CS 61A Spring 2026

SQL

Discussion 12: April 29, 2026

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.

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 (name TEXT, open INTEGER, close INTEGER);

INSERT INTO pizzas VALUES
  ("Artichoke", 12, 15),
  ("La Val's", 11, 22),
  ("Sliver", 11, 20),
  ("Cheeseboard", 16, 23),
  ("Emilia's", 13, 18);

CREATE TABLE meals (meal TEXT, time INTEGER);

INSERT INTO meals VALUES
  ("breakfast", 11),
  ("lunch", 13),
  ("dinner", 19),
  ("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. To test what table your query outputs, press the green play button in 61A Code!

`opening` table:

name
Sliver
La Val's
Artichoke

```
-- Pizza places that open before 1pm in alphabetical order
CREATE TABLE opening AS
  SELECT name FROM pizzas WHERE open < 13 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
Artichoke	2
Emilia's	1
Cheeseboard	0

```
-- Pizza places and the duration of a study break that ends at 14 o'clock
CREATE TABLE study AS
  SELECT name, MAX(14 - open, 0) AS duration FROM pizzas 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
CREATE TABLE late AS
  SELECT name || " closes at " || close AS status
  FROM pizzas JOIN meals
  ON time <= close
  WHERE meal="snack";
```

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
CREATE TABLE double AS
  SELECT a.meal AS first, b.meal AS second, name
         FROM meals AS a
         JOIN meals AS b ON b.time > a.time + 6
         JOIN pizzas ON open <= a.time AND a.time <= close AND open <= b.time AND b.time
    <= close;
```

A Final Exam About Final Exams

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 data above the result should be:

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

```
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 course, SUM(seats) AS total
  FROM finals JOIN sizes ON hall=room
  GROUP BY course HAVING COUNT(*) > 1;
```

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.

`COUNT(DISTINCT x)` evaluates to the number of distinct values that appear in column `x` for a group. For example, `SELECT COUNT(DISTINCT seats) from sizes;` would output 8, because there are 9 rows in `sizes`, but two rows have 300 seats, so there are only 8 distinct values.

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 a.course, COUNT(DISTINCT a.hall) AS shared
  FROM finals AS a JOIN finals AS b
  ON a.hall = b.hall AND a.course != b.course
  GROUP BY a.course;
```

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.

celebrate your last discussion section!