
12	1	Mary	Smith	mary.smith@sakilacustome
13	12	Nancy	Thomas	nancy.thomas@sakilacustom
14	2	Patricia	Johnson	patricia.johnson@sakilacustom
15	8	Susan	Wilson	susan.wilson@sakilacustomer.org

Total rows: 15 of 15 Query complete 00:00:00.301 Ln 2, Col 8

Functions In SQL

Functions in SQL are the database objects that contains a set of SQL statements to perform a specific task. A function accepts input parameters, perform actions, and then return the result.

Types of Function:

1. System Defined Function : these are built-in functions
   - Example: rand(), round(), upper(), lower(), count(), sum(), max(), etc
2. User-Defined Function : Once you define a functio can call it in the same way as the built-in functions

Most Used String Functions

String functions are used to perform an operation on input string and return an output string

- **UPPER()** converts the value of a field to uppercase
- **LOWER()** converts the value of a field to lowercase
- **LENGTH()** returns the length of the value in a text field
- **SUBSTRING()** extracts a substring from a string
- **NOW()** returns the current system date and time
- **FORMAT()** used to set the format of a field
- **CONCAT()** adds two or more strings together
- **REPLACE()** Replaces all occurrences of a substring within a string, with a new subst
- **TRIM()** removes leading and trailing spaces (or other specified characters) from a st

Most Used Aggregate Functions

Aggregate function performs a calculation on multiple values and returns a single value.

And Aggregate functiona are often used with GROUP BY & SELECT statement

- **COUNT()** returns number of values
- **SUM()** returns sum of all values
- **AVG()** returns average value
- **MAX()** returns maximum value
- **MIN()** returns minimum value
- **ROUND()** Rounds a number to a specified number of decimal places

GROUP BY Statement

The GROUP BY statement group rows that have the same values into summary rows.

It is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns

- **Syntax**

```
SELECT column_name(s)
FROM table_name
GROUP BY column_name(s);
```

- **Example**

```
SELECT mode, SUM(amount) AS total
FROM payment
GROUP BY mode
```

Browser

Properties | SQL | Statistics | Dependencies | Dependents | rnm_db/postgres@PostgreSQL 14*

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (2)
      - postgres
      - rnm_db
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates
            - Foreign Tables
            - Functions

rnm_db/postgres@PostgreSQL 14

No limit

Query | Query History

```
1 SELECT mode, SUM(amount) AS Total
2 FROM payment
3 GROUP BY mode
```

Data output | Messages | Notifications

	mode character varying (50)	total numeric
1	Mobile Payment	120
2	Netbanking	90
3	Debit Crad	90
4	Credit Card	200
5	Cash	220

Total rows: 5 of 5 | Query complete 00:00:00.317 | Ln 3, Col 15

Browser

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (2)
      - postgres
      - rnm_db
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates
            - Foreign Tables
            - Functions

Properties | SQL | Statistics | Dependencies | Dependents | rnm_db/postgres@PostgreSQL 14*

rnm_db/postgres@PostgreSQL 14

No limit

Query | Query History

```
1  SELECT mode, SUM(amount) AS total
2  FROM payment
3  GROUP BY mode
4  ORDER BY total ASC
```

Data output | Messages | Notifications

	mode character varying (50)	total numeric
1	Netbanking	90
2	Debit Crad	90
3	Mobile Payment	120
4	Credit Card	200
5	Cash	220

Total rows: 5 of 5 | Query complete 00:00:00.458 | Ln 1, Col 30

HAVING Clause

The **HAVING** clause is used to apply a filter on the result of **GROUP BY** based on the specified condition.

The **WHERE** clause places conditions on the selected columns, whereas the **HAVING** clause places conditions on groups created by the **GROUP BY** clause

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition(s)
GROUP BY column_name(s)
HAVING condition(s)
```

- **Example**

```
SELECT mode, COUNT(amount) AS total
FROM payment
GROUP BY mode
HAVING COUNT(amount) >= 3
ORDER BY total DESC
```

Browser

Properties SQL Statistics Dependencies Dependents rnm_db/postgres@PostgreSQL 14*

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (2)
      - postgres
      - rnm_db
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates
            - Foreign Tables
            - Functions

rnm_db/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SELECT mode, COUNT(amount) AS total
2  FROM payment
3  GROUP BY mode
4  HAVING COUNT(amount) >= 2 AND COUNT(amount) < 4
5  ORDER BY total DESC
6
```

Data output Messages Notifications

	mode character varying (50)	total bigint
1	Mobile Payment	3
2	Netbanking	2
3	Debit Crad	2

Total rows: 3 of 3 Query complete 00:00:00.241 Ln 5, Col 21

Quick Assignment: 01

Order of execution in SQL:

SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, LIMIT

?

Answer in video's comment

(no cheating)

TIMESTAMP

The **TIMESTAMP** data type is used for values that contain both date and time parts

- **TIME** contains only time, format HH:MI:SS
- **DATE** contains on date, format YYYY-MM-DD
- **YEAR** contains on year, format YYYY or YY
- **TIMESTAMP** contains date and time, format YYYY-MM HH:MI:SS
- **TIMESTAMPTZ** contains date, time and time zone

TIMESTAMP functions/operators

Below are the TIMESTAMP functions and operators in SQL:

- SHOW TIMEZONE
- SELECT NOW()
- SELECT TIMEOFDAY()
- SELECT CURRENT_TIME
- SELECT CURRENT_DATE

pgAdmin File Object Tools Help

Browser

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (3)
      - postgres
      - rnm_db
      - rnmcopy
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates

Properties SQL Statistics Dependencies Dependents rnmcopy/postgres@PostgreSQL 14*

rnmcopy/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SHOW TIMEZONE
2  SELECT NOW()
3  SELECT TIMEOFDAY()
4  SELECT CURRENT_TIME
5  SELECT CURRENT_DATE
6
```

Data output Messages Notifications

	current_time time with time zone
1	18:13:49.605630+0...

EXTRACT Function

The **EXTRACT()** function extracts a part from a given date value.

Syntax: SELECT **EXTRACT**(MONTH FROM date_field) FROM Table

- **YEAR**
- **QUARTER**
- **MONTH**
- **WEEK**
- **DAY**
- **HOUR**
- **MINUTE**
- **DOW** – day of week
- **DOY** – day of year

Browser

Properties SQL Statistics Dependencies Dependents rnmcopy/postgres@PostgreSQL 14*

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (3)
      - postgres
      - rnm_db
      - rnmcopy
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates

rnmcopy/postgres@PostgreSQL 14

No limit

Query Query History

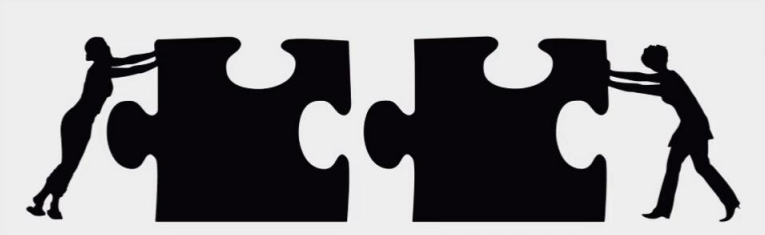
```
1  SELECT EXTRACT(MONTH FROM payment_date) AS payment_month, payment_date
2  FROM payment
3
```

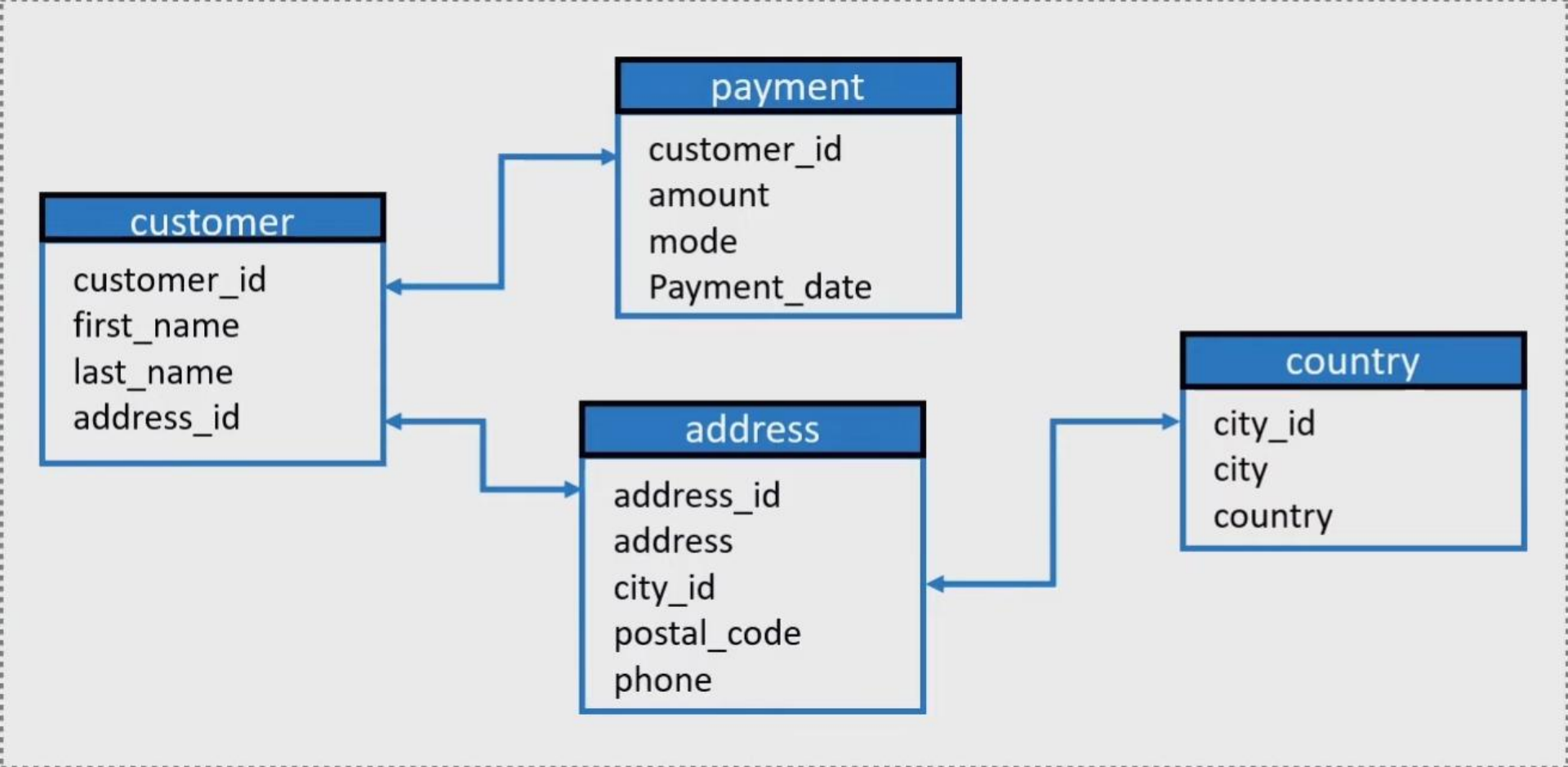
Data output Messages Notifications

	payment_month numeric	payment_date date
1	9	2020-09-24
2	4	2020-04-27
3	1	2021-01-26
4	2	2021-02-28
5	3	2021-03-01

SQL JOIN

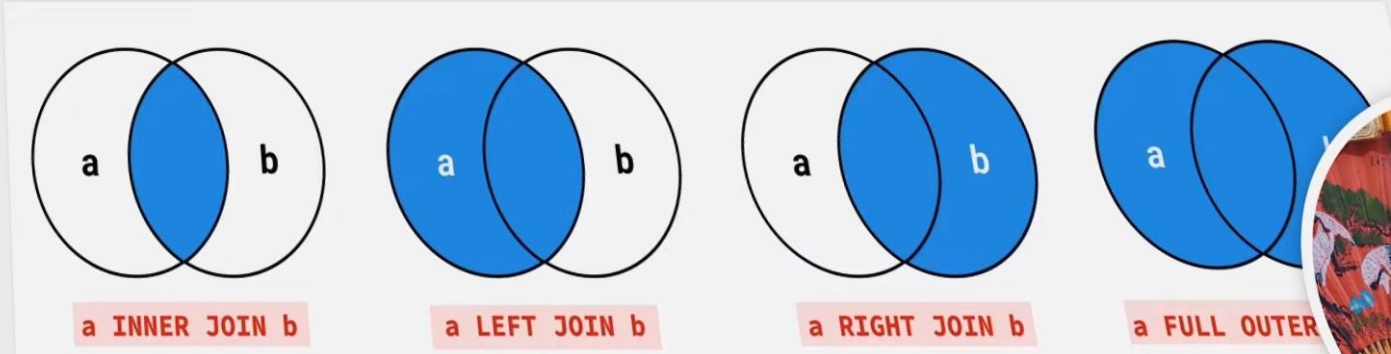
- **JOIN** means to combine something.
- A **JOIN** clause is used to combine data from two or more tables, based on a related column between them
- Let's understand the joins through an example:




payment
customer_id
amount
mode
Payment_date
customer
customer_id
first_name
last_name
address_id
country
city_id
city
country
address
address_id
address
city_id
postal_code
phone

TYPES OF JOINS

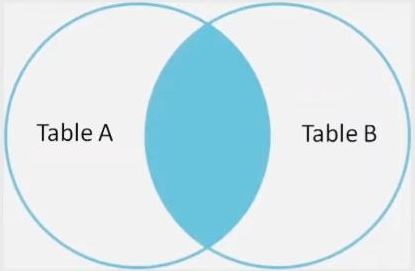
- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN



INNER JOIN

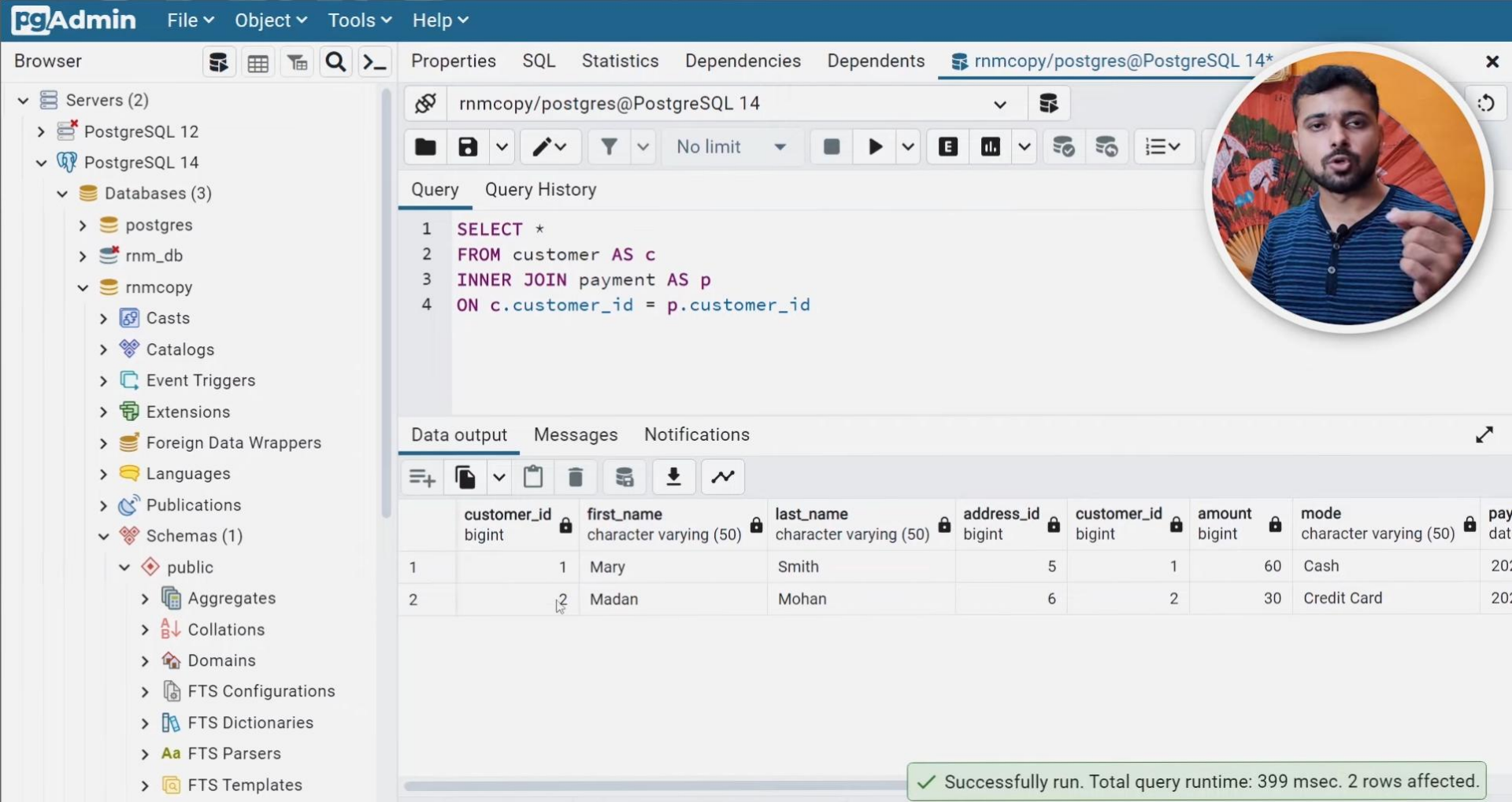
- **Syntax**

```
SELECT column_name(s)
FROM TableA
INNER JOIN TableB
ON TableA.col_name = TableB.col_name
```

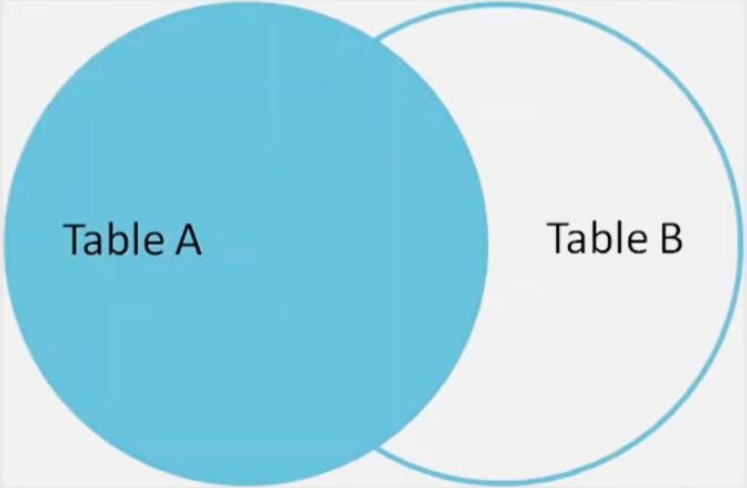


- **Example**

```
SELECT *
FROM customer AS c
INNER JOIN payment AS p
ON c.customer_id = p.customer_id
```


pgAdmin
File
Object
Tools
Help
Browser
Properties
SQL
Statistics
Dependencies
Dependents
rnmcopy/postgres@PostgreSQL 14*
Servers (2)
PostgreSQL 12
PostgreSQL 14
Databases (3)
postgres
rnm_db
rnmcopy
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
rnmcopy/postgres@PostgreSQL 14
No limit
Query
Query History
1 SELECT *
2 FROM customer AS c
3 INNER JOIN payment AS p
4 ON c.customer_id = p.customer_id
Data output
Messages
Notifications
customer_id bigint
first_name character varying (50)
last_name character varying (50)
address_id bigint
customer_id bigint
amount bigint
mode character varying (50)
1 1 Mary Smith 5 1 60 Cash
2 2 Madan Mohan 6 2 30 Credit Card
Successfully run. Total query runtime: 399 msec. 2 rows affected.

- Returns all records from the left table, and the matched records from the right table



left_join(x, y)

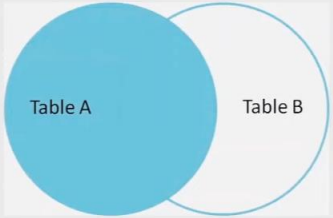
1	x1
2	x2
3	x3

1	y1
2	y2
4	

LEFT JOIN

- **Syntax**

```
SELECT column_name(s)
FROM TableA
LEFT JOIN TableB
ON TableA.col_name = TableB.col_name
```



- **Example**

```
SELECT *
FROM customer AS c
LEFT JOIN payment AS p
ON c.customer_id = p.customer_id
```

Browser

Properties | SQL | Statistics | Dependencies | Dependents | rnmcopy/postgres@PostgreSQL 14*

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (3)
      - postgres
      - rnm_db
      - rnmcopy
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates

rnmcopy/postgres@PostgreSQL 14

No limit

Query | Query History

```
1  SELECT *
2  FROM customer AS c
3  LEFT JOIN payment AS p
4  ON c.customer_id = p.customer_id
5
```

Data output | Messages | Notifications

	customer_id [PK] bigint	first_name character varying (50)	last_name character varying (50)	address_id bigint
1	1	Mary	Smith	5
2	3	Linda	Williams	7
3	4	Barbara	Jones	8
4	2	Madan	Mohan	6

Total rows: 4 of 4 | Query complete 00:00:00.222

Ln 2, Col 19

Browser

Properties SQL Statistics Dependencies Dependents rnmcopy/postgres@PostgreSQL 14*

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (3)
      - postgres
      - rnm_db
      - rnmcopy
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates

rnmcopy/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SELECT *
2  FROM customer AS c
3  LEFT JOIN payment AS p
4  ON c.customer_id = p.customer_id
5
```

Data output Messages Notifications

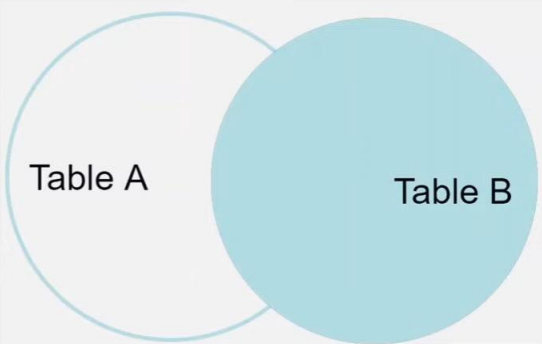
	customer_id bigint	first_name character varying (50)	last_name character varying (50)	address_id bigint	customer_id bigint	amount bigint	mode character varying (50)	pay dat
1	1	Mary	Smith	5	1	60	Cash	202
2	2	Madan	Mohan	6	2	30	Credit Card	202
3	4	Barbara	Jones	8	[null]	[null]	[null]	[nu
4	3	Linda	Williams	7	[null]	[null]	[null]	[nu

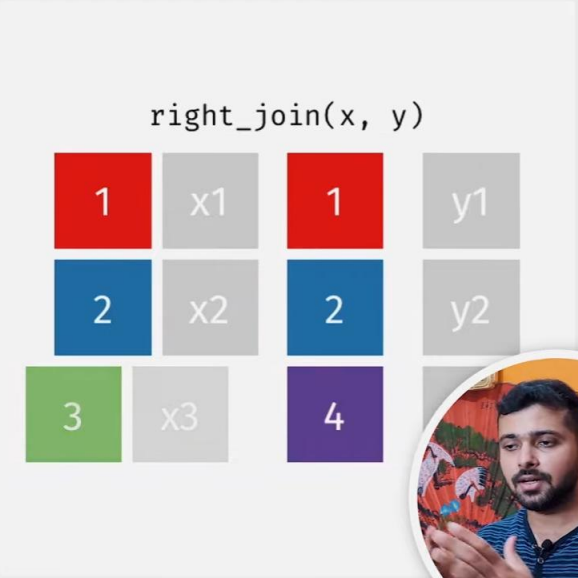
Total rows: 4 of 4 Query complete 00:00:00.244

Successfully run. Total query runtime: 244 msec. 4 rows affected.

RIGHT JOIN

- Returns all records from the right table, and the matched records from the left table

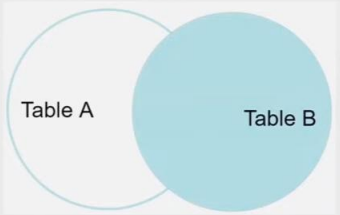




RIGHT JOIN

- **Syntax**

```
SELECT column_name(s)
FROM TableA
RIGHT JOIN TableB
ON TableA.col_name = TableB.col_name
```



- **Example**

```
SELECT *
FROM customer AS c
RIGHT JOIN payment AS p
ON c.customer_id = p.customer_id
```

Browser

Properties SQL Statistics Dependencies Dependents rnmcopy/postgres@PostgreSQL 14*

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (3)
      - postgres
      - rnm_db
      - rnmcopy
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates

rnmcopy/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SELECT *
2  FROM customer AS c
3  RIGHT JOIN payment AS p
4  ON c.customer_id = p.customer_id
5
```

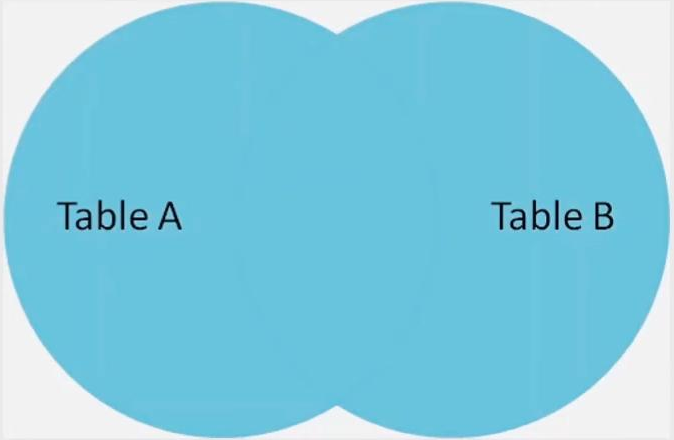
Data output Messages Notifications

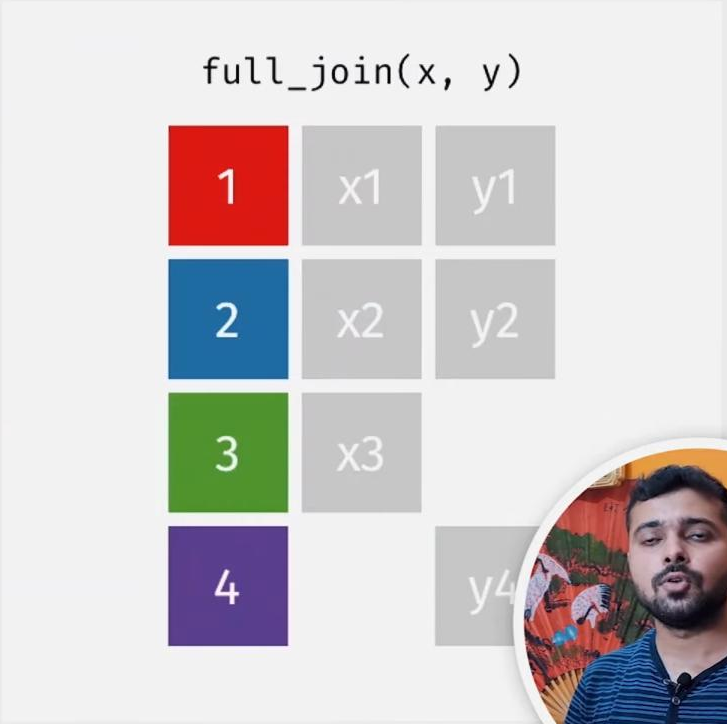
	customer_id bigint	first_name character varying (50)	last_name character varying (50)	address_id bigint	customer_id bigint	amount bigint	mode character varying (50)	pay dat
1	1	Mary	Smith	5	1	60	Cash	202
2	2	Madan	Mohan	6	2	30	Credit Card	202
3	[null]	[null]	[null]	[null]	8	110	Cash	202
4	[null]	[null]	[null]	[null]	10	70	mobile Payment	202
5	[null]	[null]	[null]	[null]	11	80	Cash	202

Total rows: 5 of 5 Query complete 00:00:00.766 Ln 5, Col 1

FULL JOIN

- Returns all records when there is a match in either left or right table

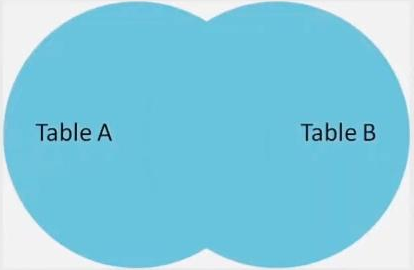




FULL JOIN

- **Syntax**

```
SELECT column_name(s)
FROM TableA
FULL OUTER JOIN TableB
ON TableA.col_name = TableB.col_name
```



- **Example**

```
SELECT *
FROM customer AS c
FULL OUTER JOIN payment AS p
ON c.customer_id = p.customer_id
```

pgAdmin File Object Tools Help

Browser

Properties SQL Statistics Dependencies Dependents rnmcopy/postgres@PostgreSQL 14*

rnmcopy/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SELECT *
2  FROM customer AS c
3  FULL OUTER JOIN payment AS p
4  ON c.customer_id = p.customer_id
5
```

Data output Messages Notifications

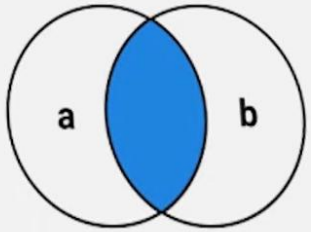
	customer_id bigint	first_name character varying (50)	last_name character varying (50)	address_id bigint	customer_id bigint	amount bigint	mode character varying (50)	pay dat
1	1	Mary	Smith	5	1	60	Cash	20
2	2	Madan	Mohan	6	2	30	Credit Card	20
3	[null]	[null]	[null]	[null]	8	110	Cash	20
4	[null]	[null]	[null]	[null]	10	70	mobile Payment	20
5	[null]	[null]	[null]	[null]	11	80	Cash	20
6	4	Barbara	Jones	8	[null]	[null]	[null]	[nu
7	3	Linda	Williams	7	[null]	[null]	[null]	[nu

Total rows: 7 of 7 Query complete 00:00:00.716 Ln 5, Col 1

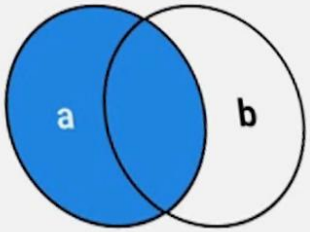
- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (3)
      - postgres
      - rnm_db
      - rnmcopy
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates

Which JOIN To Use

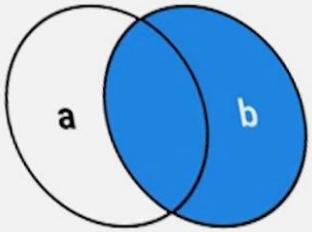
- **INNER JOIN:** Returns records that have matching values in both tables
- **LEFT JOIN:** Returns all records from the left table, and the matched records from the right table
- **RIGHT JOIN:** Returns all records from the right table, and the matched records from the left table
- **FULL JOIN:** Returns all records when there is a match in either left or right table



a INNER JOIN b



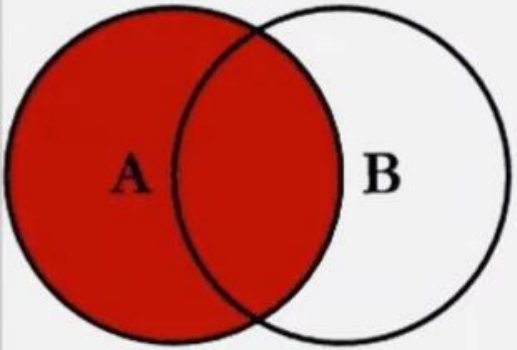
a LEFT JOIN b



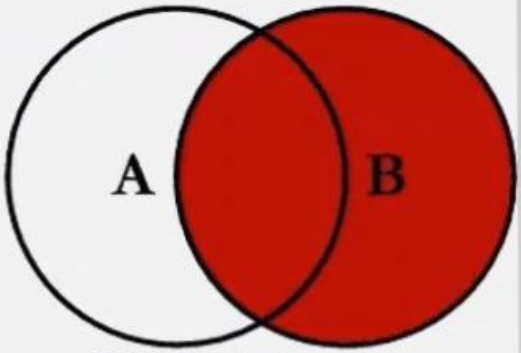
a RIGHT JOIN b

a FULL OUTER JOI

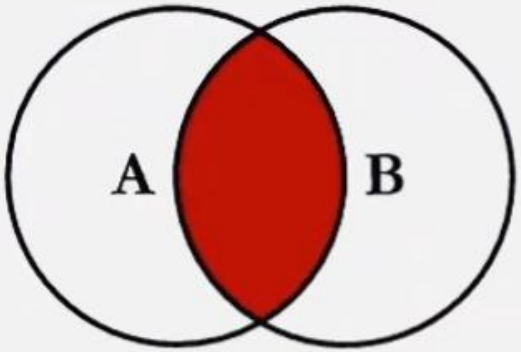
SQL JOINS



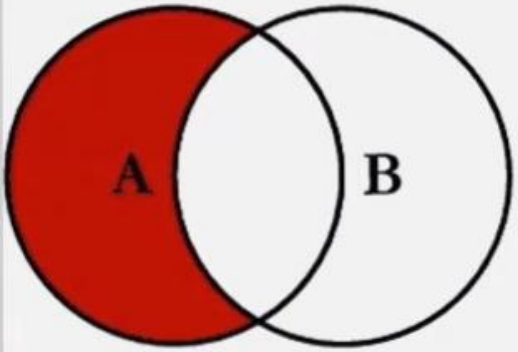

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



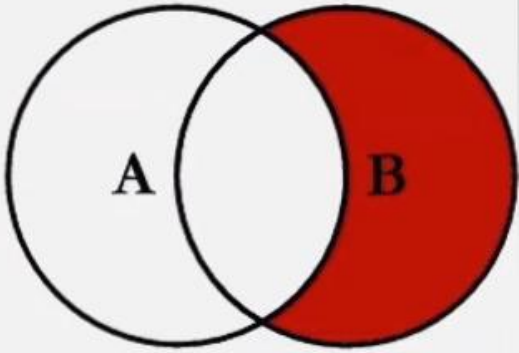

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



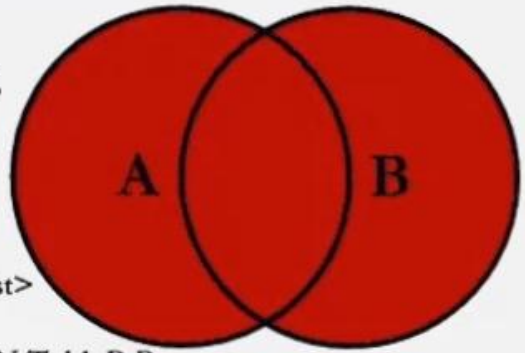

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



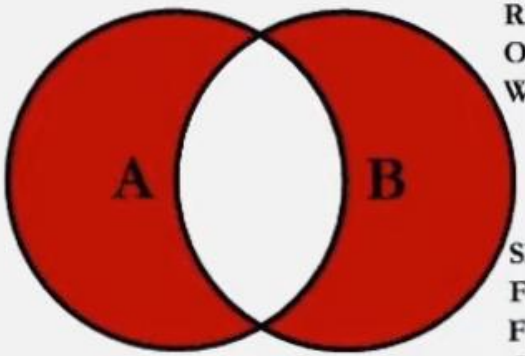

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

SELF JOIN

- A **self join** is a regular join in which a table is joined to itself
- **SELF Joins** are powerful for comparing values in a column of rows with the same table
- Syntax

```
SELECT column_name(s)
FROM Table AS T1
JOIN Table AS T2
ON T1.col_name = T2.col_name
```

empid [PK] bigint	empname character varying (50)	manager_id bigint
1	Agni	3
2	Akash	4
3	Dharti	2
4	Vayu	3

empid [PK] bigint	empname character varying (50)	manager_id bigint
1	Agni	3
2	Akash	4
3	Dharti	2
4	Vayu	3

mngr character varying (50)
Dharti
Vayu
Akash
Dharti

Browser

Properties SQL Statistics Dependencies Dependents rnmcopy/postgres@PostgreSQL 14*

rnmcopy/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SELECT *
2  FROM emp AS T1
3  JOIN emp AS T2
4  ON T2.empid = T1.manager_id
5
6
```

Data output Messages Notifications

	empid [PK] bigint	empname character varying (50)	manager_id bigint	empid bigint	empname character varying (50)	manager_id bigint
1	1	Agni	3	3	Dharti	2
2	2	Akash	4	4	Vayu	3
3	3	Dharti	2	2	Akash	4
4	4	Vayu	3	3	Dharti	2

Total rows: 4 of 4 Query complete 00:00:00.193 Ln 4, Col 28

- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Publications
- Schemas (1)
  - public
    - Aggregates
    - Collations
    - Domains
    - FTS Configurations
    - FTS Dictionaries
    - FTS Parsers
    - FTS Templates
    - Foreign Tables
    - Functions
    - Materialized Views
    - Operators
    - Procedures
    - Sequences
    - Tables (5)
      - address
      - country
      - customer

Browser

Properties SQL Statistics Dependencies Dependents rnmcopy/postgres@PostgreSQL 14*

rnmcopy/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SELECT T1.empname AS employee_name, T2.empname AS manager_name
2  FROM emp AS T1
3  JOIN emp AS T2
4  ON T2.empid = T1.manager_id
5
6
```

Data output Messages Notifications

	employee_name character varying (50)	manager_name character varying (50)
1	Agni	Dharti
2	Akash	Vayu
3	Dharti	Akash
4	Vayu	Dharti

Total rows: 4 of 4 Query complete 00:00:00.219 Ln 1, Col 63

- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Publications
- Schemas (1)
  - public
    - Aggregates
    - Collations
    - Domains
    - FTS Configurations
    - FTS Dictionaries
    - FTS Parsers
    - FTS Templates
    - Foreign Tables
    - Functions
    - Materialized Views
    - Operators
    - Procedures
    - Sequences
    - Tables (5)
      - address
      - country
      - customer

UNION

The SQL **UNION** clause/operator is used to combine/concatenate the results of two or more SELECT statements without returning any duplicate rows and keeps **unique records**

To use this UNION clause, each SELECT statement must have

- The same number of columns selected and expressions
- The same data type and
- Have them in the same order
- **Syntax**

```
SELECT column_name(s) FROM TableA
UNION
SELECT column_name(s) FROM TableB
```

- **Example**

```
SELECT cust_name, cust_amount from custA
UNION
SELECT cust_name, cust_amount from custB
```

UNION ALL

In **UNION ALL** everything is same as **UNION**, it combines/concatenate two or more table but keeps all records, **including duplicates**

- **Syntax**

```
SELECT column_name(s) FROM TableA
UNION ALL
SELECT column_name(s) FROM TableB
```

- **Example**

```
SELECT cust_name, cust_amount from custA
UNION ALL
SELECT cust_name, cust_amount from custB
```

cust_name character (30)	cust_amount bigint
Madan Mohan	2100

cust_name character (30)
Gopal Bhat

Browser

Properties SQL Statistics Dependencies Dependents rnmcopy/postgres@PostgreSQL 14*

- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Publications
- Schemas (1)
  - public
    - Aggregates
    - Collations
    - Domains
    - FTS Configurations
    - FTS Dictionaries
    - FTS Parsers
    - FTS Templates
    - Foreign Tables
    - Functions
    - Materialized Views
    - Operators
    - Procedures
    - Sequences
    - Tables (5)
      - address
      - country
      - customer

rnmcopy/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SELECT cust_name, cust_amount
2  FROM custA
3  UNION
4  SELECT cust_name, cust_amount
5  FROM custB
```

Data output Messages Notifications

	cust_name character (30)	cust_amount bigint
1	Gopal Bhat	1500
2	Madan Mohan	2100
3	Govind Dev	5000
4	Gopi Nath	1200

Total rows: 4 of 4 Query complete 00:00:00.201 Ln 3, Col 6

Browser

Properties SQL Statistics Dependencies Dependents rnmcopy/postgres@PostgreSQL 14*

- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Publications
- Schemas (1)
  - public
    - Aggregates
    - Collations
    - Domains
    - FTS Configurations
    - FTS Dictionaries
    - FTS Parsers
    - FTS Templates
    - Foreign Tables
    - Functions
    - Materialized Views
    - Operators
    - Procedures
    - Sequences
    - Tables (5)
      - address
      - country
      - customer

rnmcopy/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SELECT cust_name, cust_amount
2  FROM custA
3  UNION ALL
4  SELECT cust_name, cust_amount
5  FROM custB
```

Data output Messages Notifications

	cust_name character (30)	cust_amount bigint
1	Madan Mohan	2100
2	Gopi Nath	1200
3	Govind Dev	5000
4	Gopal Bhat	1500
5	Madan Mohan	2100

Total rows: 5 of 5 Query complete 00:00:00.124 Ln 3, Col 10

SUB QUERY

A **Subquery** or Inner query or a Nested query allows us to create complex query on the output of another query

- Sub query syntax involves two SELECT statements

- **Syntax**

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator
   ( SELECT column_name  FROM table_name WHE
```

	customer_id [PK] bigint	amount bigint	mode character varying (50)	payment_date date
1	1	60	Cash	2020-09-24
2	2	30	Credit Card	
3	8	110	Cash	
4	10	70	mobile Payment	
5	11	80	Cash	

Browser

- rnmcopy
  - Casts
  - Catalogs
  - Event Triggers
  - Extensions
  - Foreign Data Wrappers
  - Languages
  - Publications
  - Schemas (1)
    - public
      - Aggregates
      - Collations
      - Domains
      - FTS Configurations
      - FTS Dictionaries
      - FTS Parsers
      - FTS Templates
      - Foreign Tables
      - Functions
      - Materialized Views
      - Operators
      - Procedures
      - Sequences

Properties | SQL | Statistics | Dependencies | Dependents | rnmcopy/postgres@PostgreSQL 14*

rnmcopy/postgres@PostgreSQL 14

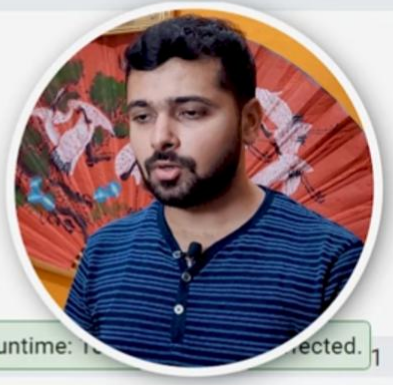
No limit

Query | Query History

```
1  select avg(amount) from payment
2  -- find the average value
3
4
```

Data output | Messages | Notifications

	avg numeric
1	164.0000000



Total rows: 1 of 1 | Query complete 00:00:00.186

Successfully run. Total query runtime: 1 ... rected.

Browser

Properties | SQL | Statistics | Dependencies | Dependents | rnmcopy/postgres@PostgreSQL 14*

rnmcopy/postgres@PostgreSQL 14

No limit

Query | Query History

```
1  select avg(amount) from payment
2  -- find the average value
3  -- filter the customer data > avg value
4
5  SELECT *
6  FROM payment
7  WHERE amount > 164
8
```

Data output | Messages | Notifications

	customer_id [PK] bigint	amount bigint	mode character varying (50)	payment_date date
1	2	500	Credit Card	2020-04-27

Total rows: 1 of 1 | Query complete 00:00:01.086 | Ln 5, Col 1

- rnmcopy
  - Casts
  - Catalogs
  - Event Triggers
  - Extensions
  - Foreign Data Wrappers
  - Languages
  - Publications
  - Schemas (1)
    - public
      - Aggregates
      - Collations
      - Domains
      - FTS Configurations
      - FTS Dictionaries
      - FTS Parsers
      - FTS Templates
      - Foreign Tables
      - Functions
      - Materialized Views
      - Operators
      - Procedures
      - Sequences

Browser

Properties | SQL | Statistics | Dependencies | Dependents | rnmcopy/postgres@PostgreSQL 14*

rnmcopy/postgres@PostgreSQL 14

No limit

Query | Query History

```
6  SELECT *
7  FROM payment
8  WHERE amount > 164
9
10 -- sub query
11 SELECT *
12 FROM payment
13 WHERE amount > (select avg(amount) from payment)
14
```

Data output | Messages | Notifications

	customer_id [PK] bigint	amount bigint	mode character varying (50)	payment_date date
1	2	500	Credit Card	2020-04-27

Total rows: 1 of 1 | Query complete 00:00:00.262 | Ln 11, Col 9

- rnmcopy
  - Casts
  - Catalogs
  - Event Triggers
  - Extensions
  - Foreign Data Wrappers
  - Languages
  - Publications
  - Schemas (1)
    - public
      - Aggregates
      - Collations
      - Domains
      - FTS Configurations
      - FTS Dictionaries
      - FTS Parsers
      - FTS Templates
      - Foreign Tables
      - Functions
      - Materialized Views
      - Operators
      - Procedures
      - Sequences

Browser

Properties | SQL | Statistics | Dependencies | Dependents | rnmcopy/postgres@PostgreSQL 14*

rnmcopy/postgres@PostgreSQL 14

No limit

Query | Query History

```
12  FROM payment
13  WHERE amount > (select avg(amount) from payment)
14
15
16  SELECT customer_id, amount, mode
17  from payment
18  where customer_id IN (select customer_id from customer)
19
20
```

Data output | Messages | Notifications

	customer_id [PK] bigint	amount bigint	mode character varying (50)
1	1	60	Cash
2	2	500	Credit Card

Total rows: 2 of 2 | Query complete 00:00:04.357 | Ln 17, Col 6

- rnmcopy
  - Casts
  - Catalogs
  - Event Triggers
  - Extensions
  - Foreign Data Wrappers
  - Languages
  - Publications
  - Schemas (1)
    - public
      - Aggregates
      - Collations
      - Domains
      - FTS Configurations
      - FTS Dictionaries
      - FTS Parsers
      - FTS Templates
      - Foreign Tables
      - Functions
      - Materialized Views
      - Operators
      - Procedures
      - Sequences

Browser

Properties SQL Statistics Dependencies Dependents rnmcopy/postgres@PostgreSQL 14*

rnmcopy/postgres@PostgreSQL 14

No limit

- rnmcopy
  - Casts
  - Catalogs
  - Event Triggers
  - Extensions
  - Foreign Data Wrappers
  - Languages
  - Publications
  - Schemas (1)
    - public
      - Aggregates
      - Collations
      - Domains
      - FTS Configurations
      - FTS Dictionaries
      - FTS Parsers
      - FTS Templates
      - Foreign Tables
      - Functions
      - Materialized Views
      - Operators
      - Procedures
      - Sequences

Query Query History

```
1  SELECT customer_id, amount
2  FROM payment
3  WHERE amount > 100
```

Data output Messages Notifications

	customer_id [PK] bigint	amount bigint
1	8	110
2	2	500

Total rows: 2 of 2 Query complete 00:02:40.328

Ln 1, Col 1

Browser

Properties SQL Statistics Dependencies Dependents rnmcopy/postgres@PostgreSQL 14*

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (3)
      - postgres
      - rnm_db
      - rnmcopy
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates

rnmcopy/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SELECT first_name, last_name
2  FROM customer c
3  WHERE EXISTS (  SELECT customer_id, amount
4                  FROM payment p
5                  WHERE p.customer_id = c.customer_id
6                  AND amount > 100)
```

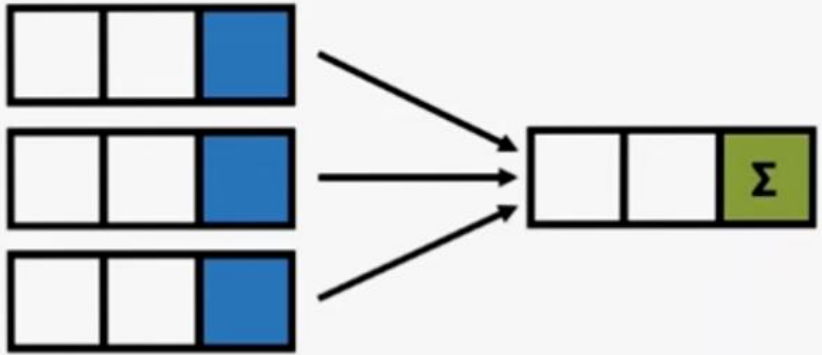
Data output Messages Notifications

	first_name character varying (50)	last_name character varying (50)
1	Madan	Mohan

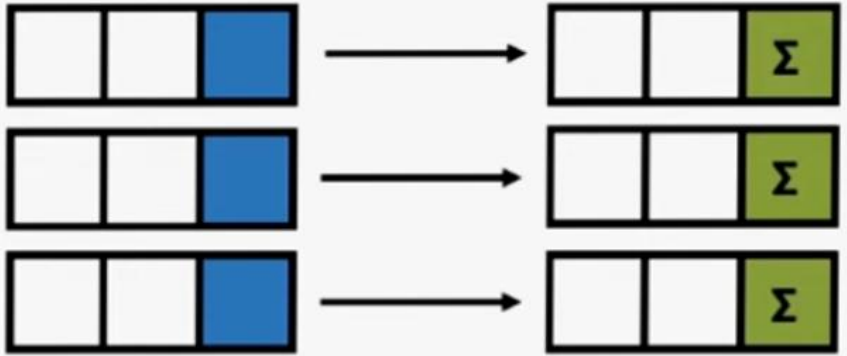
Total rows: 1 of 1 Query complete 00:00:00.247

Ln 1, Col 1

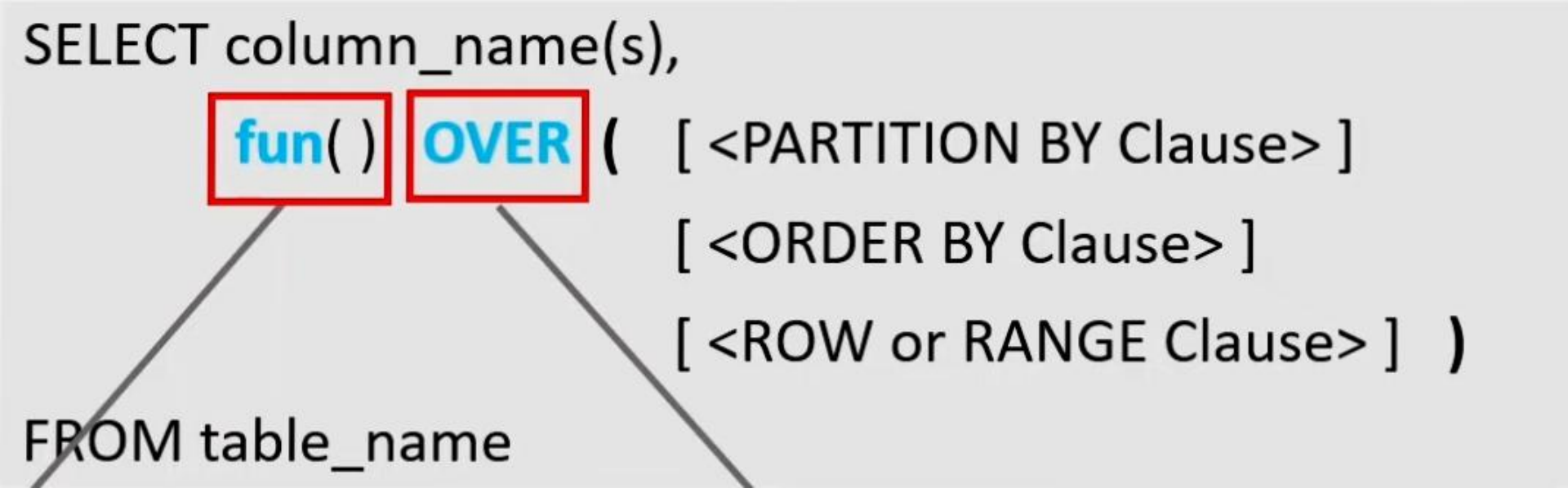
Aggregate Functions (SUM, AVG, etc.)

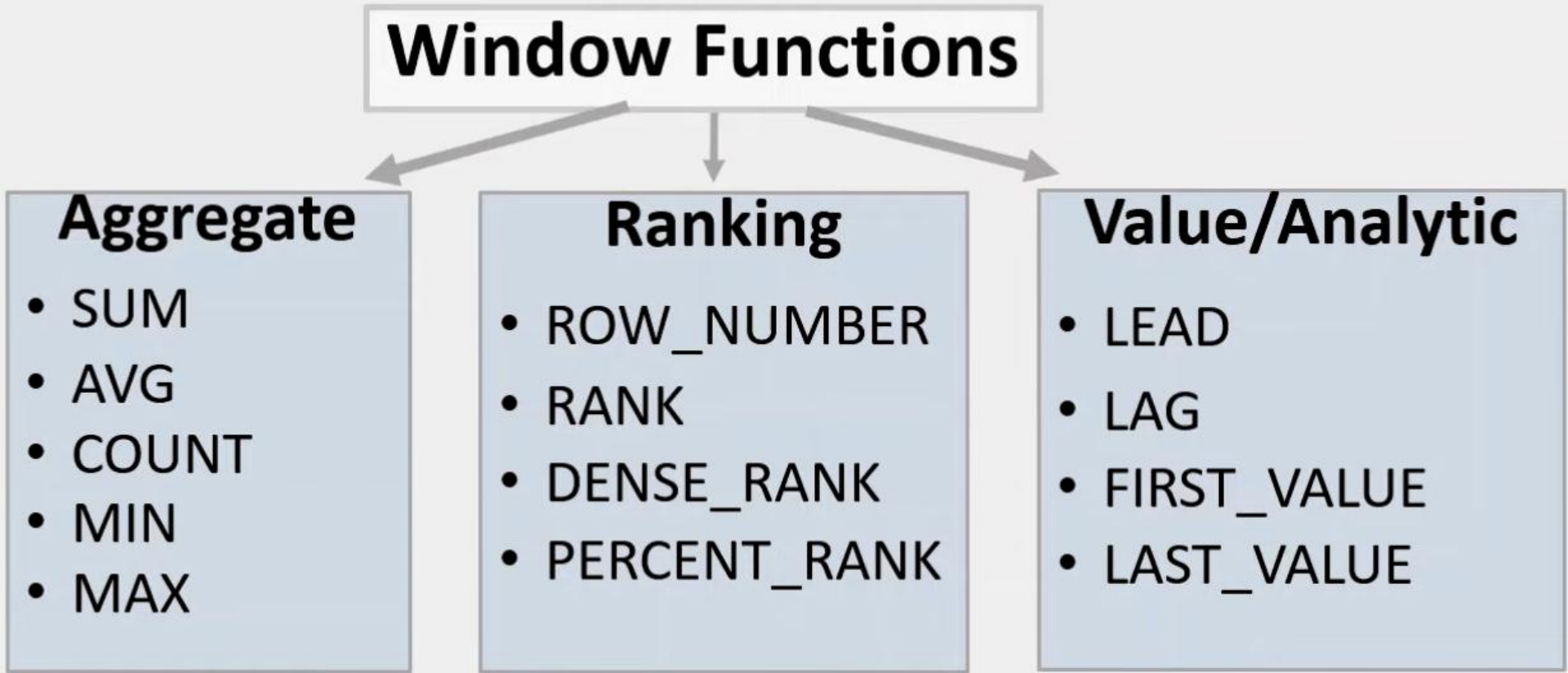


Window Functions



```
SELECT column_name(s),
       fun( ) OVER (  [ <PARTITION BY Clause> ]
                      [ <ORDER BY Clause> ]
                      [ <ROW or RANGE Clause> ]  )
FROM table_name
```


SELECT column_name(s),
fun( )
OVER
(
[<PARTITION BY Clause>]
[<ORDER BY Clause>]
[<ROW or RANGE Clause>]
)
FROM table_name


Window Functions
Aggregate
• SUM
• AVG
• COUNT
• MIN
• MAX
Ranking
• ROW_NUMBER
• RANK
• DENSE_RANK
• PERCENT_RANK
Value/Analytic
• LEAD
• LAG
• FIRST_VALUE
• LAST_VALUE

new_id	new_cat	Total	Average	Count	Min	Max
100	Agni	300	150	2	100	200
200	Agni	300	150	2	100	200
500	Dharti	1200	600	2	500	700
700	Dharti	1200	600	2	500	700
200	Vayu	1000	333.33333	3	200	500
300	Vayu	1000	333.33333	3	200	500
500	Vayu	1000	333.33333	3	200	500

AGGREGATE FUNCTION Example

new_id	new_cat	Total	Average	Count	Min	Max
100	Agni	2500	357.14286	7	100	700
200	Agni	2500	357.14286	7	100	700
200	Vayu	2500	357.14286	7	100	700
300	Vayu	2500	357.14286	7	100	700
500	Vayu	2500	357.14286	7	100	700
500	Dharti	2500	357.14286	7	100	700
700	Dharti	2500	357.14286	7	100	700

NOTE: Above we have used: "**ROWS** BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING" which will give a SINGLE output based on all INPUT Values/PARTITION (if used)

new_id	ROW_NUMBER	RANK	DENSE_RANK	PERCENT_RANK
100	1	1	1	0
200	2	2	2	0.166
200	3	2	2	0.166
300	4	4	3	0.5
500	5	5	4	0.666

Browser

Properties | SQL | Statistics | Dependencies | Dependents | rnmcopy/postgres@PostgreSQL 14*

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (3)
      - postgres
      - rnm_db
      - rnmcopy
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates
            - Foreign Tables
            - Functions

rnmcopy/postgres@PostgreSQL 14

No limit

Query | Query History

```
1  SELECT new_id,
2  ROW_NUMBER() OVER(ORDER BY new_id) AS "ROW_NUMBER",
3  RANK() OVER(ORDER BY new_id) AS "RANK",
4  DENSE_RANK() OVER(ORDER BY new_id) AS "DENSE_RANK",
5  PERCENT_RANK() OVER(ORDER BY new_id) AS "PERCENT_RANK"
6  FROM test_data
7
```

Data output | Messages | Notifications

	new_id bigint	ROW_NUMBER bigint	RANK bigint	DENSE_RANK bigint	PERCENT_RANK double precision
1	100	1	1	1	0
2	200	2	2	2	0.166666666666666
3	200	3	2	2	0.166666666666666
4	300	4	4	3	0.5
5	500	5	5	4	0.666666666666666
6	500	6	5	4	0.666666666666666
7	700	7	7	5	1

new_id	FIRST_VALUE	LAST_VALUE	LEAD	LAG
100	100	100	200	null
200	100	200	200	100
200	100	200	300	200
300	100	300	500	200
500	100	500	500	300
500	100	500	700	500
700	100	700	null	500

new_id	LEAD	LAG
100	200	NULL
200	300	NULL
200	500	100
300	500	200
500	700	200
500	NULL	300
700	NULL	500

new_id	LEAD_by2	LAG_by2
100	200	null
200	300	null
200	500	100
300	500	200
500	700	200
500	null	300
700	null	500

CASE Expression

- The CASE expression goes through conditions and returns a value when the first condition is met (like if-then-else statement). 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.

Browser

Properties | SQL | Statistics | Dependencies | Dependents | rnmcopy/postgres@PostgreSQL 14*

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (3)
      - postgres
      - rnm_db
      - rnmcopy
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates
            - Foreign Tables
            - Functions
            - Materialized Views

rnmcopy/postgres@PostgreSQL 14

No limit

Query | Query History

```
1  SELECT customer_id, amount,
2  CASE
3      WHEN amount > 100 THEN 'Expensive product'
4      WHEN amount = 100 THEN 'Moderate product'
5      ELSE 'Inexpensive product'
6  END AS ProductStatus
7  FROM payment
8
```

Data output | Messages | Notifications

	customer_id [PK] bigint	amount bigint	productstatus text
1	1	60	Inexpensive product
2	10	70	Inexpensive product
3	11	80	Inexpensive product
4	2	500	Expensive product
5	8	100	Moderate product

Total rows: 5 of 5 | Query complete 00:00:00.605 | Ln 4, Col 22

- **General CASE Syntax**

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

- **Example:**

```
SELECT customer_id, a
CASE
    WHEN amount > 100
    WHEN amount = 100 T
    ELSE 'Inexpensive produ
END AS ProductStatus
FROM payment
```

- **CASE Expression Syntax**

```
CASE Expression
    WHEN value1 THEN result1
    WHEN value2 THEN result2
    WHEN valueN THEN resultN
    ELSE other_result
END;
```

- **Example:**

```
SELECT customer_id,
 CASE amount
    WHEN 500 THEN
```

Browser

Properties SQL Statistics Dependencies Dependents rnmcopy/postgres@PostgreSQL 14*

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (3)
      - postgres
      - rnm_db
      - rnmcopy
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggregates
            - Collations
            - Domains
            - FTS Configurations
            - FTS Dictionaries
            - FTS Parsers
            - FTS Templates
            - Foreign Tables
            - Functions

rnmcopy/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SELECT customer_id,
2  CASE amount
3      WHEN 500 THEN 'Prime Customer'
4      WHEN 100 THEN 'Plus Customer'
5      ELSE 'Regular Customer'
6  END AS CustomerStatus
7  FROM payment
8
9  SELECT * FROM payment
```

Data output Messages Notifications

	customer_id [PK] bigint	customerstatus text
1	1	Regular Customer
2	10	Regular Customer
3	11	Regular Customer
4	2	Prime Customer
5	8	Plus Customer

Common Table Expression (CTE)

- A common table expression, or CTE, is a temporary named result set created from a simple SELECT statement that can be used in a subsequent SELECT statement
- We can define CTEs by adding a WITH clause directly before SELECT, INSERT, UPDATE, DELETE, or MERGE statement.
- The **WITH** clause can include one or more CTEs s
  by commas

Common Table Expression (CTE)

- **Syntax**

```
WITH my_cte AS (
        SELECT a,b,c
        FROM Table1 )
SELECT a,c
FROM my_cte
```

CTE query

Main query

The name of this CTE is my_cte, and the CTE query is SELECT a,b,c FROM Table1
with the **WITH** keyword, after which you specify the name of your CTE, then the
query in parentheses. The main query comes after the closing parenthesis and re
CTE. Here, the main query (also known as the outer query) is SELECT a,c FROM my

Browse | Properties | SQL | Statistics | Dependencies | Dependents | rnmcopy/postgres@PostgreSQL 14*

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (4)
      - Easy
      - postgres
      - rnm_db
      - rnmcopy
        - Casts
        - Catalogs
        - Event Trigger
        - Extensions
        - Foreign Data
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggrega
            - Collatio
            - Domain
            - FTS Cor
            - FTS Dict
            - FTS Par
            - FTS Ten
            - Foreign

rnmcopy/postgres@PostgreSQL 14

No limit

Query | Query History

```
1  WITH my_cte AS (
2      SELECT  *, AVG(amount) OVER(ORDER BY p.customer_id) AS "Average_Price",
3      COUNT(address_id) OVER(ORDER BY c.customer_id) AS "Count"
4      FROM payment as p
5      INNER JOIN customer AS c
6      ON p.customer_id = c.customer_id
7  )
```

Data output | Messages | Notifications

	customer_id bigint	amount bigint	mode character varying (50)	payment_date date	customer_id bigint	first_name character varying (50)	last_name character varying (50)	address_id bigint	Average_Price numeric	Cou bigi
1	1	60	Cash	2020-09-24	1	Mary	Smith	5	60.000000000000	
2	2	500	Credit Card	2020-04-27	2	Madan	Mohan	6	280.000000000000	
3	17	250	Credit Card	2021-04-01	17	R	Madhav	9	270.000000000000	

Browse | Properties | SQL | Statistics | Dependencies | Dependents | rnmcopy/postgres@PostgreSQL 14*

- Servers (2)
  - PostgreSQL 12
  - PostgreSQL 14
    - Databases (4)
      - Easy
      - postgres
      - rnm_db
      - rnmcopy
        - Casts
        - Catalogs
        - Event Trigger
        - Extensions
        - Foreign Data
        - Languages
        - Publications
        - Schemas (1)
          - public
            - Aggrega
            - Collatio
            - Domain
            - FTS Cor
            - FTS Dict
            - FTS Par
            - FTS Ten
            - Foreign

rnmcopy/postgres@PostgreSQL 14

No limit

Query | Query History

```
1  WITH my_cte AS (
2    SELECT mode, MAX(amount) AS highest_price, SUM(amount) AS total_price
3    FROM payment
4    GROUP BY mode
5  )
6  SELECT payment.*, my.highest_price, my.total_price
7  FROM payment
8  JOIN my_cte my
9  ON payment.mode = my.mode
10 ORDER BY payment.mode
```

Data output | Messages | Notifications

No data output. Execute a query to get output.