Lecture 02: Being Selective with SQL

DATA 351: Data Management with SQL

Lucas P. Cordova, Ph.D.

2026-01-14

This lecture covers the SELECT, WHERE, ORDER BY, and LIMIT clauses of SQL.

Table of contents

1 Announcements

1.0.1 Today's Agenda

- We will cover all the SQL you need for the assignment today
- Topics:
  - SELECT and column aliases
  - WHERE clause and filtering
  - Combining conditions with AND/OR
  - ORDER BY for sorting
  - LIMIT for restricting results

1.0.2 About Homework 1

- Uses the **Stack Overflow 2025 Developer Survey** dataset
- 10 SQL queries exploring developer demographics, AI attitudes, and work preferences
- Autograded through CodeGrade
- Each query builds on today's concepts

1.1 Review

1.1.1 Quick Review

What do you think this query will return?

```
SELECT * FROM survey25;
```

. . .

All columns and all rows from the `survey25` table.

1.1.2 Review Question

Which SQL keyword do you think you would use to retrieve only the unique values in a column?

A. `UNIQUE`

B. `SINGLE`

C. `DISTINCT`

D. `DIFFERENT`

. . .

Answer: C - `DISTINCT` returns only unique values

1.2 Selecting Columns

1.2.1 The SELECT Statement

The most fundamental SQL operation is selecting data from a table.

Basic syntax:

```
SELECT column1, column2, column3
FROM table_name;
```

This returns the specified columns for every row in the table.

1.2.2 Selecting All Columns

Use the asterisk `*` to select all columns:

```
SELECT *
FROM survey25;
```

Useful for exploration, but in practice you should specify only the columns you need.

1.2.3 Selecting Specific Columns

For the developer survey, we might want just a few columns:

```
SELECT response_id, age, country
FROM survey25;
```

This returns only those three columns for all respondents.

1.2.4 Why Be Selective?

Selecting only needed columns has benefits:

- **Performance**: Less data to transfer and process
- **Clarity**: Results are easier to read and understand
- **Memory**: Reduces memory usage for large tables

1.2.5 Column Aliases with AS

Sometimes column names are unclear or too long. Use `AS` to rename them in your output:

```
SELECT response_id AS id,
       age AS age_range,
       country AS location
FROM survey25;
```

The original table is unchanged; only the output uses the new names.

1.2.6 Alias Syntax Options

Both of these work in PostgreSQL:

```
-- With AS keyword (preferred for clarity)
SELECT response_id AS id FROM survey25;

-- Without AS keyword (also valid)
SELECT response_id id FROM survey25;
```

Use `AS` for readability.

1.2.7 When to Use Aliases

Aliases are helpful when:

- Column names are cryptic (`i_cor_pm` becomes `role_type`)
- You want more descriptive output (`converted_comp_yearly` becomes `salary_usd`)
- Column names contain special characters
- You need to distinguish columns from different tables (joins)

1.2.8 Think-Pair-Share: Column Selection

Think (1 min): Write a query to select the `dev_type` and `remote_work` columns from `survey25`, renaming them to `job_title` and `work_style`.

Pair (2 min): Compare your answer with a neighbor.

Share: Let's see some solutions!

. . .

```
SELECT dev_type AS job_title,
       remote_work AS work_style
FROM survey25;
```

1.3 Filtering with WHERE

1.3.1 Why Filter?

Real databases have thousands or millions of rows. You rarely want all of them.

The `WHERE` clause filters rows based on conditions:

```
SELECT column1, column2
FROM table_name
WHERE condition;
```

1.3.2 Basic Equality Filter

Find all remote workers:

```
SELECT response_id, dev_type, country
FROM survey25
WHERE remote_work = 'Remote';
```

Note: String values must be in **single quotes**.

1.3.3 Comparison Operators

SQL provides several comparison operators:

Operator	Meaning
`=`	Equal to
`<>` or `!=`	Not equal to
`>`	Greater than
`<`	Less than
`>=`	Greater than or equal
`<=`	Less than or equal

1.3.4 Numeric Comparisons

Find developers with more than 15 years of work experience:

```
SELECT response_id, work_exp, dev_type
FROM survey25
WHERE work_exp > 15;
```

No quotes needed for numbers.

1.3.5 String Comparisons

Find developers in Canada:

```
SELECT response_id, country, dev_type
FROM survey25
WHERE country = 'Canada';
```

String comparisons are **case-sensitive** by default.

1.3.6 Understanding Check

What will this query return?

```
SELECT response_id, age
FROM survey25
WHERE age = '18-24 years old';
```

A. All developers aged exactly 18 to 24

B. All developers with age column containing that exact string

C. An error because age should be a number

D. Nothing because no one is exactly that age

. . .

Answer: B - The `age` column contains text values like '18-24 years old'

1.3.7 The Problem with NULL

`NULL` represents missing or unknown data. It behaves strangely:

```
-- This will NOT work as expected!
SELECT * FROM survey25
WHERE converted_comp_yearly = NULL;
```

This returns **zero rows** because `NULL` comparisons always return `NULL`, not true or false.

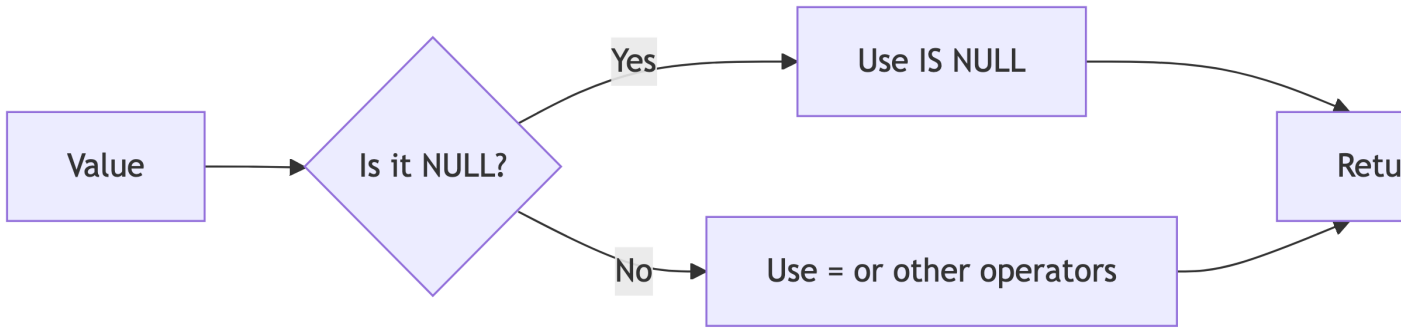
1.3.8 IS NULL and IS NOT NULL

To check for NULL values, use special syntax:

```sql
-- Find rows WITH salary data
SELECT response_id, converted_comp_yearly
FROM survey25
WHERE converted_comp_yearly IS NOT NULL;

-- Find rows WITHOUT salary data
SELECT response_id, converted_comp_yearly
FROM survey25
WHERE converted_comp_yearly IS NULL;
```

1.3.9 NULL Visualization



1.3.10 Think-Pair-Share: Filtering

Think (1 min): Write a query to find all developers who believe AI is a threat to their job (`ai_threat = 'Yes'`).

Pair (2 min): Share with a neighbor.

Share: What columns would be interesting to include?

. . .

```sql
SELECT response_id, age, dev_type, ai_threat
FROM survey25
WHERE ai_threat = 'Yes';
```

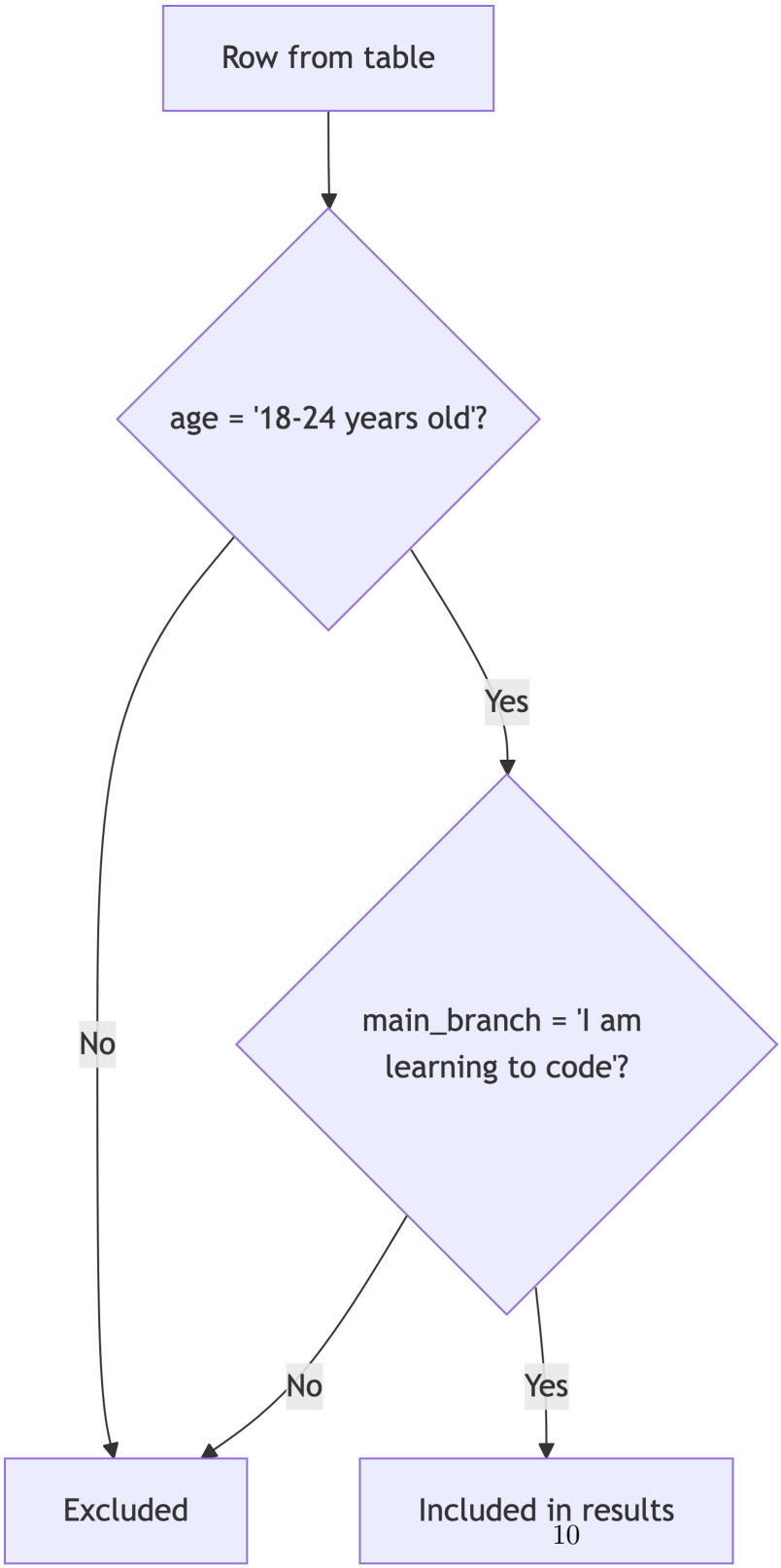
1.4 Combining Conditions

1.4.1 Multiple Conditions with AND

Often you need to filter on multiple criteria. Use `AND` when **all** conditions must be true:

```
SELECT response_id, age, main_branch
FROM survey25
WHERE age = '18-24 years old'
  AND main_branch = 'I am learning to code';
```

1.4.2 AND Logic Visualized



1.4.3 Multiple Conditions with OR

Use `OR` when **any** condition being true is sufficient:

```
SELECT response_id, remote_work
FROM survey25
WHERE remote_work = 'Remote'
   OR remote_work = 'Hybrid';
```

1.4.4 AND vs OR

Operator	Behavior	Use When
`AND`	All conditions must be true	Narrowing results
`OR`	At least one must be true	Expanding results

1.4.5 Order of Operations

`AND` has higher precedence than `OR`. Use parentheses to be explicit:

```
-- Without parentheses (AND evaluated first)
SELECT * FROM survey25
WHERE industry = 'Software Development'
  AND ai_threat = 'Yes'
   OR work_exp > 20;

-- With parentheses (clearer intent)
SELECT * FROM survey25
WHERE industry = 'Software Development'
  AND (ai_threat = 'Yes' OR work_exp > 20);
```

1.4.6 Real Example: AI Enthusiasts

Find developers who use AI daily AND have very favorable AI sentiment:

```
SELECT response_id, dev_type, years_code AS coding_experience
FROM survey25
WHERE ai_select = 'Yes, I use AI tools daily'
  AND ai_sent = 'Very favorable';
```

1.4.7 Understanding Check

How many conditions must be true for a row to be returned?

```
SELECT * FROM survey25
WHERE country = 'United States'
  AND work_exp > 10
  AND remote_work = 'Remote';
```

A. At least one

B. At least two

C. All three

D. Exactly one

. . .

Answer: C - With `AND`, all conditions must be true

1.4.8 Think-Pair-Share: Combined Filters

Think (2 min): Write a query to find full-stack developers (`dev_type = 'Developer, full-stack'`) with more than 15 years of experience.

Pair (2 min): Compare queries with your neighbor.

. . .

```
SELECT response_id, work_exp, org_size
FROM survey25
WHERE dev_type = 'Developer, full-stack'
  AND work_exp > 15;
```

1.5 Sorting with ORDER BY

1.5.1 Why Sort Results?

Without explicit ordering, SQL returns rows in an **unpredictable** order.

Use `ORDER BY` to sort your results:

```
SELECT column1, column2
FROM table_name
ORDER BY column1;
```

1.5.2 Ascending Order (Default)

Sort developers by years of experience (lowest first):

```
SELECT response_id, work_exp, dev_type
FROM survey25
WHERE work_exp IS NOT NULL
ORDER BY work_exp;
```

`ASC` (ascending) is the default and can be omitted.

1.5.3 Descending Order

Sort by salary (highest first):

```
SELECT response_id, country, converted_comp_yearly AS salary_usd
FROM survey25
WHERE converted_comp_yearly IS NOT NULL
ORDER BY converted_comp_yearly DESC;
```

1.5.4 ORDER BY with Aliases

You can order by a column alias:

```
SELECT response_id,
       converted_comp_yearly AS salary_usd
FROM survey25
WHERE converted_comp_yearly IS NOT NULL
ORDER BY salary_usd DESC;
```

Or use the original column name. Both work.

1.5.5 Sorting Text Columns

Text is sorted alphabetically:

```
SELECT DISTINCT country
FROM survey25
ORDER BY country;
```

Sorting depends on the database **collation** (locale settings).

1.5.6 NULL Values in Sorting

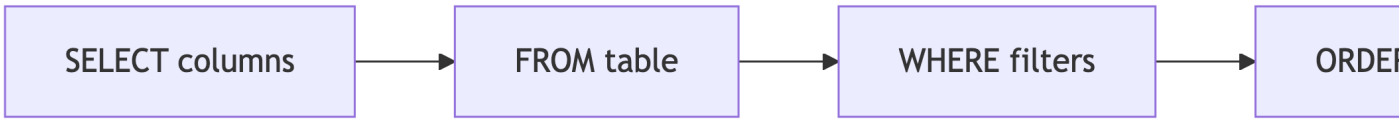
By default in PostgreSQL:

- `ASC`: NULLs appear **last**
- `DESC`: NULLs appear **first**

You can control this:

```
ORDER BY column ASC NULLS FIRST
ORDER BY column DESC NULLS LAST
```

1.5.7 Sorting Flow



ORDER BY is processed **after** filtering.

1.6 Limiting Results

1.6.1 The LIMIT Clause

Sometimes you only want a few rows. Use `LIMIT`:

```
SELECT response_id, country, converted_comp_yearly AS salary_usd
FROM survey25
WHERE converted_comp_yearly IS NOT NULL
ORDER BY converted_comp_yearly DESC
LIMIT 10;
```

This returns only the **top 10** highest earners.

1.6.2 LIMIT Position

`LIMIT` always comes **last** in the query:

```
SELECT columns
FROM table
WHERE conditions
ORDER BY column
LIMIT n;
```

1.6.3 Common LIMIT Use Cases

LIMIT is useful for:

- Finding top N or bottom N results
- Sampling data to understand table structure
- Pagination (showing results in pages)
- Testing queries on large tables

1.6.4 Top 5 Happiest Developers

```
SELECT response_id,
       job_sat AS satisfaction_score,
       dev_type,
       remote_work AS work_arrangement
FROM survey25
WHERE job_sat IS NOT NULL
ORDER BY job_sat DESC
LIMIT 5;
```

1.6.5 Understanding Check

What does this query return?

```
SELECT response_id, work_exp
FROM survey25
ORDER BY work_exp ASC
LIMIT 3;
```

A. The 3 most experienced developers

B. The 3 least experienced developers

C. 3 random developers

D. An error

. . .

Answer: B - ASC sorts lowest first, LIMIT 3 takes the first three

1.7 Putting It All Together

1.7.1 Query Order of Execution

When SQL runs your query, it processes clauses in this order:

1. `FROM` - Which table?
2. `WHERE` - Which rows match?
3. `SELECT` - Which columns?
4. `ORDER BY` - How to sort?
5. `LIMIT` - How many rows?

1.7.2 Query Writing Order

You **write** queries in this order:

```
SELECT columns      -- 1st: What to show
FROM table          -- 2nd: Where from
WHERE conditions    -- 3rd: Which rows
ORDER BY column     -- 4th: How to sort
LIMIT n;            -- 5th: How many
```

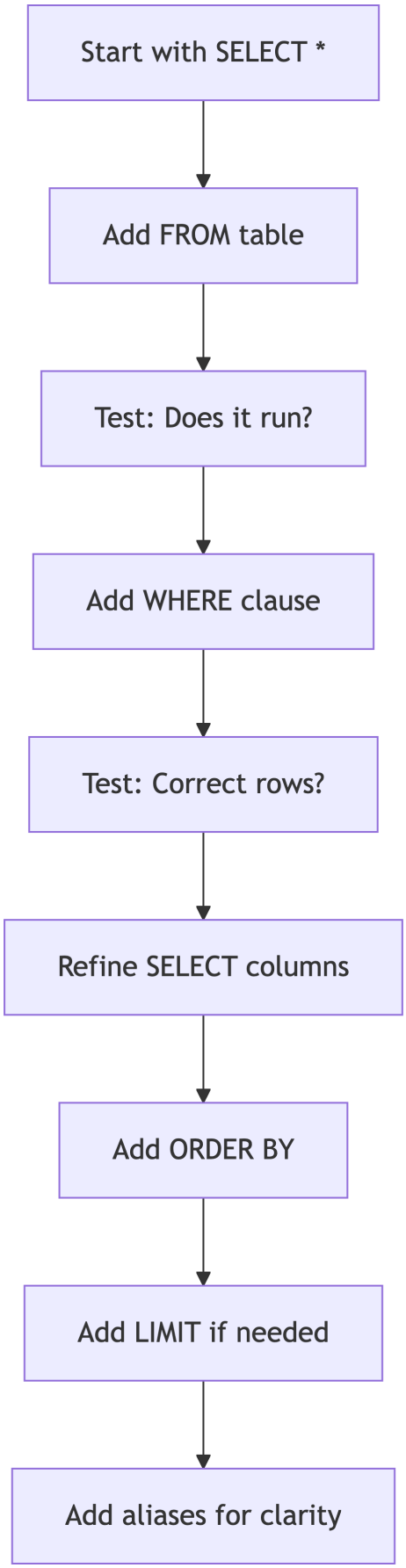
1.7.3 Complete Example

Find the top 10 highest-paid developers with valid salary data:

```
SELECT response_id,
       country,
       dev_type,
       converted_comp_yearly AS salary_usd
FROM survey25
WHERE converted_comp_yearly IS NOT NULL
```

```
7  ORDER BY converted_comp_yearly DESC
8  LIMIT 10;
```

1.7.4 Query Building Strategy



1.7.5 Think-Pair-Share: Complete Query

Think (3 min): Write a query to find developers with less than 3 years of experience at companies with 10,000+ employees. Order by experience (lowest first).

Pair (2 min): Compare with your neighbor.

. . .

```
SELECT response_id, age, ed_level, work_exp, org_size
FROM survey25
WHERE work_exp < 3
  AND org_size = '10,000 or more employees'
ORDER BY work_exp ASC;
```

1.8 Practice Problems

1.8.1 Practice 1: Stack Overflow Power Users

Write a query to find developers who:

- Visit Stack Overflow multiple times per day
- Definitely consider themselves community members

Show their response_id, years_code, and relevant columns with good aliases.

. . .

```
SELECT response_id,
       years_code,
       so_visit_freq AS visit_frequency,
       so_comm AS community_member
FROM survey25
WHERE so_visit_freq = 'Multiple times per day'
  AND so_comm = 'Yes, definitely';
```

1.8.2 Practice 2: AI Skeptics

Write a query to find developers in Software Development who believe AI threatens their job.

. . .

```sql
SELECT response_id,
       age,
       dev_type,
       ai_threat AS ai_job_threat
FROM survey25
WHERE industry = 'Software Development'
  AND ai_threat = 'Yes';
```

1.8.3 Practice 3: Happy Remote Workers

Write a query to find the 5 happiest developers. Exclude NULL satisfaction scores.

. . .

```sql
SELECT response_id,
       job_sat AS satisfaction_score,
       dev_type,
       remote_work AS work_arrangement
FROM survey25
WHERE job_sat IS NOT NULL
ORDER BY job_sat DESC
LIMIT 5;
```

1.9 Common Mistakes

1.9.1 Mistake 1: Wrong Quotes

```sql
-- WRONG: Double quotes for strings
SELECT * FROM survey25 WHERE country = "Canada";

-- CORRECT: Single quotes for strings
SELECT * FROM survey25 WHERE country = 'Canada';
```

Double quotes are for identifiers (column/table names), not string values.

1.9.2 Mistake 2: NULL Comparisons

```sql
-- WRONG: Using = with NULL
SELECT * FROM survey25 WHERE job_sat = NULL;

-- CORRECT: Using IS NULL
SELECT * FROM survey25 WHERE job_sat IS NULL;
```

```sql

-- CORRECT: Using IS NOT NULL
SELECT * FROM survey25 WHERE job_sat IS NOT NULL;
```

1.9.3 Mistake 3: Case Sensitivity

```sql
-- May return nothing if data is 'Remote'
SELECT * FROM survey25 WHERE remote_work = 'remote';

-- Exact match required
SELECT * FROM survey25 WHERE remote_work = 'Remote';
```

String comparisons are case-sensitive. Check your data values.

1.9.4 Mistake 4: Missing Quotes

```sql
-- WRONG: String without quotes
SELECT * FROM survey25 WHERE country = Canada;

-- CORRECT: String with quotes
SELECT * FROM survey25 WHERE country = 'Canada';

-- CORRECT: Numbers don't need quotes
SELECT * FROM survey25 WHERE work_exp > 10;
```

1.9.5 Mistake 5: Wrong Clause Order

```sql
-- WRONG: LIMIT before ORDER BY
SELECT * FROM survey25
LIMIT 10
ORDER BY work_exp;

-- CORRECT: ORDER BY before LIMIT
SELECT * FROM survey25
ORDER BY work_exp
LIMIT 10;
```

1.10 Summary

1.10.1 Key Concepts

Today we covered:

- `SELECT` specifies which columns to return
- `AS` creates column aliases for clearer output
- `WHERE` filters rows based on conditions
- `AND`/`OR` combine multiple conditions
- `IS NULL` and `IS NOT NULL` handle missing data
- `ORDER BY` sorts results (ASC or DESC)
- `LIMIT` restricts the number of rows returned

1.10.2 Query Template

```sql
SELECT column1,
       column2 AS alias
FROM table_name
WHERE condition1
  AND condition2
ORDER BY column1 DESC
LIMIT n;
```

1.10.3 For Homework 1

You now have all the tools needed:

- 10 queries using the Stack Overflow survey
- Each builds on these concepts
- Test your queries before submitting
- Pay attention to exact column aliases
- String values must match exactly

1.10.4 Next Class

We will cover:

- More advanced filtering with `BETWEEN` and `IN`
- Pattern matching with `LIKE` and `ILIKE`
- Working with dates and times

Read Chapter 3 before next class.