Pre-Lab 2

SELECT & WHERE

MIS 3220 - Fall 2026

Read before class on Tuesday, September 8. Plan on 15 to 20 minutes.

How pre-labs work

This is the first pre-lab of the semester, and they all work the same way: a short read before Tuesday, a preview of Thursday's lab, and a few exercises to try on your own. The **Pre-Lab Check** on Canvas (3 points, auto-graded) is due before Tuesday's class begins at 2:00 PM, and its questions come straight from this reading. Tuesday's **quiz** samples this reading and the ungraded practice set.

Part 1: Foundation

Think about the last time you filtered anything on your phone: shoes under 50 dollars, bank charges over 20, flights out of MSP on a Friday. Each time, a database somewhere ran a SELECT statement with a WHERE clause and sent back only the rows that passed the test. This week you learn that exact move: first choosing columns, then filtering rows.

1.1 The shape of every query this week

```
SELECT columns
FROM table
WHERE condition;
```

SQL reads it in a different order than you write it. First, `FROM` finds the table. Then `WHERE` walks through the rows one at a time and keeps the ones that pass your test. Last, `SELECT` trims the survivors down to the columns you asked for.

Meet the database. All examples use `northshore_coffee.db`, a fictional chain of eight coffee shops along Lake Superior's North Shore. The file is posted with this pre-lab on Canvas, and Thursday's lab uses the same one. Four tables: `shops`, `menu_items`, `employees`, and `orders` (which sits out until later weeks).

1.2 Choosing your columns

A table hands you everything. Most of the time you want less. In Week 1 you used `SELECT *` to see a whole table. Naming columns is how you take control:

```
SELECT name, city
FROM shops;
```

`FROM shops` loads the whole table (8 rows, 7 columns). There is no WHERE, so all 8 rows stay. `SELECT name, city` keeps just those two columns, in the order you listed them. A habit worth building now: name your columns. `SELECT *` is fine for a first look at a table, but real queries ask for exactly what they need.

1.3 Comparison operators and the boundary trap

WHERE tests one row at a time with `=`, `<>`, `>`, `<`, `>=`, or `<=`. Say the owner wants to know who earns more than 20 dollars an hour:

```
SELECT first_name, last_name, hourly_wage
FROM employees
WHERE hourly_wage > 20.00;
```

Five of the 25 staff rows pass: Sarah Johnson (22.50), Anna Peterson (21.50), Daniel Rivera (21.00), Olivia Lindgren (21.00), and Liam Gustafson (20.50). Now look at who is missing. Ella Swenson and Zoe Hartman both earn exactly 20.00, and neither made the list. The `>` test is strict. When you want the boundary included, that is what `>=` is for.

1.4 Combining tests with AND, OR, NOT

`AND` keeps a row only when both tests pass. `OR` keeps it when at least one passes. `NOT` flips a test. Say we want teas under 5 dollars:

```
SELECT name, size, price
FROM menu_items
WHERE category = 'Tea' AND price < 5.00;
```

Walk one row through it. The Medium Chai Tea Latte is a Tea, so the first test passes. Its price is 5.25, so the second test fails. AND needs both, so the row is out. Across the whole menu, only three rows pass both tests: Green Tea (Medium, 3.25), Earl Grey (Medium, 3.25), and Hibiscus Iced Tea (Large, 4.50).

One warning before you mix AND with OR in the same WHERE: AND binds tighter than OR. Put parentheses around the OR part so the query does what you meant. Exercise 3 in Part 3 will show you why.

1.5 LIKE: matching patterns

Sometimes you do not know the exact text, only its shape. LIKE matches patterns using two wildcards: `%` stands for any run of characters, including none at all, and `_` stands for exactly one character. So `'Latte%'` matches names that start with Latte, `'%Latte'` matches names that end with Latte, and `'%Latte%'` matches Latte anywhere.

```
SELECT name, size, price
FROM menu_items
WHERE name LIKE '%Latte%';
```

This returns 8 rows: Vanilla Latte (Medium and Large), Pumpkin Spice Latte, Maple Latte, Campfire S'mores Latte, Chai Tea Latte (Medium and Large), and London Fog Latte. Notice what did not show up. The **Flat White** is a latte in the coffee sense, but its name contains no "Latte", so LIKE leaves it out. LIKE matches characters, not meaning.

1.6 IN: matching a list

IN checks a value against a list. Which shops sit in Duluth or Two Harbors?

```
SELECT name, city
FROM shops
WHERE city IN ('Duluth', 'Two Harbors');
```

Four rows come back: the three Duluth shops and the Two Harbors shop. IN is exact shorthand for a chain of ORs on one column: `city = 'Duluth' OR city = 'Two Harbors'`. The IN version is shorter, and because it counts as one single test, it keeps you out of the parentheses trouble from section 1.4.

1.7 BETWEEN: matching a range

BETWEEN keeps values inside a range, **and it includes both ends**. What sells in the 3.75 to 4.00 range?

```
SELECT name, price
FROM menu_items
WHERE price BETWEEN 3.75 AND 4.00;
```

Five rows: Americano (4.00), Double Espresso (3.75), Blueberry Muffin (3.75), Maple Scone (3.95), and Lemon Poppy Seed Muffin (3.75). Both edges made the list. `BETWEEN a AND b` means greater than or equal to a, and less than or equal to b. Compare that with the strict `>` from section 1.3. The two behave differently right at the boundary, and keeping that difference straight will save you often.

Part 2: What Thursday's Lab Asks

Thursday: **Lab 2: SELECT and WHERE (25 points)**, `northshore_coffee.db`, tables `shops`, `menu_items`, and `employees` (`orders` sits out until later weeks). Maren Josephson owns it and has fifteen questions for you.

- Questions 1 to 5 are SELECT work: choosing columns, renaming them with AS, doing small calculations, and joining text together. Those last three moves are new; Tuesday's class and the Week 2 notes cover them.
- Questions 6 to 15 are WHERE work: everything in Part 1 above, plus finding rows where data is missing (NULL), which the notes also cover.
- The lab starts in class Thursday, September 10. The report is due Tuesday, September 15, 2:00 PM, on Canvas; the window stays open in case you need to tidy up at home.

Part 3: Try These

These are warm-ups, not homework. Nothing here is collected, and answers are at the end of this document. Open `northshore_coffee.db` in VS Code the way you did in Week 1, start a scratch `.sql` file, and try these before Tuesday.

1. The owner wants a premium price list. Show the `name` and `price` of every menu item that costs more than 7 dollars. Then check your list: is the Berry Blast Smoothie on it? Say why or why not.

2. Duluth is full of Scandinavian family names. Show the `first_name` and `last_name` of every employee whose last name ends in "son".

3. Lunch combo planning. Show the `name`, `category`, and `price` of every menu item that is a Smoothie or a Sandwich and costs between 7 and 8 dollars. One heads-up: if the Mango Smoothie shows up in your results, your logic needs another look.

4. **(One from Week 1.)** Ask the `shops` table for everything: every column, every row. Count the columns you get. Then write a second query that narrows it down to just `name`, `city`, and `has_drive_thru`. It is the same two-step you did last week, on this week's data.

5. How many different cities does Northshore operate in? Write a query that lists each city exactly once. (Hint: the keyword that removes duplicates is in Tuesday's material and the Week 2 notes: DISTINCT.)

If one breaks, write down what you tried and bring it Tuesday.

Answers to Try These

1. `SELECT name, price FROM menu_items WHERE price > 7.00;`
   7 rows: Turkey Club Sandwich 8.50, Ham & Swiss Panini 8.25, Veggie Wrap 7.75, Green Power Smoothie 7.50, Caprese Sandwich 7.50, Campfire S'mores Latte 7.25, Tropical Sunrise Smoothie 7.25. The Berry Blast Smoothie costs exactly 7.00, so the strict `>` correctly leaves it out. That is the boundary lesson from section 1.3.

2. `SELECT first_name, last_name FROM employees WHERE last_name LIKE '%son';`
   9 rows: Sarah Johnson, Jake Anderson, Emily Larson, Anna Peterson, Chloe Olson, Ethan Johansson, Maya Thompson, Liam Gustafson, Ella Swenson. Note that `'%son%'` happens to return the same 9 rows on this data, but the two patterns mean different things: `'%son'` anchors "son" to the end of the name.

3. `SELECT name, category, price FROM menu_items WHERE category IN ('Smoothie', 'Sandwich') AND price BETWEEN 7.00 AND 8.00;`
   5 rows: Berry Blast Smoothie 7.00, Green Power Smoothie 7.50, Tropical Sunrise Smoothie 7.25, Veggie Wrap 7.75, Caprese Sandwich 7.50. The OR form also works if you parenthesize it: `WHERE (category = 'Smoothie' OR category = 'Sandwich') AND price BETWEEN 7.00 AND 8.00`. Drop the parentheses and AND binds before OR, the Mango Smoothie (6.50) sneaks in, and you get 6 rows. Also notice: Berry Blast at exactly 7.00 IS included here, because BETWEEN includes its endpoints. Compare with exercise 1.

4. `SELECT * FROM shops;` gives 8 rows and 7 columns (id, name, city, address, phone, open_date, has_drive_thru). `SELECT name, city, has_drive_thru FROM shops;` gives the same 8 rows, 3 columns. If you noticed blanks in the phone column: three shops have a NULL phone, which previews Thursday's IS NULL questions.

5. `SELECT DISTINCT city FROM shops;`
   6 rows: Beaver Bay, Duluth, Grand Marais, Silver Bay, Tofte, Two Harbors. Without DISTINCT you get 8 rows, because Duluth appears three times.

Before the check. You should be able to answer these: (1) what changes when I swap `SELECT *` for a list of column names; (2) how `>` and `>=` differ at the boundary; (3) what `%` and `_` each stand for in a LIKE pattern; (4) what BETWEEN does at the two ends of its range, and how to rewrite an IN list as a chain of ORs. If one is fuzzy, reread that section before the check.