Lecture 03: Data Types, Importing Data, and SQL Math

DATA 503: Fundamentals of Data Engineering

Lucas P. Cordova, Ph.D.

2026-01-19

This lecture covers the data types, importing data, and SQL math.

Table of contents

1 Part 1: Getting Data Into PostgreSQL 2

2 Data Types: Choosing Wisely 8

3 Part 2: Math Operators and Functions 13

4 Aggregate Functions 18

5 Part 3: Statistics and Percentiles 24

6 Date Arithmetic 28

7 Homework 2 Preview 31

0.1 Today's Agenda

Part 1: Hands-On Setup + Data Types

- Import our sample dataset
- Understanding PostgreSQL data types
- Choosing the right type for your data

0.2 Today's Agenda (continued)

Part 2: Math Operators and Functions

- Arithmetic in SQL
- Rounding, absolute values, and precision
- Aggregate functions (SUM, AVG, COUNT)

0.3 Today's Agenda (continued)

Part 3: Statistical Functions + Assignment Preview

- Finding medians and percentiles
- Date arithmetic and intervals
- Homework 2 walkthrough

1 Part 1: Getting Data Into PostgreSQL

1.1 Why Not Just INSERT Everything?

Imagine you have 65,000 survey responses...

```sql
INSERT INTO survey VALUES (1, 'USA', 'Developer', 85000);
INSERT INTO survey VALUES (2, 'UK', 'Designer', 72000);
INSERT INTO survey VALUES (3, 'Germany', 'Developer', 91000);
-- ... 64,997 more times
```

This is:

- Slow (each INSERT is a separate transaction)
- Error-prone (one typo and you start over)
- Painful (your fingers will hate you)

Solution: Bulk import with `\COPY`

1.2 The Office: Our Sample Dataset

Today we will work with some Dunder Mifflin employee data again.

1.3 The Dataset

employee_id	full_name	department	salary_usd
1	Michael Scott	Management	75000.00
2	Dwight Schrute	Sales	62000.00
3	Pam Beesly	Reception	42000.00
4	Jim Halpert	Sales	61000.00

1.4 Expanded Dataset for Today

I have added a few more employees and columns so we can practice more SQL:

employee_id	full_name	department	salary_usd	performance_rating	years_experience
1	Michael Scott	Management	75000.00	3.2	15
2	Dwight Schrute	Sales	62000.00	4.8	12
3	Pam Beesly	Reception	42000.00	3.9	8
4	Jim Halpert	Sales	61000.00	4.1	10
5	Angela Martin	Accounting	52000.00	4.5	11
6	Kevin Malone	Accounting	48000.00	2.1	9
7	Oscar Martinez	Accounting	54000.00	4.7	13
8	Stanley Hudson	Sales	58000.00	3.0	20

1.5 Hands-On: Create the CSV File

Step 1: Create a file named `employees_import.csv`, I recommend in your Downloads directory for now.

Copy this exact content (hover over the data and click the copy button that appearsto the right):

```
employee_id,full_name,department,email,hire_date,salary_usd,is_manager,performance_rating,yea
1,Michael Scott,Management,michael.scott@dundermifflin.com,2005-03-24,75000.00,true,3.2,15,,2
2,Dwight Schrute,Sales,dwight.schrute@dundermifflin.com,2006-04-12,62000.00,false,4.8,12,0.08
3,Pam Beesly,Reception,pam.beesly@dundermifflin.com,2007-07-02,42000.00,false,3.9,8,,
4,Jim Halpert,Sales,jim.halpert@dundermifflin.com,2005-10-05,61000.00,false,4.1,10,0.07,2026-
5,Angela Martin,Accounting,angela.martin@dundermifflin.com,2006-08-15,52000.00,false,4.5,11,
6,Kevin Malone,Accounting,kevin.malone@dundermifflin.com,2007-02-28,48000.00,false,2.1,9,,202
7,Oscar Martinez,Accounting,oscar.martinez@dundermifflin.com,2005-06-01,54000.00,false,4.7,13
8,Stanley Hudson,Sales,stanley.hudson@dundermifflin.com,2004-11-20,58000.00,false,3.0,20,0.05
```

1.6 Hands-On: Move the CSV to a Safe Location

Why a "safe" location?

PostgreSQL needs permission to read your file. Some folders are restricted.

macOS / Linux:

```
cp ~/Downloads/employees_import.csv /tmp/employees_import.csv
ls -l /tmp/employees_import.csv
```

Windows (PowerShell):

```
Copy-Item $HOME\Downloads\employees_import.csv C:\Users\Public\employees_import.csv
Get-Item C:\Users\Public\employees_import.csv
```

1.7 Hands-On: Connect to PostgreSQL

Open your terminal and connect:

```
psql -U postgres -h localhost
```

You should see a prompt like:

```
postgres=#
```

⚠ Warning

- If your prompt spits out `command not found: psql` or something similar, psql is not in your PATH. Check out the resource on Canvas → Week 2 Lesson Plan → Adding PSQL to your PATH.
- If you see `psql: could not connect to server: No such file or directory` or something similar, you are not connected to the database or there are credential

issues. Try `psql -U postgres -h localhost` instead.

1.8 Hands-On: Create the Database

At the `postgres=#` prompt, run:

```
DROP DATABASE IF EXISTS office_db;
CREATE DATABASE office_db;
\c office_db
```

You should see:

```
You are now connected to database "office_db" as user "postgres".
```

Note

The `\c` command connects you to the new database.

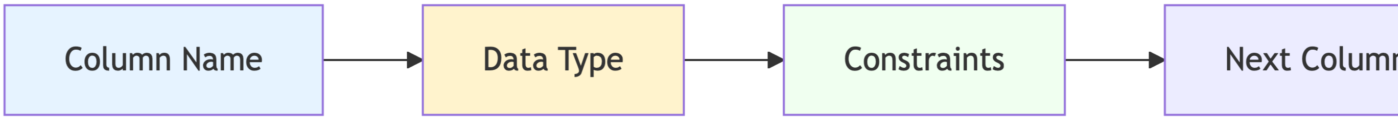
1.9 Hands-On: Create the Table

Now we need a table that matches our CSV columns exactly. Copy this exact content (hover over the data and click the copy button that appears to the right):

```
CREATE TABLE employees (
    employee_id        INTEGER PRIMARY KEY,
    full_name          TEXT NOT NULL,
    department         TEXT NOT NULL,
    email              TEXT,
    hire_date          DATE NOT NULL,
    salary_usd         NUMERIC(10,2) NOT NULL,
    is_manager         BOOLEAN NOT NULL,
    performance_rating NUMERIC(2,1),
    years_experience   INTEGER,
    commission_rate    NUMERIC(3,2),
    last_login         TIMESTAMP
);
```

Verify with `\d employees` to see the structure.

1.10 Understanding the CREATE TABLE Statement



Each column definition has:

- **Name:** What you call the column (e.g., `salary_usd`)
- **Data Type:** What kind of data it holds (e.g., `NUMERIC(10,2)`)
- **Constraints:** Rules the data must follow (e.g., `NOT NULL`)

1.11 Hands-On: Import the CSV

The moment of truth!

macOS / Linux:

```
\COPY employees FROM '/tmp/employees_import.csv' WITH (FORMAT csv, HEADER true)
```

Windows:

```
\COPY employees FROM 'C:\Users\Public\employees_import.csv' WITH (FORMAT csv, HEADER true)
```

You should see: `COPY 8`

That means 8 rows were imported successfully!

1.12 Hands-On: Verify Your Data

Count the rows:

```
SELECT COUNT(*) FROM employees;
```

count
8

View all the data:

```
SELECT * FROM employees ORDER BY employee_id;
```

Take a moment to verify the data looks correct.

1.13 The Anatomy of \COPY

```
\COPY employees FROM '/tmp/employees_import.csv' WITH (FORMAT csv, HEADER true)
```

Let's break this down:

Part	Meaning
`\COPY`	Client-side copy command
`employees`	Target table name
`FROM '/tmp/...'`	Source file path
`FORMAT csv`	File is comma-separated
`HEADER true`	First row is column names

1.14 \COPY vs COPY: What's the Difference?

Feature	`\COPY`	`COPY`
Runs on	Your computer (client)	Database server
File location	Your filesystem	Server filesystem
Permissions	Your user permissions	postgres user permissions
Best for	Development, small files	Production, large files

Rule of thumb: Use `\COPY` in this class. It is safer and easier.

1.15 Exercise: Check Your Import

Write queries to answer (work with a neighbor if your system is acting up):

1. How many employees are in the Sales department?
2. What is Michael Scott's email?
3. Which employee has the highest performance rating?

Take 3 minutes, then we will review.

1.16 Exercise Solutions

1. Sales department count:

```
SELECT COUNT(*) FROM employees WHERE department = 'Sales';
```

count
3

2. Michael's email:

```sql
SELECT email FROM employees WHERE full_name = 'Michael Scott';
```

email
michael.scott@dundermifflin.com

3. Highest performance rating:

```sql
SELECT full_name, performance_rating
FROM employees
ORDER BY performance_rating DESC
LIMIT 1;
```

full_name	performance_rating
Dwight Schrute	4.8

2 Data Types: Choosing Wisely

2.1 Why Data Types Matter

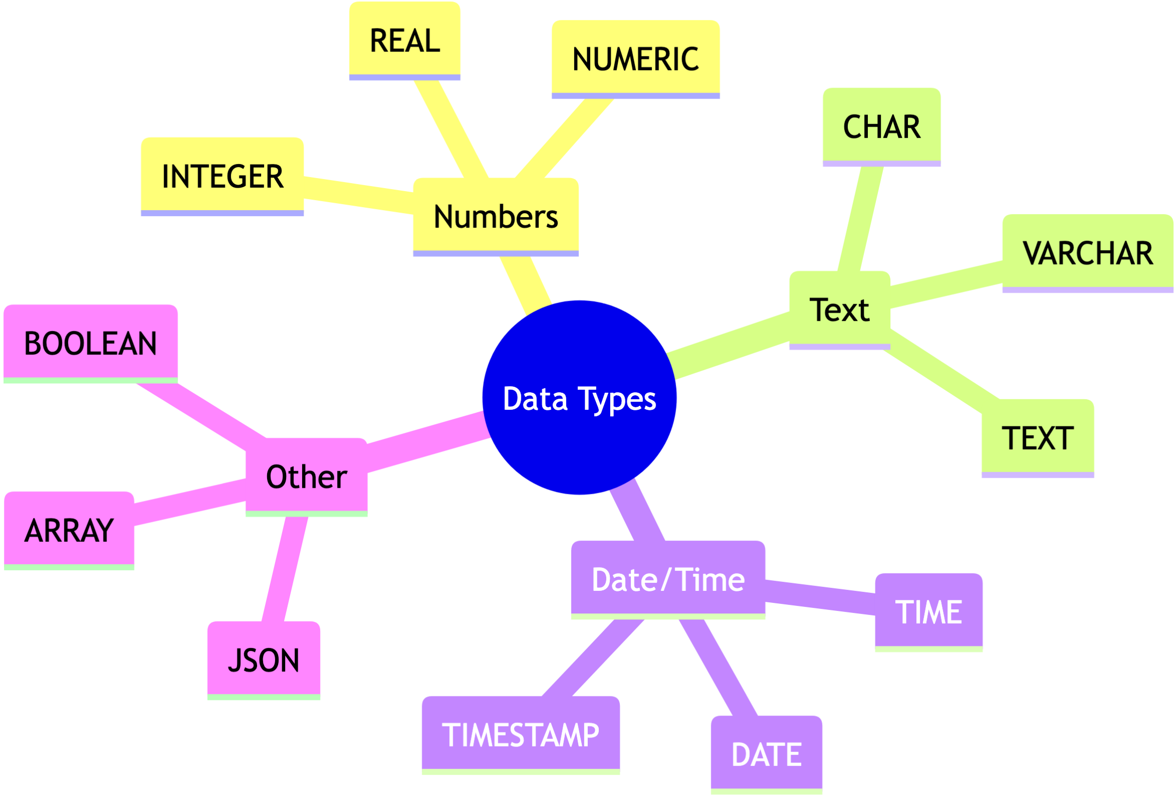
Consider this question: What is `'10' + '5'`?

- In some languages: `'105'` (string concatenation)
- In math: `15` (addition)

⚠ Warning

Data types tell PostgreSQL how to interpret and operate on your data.
Wrong data type = wrong results, wasted storage, or errors.

2.2 PostgreSQL Data Type Categories



2.3 Numeric Types: The Big Three

Type	Description	Example
`INTEGER`	Whole numbers	`42`, `-7`, `1000000`
`NUMERIC(p,s)`	Exact decimals	`75000.00`, `3.14159`
`REAL`	Approximate decimals	Scientific calculations

When to use each:

- `INTEGER`: Counts, IDs, quantities
- `NUMERIC`: Money, precise measurements
- `REAL`: Scientific data where approximation is OK

2.4 NUMERIC(precision, scale) Explained

```
salary_usd NUMERIC(10,2)
```

- **Precision (10):** Total digits allowed
- **Scale (2):** Digits after decimal point

Value	Valid for NUMERIC(10,2)?
75000.00	Yes (7 digits total)
123456789.99	No (11 digits total)
75000.001	Rounded to 75000.00

2.5 Quick Check: Our Employee Table

Look at how we defined numeric columns:

```
salary_usd         NUMERIC(10,2)    -- Up to 99,999,999.99
performance_rating NUMERIC(2,1)     -- 0.0 to 9.9
commission_rate    NUMERIC(3,2)     -- 0.00 to 9.99
years_experience   INTEGER          -- Whole years only
```

Question: Why not use INTEGER for salary?

Because we want cents! `$75,000.00` not `$75000`

2.6 Text Types: Three Flavors

Type	Description	Use Case
`CHAR(n)`	Fixed length, padded	Codes like state abbreviations
`VARCHAR(n)`	Variable length, max n	Names, emails with limits
`TEXT`	Unlimited length	Long descriptions, notes

ℹ Note

In practice: `TEXT` and `VARCHAR` are equally fast in PostgreSQL.
I generally use `TEXT` unless I have a specific length constraint.

2.7 Date and Time Types

Type	Stores	Example
`DATE`	Year, month, day	`'2026-01-15'`
`TIME`	Hour, minute, second	`'09:30:00'`
`TIMESTAMP`	Both date and time	`'2026-01-15 09:30:00'`

Our table uses:

```
hire_date  DATE       -- Just the date they started
last_login TIMESTAMP  -- Date AND time of login
```

2.8 Boolean: True or False

The `BOOLEAN` type stores `TRUE` or `FALSE`.

PostgreSQL accepts multiple formats:

True Values	False Values
`TRUE`, `'t'`, `'yes'`, `'1'`	`FALSE`, `'f'`, `'no'`, `'0'`

```
SELECT full_name, is_manager
FROM employees
WHERE is_manager = TRUE;
```

full_name	is_manager
Michael Scott	t

2.9 NULL: The Absence of Data

`NULL` means “unknown” or “not applicable.”

Look at our data for commissions:

```
SELECT full_name, commission_rate, last_login
FROM employees
WHERE commission_rate IS NULL;
```

full_name	commission_rate	last_login
Michael Scott	NULL	2026-01-15 09:12:00
Pam Beesly	NULL	NULL
Angela Martin	NULL	2026-01-15 07:45:00

> ℹ Note
>
> Pam has no commission (not in sales) AND no last_login (maybe she uses paper).

2.10 NULL Gotchas

NULL is not equal to anything, not even itself!

```sql
-- This finds NOTHING:
SELECT * FROM employees WHERE commission_rate = NULL;


-- This works:
SELECT * FROM employees WHERE commission_rate IS NULL;
```

2.11 NULL Gotchas: Math with NULL

Math with NULL returns NULL

```sql
SELECT 100 + NULL;  -- Returns NULL
SELECT NULL * 5;    -- Returns NULL
```

> ℹ Note
>
> We will learn to handle this with `COALESCE` and `NULLIF` later.

2.12 Exercise: Data Type Detective

Look at this data and choose the best PostgreSQL data type:

1. Social Security Number: `'123-45-6789'`
2. Product price: `$29.99`
3. Is item in stock: `yes` or `no`
4. Number of items in warehouse: `1,547`

5. Customer review text: `'Great product! Would buy again...'`

Take 2 minutes to decide, then we discuss.

2.13 Exercise: Data Type Answers

1. **SSN:** `CHAR(11)` or `VARCHAR(11)` - It is text, not a number (leading zeros, dashes)
2. **Price:** `NUMERIC(10,2)` - Money needs exact precision
3. **In stock:** `BOOLEAN` - True/false value
4. **Warehouse count:** `INTEGER` - Whole numbers only
5. **Review text:** `TEXT` - Variable length, potentially long

2.14 Next up: Math Operators and Functions

Stretch or grab coffee, then verify your import worked.

```sql
-- Quick verification
SELECT COUNT(*) FROM employees;
-- Should return 8
```

3 Part 2: Math Operators and Functions

3.1 SQL as a Calculator

PostgreSQL can do math! Let's start simple:

```sql
SELECT 2 + 2;
```

?column?
4

No table needed. SQL evaluates the expression and returns the result.

3.2 Basic Math Operators

Operator	Operation	Example	Result
+	Addition	5 + 3	8
-	Subtraction	10 - 4	6
*	Multiplication	6 * 7	42
/	Division	15 / 4	3
%	Modulo (remainder)	15 % 4	3

Wait, what?

15 / 4 = 3? d Yes! Integer division truncates.

3.3 Integer Division Trap

```
SELECT 15 / 4;        -- Returns 3 (integer division)
SELECT 15.0 / 4;      -- Returns 3.75 (decimal division)
SELECT 15 / 4.0;      -- Returns 3.75 (decimal division)
SELECT 15::NUMERIC / 4;  -- Returns 3.75 (cast to numeric)
```

Note

Rule: If both operands are integers, result is integer.
Make at least one operand a decimal to get decimal results.

3.4 Hands-On: Math with Employee Data

Calculate annual bonus (10% of salary):

```
SELECT
    full_name,
    salary_usd,
    salary_usd * 0.10 AS annual_bonus
FROM employees;
```

full_name	salary_usd	annual_bonus
Michael Scott	75000.00	7500.0000
Dwight Schrute	62000.00	6200.0000
Pam Beesly	42000.00	4200.0000

3.5 Calculating Percent Change

The formula: `((new - old) / old) * 100`

Example: If salary goes from $50,000 to $55,000:

```
SELECT ((55000 - 50000) / 50000.0) * 100 AS pct_increase;
```

pct_increase
10.0

A 10% raise. (Nice!)

3.6 Hands-On: Salary Per Year of Experience

How much does each employee earn per year of experience?

```
SELECT
    full_name,
    salary_usd,
    years_experience,
    ROUND(salary_usd / years_experience, 2) AS salary_per_year
FROM employees
ORDER BY salary_per_year DESC;
```

full_name	salary_usd	years_experience	salary_per_year
Pam Beesly	42000.00	8	5250.00
Jim Halpert	61000.00	10	6100.00
Dwight Schrute	62000.00	12	5166.67

3.7 The ROUND() Function

`ROUND(value, decimal_places)` rounds to specified precision:

```
SELECT ROUND(3.14159, 2);    -- Returns 3.14
SELECT ROUND(3.14159, 0);    -- Returns 3
SELECT ROUND(3.5, 0);        -- Returns 4 (rounds up)
SELECT ROUND(2.5, 0);        -- Returns 3 (banker's rounding)
```

ℹ Note

Always ROUND money calculations for clean output!

3.8 The ABS() Function

`ABS()` returns the absolute value (distance from zero):

```sql
SELECT ABS(-42);     -- Returns 42
SELECT ABS(42);      -- Returns 42
SELECT ABS(0);       -- Returns 0
```

3.9 Using ABS()

Use case: Finding differences regardless of direction:

```sql
SELECT
    full_name,
    salary_usd,
    ABS(salary_usd - 55000) AS distance_from_average
FROM employees;
```

3.10 Hands-On: Distance from Average Salary

Let's find how far each employee's salary is from $55,000:

```sql
SELECT
    full_name,
    salary_usd,
    salary_usd - 55000 AS difference,
    ABS(salary_usd - 55000) AS absolute_difference
FROM employees
ORDER BY absolute_difference DESC;
```

full_name	salary_usd	difference	absolute_difference
Michael Scott	75000.00	20000.00	20000.00
Pam Beesly	42000.00	-13000.00	13000.00
Dwight Schrute	62000.00	7000.00	7000.00

3.11 Exponents and Roots

Function	Description	Example	Result
^	Exponentiation	2 ^ 3	8
|/	Square root	|/ 16	4
||/	Cube root	||/ 27	3
SQRT()	Square root (function)	SQRT(16)	4

```
SELECT
    |/ 25 AS square_root,
    SQRT(25) AS also_square_root,
    2 ^ 10 AS two_to_the_tenth;
```

3.12 Exercise: Calculate Commission

Sales employees have a commission rate. Calculate their potential commission on a $10,000 sale:

```
SELECT
    full_name,
    commission_rate,
    -- Your calculation here: commission on $10,000 sale
FROM employees
WHERE commission_rate IS NOT NULL;
```

Hint: Commission = sale_amount * commission_rate

Take 2 minutes.

3.13 Exercise Solution: Commission Calculation

```
SELECT
    full_name,
    commission_rate,
    10000 * commission_rate AS commission_on_10k
FROM employees
WHERE commission_rate IS NOT NULL;
```

full_name	commission_rate	commission_on_10k
Dwight Schrute	0.08	800.00

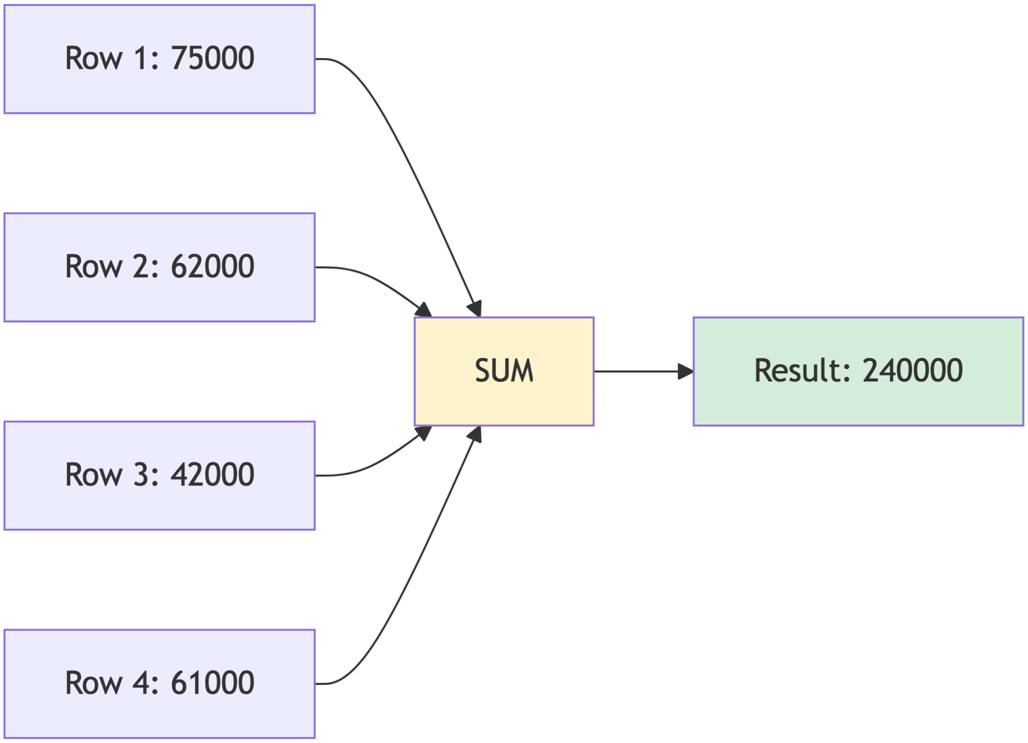
full_name	commission_rate	commission_on_10k
Jim Halpert	0.07	700.00
Stanley Hudson	0.05	500.00

Dwight’s higher rate reflects his #1 salesman status. Obviously.

4 Aggregate Functions

4.1 What Are Aggregate Functions?

Aggregate functions compute a **single result from multiple rows**.



4.2 The Big Five Aggregates

Function	Description	Example
`SUM()`	Total of all values	Total payroll

Function	Description	Example
AVG()	Average (mean)	Average salary
COUNT()	Number of rows	How many employees
MIN()	Smallest value	Lowest salary
MAX()	Largest value	Highest salary

4.3 SUM(): Total Values

What is our total payroll?

```sql
SELECT SUM(salary_usd) AS total_payroll
FROM employees;
```

total_payroll
452000.00

We spend $452,000 on salaries. (Michael probably thinks he deserves half.)

4.4 AVG(): Calculate the Mean

What is the average salary?

```sql
SELECT AVG(salary_usd) AS avg_salary
FROM employees;
```

avg_salary
56500.000000

That is a lot of decimal places. Let's fix that:

```sql
SELECT ROUND(AVG(salary_usd), 2) AS avg_salary
FROM employees;
```

avg_salary
56500.00

4.5 COUNT(): How Many?

Three ways to count:

```
-- Count all rows
SELECT COUNT(*) FROM employees;

-- Count non-NULL values in a column
SELECT COUNT(commission_rate) FROM employees;

-- Count distinct values
SELECT COUNT(DISTINCT department) FROM employees;
```

count(*)	count(commission)	count(distinct dept)
8	3	4

Only 3 employees have commission rates!

4.6 MIN() and MAX(): The Extremes

Salary range:

```
SELECT
    MIN(salary_usd) AS lowest_salary,
    MAX(salary_usd) AS highest_salary,
    MAX(salary_usd) - MIN(salary_usd) AS salary_range
FROM employees;
```

lowest_salary	highest_salary	salary_range
42000.00	75000.00	33000.00

Pam makes the least, Michael makes the most. Shocking.

4.7 Combining Multiple Aggregates

You can calculate several aggregates in one query:

```sql
SELECT
    COUNT(*) AS num_employees,
    SUM(salary_usd) AS total_payroll,
    ROUND(AVG(salary_usd), 2) AS avg_salary,
    MIN(salary_usd) AS min_salary,
    MAX(salary_usd) AS max_salary
FROM employees;
```

num_employees	total_payroll	avg_salary	min_salary	max_salary
8	452000.00	56500.00	42000.00	75000.00

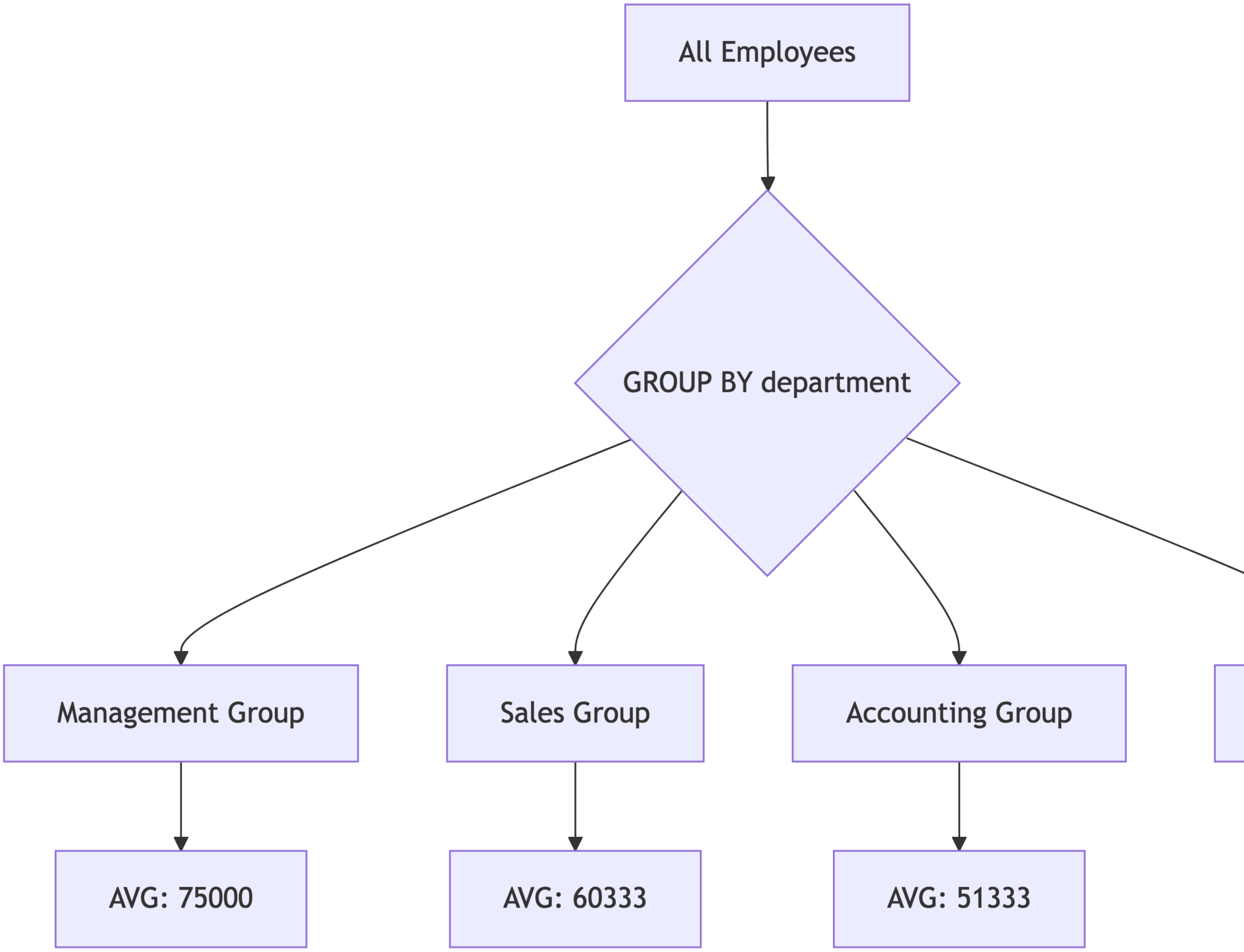
4.8 GROUP BY: Aggregates by Category

What if we want average salary BY DEPARTMENT?

```sql
SELECT
    department,
    COUNT(*) AS num_employees,
    ROUND(AVG(salary_usd), 2) AS avg_salary
FROM employees
GROUP BY department
ORDER BY avg_salary DESC;
```

department	num_employees	avg_salary
Management	1	75000.00
Sales	3	60333.33
Accounting	3	51333.33
Reception	1	42000.00

4.9 How GROUP BY Works



GROUP BY splits data into buckets, then aggregates each bucket separately.

4.10 HAVING: Filter After Grouping

Show only departments with more than 1 employee:

```
SELECT
    department,
    COUNT(*) AS num_employees,
```

```sql
    ROUND(AVG(salary_usd), 2) AS avg_salary
FROM employees
GROUP BY department
HAVING COUNT(*) > 1
ORDER BY avg_salary DESC;
```

department	num_employees	avg_salary
Sales	3	60333.33
Accounting	3	51333.33

`WHERE` filters rows BEFORE grouping. `HAVING` filters AFTER grouping.

4.11 Exercise: Department Analysis

Write a query that shows for each department:

- Department name
- Number of employees
- Total salary expense
- Average performance rating (rounded to 1 decimal)

Only include departments where average performance rating > 3.5

Take 4 minutes.

4.12 Exercise Solution

```sql
SELECT
    department,
    COUNT(*) AS num_employees,
    SUM(salary_usd) AS total_salary,
    ROUND(AVG(performance_rating), 1) AS avg_rating
FROM employees
GROUP BY department
HAVING AVG(performance_rating) > 3.5
ORDER BY avg_rating DESC;
```

department	num_employees	total_salary	avg_rating
Accounting	3	154000.00	3.8
Sales	3	181000.00	3.6

Management and Reception did not make the cut!

4.13 10 Minute Break

When we return: Statistical functions and percentiles!

Up next: Finding the median (and why it matters more than average).

5 Part 3: Statistics and Percentiles

5.1 The Problem with Averages

Pop quiz: A company has 5 employees with these salaries:

$40,000, $42,000, $45,000, $48,000, $500,000

What is the average salary?

```sql
SELECT AVG(salary) FROM company;
-- Returns: $135,000
```

Is $135,000 a good representation of "typical" salary? **No!**

The CEO's salary skews the average dramatically.

5.2 Median: The Middle Value

The **median** is the middle value when data is sorted.

$40,000, $42,000, **$45,000**, $48,000, $500,000

The median is $45,000, which better represents "typical."

Tip

When to use which:

- **Mean (average):** Data is normally distributed, no outliers
- **Median:** Data is skewed or has outliers

5.3 Finding Median with percentile_cont()

PostgreSQL does not have a built-in `MEDIAN()` function, but we can use:

```
SELECT
    percentile_cont(0.5) WITHIN GROUP (ORDER BY salary_usd) AS median_salary
FROM employees;
```

median_salary
55000

The `0.5` means "50th percentile" which is the median.

5.4 The WITHIN GROUP Syntax

```
percentile_cont(0.5) WITHIN GROUP (ORDER BY salary_usd)
```

Let's break this down:

Part	Meaning
`percentile_cont(0.5)`	Find the 50th percentile (median)
`WITHIN GROUP`	Required syntax for ordered-set aggregates
`ORDER BY salary_usd`	Which column to calculate percentile on

5.5 Hands-On: Median vs Average

Compare median and average for our employee salaries:

```
SELECT
    ROUND(AVG(salary_usd), 2) AS mean_salary,
    percentile_cont(0.5) WITHIN GROUP (ORDER BY salary_usd) AS median_salary
FROM employees;
```

mean_salary	median_salary
56500.00	55000

Pretty close! Our data does not have extreme outliers.

5.6 What is a Percentile?

A percentile tells you what percentage of values fall below a point.

- **25th percentile:** 25% of values are below this
- **50th percentile:** 50% are below (the median)
- **75th percentile:** 75% are below this
- **90th percentile:** 90% are below this

If you score in the 90th percentile on a test, you beat 90% of test-takers.

5.7 Calculating Quartiles

Quartiles divide data into four equal parts:

```
SELECT
    percentile_cont(0.25) WITHIN GROUP (ORDER BY salary_usd) AS q1,
    percentile_cont(0.50) WITHIN GROUP (ORDER BY salary_usd) AS q2_median,
    percentile_cont(0.75) WITHIN GROUP (ORDER BY salary_usd) AS q3
FROM employees;
```

q1	q2_median	q3
49000	55000	61250

5.8 Using Arrays for Multiple Percentiles

Instead of separate function calls, use an array:

```
SELECT
    percentile_cont(ARRAY[0.25, 0.5, 0.75])
        WITHIN GROUP (ORDER BY salary_usd) AS quartiles
FROM employees;
```

quartiles
{49000,55000,61250}

The result is an array. Curly braces indicate array values.

5.9 percentile_cont vs percentile_disc

Two versions exist:

Function	Behavior	Best For
`percentile_cont`	Interpolates between values	Continuous data
`percentile_disc`	Returns actual value from data	Discrete data

```sql
SELECT
    percentile_cont(0.5) WITHIN GROUP (ORDER BY salary_usd) AS cont,
    percentile_disc(0.5) WITHIN GROUP (ORDER BY salary_usd) AS disc
FROM employees;
```

Tip

For salaries, `percentile_cont` is usually more appropriate.

5.10 Exercise: Salary Percentile Analysis

Write a query that shows:

- The 10th percentile salary (low end)
- The median salary
- The 90th percentile salary (high end)

Use a single `percentile_cont` call with an array.

Take 2 minutes.

5.11 Exercise Solution

```sql
SELECT
    percentile_cont(ARRAY[0.10, 0.50, 0.90])
        WITHIN GROUP (ORDER BY salary_usd) AS salary_percentiles
FROM employees;
```

salary_percentiles
{43400,55000,69550}

Interpretation:

- 10% of employees make less than $43,400
- 50% make less than $55,000 (median)
- 90% make less than $69,550

6 Date Arithmetic

6.1 Working with Dates in PostgreSQL

PostgreSQL makes date math intuitive:

```
SELECT '2026-01-15'::DATE - '2025-01-15'::DATE AS days_between;
```

days_between
365

Subtracting dates gives you the number of days between them.

6.2 Hands-On: Calculate Tenure

How long has each employee been with the company?

```
SELECT
    full_name,
    hire_date,
    CURRENT_DATE AS today,
    CURRENT_DATE - hire_date AS days_employed
FROM employees
ORDER BY days_employed DESC;
```

full_name	hire_date	today	days_employed
Stanley Hudson	2004-11-20	2026-01-19	7730
Michael Scott	2005-03-24	2026-01-19	7606
Jim Halpert	2005-10-05	2026-01-19	7411

6.3 EXTRACT(): Pull Parts from Dates

`EXTRACT(part FROM date)` gets specific components:

```sql
SELECT
    full_name,
    hire_date,
    EXTRACT(YEAR FROM hire_date) AS hire_year,
    EXTRACT(MONTH FROM hire_date) AS hire_month,
    EXTRACT(DOW FROM hire_date) AS day_of_week
FROM employees;
```

full_name	hire_date	hire_year	hire_month	day_of_week
Michael Scott	2005-03-24	2005	3	4

DOW = Day of Week (0=Sunday, 1=Monday, etc.)

6.4 DATE_PART(): Alternative Syntax

`DATE_PART('part', date)` does the same thing:

```sql
SELECT
    full_name,
    DATE_PART('year', hire_date) AS hire_year,
    DATE_PART('quarter', hire_date) AS hire_quarter
FROM employees;
```

full_name	hire_year	hire_quarter
Michael Scott	2005	1
Angela Martin	2006	3

Use whichever syntax you prefer. They are equivalent.

6.5 Grouping by Date Parts

How many employees were hired each year?

```
SELECT
    EXTRACT(YEAR FROM hire_date) AS hire_year,
    COUNT(*) AS num_hired
FROM employees
GROUP BY EXTRACT(YEAR FROM hire_date)
ORDER BY hire_year;
```

hire_year	num_hired
2004	1
2005	3
2006	2
2007	2

6.6 Interval Arithmetic

You can add intervals to dates:

```
SELECT
    hire_date,
    hire_date + INTERVAL '1 year' AS one_year_later,
    hire_date + INTERVAL '90 days' AS ninety_days_later
FROM employees
WHERE full_name = 'Jim Halpert';
```

hire_date	one_year_later	ninety_days_later
2005-10-05	2006-10-05	2006-01-03

6.7 Exercise: Login Analysis

Write a query that shows:

- Employee name
- Their last login date/time
- How many days ago they logged in (from CURRENT_DATE)

Only include employees who HAVE logged in (not NULL).

Order by most recent login first.

Take 3 minutes.

6.8 Exercise Solution

```sql
SELECT
    full_name,
    last_login,
    CURRENT_DATE - last_login::DATE AS days_since_login
FROM employees
WHERE last_login IS NOT NULL
ORDER BY last_login DESC;
```

full_name	last_login	days_since_login
Dwight Schrute	2026-01-16 08:01:00	3
Jim Halpert	2026-01-16 08:03:00	3
Kevin Malone	2026-01-16 09:30:00	3

Note: I cast `last_login` to DATE to get whole days.

7 Homework 2 Preview

7.1 Assignment Overview

Homework 2 covers everything we learned today:

- **Q1:** CREATE TABLE with correct data types
- **Q2:** Import data with `\COPY`
- **Q3-Q5:** Math operators and aggregates
- **Q6-Q7:** Percentile functions
- **Q8-Q10:** Date arithmetic and grouping

You will use the `stackoverflow` database with three tables:

- `currency_rates` (CSV). Create table and import a CSV.
- `country_stats` (SQL to execute)
- `response_timeline` (SQL to execute)

7.2 Q1 and Q2 Tips: Creating and Importing

You will define a `currency_rates` table based on sample CSV data.

Sample data you will see:

```
rate_date,currency_code,currency_name,exchange_rate,is_major_currency
2025-01-01,USD,US Dollar,1.000000,true
2025-01-01,EUR,Euro,0.8523,true
```

Choose types carefully:

- `rate_date` needs a DATE type
- `currency_code` is always 3 characters
- `exchange_rate` needs decimal precision

7.3 Q3 and Q4 Tips: Math Functions

Q3 asks for ABS():

Remember that ABS gives the distance from zero:

```
ABS(exchange_rate - 1.0)  -- Distance from USD rate
```

Q4 asks for ROUND() and percentages:

```
ROUND((part / whole) * 100, 2)  -- Two decimal places
```

Watch for integer division! Cast if needed.

7.4 Q5 Tips: GROUP BY with HAVING

You will group by currency and filter with HAVING.

Common mistake: Using WHERE instead of HAVING for aggregate conditions.

```
-- WRONG: WHERE cannot filter on aggregates
SELECT currency_code, COUNT(*)
FROM currency_rates
WHERE COUNT(*) = 12  -- ERROR!
GROUP BY currency_code;

-- RIGHT: Use HAVING for aggregate conditions
SELECT currency_code, COUNT(*)
```

```
FROM currency_rates
GROUP BY currency_code
HAVING COUNT(*) = 12;  -- Correct!
```

7.5 Q6 and Q7 Tips: Percentiles

Q6: Finding the median

```
percentile_cont(0.5) WITHIN GROUP (ORDER BY column_name)
```

Do not forget `WITHIN GROUP`! It is required.

Q7: Using arrays for quartiles

```
percentile_cont(ARRAY[0.25, 0.5, 0.75]) WITHIN GROUP (ORDER BY column_name)
```

The ARRAY keyword is essential.

7.6 Q8 and Q9 Tips: Date Arithmetic

Q8: Subtracting dates gives days:

```
response_date - survey_start_date AS days_since_start
```

Q9: EXTRACT for grouping by time parts:

```
EXTRACT(MONTH FROM rate_date) AS month_num
```

Remember to GROUP BY the same expression you SELECT.

7.7 Q10 Tips: Self-Join

The hardest question! You need to compare January and December rates.

Strategy: Join the table to itself:

```
FROM currency_rates jan
JOIN currency_rates dec
    ON jan.currency_code = dec.currency_code
WHERE jan.rate_date = '2025-01-01'
  AND dec.rate_date = '2025-12-01'
```

Use table aliases (`jan`, `dec`) to distinguish the two instances.

7.8 Common Mistakes to Avoid

1. **Forgetting to alias calculated columns**

```sql
-- Bad: No alias
SELECT salary * 0.10 FROM employees;

-- Good: Clear alias
SELECT salary * 0.10 AS bonus FROM employees;
```

2. **Integer division giving wrong results**

```sql
-- Returns 0 (integer division)
SELECT 3 / 4;

-- Returns 0.75
SELECT 3.0 / 4;
```

3. **Using WHERE instead of HAVING with aggregates**

7.9 Testing Your Queries Locally

Before submitting:

1. Run your query in Beekeeper Studio or psql
2. Verify the output looks reasonable
3. Check that column aliases match what the question asks
4. Make sure ORDER BY direction is correct (ASC vs DESC)

Pro tip: Start simple, then add complexity.

7.10 Questions?

Topics we covered today:

- Data types (INTEGER, NUMERIC, TEXT, DATE, BOOLEAN)
- Importing data with `\COPY`
- Math operators (+, -, *, /, %, ABS, ROUND)
- Aggregates (SUM, AVG, COUNT, MIN, MAX)
- Percentiles (percentile_cont with WITHIN GROUP)
- Date arithmetic (subtraction, EXTRACT)

What questions do you have?

7.11 Next Week Preview

Chapter 6: Joining Tables

We will learn how to combine data from multiple tables:

```sql
SELECT employees.name, departments.budget
FROM employees
JOIN departments ON employees.dept_id = departments.id;
```

This is where SQL gets really powerful!

7.12 Thank You!

Reminders:

- Homework 2 due date: Check Canvas
- Office hours: Check Canvas for schedule
- Questions: Post on Discord or email

Good luck with the assignment!