CROSS TABULATIONS

Jed Rembold

Monday, October 28, 2024

ANNOUNCEMENTS

- I'm working on scoring HW7!
- Test 2 on Wednesday!
  - No HW due
  - In addition to the study guide, there *will* be something with data normalization
    - I'll get some sample questions on that up later tonight
- 3+1 Application workshop: QUAD Center 12-1pm tomorrow, if you are a junior looking to apply!
- Polling today: polling.jedrembold.prof

REVIEW QUESTION

Given the starting table called `rev`, what is the output of the query?

name	num
A	1
B	2
C	3
D	4
E	5

```
SELECT name
FROM rev
WHERE num IN (
  SELECT
    num - (SELECT avg(num) FROM rev)
  FROM rev
  WHERE num >= ALL (
    SELECT num FROM rev
  )
);
```

REUSING DERIVED TABLES

COMMON TABLE EXPRESSIONS

- Sometimes you may want to use the same derived tables multiple places in a query
- You could copy and paste those subqueries, but there is a better way
- *Common Table Expressions* or CTEs are a way for you to define up front a derived table that can be referenced anywhere in the rest of the query
- To do so, you use a `WITH` keyword at the *start* of the query:

```
WITH
  derived table name (column names) AS (
    subquery
  )
SELECT ...
```

CTE DETAILS

- The number of columns returned by your subquery must match the number of column names you initially define
  - If you aren't going to rename the columns, you do not even really need to include the new column names, but it can be nice for clarity
- You do not need to include data types, as these will be inherited from the subquery columns
- You can include multiple derived tables in the CTE statement, separated with a comma
- Good use of CTE's can help clarify and simplify queries
  - *Especially* if you have a subquery that you need to reference a few times, use a CTE!

EXAMPLE

Using a CTE and other subqueries, let's extract the middle 50% of the taxi_rides by duration.

PIVOT TABLES

WHY CROSS TABULATION?

- SQL tables tend to operate with observations (records/rows) and variables (columns)
  - Commonly called a “tidy” data format
- This isn’t always the most *human-readable* format
- Cross-tabulations or pivot tables compare aggregates of variables to other variables

Name	Color	Condition	Number
Apple	Red	Good	5
Orange	Orange	Bad	3
Cherry	Red	Good	10
Banana	Yellow	Good	1
Pineapple	Yellow	Bad	4

Color	Good	Bad
Red	15	0
Orange	0	3
Yellow	1	4

EXTENSIONS

- Standard SQL has no way to create these cross-tabulations
  - They can't be accomplished just through grouping, as you are needing to create new columns
- Postgres has a function to help, but it is located in an *extension* or *module*
  - Extensions to Postgres are collections of functions, date types, and more which add some functionality to a particular database
- To use an extension, you need only enable it for your particular database

```
CREATE EXTENSION extension name;
```

- To facilitate making pivot tables, we need to add the `tablefunc` extension to our database, which will add the `crosstab` function
- You can always remove an extension using `DROP` if you need later

PIECES OF THE CROSSTAB

- The `crosstab` function has a lot going on, so let's break things down

```
SELECT *
FROM crosstab(
  subquery 1 string,
  subquery 2 string
  )
AS (
  row label column TEXT,
  column 1  type 1,
  column 2  type 2,
   ⋮
);
```

PIECES OF THE CROSSTAB

- The `crosstab` function has a lot going on, so let's break things down
  - Subquery 1 must return 3 columns

```
SELECT *
FROM crosstab(
  subquery 1 string,
  subquery 2 string
  )
AS (
  row label column TEXT,
  column 1  type 1,
  column 2  type 2,
   ⋮
);
```

PIECES OF THE CROSSTAB

- The `crosstab` function has a lot going on, so let's break things down

```
SELECT *
FROM crosstab(
  subquery 1 string,
  subquery 2 string
  )
AS (
  row label column TEXT,
  column 1  type 1,
  column 2  type 2,
   ⋮
);
```

- Subquery 1 must return 3 columns
  - The first is the row label values

PIECES OF THE CROSSTAB

- The `crosstab` function has a lot going on, so let's break things down

```
SELECT *
FROM crosstab(
   subquery 1 string,
   subquery 2 string
   )
AS (
   row label column TEXT,
   column 1  type 1,
   column 2  type 2,
    ⋮
);
```

- Subquery 1 must return 3 columns
  - The first is the row label values
  - The second is the column label values

PIECES OF THE CROSSTAB

- The `crosstab` function has a lot going on, so let's break things down

```
SELECT *
FROM crosstab(
  subquery 1 string,
  subquery 2 string
  )
AS (
  row label column TEXT,
  column 1  type 1,
  column 2  type 2,
   ⋮
);
```

- Subquery 1 must return 3 columns
  - The first is the row label values
  - The second is the column label values
  - The third is the particular data values

PIECES OF THE CROSSTAB

- The `crosstab` function has a lot going on, so let's break things down

```
SELECT *
FROM crosstab(
  subquery 1 string,
  subquery 2 string
  )
AS (
  row label column TEXT,
  column 1  type 1,
  column 2  type 2,
   ⋮
);
```

- Subquery 1 must return 3 columns
  - The first is the row label values
  - The second is the column label values
  - The third is the particular data values
- Subquery 2 returns a single column

PIECES OF THE CROSSTAB

- The `crosstab` function has a lot going on, so let's break things down

```
SELECT *
FROM crosstab(
  subquery 1 string,
  subquery 2 string
  )
AS (
  row label column TEXT,
  column 1  type 1,
  column 2  type 2,
   ⋮
);
```

- Subquery 1 must return 3 columns
  - The first is the row label values
  - The second is the column label values
  - The third is the particular data values
- Subquery 2 returns a single column
  - What column categories should rows be placed into?

PIECES OF THE CROSSTAB

- The `crosstab` function has a lot going on, so let's break things down

```
SELECT *
FROM crosstab(
  subquery 1 string,
  subquery 2 string
  )
AS (
  row label column TEXT,
  column 1  type 1,
  column 2  type 2,
   ⋮
);
```

- Subquery 1 must return 3 columns
  - The first is the row label values
  - The second is the column label values
  - The third is the particular data values
- Subquery 2 returns a single column
  - What column categories should rows be placed into?
- You still need to define the column names and data types

PIECES OF THE CROSSTAB

- The `crosstab` function has a lot going on, so let's break things down

```
SELECT *
FROM crosstab(
  subquery 1 string,
  subquery 2 string
  )
AS (
  row label column TEXT,
  column 1  type 1,
  column 2  type 2,
   ⋮
);
```

- Subquery 1 must return 3 columns
  - The first is the row label values
  - The second is the column label values
  - The third is the particular data values
- Subquery 2 returns a single column
  - What column categories should rows be placed into?
- You still need to define the column names and data types
  - First should be the row label name and data type

PIECES OF THE CROSSTAB

- The `crosstab` function has a lot going on, so let's break things down

```
SELECT *
FROM crosstab(
  subquery 1 string,
  subquery 2 string
  )
AS (
  row label column TEXT,
  column 1  type 1,
  column 2  type 2,
   ⋮
);
```

- Subquery 1 must return 3 columns
  - The first is the row label values
  - The second is the column label values
  - The third is the particular data values
- Subquery 2 returns a single column
  - What column categories should rows be placed into?
- You still need to define the column names and data types
  - First should be the row label name and data type
  - Rest should be whatever column categories you are creating

SUBQUERY STRINGS

- You must provide the subqueries to `crosstab` as strings
- Surrounding the entire query in single quotes though can be incredibly annoying if your subquery also involves strings
  - You would have to “escape” out all the interior single quotes with `\`
- Usually *far* easier to use double `$$` to indicate the start and end of the string
  - Has the side benefit of leaving the subquery perfectly capable of being selected and run independently

EXAMPLE

Let’s construct a pivot table comparing the total number of taxi rides at different days of the week across different hours.

A CASE STUDY

ON THE CASE

- It can be useful in certain situations to employ a little conditional logic *outside* a filtering statement
- Can be particularly useful to tweak or reclassify values
- Standard SQL has the `CASE` statement to accomplish this
  - Essentially works like many programming language's if…else if…else blocks

```
CASE
  WHEN some condition THEN output
  WHEN some other condition THEN diff output
   ⋮
  ELSE more diff output
END
```

A CASE STUDY

- Comparisons are made in order, so the first condition that matches, that output is used
- If you do not include an `ELSE` part, then `NULL` will be output if nothing else matches
- `CASE` statements most often show up in `SELECT` statements where column outputs are being selected, but they could potentially also show up in filtering or ordering statements
- A `CASE` statement will not evaluate results where the condition is not met, so they can be used to prevent certain errors as well (such as dividing by 0)

```
SELECT
  CASE WHEN col != 0 THEN 5/col END
FROM table;
```

EXAMPLE

Let's compute how many taxi rides were considered:

- "Time Travelers" with a duration of less than 0
- "Short Trips" with a duration less than 5 minutes
- "Medium Trips" with a duration between 5 and 15 minutes
- "Long Trips" with a duration greater than 15 minutes

COALESCE

- Similar to `CASE`, SQL also has the `COALESCE` statement
- `COALESCE` technically returns the first non-null value in the comma separate list of values
- Most frequently used to substitute a default value for `NULL` values

```
SELECT
  student_id,
  COALESCE(grade, 0)
FROM grades;
```

ACTIVITY TIME!

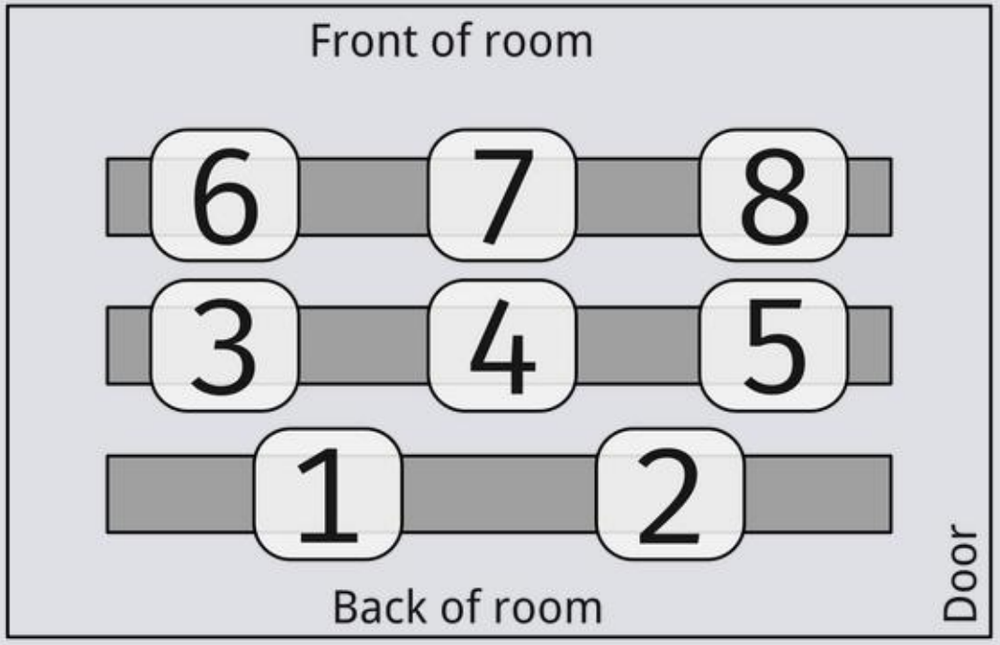
GRADE REPORTING

Using the same grade book tables from last class (here if you need them again), do the following:

- Create a pivot table for student quiz row point scores with full student names as the rows and individual quizzes as the columns. You can concatenate two strings in SQL using the `||` operator. So `fname || ' ' || lname` would give the first and last name with a space between as a single string. Feel free to use joins here.
- Create a pivot table for student letter grades on each quiz. Parts of this will look really similar to what you did in the first part, but there will be some important differences.

TODAY'S GROUPS

- Group 1: Tiffany, Jordan, Jerrick
- Group 2: Jack, Grace, Marcus
- Group 3: Mallory, Aurora, Sam J
- Group 4: Connor, Sam H, Sergio
- Group 5: Nick, Michael, Finn
- Group 6: Hannah, Dayton, Tippy
- Group 7: Matthew, Harleen, AJ
- Group 8: Greg, Evan, Haley



Group Areas