CS 61A SU26

Lecture 22: SQL and Tables

July 29, 2026

Rebecca Dang

Join pollev.com/rebeccadang831 (or scan QR code) on your phone or laptop

We'll begin at Berkeley time (10 minutes after), as per tradition!

Potpourri

10 min

- Announcements
- Computing in the News
- Eval: WWSD

Announcements

- Lecture 21 Scheme practice code has been updated to fix the bug we encountered in the `for` macro question
  - The issue was a buggy implementation of the non-tail recursive `map` procedure
- Scheme project has been released!
  - Checkpoint due Thu 8/6
  - Early due date Wed 8/12
  - Final due date Thu 8/13
  - We will not have any extensions for this assignment, unless you have DSP accommodations for extensions, please plan accordingly!
  - Also note that it's due **after** the final exam (to give you more time), but the topics are still in scope
- Submit your favorite songs so we can use that data in tomorrow's lecture!

Computing in the News

- Apple becomes second $5tn company as investors flee AI stocks (The Guardian)
  - See also: AI circular financing: How OpenAI Uses Complex and Circular Deals to Fuel Its Multibillion-Dollar Rise (NY Times, 2025)
- New York school pauses plan to deploy humanlike AI robot teacher after backlash (NPR)
- EPA says power for data centers can sidestep pollution laws (Reuters)
- Minnesota's first-in-the-nation law banning prediction markets is halted in federal court (AP)
  - "A federal judge has temporarily blocked Minnesota's first-in-the-nation law banning prediction markets just days before it was to take effect, the latest clash between President Donald Trump's administration and states over who regulates operators such as Kalshi and Polymarket."
  - Sidestepping state regulation: "prediction market" rather than "gambling"

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Eval: What would Scheme display when evaluating this expression?
 (eval (map (lambda (x) (* x x)) '(1 2 3)))

SQL

40 min

- Databases
- SQL
- Select
- Operators
- Where
- Like
- Order By
- Limit
- Joins
- Implicit Join
- Explicit Join
- Order of Execution
- Query Writing Tips

Databases (1 of 2)

- A database stores data in a **persistent** manner
  - In contrast to variables in a program, which only exist temporarily in the computer's memory during the execution of the program (turning off the power wipes the memory)
- A **database management system (DBMS)** is a piece of software that allows the user to create, retrieve, update, delete, and otherwise manage a database
- A popular kind of database is a **relational database**, where **records** are stored as **rows** in a **table**, which can have multiple **columns** (aka **fields**)
  - Each column has a single **data type**

Databases (2 of 2)

`cities` table example:

latitude	longitude	name
38	122	'Berkeley'
41	74	'New York'
45	93	'Minneapolis'
34	118	'Los Angeles'

SQL

- **Structured Query Language (SQL)** is a declarative programming language for database manipulation
  - Pronounced "sequel" or "S-Q-L"
  - There are many SQL variants; in 61A we use **SQLite**
  - We will focus on a subset of SQL, the **Data Query Language (DQL)** which is only used to **read** databases
- Unlike imperative, functional, or object-oriented programming, in **declarative programming**, the programmer describes the output they want and the computer figures out the rest (take CS 186 or DATA 101 to find out how)
- **SQL is case-insensitive**, but by convention, keywords are written in all capital letters
- All SQL statements **must end with a semicolon** (`;`)
- Like Scheme, **whitespace is mainly for readability** (at minimum you have to separate keywords/column/table names with a single space)
- Single line comments with `--`

Select (1 of 2)

- Creates a new table
  - Create a new table from scratch or take columns `FROM` existing table
  - Can select 1+ columns, separated by commas
  - Can select all columns with `*`
- Use `AS` to rename columns or expressions
  - Also referred to as "**aliasing**" a column or "giving the column an alias"

```
SELECT <expression>;

SELECT [col1], ... FROM <table>;

SELECT * FROM <table>;
```

Select (2 of 2)

```
SELECT latitude, name FROM cities;
```

latitude	name
38	'Berkeley'
41	'New York'
45	'Minneapolis'
34	'Los Angeles'

Operators

- SQL supports basic arithmetic operations like Python (`+`, `*`, `-`, `/`)
- SQL also supports the same comparison (e.g. `>`, `<=`) and boolean operators (e.g. `AND`, `OR`, and `NOT`) as Python, with some slight changes:
  - Equality: `=` is preferred over `==` but they are equivalent
  - Inequality: `<>` is preferred over `!=` but they are equivalent
- SQL string concatenation operator is `||` (instead of `+`)

Where (1 of 3)

The WHERE clause is used to filter out rows based on some condition.

```
SELECT <expression(s) and/or column(s)>
FROM <table>
WHERE <condition>;
```

Where (2 of 3)

```
SELECT latitude, name FROM cities WHERE latitude > 40;
```

latitude	name
~~38~~	~~'Berkeley'~~
41	'New York'
45	'Minneapolis'
~~34~~	~~'Los Angeles'~~

Where (3 of 3)

```
SELECT latitude, name FROM cities WHERE latitude > 40;
```

latitude	name
41	'New York'
45	'Minneapolis'

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

True or false: The WHERE clause can only contain columns that are in the SELECT clause.

Like (1 of 2)

- The `LIKE` clause is used to perform string matching.
- Wildcards allow us to match string patterns
  - `%` = 0 or more of any character
  - `_` = exactly 1 of any character
- **NOTE:** `LIKE` matching is case **insensitive** in SQLite

```
SELECT name FROM cities WHERE name LIKE '% %';
```

name
'Berkeley'
'New York'
'Minneapolis'
'Los Angeles'

name
~~'Berkeley'~~
'New York'
~~'Minneapolis'~~
'Los Angeles'

name
'New York'
'Los Angeles'

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Select all of the following strings that would be matched by the clause: LIKE '%a_'

Like (2 of 2)

Pattern	Matches	Does not match

`'%hello%'`	`'hello'`
	`'HELLO'`
	`'a hello b'`
	`'a hello'`
	`'hello b'`
		`'bye'`
		`'ello'`

`'%hello'`	`'hello'` `'a hello'`	`'hello b'`
`'hello%'`	`'hello'` `'hello b'`	`'a hello'`

`'_at'`	`'bat'`
	`'cat'`
	`'fat'`
		`'chat'`
		`'a bat'`

`'%a_'`	`'at'`
	`'fat'`
	`'chat'`
	`'a bat'`
		`'hello'`

Order By (1 of 3)

- The `ORDER BY` clause sorts the rows of the table based on one or more columns (ascending order by default, use `DESC` to specify descending order)
  - If sorting on multiple columns, break ties based on ordering from left to right

```
SELECT <columns>
FROM <table>
WHERE <condition>
ORDER BY <columns> ASC/DESC;
```

Order By (2 of 3)

Updated `cities` table:

latitude	longitude	name
38	122	'Berkeley'
41	74	'New York'
45	93	'Minneapolis'
34	118	'Los Angeles'
38	130	'Metropolis'

Order By (3 of 3)

```
SELECT * FROM cities ORDER BY latitude, name DESC;
```

latitude	longitude	name
34	118	'Los Angeles'
38	130	'Metropolis'
38	122	'Berkeley'
41	74	'New York'
45	93	'Minneapolis'

Limit

Adding a `LIMIT` clause ensures we only keep the first `n` rows of output.

```
SELECT <columns>
FROM <table>
WHERE <condition>
ORDER BY <columns>
LIMIT <n>;
```

Joins (1 of 4)

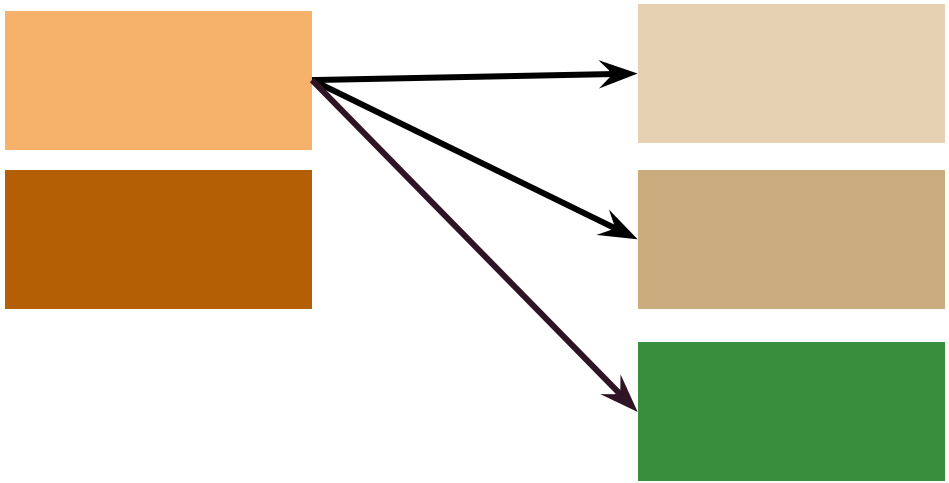
- What if we want to combine data from multiple tables? Ex:
  - A university database might have a `courses` table and `students` table, and we want to find out which students are taking which courses
- We can do this by performing a database operation called a **join**
  - The simplest kind of join is a **cross join**, where you match each row in table A with each row in table B ("nested for loop")

Joins (2 of 4)

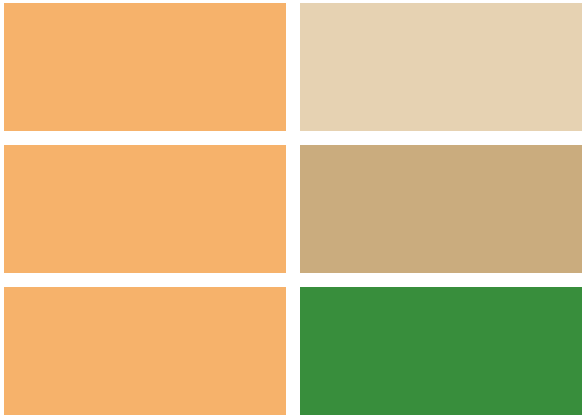
2×1 Table **join** with 3×1 Table **Resulting Table**

Joins (3 of 4)

2×1 Table **join** with 3×1 Table

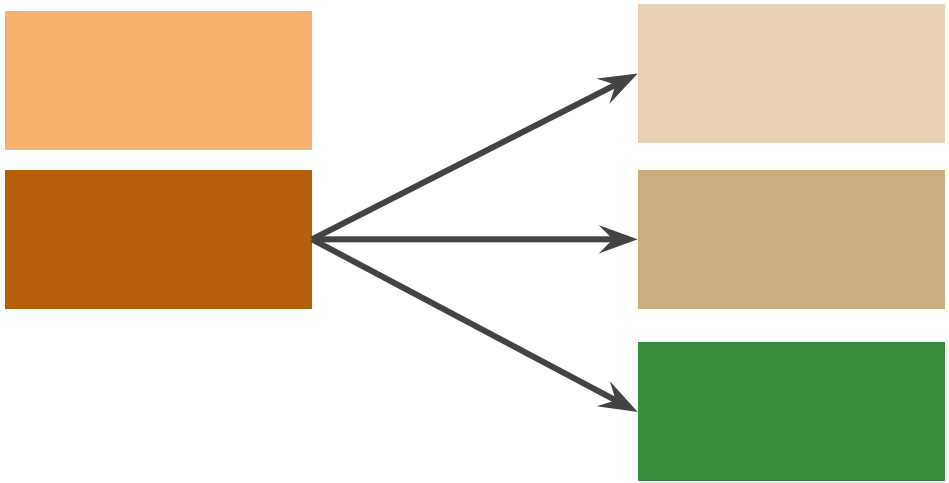


Resulting Table

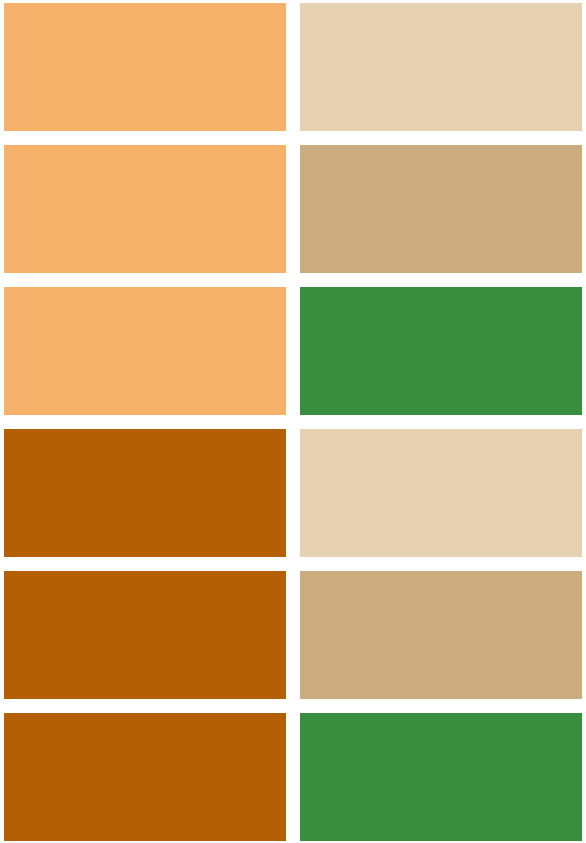


Joins (4 of 4)

2×1 Table **join** with 3×1 Table



Resulting Table



When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Suppose table A has n rows and table B has m rows. What is the number of rows in the output table when performing a cross join on A and B?

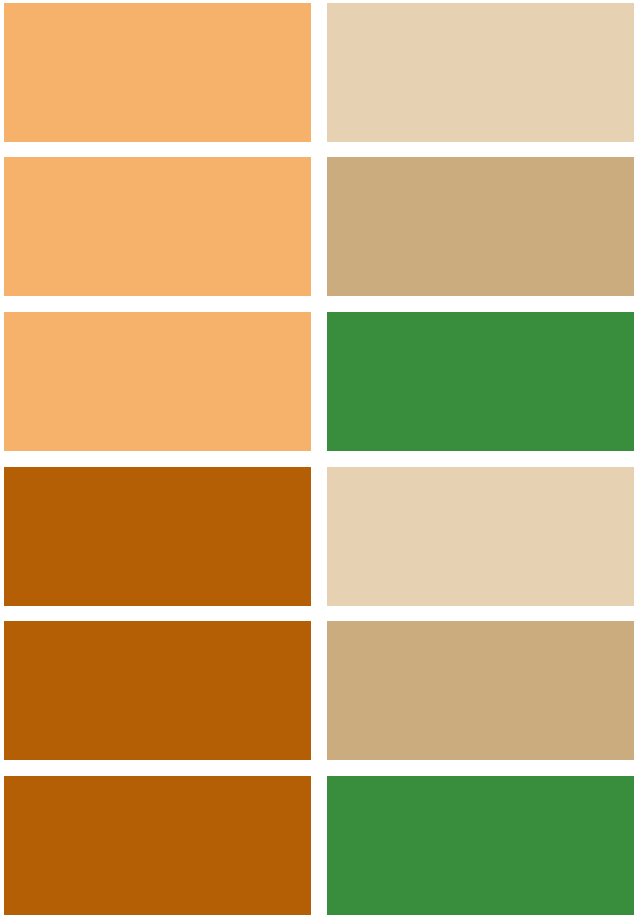
Implicit Join (1 of 5)

An **implicit join** is a join that does not involve the JOIN keyword (instead you specify 2 or more tables in the FROM clause). Ex:

```
SELECT <columns>
FROM <table1>, <table2>
WHERE <condition>;
```

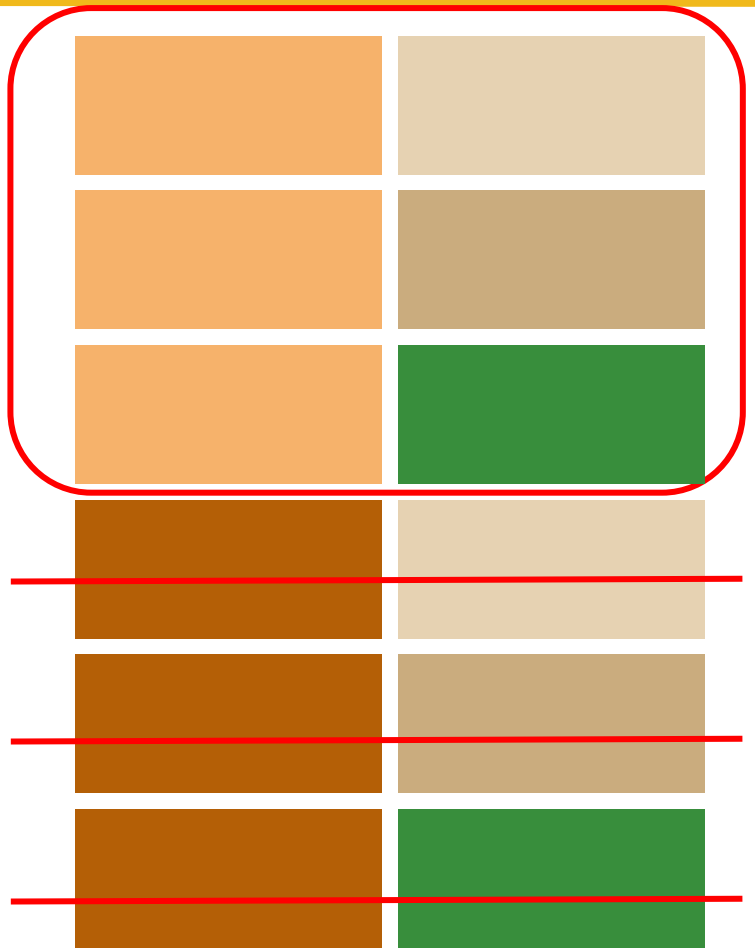
Implicit Join (2 of 5)

```
SELECT * FROM table1, table2
WHERE left_color = orange AND
    right_color != green;
```



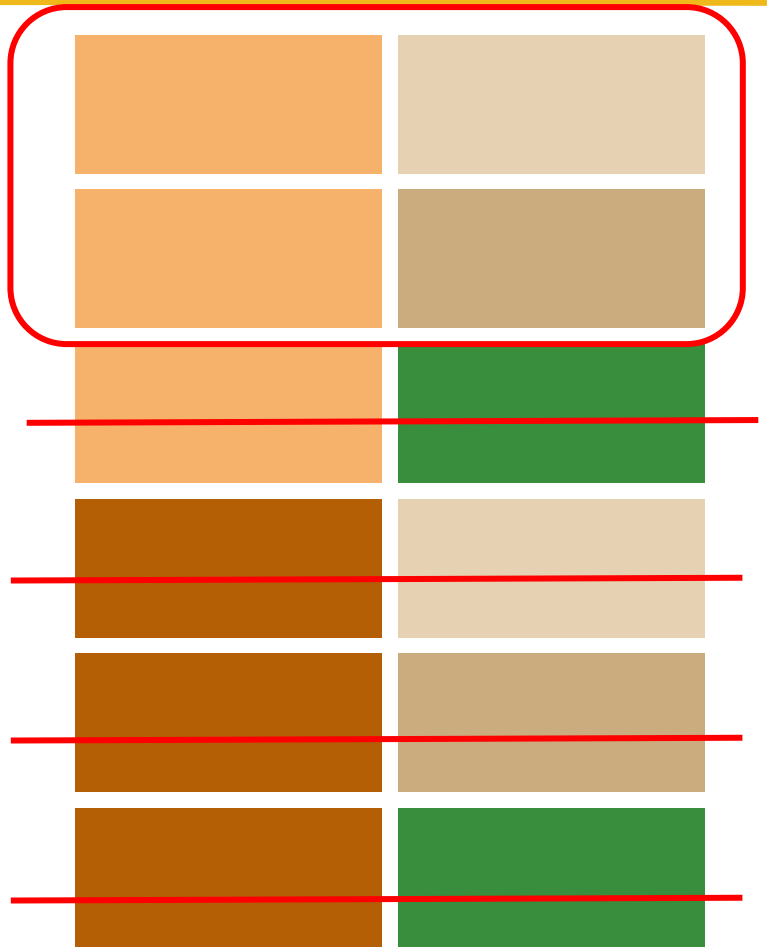
Implicit Join (3 of 5)

```
SELECT * FROM table1, table2
WHERE left_color = orange AND
    right_color != green;
```



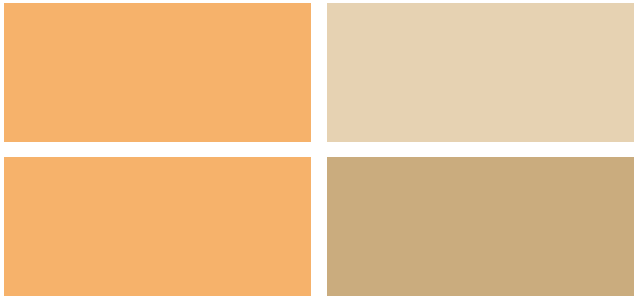
Implicit Join (4 of 5)

```
SELECT * FROM table1, table2
WHERE left_color = orange AND
    right_color != green;
```



Implicit Join (5 of 5)

```
SELECT * FROM table1, table2
WHERE left_color = orange AND
    right_color != green;
```



Dot Notation

- To specify that you want to retrieve a column from a particular table, you can use **dot notation**.
  - You can use dot notation with the full table name or its alias
- This is only required if the column names are ambiguous (e.g. the tables have duplicate column names), but it is good practice to always use dot notation for clarity.
- **Demo visualization** (can run this in 61A Code's SQL interpreter):

```
SELECT r.name, division, title, salary, salary2023
FROM records AS r, salaries AS s
WHERE r.name = s.name;
```

Explicit Join (1 of 2)

An **explicit join** is a join that ***does*** involve the `JOIN` keyword. You put the **join predicate** (aka condition) in the `ON` clause. (You can still have a `WHERE` clause if you want.) Ex:

```
SELECT <columns>

FROM <table1>

JOIN <table2>

ON <condition>;
```

Explicit Join (2 of 2)

```sql
-- Equivalent to implicit join example
SELECT r.name, division, title, salary, salary2023
FROM records AS r
JOIN salaries AS s
ON r.name = s.name;
```

Order of Execution

Because SQL is declarative, SQL code is not necessarily executed top to bottom, left to right! Instead, this is the order of execution for the clauses we've learned so far:

1. `FROM`, `JOIN … ON …`
2. `WHERE`, `LIKE`
3. `SELECT`
4. `AS`
5. `ORDER BY`
6. `LIMIT`

Query Writing Tips

1. Where is the data located? → `FROM, JOIN … ON …`
2. Do we need to filter out any rows? → `WHERE`, `LIKE`
3. Which columns should be included in the output? → `SELECT`
4. Do we need to rename any columns? → `AS`
5. Do we need to sort the final output? → `ORDER BY`
6. Do we need to truncate the output (only include the first `n` rows)? → `LIMIT`

Hint: This is very similar to the SQL order of execution!

5 min break

Practice

(reminder to self to use high contrast theme)

30 min

- Boba

Boba

- Implement Q1 - Q5
- Download starter code `22.sql` from the course website under Lecture 22
- Manually test your code by copying and pasting it into a new file in code.cs61a.org, clicking on the green play button on the top right, and selecting all rows from that question's table
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?