awake
to Power Communications
...excellence & reach

WEEK 5

Week 5

ER Wait Time Crisis | PostgreSQL

Healthcare Case with Depth

Master SQL with a Real -World Healthcare Case Study.

```
WITH wait_times AS(
    SELECT
    patient_id,
    departments_id,
    ROUND(EXTRACT(EPOCH FROM (check_out_time-check_in_time))/60,2) AS wait_minutes
    FROM visits
    WHERE check_out_time IS NOT NULL AND check_in_time IS NOT NULL
)
SELECT
    d.department_name,
    ROUND(AVG(w.wait_minutes),2) AS avg_wait_time_minutes
    FROM wait_times w
    JOIN departments d ON w.departments_id = d.department_id
    GROUP BY d.department_name
    ORDER BY avg_wait_time_minutes DESC;
```

BUSINESS CASE BACKGROUND

1. Introduction

This document is designed to help students understand how to apply SQL to solve real-world business problems using a healthcare case study. The scenario focuses on emergency room (ER) wait times at CVS Pharmacy walk-in clinics. SQL is used to clean data, generate key performance metrics, and support decision-making.

2. Problem Statement

The executive team at CVS Pharmacy is concerned that 20% of their ER locations have average wait times exceeding 2 hours. This impacts patient satisfaction, clinic efficiency, and brand reputation.

They need a data-driven approach to identify:

1. Which locations have the longest wait times
2. Which departments are causing the delays
3. What percentage of locations are underperforming

3. Objectives

By the end of this analysis, students will be able to:

1. Use SQL to calculate wait times from timestamps
2. Aggregate and summarize performance by location and department
3. Identify problem areas based on business-defined KPIs
4. Interpret query outputs to make strategic decisions

CREATING TABLES

4. Step-by-Step SQL Walkthrough

4.0 Creating tables with proper constraints

```
CREATE TABLE departments (
  department_id INT PRIMARY KEY,
  department_name VARCHAR(100) NOT NULL
);

CREATE TABLE locations (
  location_id INT PRIMARY KEY,
  location_name VARCHAR(100) NOT NULL
);

CREATE TABLE patients (
  patient_id INT PRIMARY KEY,
  check_in_time TIMESTAMPTZ NOT NULL,
  check_out_time TIMESTAMPTZ NOT NULL,
  department_id INT NOT NULL REFERENCES departments(department_id),
  location_id INT NOT NULL REFERENCES locations(location_id),
  CHECK (check_out_time >= check_in_time)
);
```

Describing what each line does

Line	Purpose
patient_id INT PRIMARY KEY	Unique identifier for each patient record.
check_in_time TIMESTAMPTZ NOT NULL	Stores the date and time the patient checked in (with timezone).
check_out_time TIMESTAMPTZ NOT NULL	Stores the date and time the patient checked out.
department_id INT NOT NULL REFERENCES...	Links the patient to a department (foreign key).
location_id INT NOT NULL REFERENCES...	Links the patient to a location (foreign key).
CHECK (check_out_time >= check_in_time)	Prevents impossible entries (e.g., checkout before check-in).
);	Closes the table definition properly.

WAIT TIME

4.1 Calculating Wait Time Per Patient

Imagine you're a data analyst at CVS Health. Executives are complaining that 20% of ER locations have long wait times (patients wait over 2 hours). They ask:

"How long does each patient spend in the ER, and where?"

But before we analyze average times, departments, or locations, we must first calculate the actual wait time for every patient. That's what this query does.

SQL Snippet:

```
WITH wait_times AS (
  SELECT
    patient_id,
    location_id,
    department_id,
    ROUND(EXTRACT(EPOCH FROM (check_out_time - check_in_time))/60,2) AS wait_minutes
  FROM patients
  WHERE check_out_time IS NOT NULL AND check_in_time IS NOT NULL
)

SELECT * FROM wait_times;
```

WAIT TIME

Line by Line Breakdown

WITH wait_times **AS** (....)

This defines a Common Table Expression (CTE). It is a temporary table that stores intermediate results. It makes the query easier to read and reuse.

SELECT

We are selecting these fields from the patients table

patient_id, --this is a unique ID for each patient
location_id, --which CVS branch or clinic the patient visited
department_id,-- Which department the patient interacted with

ROUND(EXTRACT(EPOCH **FROM** (check_out_time - check_in_time))/60,2) **AS** wait_minutes

This calculates the **wait time in minutes** and rounds it to 2 decimal places.

Let's break it further:

check_out_time - check_in_time --this gives a time interval

EXTRACT(EPOCH **FROM** --this converts that interval into seconds

.../60 --this converts seconds into minutes

ROUND(...,2) --rounds the minutes to 2 decimal places

WHERE **check_out_time** IS NOT NULL AND **check_in_time** IS NOT NULL --It only returns rows where both check_in_time and check_out_time have actual values; but this line is technically not necessary because we already created a NOT NULL constraint when we were creating the tables.

AVERAGE WAIT TIME

4.2 Average Wait Time by Location

SQL Snippet:

```sql
WITH wait_times AS (
  SELECT
    patient_id,
    location_id,
    department_id,
    ROUND(EXTRACT(EPOCH FROM (check_out_time - check_in_time))/60,2) AS wait_minutes
  FROM patients
  WHERE check_in_time IS NOT NULL AND check_out_time IS NOT NULL
),
location_avg_waits AS (
  SELECT
    l.location_name,
    ROUND(AVG(w.wait_minutes), 2) AS avg_wait_time_minutes
  FROM wait_times w
  JOIN locations l ON w.location_id = l.location_id
  GROUP BY l.location_name
)
SELECT *
FROM location_avg_waits
ORDER BY avg_wait_time_minutes DESC;
```

Block by Block Breakdown

WITH wait_times **AS** (--Block 1 wait time CTE
SELECT
patient_id,
location_id,
department_id,
ROUND(EXTRACT(EPOCH **FROM** (check_out_time - check_in_time))/60,2) **AS** wait_minutes
FROM patients
WHERE check_out_time **IS NOT NULL AND** check_in_time **IS NOT NULL**
),
Block 1 creates a temporary result (called a CTE or Common Table Expression) named wait_times. It calculates how long each patient spent between check-in and check-out, in minutes.

Function	Purpose
EXTRACT(EPOCH FROM interval)	Converts a timestamp interval into seconds.
check_out_time - check_in_time	Computes the duration a patient waited.
/60	Converts seconds to minutes.
ROUND(..., 2)	Rounds the minutes to 2 decimal places.

Example:

If a patient checked in at 10:00 and out at 10:45, the difference is:

2700 seconds == 45 minutes == 45.00 after rounding

AVERAGE WAIT TIME

location_avg_waits AS (**-- Block 2: location_avg_waits CTE**

```
 SELECT
   l.location_name,
   ROUND(AVG(w.wait_minutes), 2) AS avg_wait_time_minutes
FROM wait_times w
JOIN locations l ON w.location_id = l.location_id
GROUP BY l.location_name
```

Block 2 takes the patient-level data from wait_times and joins it with the locations table to get human-readable names. Computes the average wait time per location.

SELECT * -- Block 3

FROM location_avg_waits

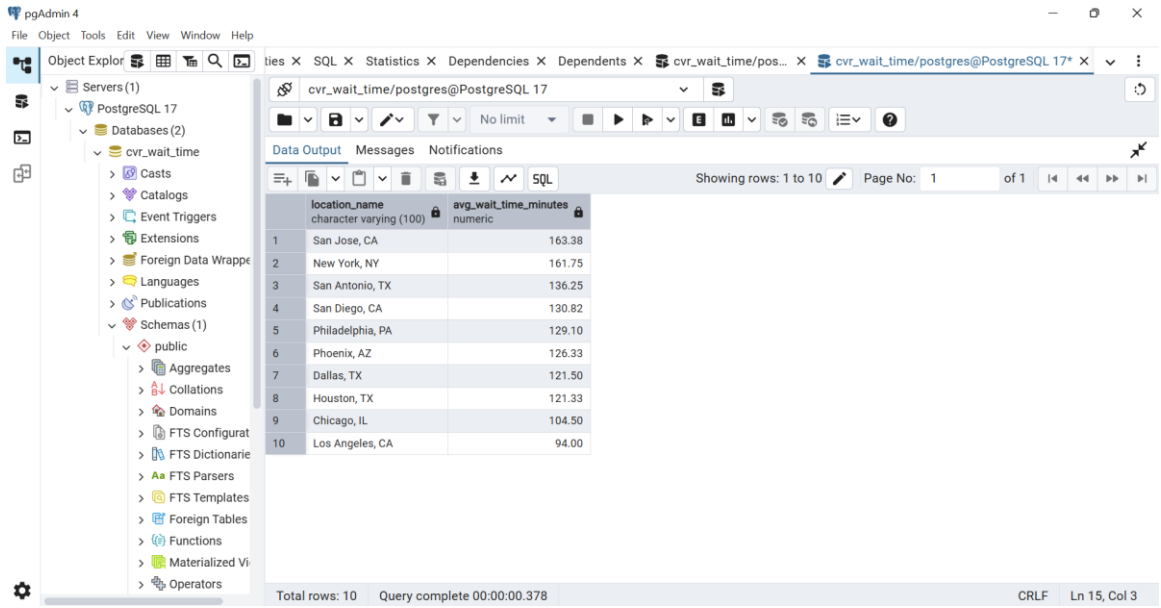
ORDER BY avg_wait_time_minutes DESC;

Explanation for block 3

Part	Meaning
SELECT *	Get all columns from the table/view.
FROM location_avg_waits	Use the data stored in the location_avg_waits table (or view).
ORDER BY avg_wait_time_minutes DESC	Sort rows so that locations with the longest average wait times appear at the top.

AVERAGE WAIT TIME

Expected output



PROBLEM LOCATION

4.3 Identifying Problem Locations (Wait > 120 Minutes)

SQL Snippet:

```
28
29  WITH wait_times AS (
30    SELECT
31      patient_id,
32      location_id,
33      EXTRACT(EPOCH FROM (check_out_time - check_in_time)) / 60 AS wait_minutes
34    FROM patients
35    WHERE check_in_time IS NOT NULL AND check_out_time IS NOT NULL
36  )
37
38  SELECT
39    l.location_name,
40    ROUND(AVG(w.wait_minutes), 2) AS avg_wait_time_minutes
41  FROM wait_times w
42  JOIN locations l ON w.location_id = l.location_id
43  GROUP BY l.location_name
44  HAVING AVG(w.wait_minutes) > 120
45  ORDER BY avg_wait_time_minutes DESC;
46
```

What each part does

Table: Breakdown of the Full Query

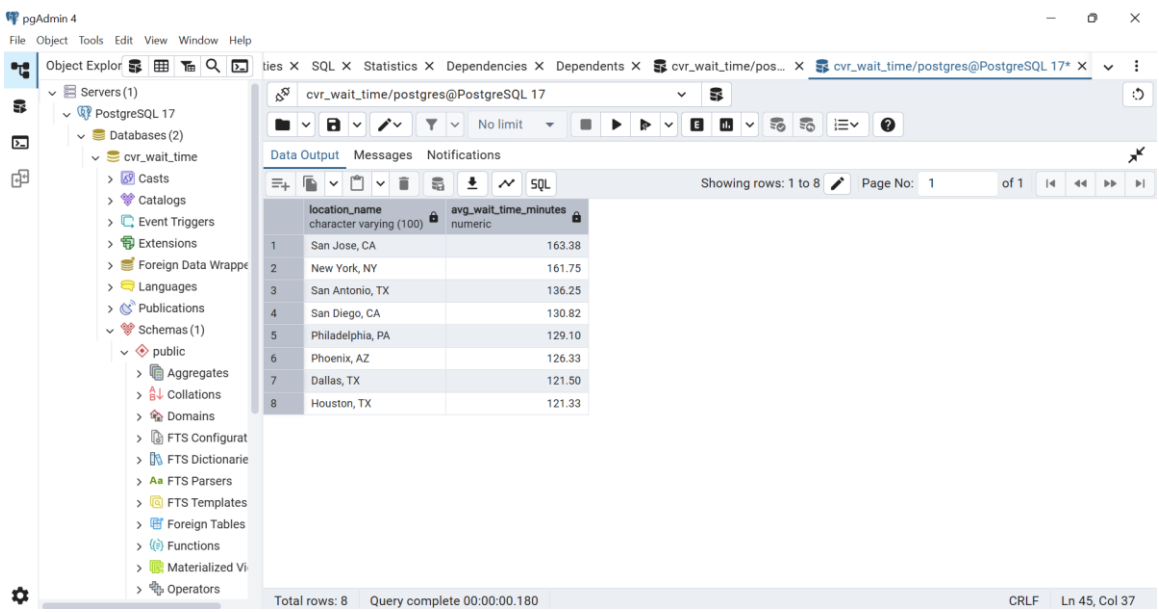
Section	Purpose
WITH wait_times AS (...)	Common Table Expression (CTE) to calculate individual patient wait times in minutes.
EXTRACT(EPOCH FROM interval)	Extracts total wait time in seconds from the interval.
/ 60 AS wait_minutes	Converts seconds to minutes.
WHERE check_in_time IS NOT NULL AND check_out_time IS NOT NULL	Ensures only complete patient visits are used (no NULLs).
SELECT l.location_name	Retrieves readable names of locations.
ROUND(AVG(w.wait_minutes), 2)	Calculates average wait time per location, rounded to 2 decimal places.
FROM wait_times w	Uses the CTE to access calculated wait times.
JOIN locations l ON w.location_id = l.location_id	Matches each wait time to its location name.
GROUP BY l.location_name	Aggregates results per location.
HAVING AVG(w.wait_minutes) > 120	Filters only those locations where average wait time is over 120 minutes.
ORDER BY avg_wait_time_minutes DESC	Sorts output from longest average wait to shortest.

PROBLEM LOCATION

Table: Explanation of Key SQL Functions and Clauses

SQL Function / Clause	What It Does
WITH ... AS	Declares a temporary result set (CTE) that can be reused in the main query.
EXTRACT(EPOCH FROM interval)	Converts time difference to total seconds (PostgreSQL-specific).
ROUND(expression, 2)	Rounds a number to 2 decimal places.
AVG(expression)	Calculates the average of all values in a group.
JOIN ... ON	Combines rows from two tables where the condition matches.
GROUP BY	Groups rows by one or more columns so aggregate functions (like AVG) can be used.
HAVING	Filters groups after aggregation (similar to WHERE, but for grouped data).
ORDER BY ... DESC	Sorts the result in descending order (largest to smallest).

Expected Output



DEPT – LEVEL BOTTLENECK

4.4 Department-Level Bottleneck Analysis

SQL Snippet:

```sql
WITH wait_times AS (
  SELECT
    patient_id,
    department_id,
    EXTRACT(EPOCH FROM (check_out_time - check_in_time)) / 60 AS wait_minutes
  FROM patients
  WHERE check_in_time IS NOT NULL AND check_out_time IS NOT NULL
)

SELECT
  d.department_name,
  ROUND(AVG(w.wait_minutes), 2) AS avg_wait_time_minutes
FROM wait_times w
JOIN departments d ON w.department_id = d.department_id
GROUP BY d.department_name
ORDER BY avg_wait_time_minutes DESC;
```

Total rows: 5 Query complete 00:00:00.098

Table: Step-by-Step Query Structure

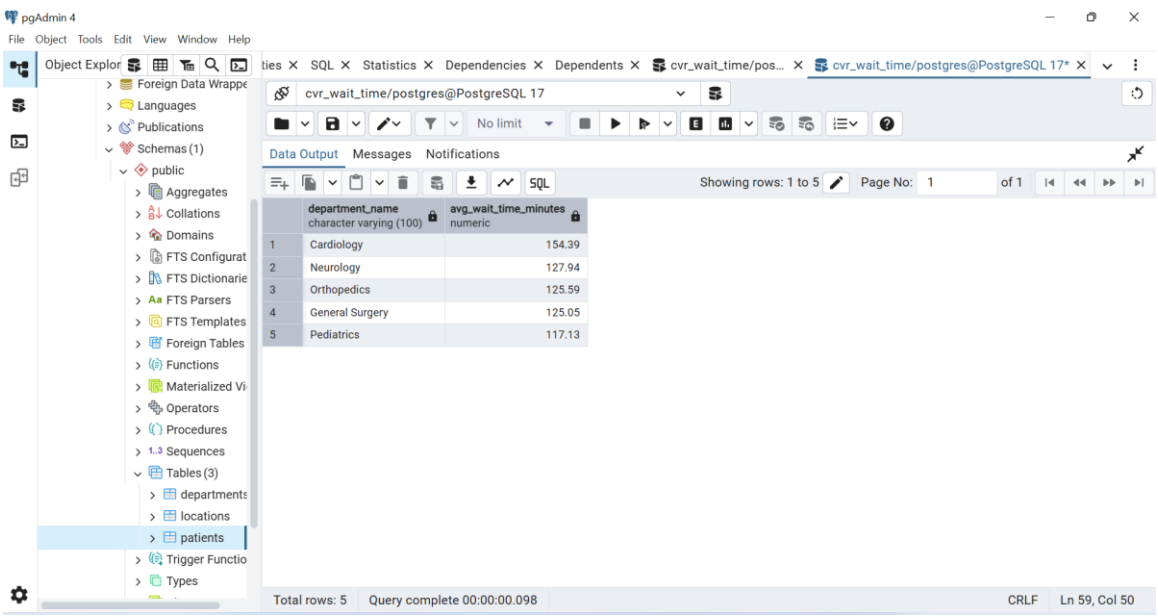
Section	Purpose
WITH wait_times AS (...)	Creates a temporary table (CTE) containing patient wait times and department IDs.
EXTRACT(EPOCH FROM interval)	Converts the interval between check-in and check-out into total seconds.
/ 60 AS wait_minutes	Converts seconds to minutes.
WHERE check_in_time IS NOT NULL AND check_out_time IS NOT NULL	Filters to only include complete visits.
SELECT d.department_name	Retrieves the name of each department.
ROUND(AVG(w.wait_minutes), 2)	Calculates the average wait time for each department, rounded to 2 decimals.
FROM wait_times w	Uses the CTE as input data.
JOIN departments d ON w.department_id = d.department_id	Links wait times to department names.
GROUP BY d.department_name	Aggregates wait times by department.
ORDER BY avg_wait_time_minutes DESC	Sorts output from highest to lowest average wait time.

DEPT – LEVEL BOTTLENECK

Table: Explanation of Functions and Clauses

SQL Function / Clause	Explanation
WITH ... AS	Declares a reusable temporary query result (CTE).
EXTRACT(EPOCH FROM ...)	Gets time duration in seconds from a timestamp difference.
/ 60	Converts seconds to minutes.
ROUND(..., 2)	Rounds to 2 decimal places for easier reading.
AVG(...)	Averages wait times within each group.
JOIN ... ON ...	Combines patient wait data with department names.
GROUP BY	Aggregates rows so that averages can be computed per department.
ORDER BY ... DESC	Shows departments with the worst wait times first.

Expected output



LOCATION WITH HIGH WAITS

4.5 KPI: % of Locations with High Wait Times

SQL Snippet:

```sql
---KPI: % of Locations With High Wait Times
WITH wait_times AS (
  SELECT location_id,
    EXTRACT(EPOCH FROM (check_out_time - check_in_time))/60 AS wait_minutes
  FROM patients
),
location_avg AS (
  SELECT location_id, AVG(wait_minutes) AS avg_wait_time
  FROM wait_times
  GROUP BY location_id
)
SELECT
  COUNT(*) FILTER (WHERE avg_wait_time > 120) AS high_wait_count,
  COUNT(*) AS total_locations,
  ROUND(COUNT(*) FILTER (WHERE avg_wait_time > 120) * 100.0 / COUNT(*), 2) AS percent_high_wait
FROM location_avg;
```

Table: Query Structure and Purpose

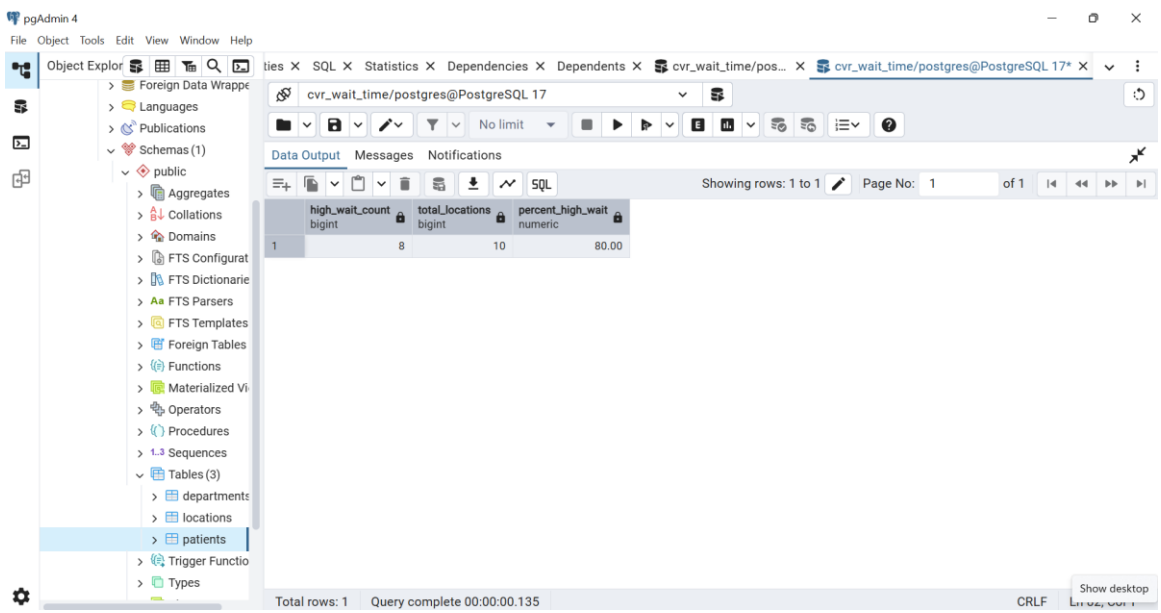
Section	Description
WITH wait_times AS (...)	Calculates individual patient wait times in minutes (per location_id).
EXTRACT(EPOCH FROM (check_out_time - check_in_time))/60 AS wait_minutes	Converts time difference from seconds to minutes.
location_avg AS (...)	Computes the average wait time per location.
SELECT COUNT(*) FILTER (WHERE avg_wait_time > 120)	Counts how many locations have high wait times (>120 mins).
COUNT(*) AS total_locations	Counts all locations represented in the data.
ROUND(... * 100.0 / COUNT(*), 2)	Calculates the percentage of high-wait locations, rounded to 2 decimals.

LOCATION WITH HIGH WAITS

Table: Explanation of SQL Functions and Clauses

Function / Clause	Explanation
WITH ... AS	Declares one or more reusable subqueries (CTEs).
EXTRACT(EPOCH FROM interval)	Gets duration in seconds between two timestamps.
/60	Converts seconds to minutes.
AVG()	Computes the average wait time per location.
GROUP BY location_id	Aggregates data for each unique location.
COUNT(*) FILTER (WHERE condition)	Counts rows that match a condition (PostgreSQL-specific syntax).
ROUND(value, 2)	Rounds a number to two decimal places.

Expected Output



% HIGH WAIT LOCATION

4.6 Monthly Trends of % of High-Wait Locations

SQL Snippet:

```sql
WITH wait_times AS (
  SELECT
    location_id,
    DATE_TRUNC('month', check_in_time) AS month,
    EXTRACT(EPOCH FROM (check_out_time - check_in_time)) / 60 AS wait_minutes
  FROM patients
  WHERE check_in_time IS NOT NULL AND check_out_time IS NOT NULL
),
location_monthly_avg AS (
  SELECT
    location_id,
    month,
    AVG(wait_minutes) AS avg_wait_time
  FROM wait_times
  GROUP BY location_id, month)
SELECT
  month,
  COUNT(*) FILTER (WHERE avg_wait_time > 120) AS high_wait_count,
  COUNT(*) AS total_locations,
  ROUND(COUNT(*) FILTER (WHERE avg_wait_time > 120) * 100.0 / COUNT(*), 2) AS percent_high_wait
FROM location_monthly_avg
GROUP BY month
ORDER BY month;
```

Total rows: 3 | Query complete 00:00:00.133 | CRLF | L

Section	Purpose
DATE_TRUNC('month', check_in_time)	Extracts the month (e.g., 2024-06-01) from each check-in timestamp.
WITH wait_times AS (...)	Calculates per-patient wait time and attaches the month.
location_monthly_avg AS (...)	Aggregates the average wait time per location per month.
SELECT month, ...	Begins final KPI computation.
COUNT(*) FILTER (WHERE avg_wait_time > 120)	Counts how many locations in each month had high wait times.
COUNT(*)	Counts all reporting locations that month.
ROUND(... * 100.0 / COUNT(*), 2)	Calculates the percentage of problem locations per month.
GROUP BY month	Aggregates by each month.
ORDER BY month	Ensures chronological output.