CS 61A Spring 2025

SQL

Discussion 12: April 30, 2025

Switch to Pensieve:

- **Everyone**: Go to pensieve.co, log in with your @berkeley.edu email, and **enter your group number** as the room number (which was in the email that assigned you to this discussion). As long as you all enter the same number (any number), you'll all be using a shared document.

Once you're on Pensieve, you don't need to return to this page; Pensieve has all the same content (but more features). If for some reason Penseive doesn't work, return to this page and continue with the discussion.

Attendance

Fill out this discussion attendance form with the unique number you receive from your TA. As soon as you get your number, fill out the form, selecting *arrival* (not *departure* – that's later).

Getting Started

If there are fewer than 3 people in your group, feel free to merge your group with another group in the room.

Everybody say your name, and then share your favorite restaurant, cafe, or boba shop near campus. (Yes, Kingpin Donuts counts as a restaurant.)

Select Statements

A `SELECT` statement describes an output table based on input rows. To write one: 1. Describe the **input rows** using `FROM` and `WHERE` clauses. 2. **Group** those rows and determine which groups should appear as output rows using `GROUP BY` and `HAVING` clauses. 3. Format and order the **output rows** and columns using `SELECT` and `ORDER BY` clauses.

`SELECT` *(Step 3)* `FROM` *(Step 1)* `WHERE` *(Step 1)* `GROUP BY` *(Step 2)* `HAVING` *(Step 2)* `ORDER BY` *(Step 3)*;

Step 1 may involve joining tables to form input rows that consist of two or more rows from existing tables.

The `WHERE`, `GROUP BY`, `HAVING` and `ORDER BY` clauses are optional.

Visualizing SQL

The CS61A SQL Web Interpreter is a great tool for visualizing and debugging SQL statements!

To get started, visit code.cs61a.org and hit `Start SQL interpreter` on the launch screen.

Most tables used in assignments are already available for use, so let's try to execute a `SELECT` statement:

In addition to displaying a visual representation of the output table, the "Step-by-step" button lets us step through the SQL execution and visualize every transformation that takes place. For our example, clicking on the next arrow will produce the following visuals, demonstrating exactly how SQL is grouping our rows to form the final output!

Pizza Time

The `pizzas` table contains the names, opening, and closing hours of great pizza places in Berkeley. The `meals` table contains typical meal times (for college students). A pizza place is open for a meal if the meal time is at or within the `open` and `close` times.

```
CREATE TABLE pizzas AS
  SELECT "Artichoke" AS name, 12 AS open, 15 AS close UNION
  SELECT "La Val's"         , 11        , 22          UNION
  SELECT "Sliver"           , 11        , 20          UNION
  SELECT "Cheeseboard"      , 16        , 23          UNION
  SELECT "Emilia's"         , 13        , 18;

CREATE TABLE meals AS
  SELECT "breakfast" AS meal, 11 AS time UNION
  SELECT "lunch"            , 13         UNION
  SELECT "dinner"           , 19         UNION
  SELECT "snack"            , 22;
```

Q1: Open Early

You'd like to have pizza before 13 o'clock (1pm). Create a `opening` table with the names of all pizza places that `open` before 13 o'clock, listed in reverse alphabetical order.

`opening` table:

name
Sliver
La Val's
Artichoke

```
-- Pizza places that open before 1pm in alphabetical order



SELECT "REPLACE THIS LINE WITH YOUR SOLUTION";
```

To order by `name` in reverse alphabitical order, write `ORDER BY name DESC`.

Q2: Study Session

You're planning to study at a pizza place from the moment it opens until 14 o'clock (2pm). Create a table `study` with two columns, the `name` of each pizza place and the `duration` of the study session you would have if you studied there (the difference between when it opens and 14 o'clock). For pizza places that are not open before 2pm, the `duration` should be zero. Order the rows by decreasing duration.

Hint: Use an expression of the form `MAX(_, 0)` to make sure a result is not below 0.

`study` table:

name	duration
La Val's	3
Sliver	3

name	duration
Artichoke	2
Emilia's	1
Cheeseboard	0

```
-- Pizza places and the duration of a study break that ends at 14 o'clock



SELECT "REPLACE THIS LINE WITH YOUR SOLUTION";
```

To order by decreasing duration, first name the column with `SELECT ..., ... AS duration ...`, then `ORDER BY duration DESC`.

Q3: Late Night Snack

What's still open for a late night `snack`? Create a `late` table with one column named `status` that has a sentence describing the closing time of each pizza place that closes at or after `snack` time. **Important:** Don't use any numbers in your SQL query! Instead, use a join to compare each restaurant's closing time to the time of a snack. The rows may appear in any order.

`late` table:

status
Cheeseboard closes at 23
La Val's closes at 22

The `||` operator in SQL concatenates two strings together, just like `+` in Python.

```
-- Pizza places that are open for late-night-snack time and when they close



  SELECT ____ || " closes at " || ____ AS status
  FROM ____
  WHERE ____;
```

To compare a pizza place's `close` time to the time of a snack: - join the `pizzas` and `meals` tables using `FROM pizzas, meals` - use only rows where the `meal` is a `"snack"` - compare the `time` of the snack to the `close` of the pizza place.

Use `name || " closes at " || close` to create the sentences in the resulting table. The `||` operator concatenates values into strings.

Q4: Double Pizza

If two meals are more than 6 hours apart, then there's nothing wrong with going to the same pizza place for both, right? Create a `double` table with three columns. The `first` column is the earlier meal, the `second` column is the later meal, and the `name` column is the name of a pizza place. Only include rows that describe two meals that are **more than 6 hours apart** and a pizza place that is open for both of the meals. The rows may appear in any order.

`double` table:

first	second	name
breakfast	dinner	La Val's
breakfast	dinner	Sliver
breakfast	snack	La Val's
lunch	snack	La Val's

```
-- Two meals at the same place




  SELECT ____ AS first, ____ AS second, name
  FROM ____, ____, pizzas
  WHERE ____;
```

Use `FROM meals AS a, meals AS b, pizzas` so that each row has info about two meals and a pizza place. Then you can write a `WHERE` clause that compares both `a.time` and `b.time` to `open` and `close` and each other to ensure all the relevant conditions are met.

A Final Exam About Final Exams

Note: The first question below was also used in lecture, so if you're running out of time, feel free to skip to the next one. (But it's good practice, which is why it's here.)

From the Spring 2023 final exam.

The `finals` table has columns `hall` (strings) and `course` (strings), and has rows for the lecture halls in which a course is holding its final exam.

The `sizes` table has columns `room` (strings) and `seats` (numbers), and has one row per unique room on campus containing the number of seats in that room. All lecture halls are rooms.

Q5: Total Seats

Create a table with two columns, `course` (strings) and `total` (numbers) that has a row for **each course that uses at least two rooms** for its final. Each row contains the name of the course and the total number of seats in final rooms for that course.

Your query should work correctly for any data that might appear in the `finals` and `sizes` table, but for the example below the result should be:

```
61A|2400
61B|1700
61C|1200
```

Discussion Time: Talk about why the output table contains what it contains. Why are CS 10 and 70 not included? Where does the number 1700 come from?

```
CREATE TABLE finals AS
  SELECT "RSF" AS hall, "61A" as course UNION
  SELECT "Wheeler"    , "61A"           UNION
  SELECT "Pimentel"   , "61A"           UNION
  SELECT "Li Ka Shing", "61A"           UNION
  SELECT "RSF"        , "61B"           UNION
  SELECT "Wheeler"    , "61B"           UNION
  SELECT "Morgan"     , "61B"           UNION
  SELECT "Wheeler"    , "61C"           UNION
  SELECT "Pimentel"   , "61C"           UNION
  SELECT "Soda 306"   , "10"            UNION
  SELECT "RSF"        , "70";

CREATE TABLE sizes AS
  SELECT "RSF" AS room, 900 as seats    UNION
  SELECT "Wheeler"    , 700             UNION
  SELECT "Pimentel"   , 500             UNION
  SELECT "Li Ka Shing", 300             UNION
  SELECT "Morgan"     , 100             UNION
  SELECT "Soda 306"   , 80              UNION
  SELECT "Soda 310"   , 40              UNION
  SELECT "Soda 320"   , 30;

SELECT _____, _____ AS total
  FROM _____ WHERE _____
  GROUP BY _____ HAVING _____;
```

Join the `finals` and `sizes` tables, but make sure that each joined row is coherent by restricting to rows in which the `hall` (from `finals`) and `room` (from `sizes`) are the same value.

Since the output has one row per course, but the same course appears in multiple rows of the `finals` table, group by `course`.

`COUNT(*)` evaluates to the number of input rows in a group, which in this case will be the number of rooms used by a course.

The expression `SUM(seats)` evaluates to the sum of the `seats` values (from the `sizes` table) for a group. If there is one group per course, then this will be the sum of seats in all lecture halls used for that course.

Q6: Room Sharing

Write one select statement that creates a table with two columns, `course` (strings) and `shared` (numbers) that has a row for **each course using at least one room that is also used by another course**. Each row contains the name of the course and the total number of rooms for that course which are also used by another course.

Reminder: `COUNT(DISTINCT x)` evaluates to the number of distinct values that appear in column `x` for a group.

Your query should work correctly for any data that might appear in the `finals` and `sizes` table, but for the example below the result should be:

```
61A|3
61B|2
61C|2
70|1
```

Discussion Time: Talk about why the output table contains what it contains. Which are the two halls for 61B that are shared?

```
CREATE TABLE finals AS
  SELECT "RSF" AS hall, "61A" as course UNION
  SELECT "Wheeler"    , "61A"           UNION
  SELECT "Pimentel"   , "61A"           UNION
  SELECT "Li Ka Shing", "61A"           UNION
  SELECT "RSF"        , "61B"           UNION
  SELECT "Wheeler"    , "61B"           UNION
  SELECT "Morgan"     , "61B"           UNION
  SELECT "Wheeler"    , "61C"           UNION
  SELECT "Pimentel"   , "61C"           UNION
  SELECT "Soda 306"   , "10"            UNION
  SELECT "RSF"        , "70";

CREATE TABLE sizes AS
  SELECT "RSF" AS room, 900 as seats    UNION
  SELECT "Wheeler"    , 700             UNION
  SELECT "Pimentel"   , 500             UNION
  SELECT "Li Ka Shing", 300             UNION
  SELECT "Morgan"     , 100             UNION
  SELECT "Soda 306"   , 80              UNION
  SELECT "Soda 310"   , 40              UNION
  SELECT "Soda 320"   , 30;

SELECT _____, _____ AS shared
  FROM _____ WHERE _____
  GROUP BY _____;
```

Join `finals` with `finals`, but make sure that the joined rows are for difference courses using the same lecture hall.

Group by the first `course` column to make one group (and one output row) per course. A `HAVING` clause is not

needed if the `WHERE` clause has already limited the input rows to those with two different courses using the same hall.

Count the distinct number of `hall` values for a course: `COUNT(DISTINCT ___)`. The `DISTINCT` restriction is needed so that a hall used by more than two courses is not counted more than once.

Document the Occasion

Let your TA know you're done so that you can each get a **departure** number, and fill out the attendance form again (this time selecting *departure* instead of *arrival*). If your TA isn't in the room, go find them next door.

If you finish early, maybe go get pizza together…