CS 61A
Fall 2023

SQL, Final Review

Discussion 13: November 29, 2023

Getting Started

Someone from your group should join Discord.

To get help from a TA, send a message to the `discuss-queue` channel with the @discuss tag and your discussion group number.

If you have only 1 or 2 people in your group, you can join the other group in the room with you.

Special Event: There is a food-based 61A gathering after discussion today. You'll need to solve the problems to find out what and where.

Q1: Ice Breaker

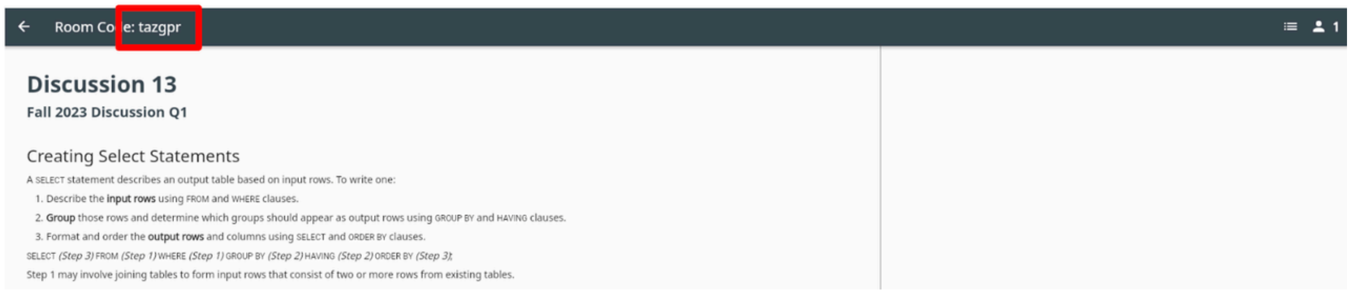
Say your name and the non-CS/EECS course that you're most excited about taking next semester. Whoever seems most excited is today's facilitator.

Pensieve

Important: Today, you'll get to try out a new Pensieve interface for discussion that synchronizes your responses across your whole group and lets you chat with an AI tutor and the course staff.

Getting started:

- **Everyone**: Go to discuss.pensieve.co and log in with your @berkeley.edu email.
- **Facilitator**: Click "Create Room" and share the room code (boxed in red in the screenshot below) with your group.
- **Others**: Don't create a room; instead, click "Join Room" after typing in the room code that the facilitator shared with you.
- Post in the `#help` channel on Discord if you have trouble.



Discussion 13
Fall 2023 Discussion Q1

Creating 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.

Pensieve room

Instructions:

- **Like a shared Google Doc, everything you type in Pensieve can be seen by your group members and the course staff.**

- Discuss first and make sure you all agree on the answer! Remember, on the final there is no "check answer" button. Once you're ready for feedback, click the `check answer` button.
- To ask a question of the AI tutor, use the right panel. First, select which question you want to ask about (boxed in red in the screenshot below). The AI Tutor can see your current solution and the question, so you don't need to paste those into the chat.

For more instructions, here's a 3 minute video guide.

The rest of this page is duplicated in the Pensieve interface, so you all can just switch there (and never come back). This page is only here in case something goes wrong with the new interface.

Creating 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.

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.

Q2: 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 course, SUM(seats) AS total
  FROM finals, sizes WHERE hall=room
  GROUP BY course HAVING COUNT(*) > 1;
```

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.

Q3: 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 a.course, COUNT(DISTINCT a.hall) AS shared
  FROM finals AS a, finals AS b WHERE a.hall = b.hall AND a.course != b.course
  GROUP BY a.course;
```

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.

Presentation time: Come up with a short explanation of why you joined the `finals` table with itself to solve this problem. (Or if you didn't join `finals` with itself but solved it anyway, come up with an explanation of your approach.) Send a message to the `#discuss-queue` channel with the `@discuss` tag, your discussion group number, and the message "Join us!" and a member of the course staff will join your voice channel to hear your explanation and give feedback.

Party Time

There is a gathering after discussion today, just for the students who participated in these experimental small discussions. Solve this problems to find out what and where, then come.

A substitution cipher replaces each word with another word in a table in order to encrypt a message. To decode an encrypted message, replace each word `x` with its corresponding `y` in a code table.

Q4: A Secret Message

Write a select statement to decode the `original` message *Can You Find The Party* using the `code` table.

```
CREATE TABLE original AS
  SELECT 1 AS n, "Can" AS word UNION
  SELECT 2     , "You"         UNION
  SELECT 3     , "Find"        UNION
  SELECT 4     , "The"         UNION
  SELECT 5     , "Party";

CREATE TABLE code AS
  SELECT "Up" AS x, "Down" AS y UNION
  SELECT "You"    , "Core" UNION
  SELECT "Party"  , "Age" UNION
  SELECT "Son"    , "Can" UNION
  SELECT "See"    , "Do" UNION
  SELECT "Can"    , "See" UNION
  SELECT "The"    , "Mess" UNION
  SELECT "It"     , "Son" UNION
  SELECT "Do"     , "It" UNION
  SELECT "Mess"   , "Thug" UNION
  SELECT "Core"   , "Not" UNION
  SELECT "Find"   , "It" UNION
  SELECT "Not"    , "Mess" UNION
  SELECT "Age"    , "Laid";

SELECT y FROM original, code WHERE word=x ORDER BY n;
```

Join the `original` and `code` tables and make sure that the joined roles have the same `word` and `x`.

The secret message is *See Core It Mess Age* which sounds like “secret message” when you say it out loud.

Q5: A Fun Time

You found a *secret message*! To find out where the party is, write another select statement to decode this secret message using the same `code` table.

```
CREATE TABLE original AS
  SELECT 1 AS n, "Can" AS word UNION
  SELECT 2     , "You"         UNION
  SELECT 3     , "Find"        UNION
  SELECT 4     , "The"         UNION
  SELECT 5     , "Party";

CREATE TABLE code AS
  SELECT "Up" AS x, "Down" AS y UNION
  SELECT "You"    , "Core" UNION
  SELECT "Party"  , "Age" UNION
  SELECT "Son"    , "Can" UNION
  SELECT "See"    , "Do" UNION
  SELECT "Can"    , "See" UNION
  SELECT "The"    , "Mess" UNION
  SELECT "It"     , "Son" UNION
  SELECT "Do"     , "It" UNION
  SELECT "Mess"   , "Thug" UNION
  SELECT "Core"   , "Not" UNION
  SELECT "Find"   , "It" UNION
  SELECT "Not"    , "Mess" UNION
  SELECT "Age"    , "Laid";

SELECT b.y
  FROM original, code AS a, code AS b
  WHERE word=a.x AND a.y=b.x
  ORDER BY n;
```

Join `original` with `code AS a` and `code AS b` to create six-column rows like: `1|Can|Can|See|See|Do`, The *Do* at the end is part of the decoded message.

The decoded message is *Do Not Son Thug Laid* which sounds like “donuts on the glade” when you say it out loud.

Document the Occasion

Please all fill out the attendance form (one submission per person per week).

Thanks for participating in this experimental discussion format for the semester!

If your group is coming to the party, please post “Incoming!” and your group number in the `#discuss-queue` channel so that we can look out for you. Now go to the party. (Still don’t know what/where this “party” is? Read the answer hint in the last question above.)

Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.