Abdou's SQL Notes

Week 2: SELECT & WHERE

MIS 3220 - Fall 2026

September 8 & 10, 2026

This Week in Class

- **Tuesday, Sep 8:** The SELECT statement. Choosing columns, renaming them with AS, doing calculations right in the query, joining text together, removing duplicates with DISTINCT, and controlling result size with LIMIT.
- **Thursday, Sep 10:** Filtering with WHERE. Comparison operators, BETWEEN, IN, NULL handling, pattern matching with LIKE, and combining conditions with AND, OR, and NOT. Lab 2 is worked in class, with help in the room, and the report is due before next Tuesday's class at 2:00 PM.

The Notes Are the Coverage Guarantee

The notes are the record. If it is here, it is fair game for labs, exams, and interviews.

Every example in these notes runs on this week's database. Open `northshore_coffee.db` from the Lab 2 files and run `.schema` first. It has four tables: `shops` (8 rows), `menu_items` (45), `employees` (25), and `orders` (204). The comment under each query tells you what came back when I ran it, so you can check your screen against the notes.

The SELECT Statement

The `SELECT` statement retrieves data from a database. It is by far the most commonly used SQL command.

Basic Syntax

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

The Asterisk (*) - Select All Columns

The asterisk (`*`) is a wildcard meaning "all columns":

```
-- Get ALL columns from the menu_items table
SELECT * FROM menu_items;
-- 45 rows, 8 columns; first row:  1, House Blend Coffee, Coffee, Medium, 3.5, 5,
0, ...

-- Get only specific columns
SELECT name, price FROM menu_items;
-- 45 rows, 2 columns; first row:  House Blend Coffee, 3.5

-- Get multiple specific columns from a different table
SELECT id, name, city, has_drive_thru FROM shops;
-- 8 rows; first row:  1, Northshore Canal Park, Duluth, 1
```

Best Practice: In real applications, avoid `SELECT *` and specify only the columns you need. This makes your queries faster and clearer.

Column Aliases with AS

Use `AS` to give columns a temporary name (alias) in the output:

```
-- Rename columns in output
SELECT name AS shop_name, city AS town FROM shops;
-- 8 rows; column headers read shop_name and town

-- Useful for calculated columns
SELECT name, price, price * 1.07 AS price_with_tax FROM menu_items;
-- 45 rows; first row:  House Blend Coffee, 3.5, 3.745

-- Use double quotes for aliases with spaces
SELECT name AS "Shop Name", phone AS "Front Desk" FROM shops;
-- 8 rows; headers read Shop Name and Front Desk
```

SQLite will also accept single quotes around an alias, but double quotes are the standard SQL form and the one PostgreSQL insists on, so get in the habit now.

Using SELECT for Calculations

SQL can perform arithmetic operations directly. You can even use SELECT without a table:

```
-- Basic arithmetic (no table needed)
SELECT 10 + 5; -- Returns 15
SELECT 100 - 25; -- Returns 75
SELECT 20 * 3; -- Returns 60
SELECT 100 / 4; -- Returns 25
SELECT 17 % 5; -- Returns 2 (modulo/remainder)

-- Combine multiple operations
SELECT (100 + 50) * 2; -- Returns 300

-- Calculate with column values:  a happy-hour price at 10 percent off
SELECT name, price, price * 0.9 AS happy_hour_price FROM menu_items;
-- 45 rows; first row:  House Blend Coffee, 3.5, 3.15

-- What would a one-dollar raise look like?
SELECT first_name, hourly_wage, hourly_wage + 1.00 AS after_raise FROM employees;
-- 25 rows; first row:  Sarah, 22.5, 23.5
```

String Concatenation

In SQLite, use `||` to join strings together:

```
-- Combine two columns with a separator
SELECT name || ', ' || city AS shop_and_city FROM shops;
-- 8 rows; first row:  Northshore Canal Park, Duluth

-- Add text to column values
SELECT 'Item:  '  || name FROM menu_items;
-- 45 rows; first row:  Item:  House Blend Coffee

-- Create formatted output
SELECT name || ' costs $' || price AS label FROM menu_items;
-- 45 rows; first row:  House Blend Coffee costs $3.5
```

DISTINCT - Removing Duplicates

Use `DISTINCT` to return only unique values:

```
-- List the category of every menu item (with duplicates)
SELECT category FROM menu_items;
-- 45 rows:  Coffee, Coffee, Coffee, Coffee, Coffee, ...

-- List unique categories only (no duplicates)
SELECT DISTINCT category FROM menu_items;
-- 6 rows:  Coffee, Tea, Smoothie, Pastry, Sandwich, Seasonal

-- DISTINCT on multiple columns:  unique category-and-size combinations
SELECT DISTINCT category, size FROM menu_items;
-- 11 rows; the food rows show a blank size because it is NULL
```

LIMIT and OFFSET - Controlling Result Size

Use `LIMIT` to restrict how many rows are returned:

```
-- Get only the first 5 rows of a big table
SELECT * FROM orders LIMIT 5;
-- 5 rows; first row:  1, 1, 2, 2026-01-13, 12.5, Credit, 2.0

-- Get only 1 row (useful for checking if data exists)
SELECT * FROM shops LIMIT 1;
-- 1 row:  Northshore Canal Park

-- Skip the first 10 rows, then get 5 rows
SELECT id, name, price FROM menu_items LIMIT 5 OFFSET 10;
-- 5 rows, ids 11 to 15; first row:  11, Mocha, 6.0

-- Pagination:  Page 1 (rows 1-10)
SELECT id, name FROM menu_items LIMIT 10 OFFSET 0;
-- 10 rows, ids 1 to 10

-- Pagination:  Page 2 (rows 11-20)
SELECT id, name FROM menu_items LIMIT 10 OFFSET 10;
-- 10 rows, ids 11 to 20
```

LIMIT without ORDER BY: When you use LIMIT without ORDER BY, the rows returned are essentially random (based on how the database stored them). In this small file they happen to come back in id order, but nothing guarantees that. Next week we add ORDER BY, which makes LIMIT predictable.

Filtering with WHERE

The `WHERE` clause filters rows based on conditions. Only rows that match the condition are returned.

Comparison Operators

Operator	Meaning	Example
`=`	Equal to	`WHERE price = 3.50`
`<>` or `!=`	Not equal to	`WHERE city <> 'Duluth'`
`>`	Greater than	`WHERE calories > 400`
`<`	Less than	`WHERE hourly_wage < 15`
`>=`	Greater than or equal	`WHERE price >= 5`
`<=`	Less than or equal	`WHERE total_amount <= 5`

```
-- Find exact matches
SELECT name, size, price FROM menu_items WHERE size = 'Large';
-- 19 rows; first row:  House Blend Coffee, Large, 4.25

-- Find shops that are NOT in Duluth
SELECT name, city FROM shops WHERE city <> 'Duluth';
-- 5 rows:  Two Harbors, Grand Marais, Silver Bay, Tofte, Beaver Bay

-- Find the heavy items
SELECT name, calories FROM menu_items WHERE calories > 400;
-- 5 rows:  Cinnamon Roll 450, Turkey Club 520, Grilled Cheese 440, Ham & Swiss
490, Caprese 410

-- Find the small orders, five dollars or less
SELECT id, total_amount FROM orders WHERE total_amount <= 5;
-- 11 rows; first row:  4, 3.5
```

Text Values Need Quotes

Always wrap text values in single quotes. Numbers do not need quotes:

```
-- Correct:  text in single quotes
SELECT * FROM employees WHERE position = 'Baker'; -- 2 rows:  Grace Kim, Noah Berg
SELECT * FROM shops WHERE name = 'Northshore Downtown'; -- 1 row, shop id 2

-- Correct:  numbers without quotes
SELECT * FROM menu_items WHERE price = 3.50;          -- 2 rows:  House Blend Coffee
(Medium), Banana Bread
SELECT * FROM shops WHERE id = 5; -- 1 row:  Northshore Grand Marais

-- WRONG: missing quotes around text
SELECT * FROM employees WHERE position = Baker;
-- Error:  no such column:  Baker.  SQLite read Baker as a column name.
```

BETWEEN - Range of Values

`BETWEEN` checks if a value falls within a range (inclusive of both endpoints):

```
-- Items with 100 to 200 calories (inclusive)
SELECT name, calories FROM menu_items WHERE calories BETWEEN 100 AND 200;
-- 5 rows:  Vanilla Latte 190, Flat White 170, Chai Tea Latte 200, London Fog 180,
Spiced Apple Cider Tea 120

-- Same as writing (note that 200 is included):
SELECT name, calories FROM menu_items WHERE calories >= 100 AND calories <= 200;
-- 5 rows, identical

-- BETWEEN works with dates stored as YYYY-MM-DD text:  everyone hired in 2023
SELECT first_name, last_name, hire_date FROM employees
WHERE hire_date BETWEEN '2023-01-01' AND '2023-12-31';
-- 6 rows; first row:  Emily, Larson, 2023-08-20

-- NOT BETWEEN - outside the range
SELECT name, calories FROM menu_items WHERE calories NOT BETWEEN 100 AND 200;
-- 34 rows.  Notice 5 + 34 = 39, not 45:  the 6 items with a NULL calorie count
match neither test.
```

IN - Matching a List of Values

`IN` checks if a value matches any value in a list:

```
-- Find everyone in a supervisory role
SELECT first_name, last_name, position FROM employees
WHERE position IN ('Manager', 'Shift Lead');
-- 12 rows; first row:  Sarah, Johnson, Manager

-- Same as writing:
SELECT first_name, last_name, position FROM employees
WHERE position = 'Manager' OR position = 'Shift Lead';
-- 12 rows, identical

-- Find specific shop IDs
SELECT * FROM shops WHERE id IN (1, 4, 7);
-- 3 rows:  Canal Park, Two Harbors, Tofte

-- NOT IN - exclude these values
SELECT name, category FROM menu_items WHERE category NOT IN ('Coffee', 'Tea');
-- 22 rows; first row:  Mango Smoothie, Smoothie
```

IS NULL / IS NOT NULL

NULL represents missing, unknown, or undefined data. It is not the same as zero or an empty string.

```
-- Find shops without a phone number on file
SELECT name, phone FROM shops WHERE phone IS NULL;
-- 3 rows:  Silver Bay, Tofte, Beaver Bay (the three newest shops)

-- Find shops that HAVE a phone number
SELECT name, phone FROM shops WHERE phone IS NOT NULL;
-- 5 rows; first row:  Northshore Canal Park, 218-555-0101

-- Find orders with no tip recorded
SELECT id, total_amount, tip_amount FROM orders WHERE tip_amount IS NULL;
-- 47 rows; first row:  3, 8.25, (blank)
```

`= NULL` runs without an error and returns 0 rows. Use `IS NULL`. The same goes for `<> NULL`; use `IS NOT NULL`.

`SELECT * FROM shops WHERE phone = NULL;` (wrong: runs, returns 0 rows)
`SELECT * FROM shops WHERE phone IS NULL;` (correct: returns the 3 shops)

LIKE - Pattern Matching

LIKE allows you to search for patterns in text using wildcards:

Wildcard	Meaning	Example
%	Zero or more characters	'J%' matches Jake, Jasmine, Jack
_	Exactly one character	'J_ck' matches Jack, Jock, Jeck

```
-- First names starting with 'J'
SELECT first_name FROM employees WHERE first_name LIKE 'J%';
-- 3 rows:  Jake, Jasmine, Jack

-- Last names ending with 'son'
SELECT last_name FROM employees WHERE last_name LIKE '%son';
-- 9 rows:  Johnson, Anderson, Larson, Peterson, Olson, Johansson, Thompson,
Gustafson, Swenson

-- Menu items containing 'Mocha' anywhere
SELECT name FROM menu_items WHERE name LIKE '%Mocha%';
-- 2 rows:  Mocha, Peppermint Mocha

-- Staff using a Gmail address for company business
SELECT first_name, email FROM employees WHERE email LIKE '%@gmail.com';
-- 3 rows:  Tyler, Ryan, Aiden
```

Using the underscore (_) wildcard:

```
-- First names with exactly 4 characters
SELECT first_name FROM employees WHERE first_name LIKE '____';
-- 8 rows:  Jake, Anna, Noah, Ryan, Maya, Liam, Ella, Jack

-- Names starting with 'J', any second character, then 'ck'
SELECT first_name FROM employees WHERE first_name LIKE 'J_ck';
-- 1 row:  Jack

-- Combine both wildcards:  'a' as the second character
SELECT first_name FROM employees WHERE first_name LIKE '_a%';
-- 9 rows:  Sarah, Jake, Marcus, Jasmine, Daniel, Maya, Jack, Hannah, Sam
```

NOT LIKE - Excluding patterns:

```
-- First names NOT starting with 'J'
SELECT first_name FROM employees WHERE first_name NOT LIKE 'J%';
-- 22 rows (25 employees minus the 3 J names)

-- Menu items whose name does not contain 'Coffee'
SELECT name FROM menu_items WHERE name NOT LIKE '%Coffee%';
-- 43 rows; only the two House Blend Coffee rows are excluded
```

AND / OR / NOT - Combining Conditions

Use logical operators to combine multiple conditions:

```
-- AND: Both conditions must be true
SELECT name, category, price FROM menu_items WHERE category = 'Tea' AND price < 4;
-- 2 rows:  Green Tea 3.25, Earl Grey 3.25

-- OR: At least one condition must be true
SELECT name, city FROM shops WHERE city = 'Duluth' OR city = 'Two Harbors';
-- 4 rows:  three Duluth shops plus Two Harbors

-- NOT: Reverses the condition
SELECT name, city FROM shops WHERE NOT city = 'Duluth';
-- 5 rows, the same five as city <> 'Duluth'

-- Combining AND and OR (use parentheses)
SELECT name, category, price FROM menu_items
WHERE (category = 'Tea' OR category = 'Pastry') AND price >= 4.5;
-- 7 rows:  five teas and two pastries, all at 4.50 or more

-- The SAME query without parentheses gives a different answer
SELECT name, category, price FROM menu_items
WHERE category = 'Tea' OR category = 'Pastry' AND price >= 4.5;
-- 9 rows:  ALL seven teas (including the 3.25 ones) plus the two pastries

-- Complex example
SELECT id, shop_id, total_amount, payment_method FROM orders
WHERE (payment_method = 'Credit' OR payment_method = 'Debit')
AND total_amount > 20
AND tip_amount IS NOT NULL;
-- 20 rows; first row:  12, 1, 22.75, Credit
```

Important: When mixing AND and OR, always use parentheses to make your intent clear. AND has higher precedence than OR, which is why the two Tea-or-Pastry queries above return 7 and 9 rows from the same words. Without the parentheses, SQL read it as "Tea, or (Pastry and 4.50 or more)."

In the Field: Answering the Daily Questions

Most questions that land on an analyst's desk are a SELECT with a WHERE clause: which shops have no phone number on file, which staff use a Gmail address for company business. Managers act on the answer without checking the query, so the condition has to be right, including the NULLs (look again at the BETWEEN example, where 6 rows dropped out with no error).

This Week's Lab

- **Lab 2: SELECT and WHERE (25 points).** You are the analyst for Northshore Coffee Co., a fictional chain of eight coffee shops along Lake Superior's North Shore, from Canal Park up to Grand Marais. The owner, Maren Josephson, has a list of questions about her shops, her menu, and her staff, and you answer each one with a filtered query against the same `northshore_coffee.db` used in these notes. It is worked in class Thursday, with help in the room, and the report is due before next Tuesday's class, at 2:00 PM.

Reading and Practice

- **Reading:** Painter-Wakefield, *A Practical Introduction to Databases*. Read the chapters on retrieving data with SELECT and on expressions and filtering with WHERE. Exact chapter locations are linked on Canvas.
- **Practice pack:** Practice Set W2 is posted with these notes on Canvas, ungraded with answers, all on `northshore_coffee.db`. Tuesday's quiz samples it, and it is your best exam preparation.

MIS 3220 - Database Management and Design
Fall 2026 - University of Minnesota Duluth