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.

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

The SELECT Block

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
```

The FROM 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.

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

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

```
SELECT 50 + 2, 51 / 2
```

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.

```
 SeLeCt  first_name,last_name
fRoM    executions
    WhErE ex_number = 145
```

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
```

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.

```
```

```
SELECT first_name, last_name, ex_number
FROM executions
WHERE first_name LIKE '%Raymond%'
 AND last_name LIKE '%Landry%'
```

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.

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

Find Napoleon Beazley's last statement.

```
SELECT last_statement
FROM executions
Where first_name = 'Napoleon'
      AND last_name = 'Beazley'
```

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.

```
```

```
SELECT COUNT(last_statement) FROM executions
```

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.

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

This query counts the number of Harris and Bexar county executions. Replace `SUM`s with `COUNT`s 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

  COUNT(CASE WHEN county='Harris' THEN 1

     ELSE NULL END),

  COUNT(CASE WHEN county='Bexar' THEN 1

     ELSE NULL END)

FROM executions
```

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
```

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.

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

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

```
SELECT DISTINCT county FROM executions
```

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
```

The GROUP BY Block

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,
 COUNT(*)
FROM executions
GROUP BY has_last_statement, county
```

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
```

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.

```
SELECT county, COUNT(*)
FROM executions
WHERE ex_age >=50
GROUP BY county
HAVING COUNT(*) >2
```

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`.

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

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>`.

```
```

```
SELECT first_name, last_name
FROM executions
WHERE LENGTH(last_statement) =
  (SELECT MAX(LENGTH(last_statement))
    FROM executions)
```

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.

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

Dates

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

```
SELECT JULIANDAY('1993-08-10') - JULIANDAY('1989-07-07') AS day_difference
```

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
```

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_difference
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
```

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_date)
  AS day_difference
FROM executions
JOIN executions previous
 ON executions.ex_number = previous.ex_number + 1
ORDER BY day_difference DESC
LIMIT 10
```