Name: Emir Dincer
Class: ECO 32500 - Python for Business Analytics
Date Due: 10/6/2024

Notes: Contains Screenshots of Answer from https://selectstarsql.com/ and MSSQL Database

Beazley's Last Statement

A First SQL Query:

A First SQL Query

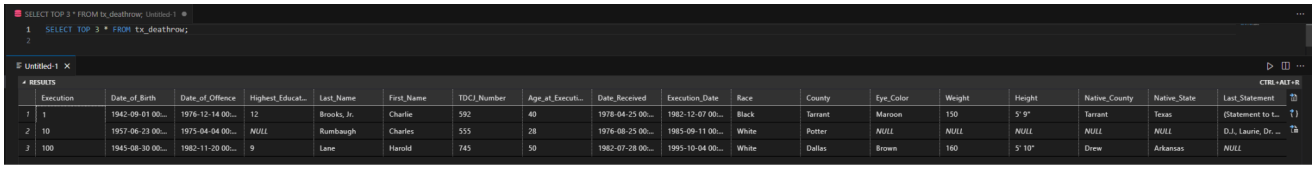
Run this query to find the first 3 rows of the 'executions' table.
Viewing a few rows is a good way to find out the columns of a table. Try to remember the column names for later use.

```
1 SELECT * FROM executions LIMIT 3
```

Run ⇩ Reset

Christopher Anthony	Young	553	34	2018-07-17	Bexar	I want to make sure the Patel family knows I love them like they love me. Make sure the kids in the world know I'm being executed and those kids I've been mentoring keep this fight going. I'm good Warden.
Danny Paul	Bible	552	66	2018-06-27	Harris	null
Juan Edward	Castillo	551	37	2018-05-16	Bexar	To everyone that has been there for me you know who you are. Love y'all. See y'all on the other side.That's it.

MSSQL:



The SELECT Block

The SELECT Block

The `SELECT` block specifies which columns you want to output. Its format is `SELECT <column>, <column>, ...`. Each column must be separated by a comma, but the space following the comma is optional. The star (ie. *) is a special character that signifies we want all the columns in the table.

In the code editor below, revise the query to select the last_statement column in addition to the existing columns.
Once you're done, you can hit Shift+Enter to run the query.

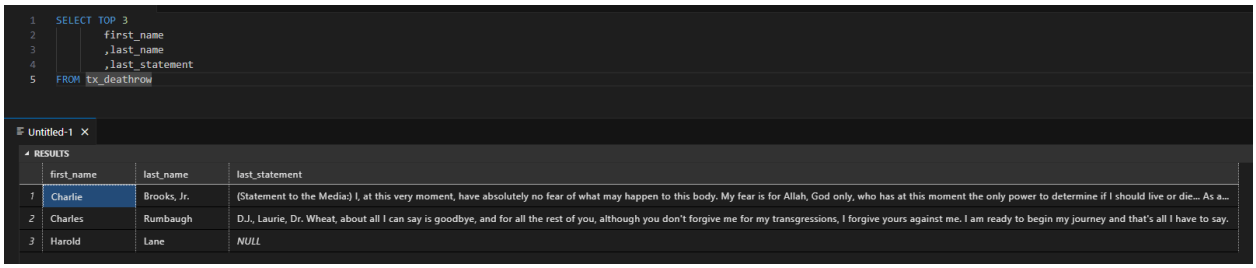
```
SELECT first_name
       ,last_name
       ,last_statement
FROM executions
LIMIT 3
```

Run ⇩ | Show Solution | Reset

Correct

first_name	last_name	last_statement
Christopher Anthony	Young	I want to make sure the Patel family knows I love them like they love me. Make sure the kids in the world know I'm being executed and those kids I've been mentoring keep this fight going. I'm good Warden.
Danny Paul	Bible	null
Juan Edward	Castillo	To everyone that has been there for me you know who you are. Love y'all. See y'all on the other side.That's it.

MSSQL:



The FROM Block

The FROM Block

The `FROM` block specifies which table we're querying from. Its format is `FROM <table>`. It always comes after the `SELECT` block.

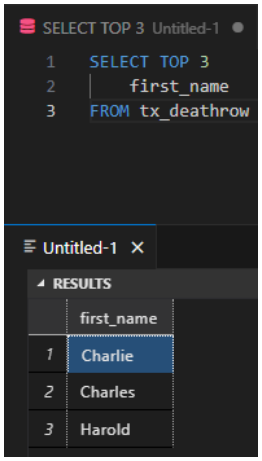
Run the given query and observe the error it produces. Fix the query.
Make it a habit to examine error messages when something goes wrong. Avoid debugging by gut feel or trial and error.

```
1 SELECT first_name FROM executions LIMIT 3
```

Run ⇩ | Show Solution | Reset

Correct

first_name
Christopher Anthony
Danny Paul
Juan Edward



Modify the query to divide 50 and 51 by 2.
SQL supports all the usual arithmetic operations.

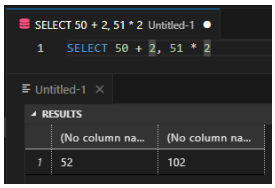
```
1 SELECT 50 + 2, 51 * 2
```

Run ⇩ | Show Solution | Reset

Incorrect

50 + 2	51 * 2
52	102

MySQL:



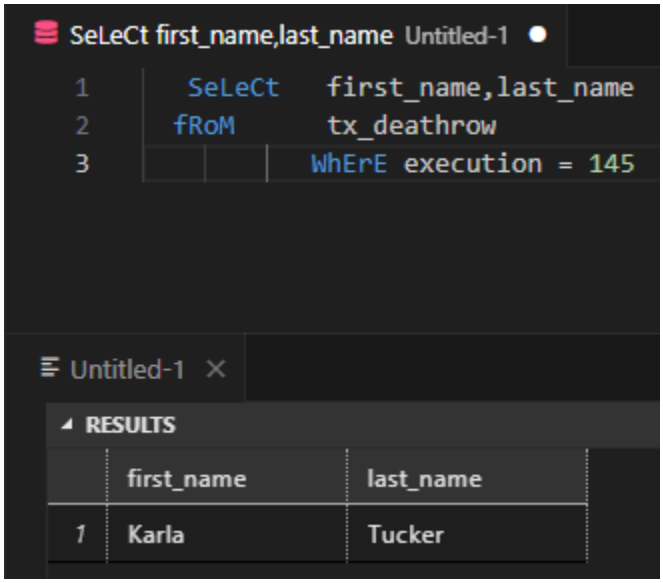
Verify that messing up capitalization and whitespace still gives a valid query.
Karla Tucker was the first woman executed in Texas since the Civil War. She was put to death for killing two people during a 1983 robbery.

```
1    SeLeCt   first_name,last_name
2   fRoM      executions
3            WhErE ex_number = 145
```

Run ⇩ Reset

first_name	last_name
Karla	Tucker

MySQL:



The WHERE Block

Find the first and last names and ages (ex_age) of inmates 25 or younger at time of execution.
Because the average time inmates spend on death row prior to execution is 10.26 years, only 6 inmates this young have been executed in Texas since 1976.

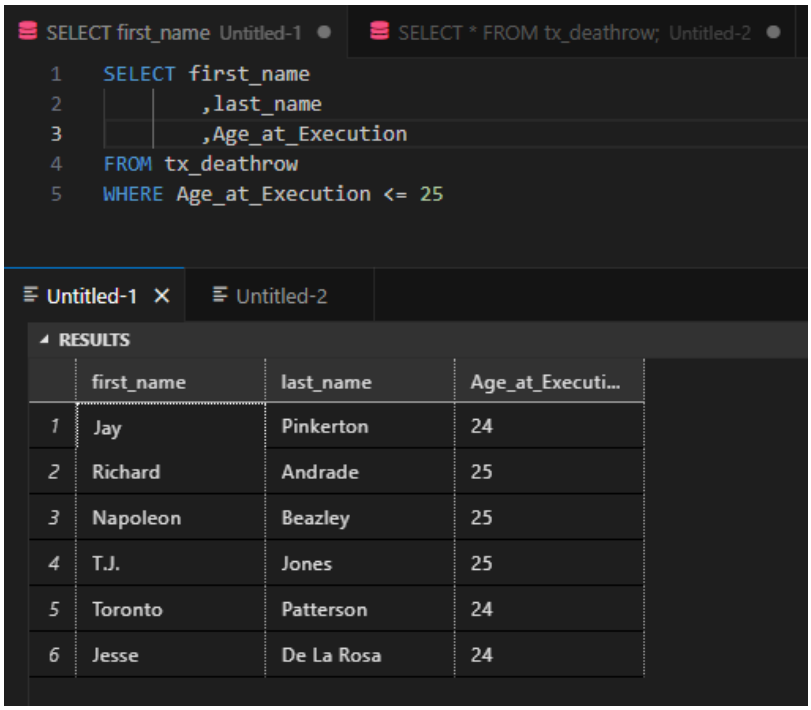
```
SELECT first_name
       ,last_name
       ,ex_age
FROM executions
WHERE ex_age <= 25
```

Run ⇩ | Show Solution | Reset

Correct

first_name	last_name	ex_age
Toronto	Patterson	24
T.J.	Jones	25
Napoleon	Beazley	25
Richard	Andrade	25
Jay	Pinkerton	24
Jesse	De La Rosa	24

MySQL:



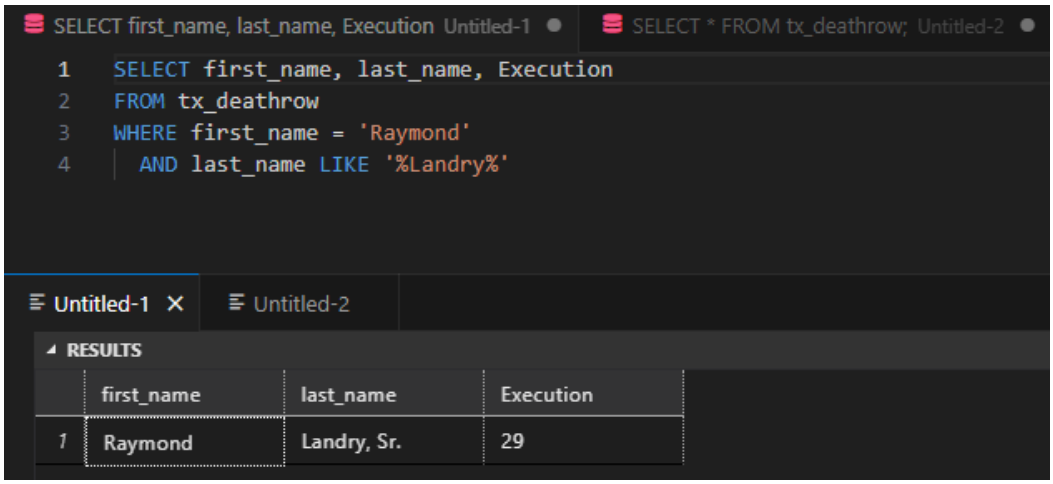
Modify the query to find the result for Raymond Landry.
You might think this would be easy since we already know his first and last name. But datasets are rarely so clean. Use the LIKE operator so you don't have to know his name perfectly to find the row.

```
1 SELECT first_name, last_name, ex_number
2 FROM executions
3 WHERE first_name = 'Raymond'
4   AND last_name LIKE '%Landry%'
```

Run ⇩ | Show Solution | Reset

Correct

first_name	last_name	ex_number
Raymond	Landry, Sr.	29



Insert a pair of parenthesis so that this statement returns 0.
Here we're relying on the fact that 1 means true and 0 means false.

```
1 SELECT 0 AND (0 OR 1)
```

Run ⇩ | Show Solution | Reset

Correct

0 AND (0 OR 1)
0

```
1  SELECT
2      CASE
3          WHEN 0 = 1 AND (0 = 1 OR 1 = 1)
4          THEN 1
5          ELSE 0
6      END AS '0 AND (0 OR 1)';
```

Untitled-1 × Untitled-2

RESULTS

	0 AND (0 OR 1)
1	0

QUIZ

Let's take a quick quiz to cement your understanding.

Select the `WHERE` blocks with valid clauses.

These are tricky. Even if you've guessed correctly, read the explanations to understand the reasoning.

- [x] WHERE 0
- [ ] WHERE ex_age == 62
- [x] WHERE ex_number < ex_age
- [ ] WHERE ex_age => 62
- [x] WHERE ex_age
- [ ] WHERE '%obert%' LIKE first_name

Check Answers | Show Explanations

All correct!

Napoleon Beazley's Last Statement

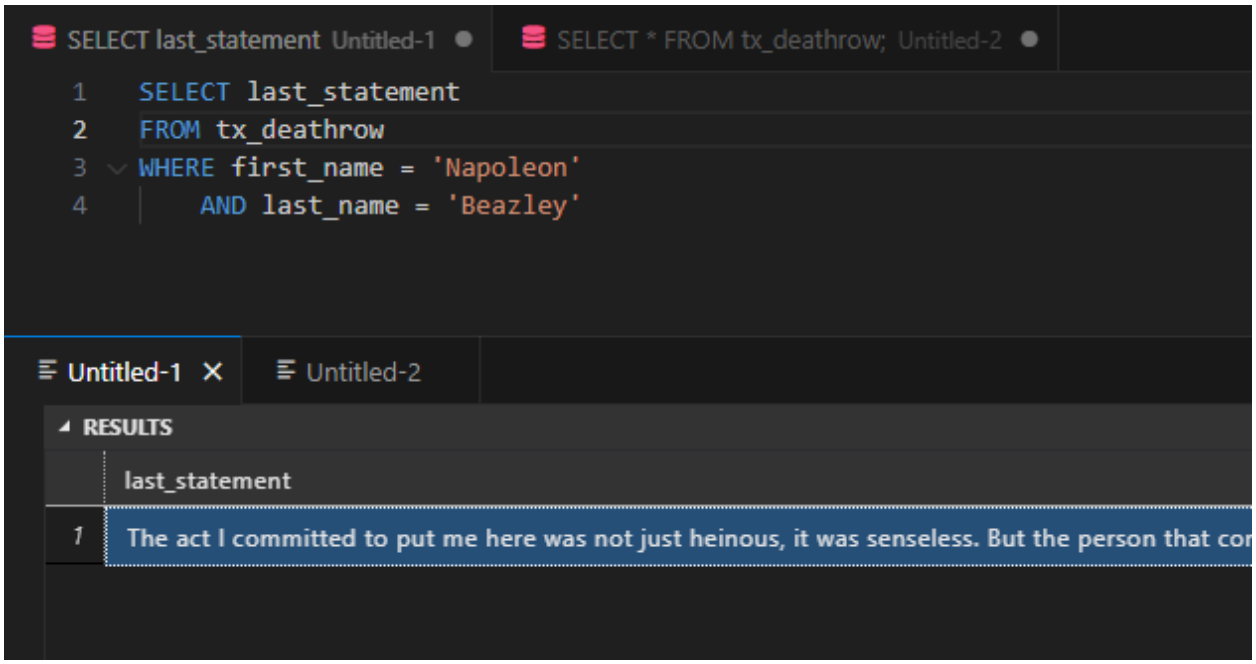
Find Napoleon Beazley's last statement.

```
1 SELECT last_statement
2 FROM executions
3 WHERE first_name = 'Napoleon'
4     AND last_name = 'Beazley'
```

Run ⇩ | Show Solution | Reset

Correct

last_statement
The act I committed to put me here was not just heinous, it was senseless. But the person that committed that act is no longer here - I am. I'm not going to struggle physically against any restraints. I'm not going to shout, use profanity or make idle threats. Understand though that I'm not only upset, but I'm saddened by what is happening here tonight. I'm not only saddened, but disappointed that a system that is supposed to protect and uphold what is just and right can be so much like me when I made the same shameful mistake. If someone tried to dispose of everyone here for participating in this killing, I'd scream a resounding, "No." I'd tell them to give them all the gift that they would not give me...and that's to give them all a second chance. I'm sorry that I am here. I'm sorry that you're all here. I'm sorry that John Luttig died. And I'm sorry that it was something in me that caused all of this to happen to begin with. Tonight we tell the world that there are no second chances in the eyes of justice...Tonight, we tell our children that in some instances, in some cases, killing is right. This conflict



Claims of Innocence
The COUNT Function

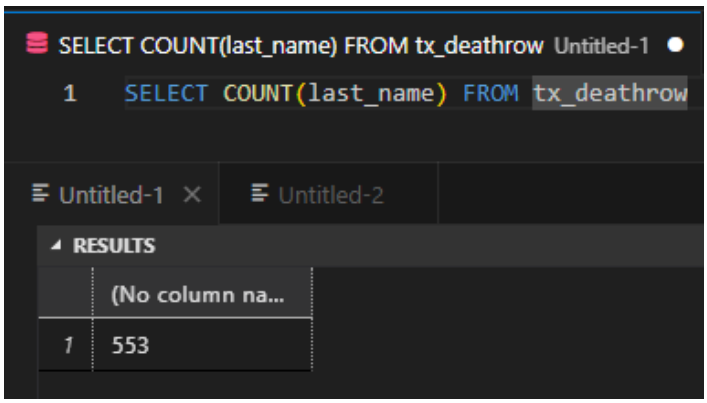
Edit the query to find how many inmates provided last statements.
We can use **COUNT** here because **NULL**s are used when there are no statements.

```
1 SELECT COUNT(last_name) FROM executions
```

Run ⇩ | Show Solution | Reset

Incorrect

COUNT(last_name)
553



Nulls

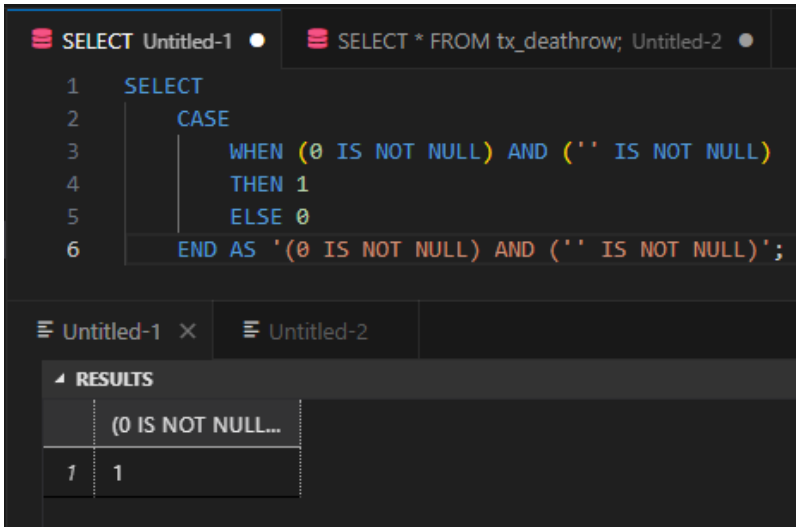
In SQL, `NULL` is the value of an empty entry. This is different from the empty string `''` and the integer `0`, both of which are *not* considered `NULL`. To check if an entry is `NULL`, use `IS` and `IS NOT` instead of `=` and `!=`.

Verify that 0 and the empty string are not considered NULL.
Recall that this is a compound clause. Both of the two `IS NOT NULL` clauses have to be true for the query to return `true`.

```
1 SELECT (0 IS NOT NULL) AND ('' IS NOT NULL)
```

Run ⇩ Reset

(0 IS NOT NULL) AND (" IS NOT NULL)
1



Find the total number of executions in the dataset.

The idea here is to pick one of the columns that you're confident has no **NULL**s and count it.

```
1 SELECT COUNT(ex_number) FROM executions
```

Run ⇩ | Show Solution | Reset

Correct

COUNT(ex_number)
553

```
1   SELECT COUNT(Execution) FROM tx_deathrow
```

Untitled-1 × | Untitled-2

RESULTS

	(No column na...
1	553

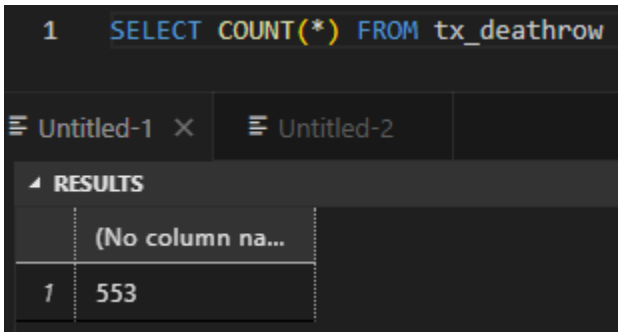
Variations on COUNT

Verify that `COUNT(*)` gives the same result as before.

```
1 SELECT COUNT(*) FROM executions
```

Run ⇩ Reset

COUNT(*)
553



This query counts the number of Harris and Bexar county executions. Replace SUMs with COUNTs and edit the CASE WHEN blocks so the query still works.

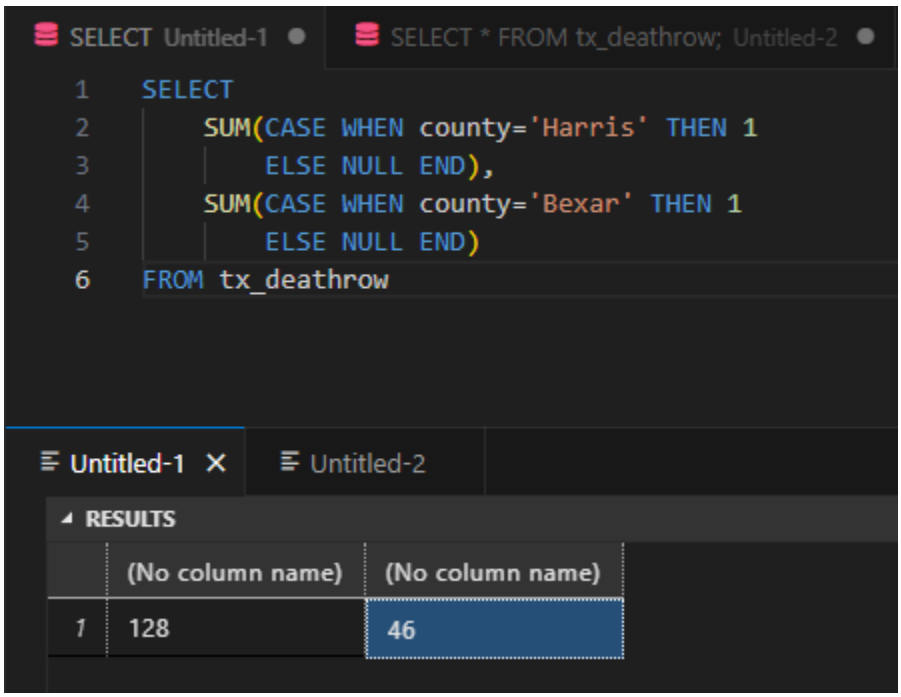
Switching **SUM** for **COUNT** alone isn't enough because **COUNT** still counts the 0 since 0 is non-null.

```
SELECT
    SUM(CASE WHEN county='Harris' THEN 1
        ELSE NULL END),
    SUM(CASE WHEN county='Bexar' THEN 1
        ELSE NULL END)
FROM executions
```

Run ⇩ | Show Solution | Reset

Correct

SUM(CASE WHEN county='Harris' THEN 1 ELSE NULL END)	SUM(CASE WHEN county='Bexar' THEN 1 ELSE NULL END)
128	46

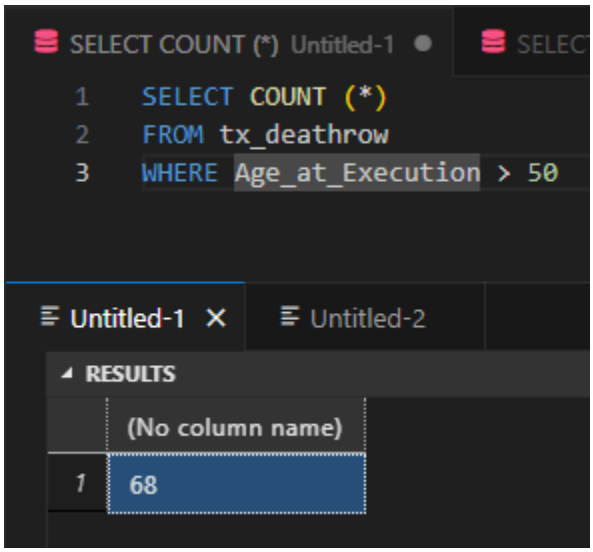


Practice

Find how many inmates were over the age of 50 at execution time.
This illustrates that the **WHERE** block filters before aggregation occurs.

```
1 SELECT COUNT (*)
2 FROM executions
3 WHERE ex_age > 50
```

Run ⇩ | Show Solution | Reset

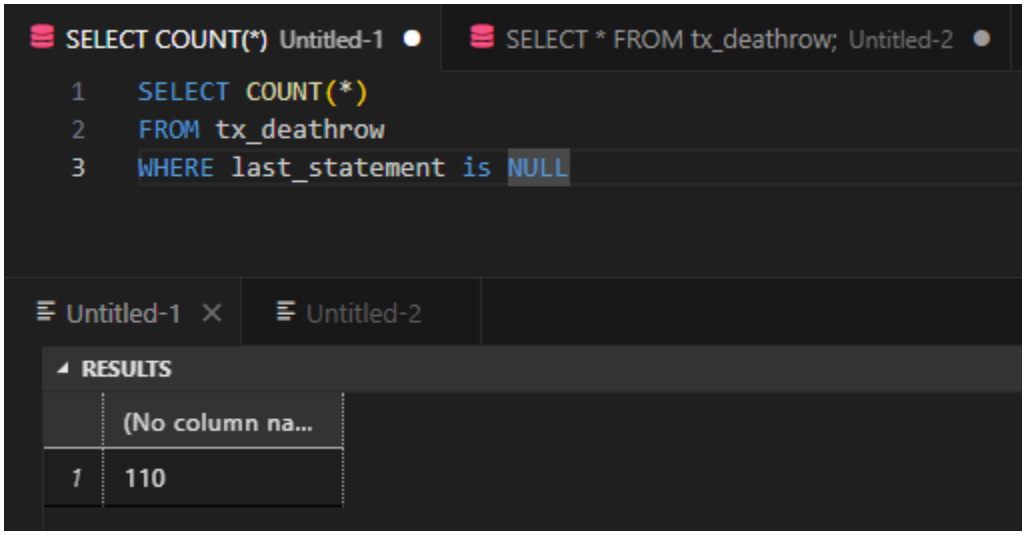


Find the number of inmates who have declined to give a last statement.
For bonus points, try to do it in 3 ways:
1) With a **WHERE** block,
2) With a **COUNT** and **CASE WHEN** block,
3) With two **COUNT** functions.

```
1 SELECT COUNT(*)
2 FROM executions
3 WHERE last_statement is NULL
```

Run ⇩ | Show Solution | Reset

COUNT(*)
110



Find the minimum, maximum and average age of inmates at the time of execution.

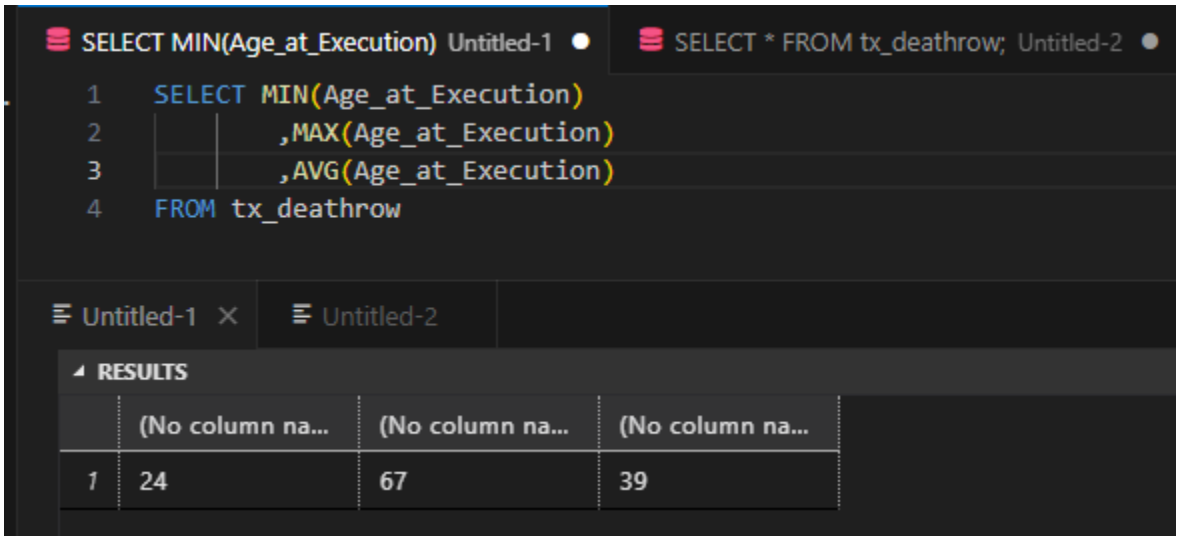
Use the **MIN**, **MAX**, and **AVG** aggregate functions.

```
SELECT MIN(ex_age)
       ,MAX(ex_age)
       ,AVG(ex_age)
FROM executions
```

Run ⇩ | Show Solution | Reset

Correct

MIN(ex_age)	MAX(ex_age)	AVG(ex_age)
24	67	39.47016274864376



Find the average length (based on character count) of last statements in the dataset.

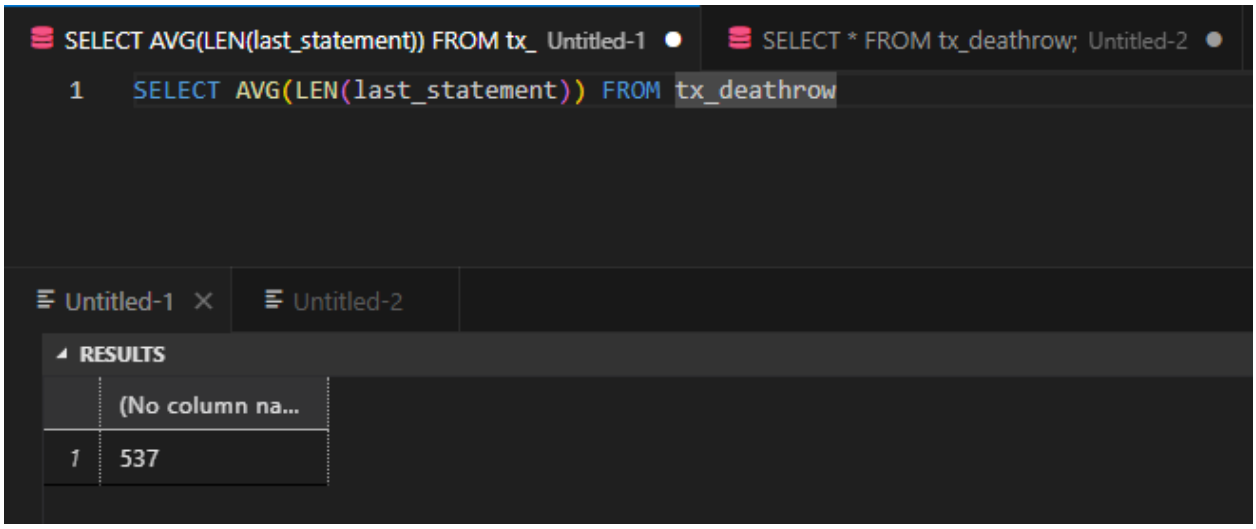
This exercise illustrates that you can compose functions. Look up the documentation to figure out which function which returns the number of characters in a string.

```
1 SELECT AVG(LENGTH(last_statement)) FROM executions
```

Run ⇩ | Show Solution | Reset

Correct

AVG(LENGTH(last_statement))
537.492099322799



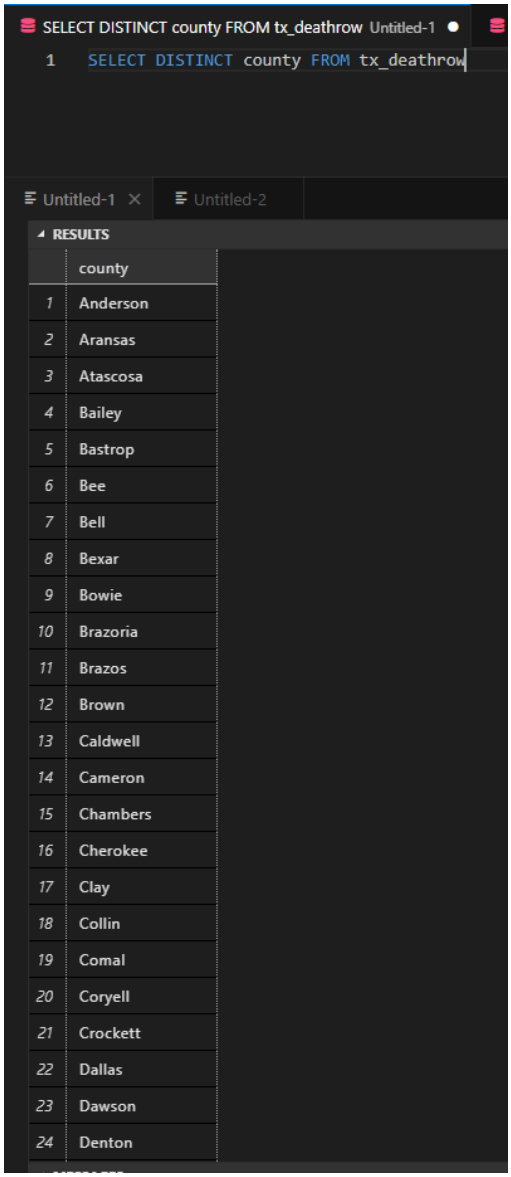
List all the counties in the dataset without duplication.
We can get unique entries by using **`SELECT DISTINCT`**. See documentation.

```
1 SELECT DISTINCT county FROM executions
```

Run ⇩ | Show Solution | Reset

Correct

county
Bexar
Harris
Tarrant
Lubbock
Dallas
Hidalgo



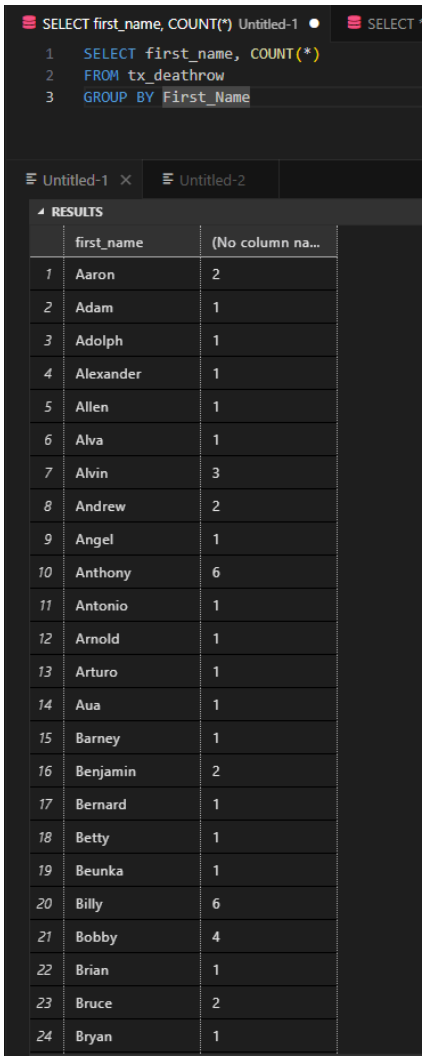
A Strange Query

Let's try it anyway and see what happens.

```
1 SELECT first_name, COUNT(*) FROM executions
```

Run ⇩ Reset

first_name	COUNT(*)
Charlie	553



```
1  SELECT first_name, COUNT(*)
2  FROM tx_deathrow
3  GROUP BY First_Name
```

Untitled-1 | Untitled-2

RESULTS

	first_name	(No column na...
1	Aaron	2
2	Adam	1
3	Adolph	1
4	Alexander	1
5	Allen	1
6	Alva	1
7	Alvin	3
8	Andrew	2
9	Angel	1
10	Anthony	6
11	Antonio	1
12	Arnold	1
13	Arturo	1
14	Aua	1
15	Barney	1
16	Benjamin	2
17	Bernard	1
18	Betty	1
19	Beunka	1
20	Billy	6
21	Bobby	4
22	Brian	1
23	Bruce	2
24	Bryan	1

Conclusion and Recap

Conclusion and Recap

Let's use what we've learned so far to complete our task:

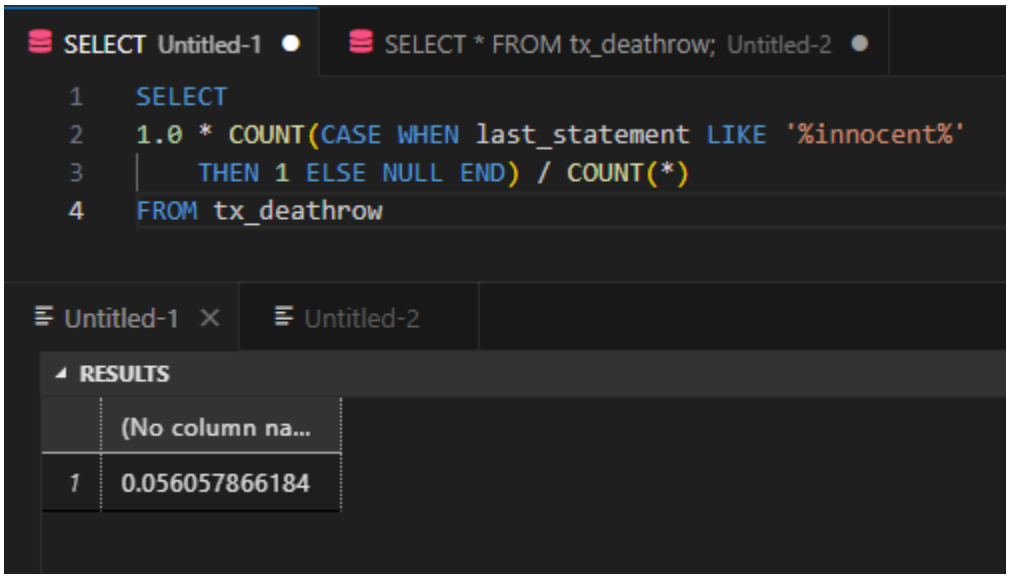
Find the proportion of inmates with claims of innocence in their last statements.
To do decimal division, ensure that one of the numbers is a decimal by multiplying it by 1.0. Use `LIKE '%innocent%'` to find claims of innocence.

```
SELECT
1.0 * COUNT(CASE WHEN last_statement LIKE '%innocent%'
    THEN 1 ELSE NULL END) / COUNT(*)
FROM executions
```

Run ⇩ | Show Solution | Reset

Correct

1.0 * COUNT(CASE WHEN last_statement LIKE '%innocent%' THEN 1 ELSE NULL END) / COUNT(*)
0.05605786618444846



Long Tails

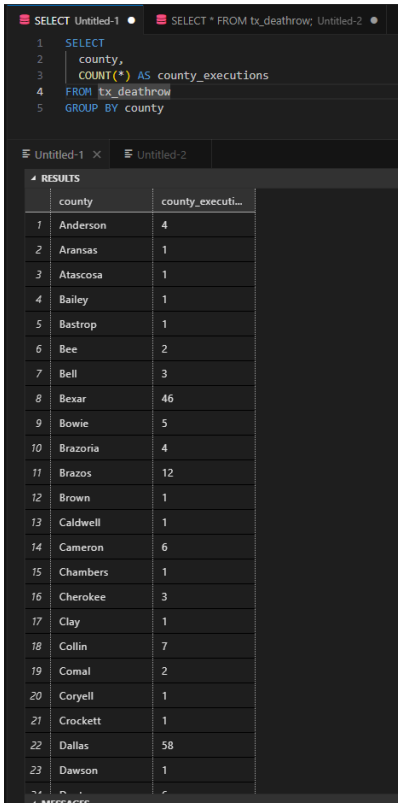
The GROUP BY Block

This query pulls the execution counts per county.

```
SELECT
  county,
  COUNT(*) AS county_executions
FROM executions
GROUP BY county
```

Run ⇩ Reset

county	county_executions
Anderson	4
Aransas	1
Atascosa	1
Bailey	1
Bastrop	1
Bee	2
Bell	3



This query counts the executions with and without last statements. Modify it to further break it down by county.

The clause `last_statement IS NOT NULL` acts as an indicator variable where 1 means true and 0 means false.

```
SELECT
  last_statement IS NOT NULL AS has_last_statement,
  county,
  COUNT(*)
FROM executions
GROUP BY has_last_statement, county
```

Run ⇩ | Show Solution | Reset

Correct

has_last_statement	county	COUNT(*)
0	Bell	1
0	Bexar	9
0	Brazoria	1
0	Brazos	2
0	Chambers	1
0	Collin	1

The HAVING Block

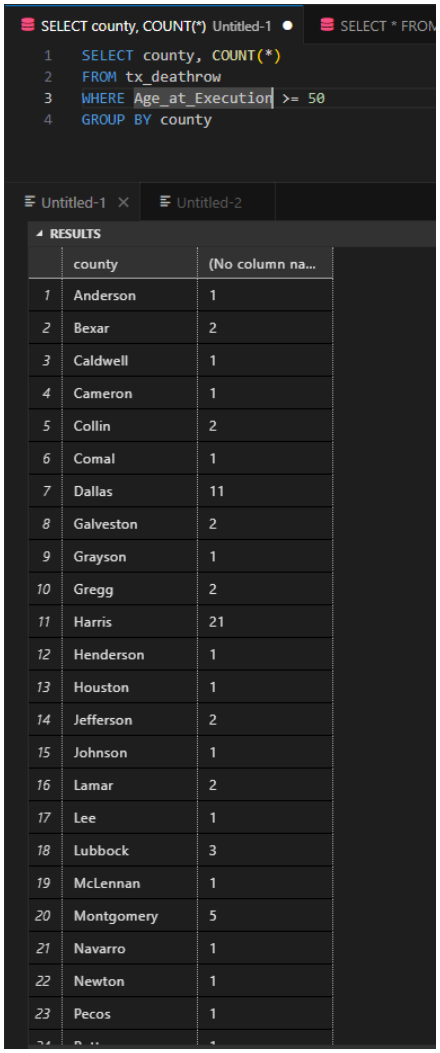
Count the number of inmates aged 50 or older that were executed in each county.
You should be able to do this using `CASE WHEN`, but try using the `WHERE` block here.

```
SELECT county, COUNT(*)
FROM executions
WHERE ex_age >= 50
GROUP BY county
```

Run ⇩ | Show Solution | Reset

Correct

county	COUNT(*)
Anderson	1
Bexar	2
Caldwell	1
Cameron	1
Collin	2
Comal	1



List the counties in which more than 2 inmates aged 50 or older have been executed.

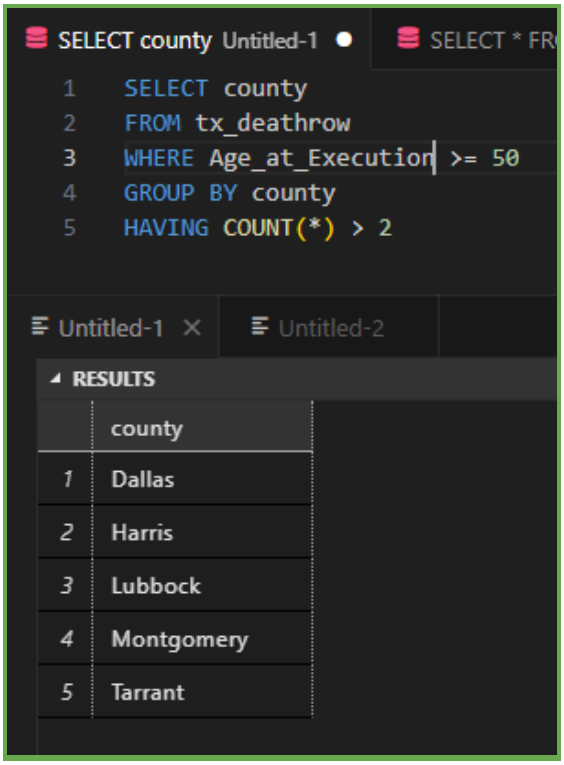
This builds on the previous exercise. We need an additional filter—one that uses the result of the aggregation. This means it cannot exist in the **WHERE** block because those filters are run before aggregation. Look up the **HAVING** block. You can think of it as a post-aggregation **WHERE** block.

```
1 SELECT county
2 FROM executions
3 WHERE ex_age >= 50
4 GROUP BY county
5 HAVING COUNT(*) > 2
```

Run ⇩ | Show Solution | Reset

Correct

county
Dallas
Harris
Lubbock
Montgomery
Tarrant



PRACTICE

Mark the statements that are true.
This query finds the number of inmates from each county and 10 year age range.

```
SELECT
  county,
  ex_age/10 AS decade_age,
  COUNT(*)
FROM executions
GROUP BY county, decade_age
```

- [x] The query is valid (ie. won't throw an error when run).
- [x] The query would return more rows if we were to use **`ex_age`** instead of **`ex_age/10`**.
- [x] The output will have as many rows as there are unique combinations of counties and decade_ages in the dataset.
- [ ] The output will have a group ('Bexar', 6) even though no Bexar county inmates were between 60 and 69 at execution time.
- [ ] The output will have a different value of county for every row it returns.
- [ ] The output can have groups where the count is 0.
- [x] The query would be valid even if we don't specify **`county`** in the **`SELECT`** block.
- [ ] It is reasonable to add **`last_name`** to the **`SELECT`** block even without grouping by it.

Check Answers | Show Explanations

All correct!

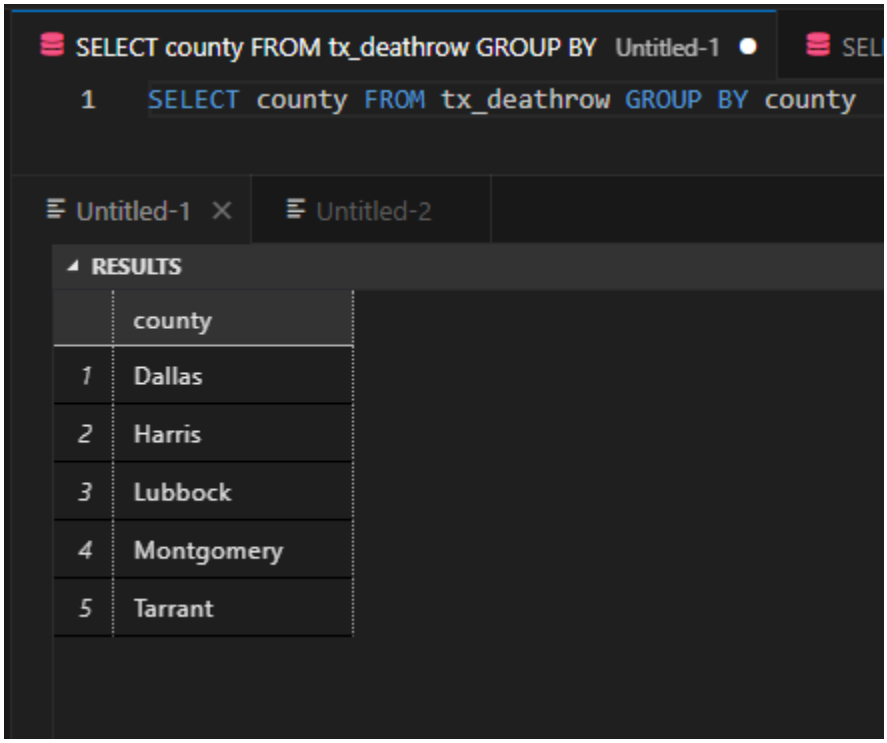
List all the distinct counties in the dataset.
We did this in the previous chapter using the **`SELECT DISTINCT`** command. This time, stick with vanilla **`SELECT`** and use **`GROUP BY`**.

```
1 SELECT county FROM executions GROUP BY county
```

Run ⇩ | Show Solution | Reset

Correct

county
Anderson
Aransas
Atascosa
Bailey
Bastrop
Bee



Nested Queries

Find the first and last name of the inmate with the longest last statement (by character count).

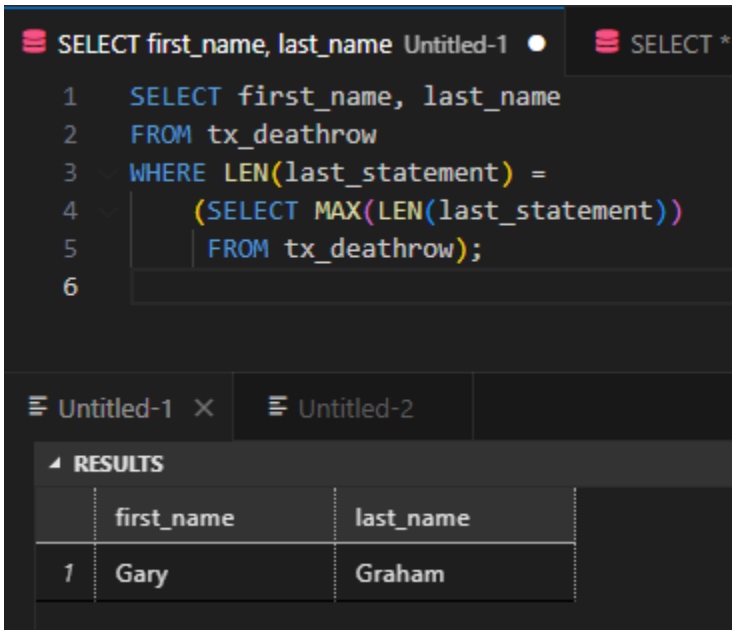
Write in a suitable query to nest in <**length-of-longest-last-statement**>.

```
1 SELECT first_name, last_name
2 FROM executions
3 WHERE LENGTH(last_statement) =
4     (SELECT MAX(LENGTH(last_statement))
5      FROM executions)
```

Run ⇩ | Show Solution | Reset

Correct

first_name	last_name
Gary	Graham



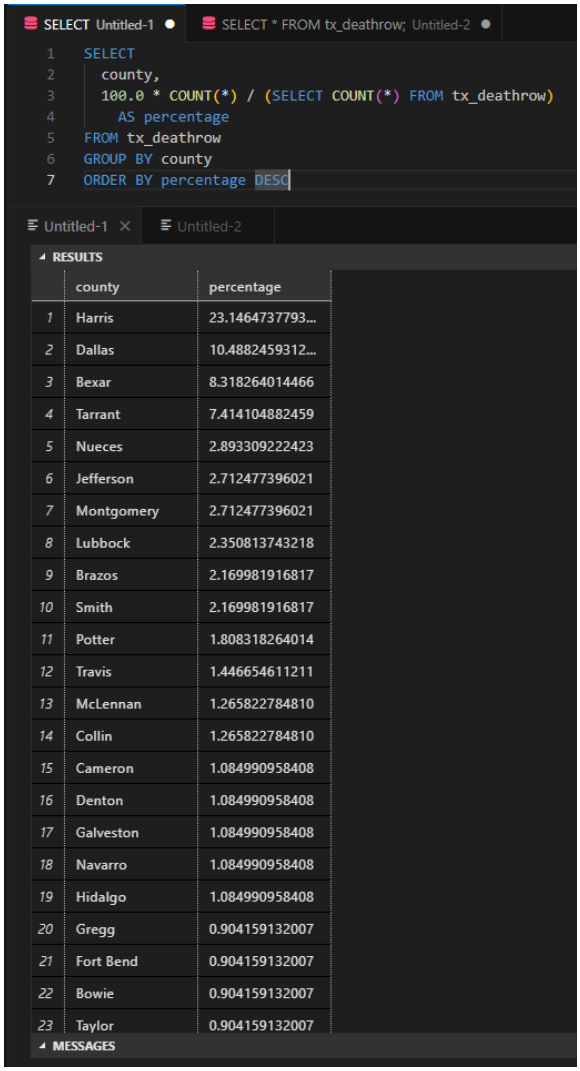
Insert the `<count-of-all-rows>` query to find the percentage of executions from each county.
`100.0` is a decimal so we can get decimal percentages.

```sql
SELECT
  county,
  100.0 * COUNT(*) / (SELECT COUNT(*) FROM executions)
    AS percentage
FROM executions
GROUP BY county
ORDER BY percentage DESC
```

Run ⇩ | Show Solution | Reset

Correct

county	percentage
Harris	23.146473779385172
Dallas	10.488245931283906
Bexar	8.318264014466546
Tarrant	7.414104882459313
Nueces	2.8933092224231465
Jefferson	2.7124773960216997



Execution Hiatuses

Hiatuses

Mark the true statements.

Suppose we have tableA with 3 rows and tableB with 5 rows.

- [x] `tableA JOIN tableB ON 1` returns 15 rows.
- [x] `tableA JOIN tableB ON 0` returns 0 rows.
- [x] `tableA LEFT JOIN tableB ON 0` returns 3 rows.
- [x] `tableA OUTER JOIN tableB ON 0` returns 8 rows.
- [x] `tableA OUTER JOIN tableB ON 1` returns 15 rows.

Check Answers | Show Explanations

All correct!

Dates

Look up the documentation to fix the query so that it returns the number of days between the dates.

```
1('1993-08-10') - JULIANDAY('1989-07-07') AS day_differen
```

Run ⇩ | Show Solution | Reset

Correct

day_difference
1495

Self Joins

Write a query to produce the `previous` table.
Remember to use aliases to form the column names **(`ex_number`, `last_ex_date`)**. Hint: Instead of shifting dates back, you could shift **`ex_number`** forward!

```
SELECT
  ex_number + 1 AS ex_number,
  ex_date AS last_ex_date
FROM executions
WHERE ex_number < 553
```

Run ⇩ | Show Solution | Reset

Correct

ex_number	last_ex_date
553	2018-06-27
552	2018-05-16
551	2018-04-25
550	2018-03-27
549	2018-02-01
548	2018-01-30

SELECT Untitled-1 ● | SELECT * FROM tx_deathr

```
SELECT
  Execution + 1 AS ex_number,
  Execution_Date AS last_ex_date
FROM tx_deathrow
WHERE Execution < 553
```

Untitled-1 × | Untitled-2

RESULTS

	ex_number	last_ex_date
1	2	1982-12-07 00:...
2	11	1985-09-11 00:...
3	101	1995-10-04 00:...
4	102	1995-12-06 00:...
5	103	1995-12-07 00:...
6	104	1995-12-11 00:...
7	105	1995-12-12 00:...
8	106	1996-02-09 00:...
9	107	1996-02-27 00:...
10	108	1996-09-18 00:...
11	109	1997-02-10 00:...
12	110	1997-03-12 00:...
13	12	1986-03-12 00:...
14	111	1997-04-02 00:...
15	112	1997-04-03 00:...
16	113	1997-04-14 00:...
17	114	1997-04-16 00:...
18	115	1997-04-21 00:...
19	116	1997-04-29 00:...
20	117	1997-05-06 00:...
21	118	1997-05-13 00:...
22	119	1997-05-16 00:...

Nest the query which generates the `previous` table into the template.
Notice that we are using a table alias here, naming the result of the nested query "previous".

```
SELECT
  last_ex_date AS start,
  ex_date AS end,
  JULIANDAY(ex_date) - JULIANDAY(last_ex_date) AS day_d:
FROM executions
JOIN (
    SELECT
      ex_number + 1 AS ex_number,
      ex_date AS last_ex_date
    FROM executions
  ) previous
  ON executions.ex_number = previous.ex_number
ORDER BY day_difference DESC
LIMIT 10
```

Run ⇩ | Show Solution | Reset

Correct

start	end	day_difference
1982-12-07	1984-03-14	463
1988-01-07	1988-11-03	301
2007-09-25	2008-06-11	260
1990-07-18	1991-02-26	223
1984-03-31	1984-10-30	213
1996-02-27	1996-09-18	204

SELECT Untitled-1 ● | SELECT * FROM tx_deathrow Untitled-2 ●

```
SELECT
  date_of_offence AS start,
  execution_date AS [end],
  DATEDIFF(DAY, date_of_offence, execution_date) AS day_difference
FROM tx_deathrow
JOIN (
    SELECT
      Execution + 1 AS ex_number,
      execution_date AS last_ex_date
    FROM tx_deathrow
) previous
ON tx_deathrow.Execution = previous.ex_number
ORDER BY day_difference DESC
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;
```

Untitled-1 ✕ | Untitled-2

RESULTS

	start	end	day_difference
1	1979-05-27 00:...	2018-06-27 00:...	14276
2	1978-05-18 00:...	2010-06-15 00:...	11716
3	1983-10-08 00:...	2015-06-03 00:...	11561
4	1987-12-24 00:...	2017-03-14 00:...	10673
5	1980-10-13 00:...	2007-03-07 00:...	9641
6	1992-04-16 00:...	2018-01-18 00:...	9408
7	1990-12-09 00:...	2016-02-16 00:...	9200
8	1982-02-02 00:...	2007-02-27 00:...	9156
9	1974-05-11 00:...	1999-03-30 00:...	9089
10	1975-02-26 00:...	1999-12-15 00:...	9058

Fill in the `JOIN ON` clause to complete a more elegant version of the previous query.

Note that we still need to give one copy an alias to ensure that we can refer to it unambiguously.

```
SELECT
  previous.ex_date AS start,
  executions.ex_date AS end,
  JULIANDAY(executions.ex_date) - JULIANDAY(previous.ex_
    AS day_difference
FROM executions
JOIN executions previous
  ON executions.ex_number = previous.ex_number + 1
ORDER BY day_difference DESC
LIMIT 10
```

Run ⇩ | Show Solution | Reset

Correct

start	end	day_difference
1982-12-07	1984-03-14	463
1988-01-07	1988-11-03	301
2007-09-25	2008-06-11	260
1990-07-18	1991-02-26	223
1984-03-31	1984-10-30	213
1996-02-27	1996-09-18	204

```
SELECT
  previous.Execution_Date AS start,
  tx_deathrow.Execution_Date AS [end],
  DATEDIFF(DAY, previous.Execution_Date, tx_deathrow.Execution_Date) AS day_difference
FROM tx_deathrow
JOIN tx_deathrow previous
  ON tx_deathrow.Execution = previous.Execution + 1
ORDER BY day_difference DESC
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;
```

Untitled-1 | Untitled-2

RESULTS

	start	end	day_difference
1	1982-12-07 00:...	1984-03-14 00:...	463
2	1988-01-07 00:...	1988-11-03 00:...	301
3	2007-09-25 00:...	2008-06-11 00:...	260
4	1990-07-18 00:...	1991-02-26 00:...	223
5	1984-03-31 00:...	1984-10-30 00:...	213
6	1996-02-27 00:...	1996-09-18 00:...	204
7	1985-09-11 00:...	1986-03-12 00:...	182
8	2016-04-06 00:...	2016-10-05 00:...	182
9	2014-04-16 00:...	2014-09-10 00:...	147
10	1996-09-18 00:...	1997-02-10 00:...	145