

# Tarea 1

Alumno: Josué Samuel Argueta Hernández 211024

## Ejercicio 1: Tutorial de SQL (20 puntos)

SQL es un lenguaje bastante común en bases de datos, por lo que existe mucho material de apoyo para su aprendizaje en línea.

Esta parte de la tarea consistirá en que lleve a cabo el siguiente tutorial para repasar los conceptos básicos de SQL vistos en la primera parte del curso:

<https://sqlbolt.com/>

Debe completar todas las lecciones del tutorial interactivo (Interactive tutorial) y leer los tópicos de Subqueries y Unions, Intersections & Exceptions.

En su entrega de la tarea debe registrar pantallazos de cada uno de los ejercicios realizados en el sitio web.

### SQL LESSON 1: SELECT QUERIES 101

Table: Movies

Title
Toy Story
A Bug's Life
Toy Story 2
Monsters, Inc.
Finding Nemo
The Incredibles
Cars
Ratatouille
WALL·E
Up

`SELECT title FROM movies;`

Exercise 1 — Tasks

- Find the title of each film ✓
- Find the director of each film
- Find the title and director of each film
- Find the title and year of each film
- Find all the information about each film

Stuck? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

Next → SQL Lesson 2: Queries with constraints (Pt. 1)  
Previous ← Introduction to SQL

Table: Movies

Director
John Lasseter
John Lasseter
John Lasseter
Pete Docter
Andrew Stanton
Brad Bird
John Lasseter
Brad Bird
Andrew Stanton
Pete Docter

`SELECT director FROM movies;`

Exercise 1 — Tasks

- Find the title of each film ✓
- Find the director of each film
- Find the title and director of each film
- Find the title and year of each film
- Find all the information about each film

Stuck? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

Next → SQL Lesson 2: Queries with constraints (Pt. 1)  
Previous ← Introduction to SQL

Table: Movies

Title	Director
Toy Story	John Lasseter
A Bug's Life	John Lasseter
Toy Story 2	John Lasseter
Monsters, Inc.	Pete Docter
Finding Nemo	Andrew Stanton
The Incredibles	Brad Bird
Cars	John Lasseter
Ratatouille	Brad Bird
WALL·E	Andrew Stanton
Up	Pete Docter

`SELECT title, director FROM movies;`

Exercise 1 — Tasks

- Find the title of each film ✓
- Find the director of each film ✓
- Find the title and director of each film ✓
- Find the title and year of each film
- Find all the information about each film

Stuck? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

Next → SQL Lesson 2: Queries with constraints (Pt. 1)  
Previous ← Introduction to SQL

Table: Movies

Title	Year
Toy Story	1995
A Bug's Life	1998
Toy Story 2	1999
Monsters, Inc.	2001
Finding Nemo	2003
The Incredibles	2004
Cars	2006
Ratatouille	2007
WALL·E	2008
Up	2009

`SELECT title, year FROM movies;`

Exercise 1 — Tasks

- Find the title of each film ✓
- Find the director of each film ✓
- Find the title and director of each film
- Find the title and year of each film ✓
- Find all the information about each film

Stuck? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

Next → SQL Lesson 2: Queries with constraints (Pt. 1)  
Previous ← Introduction to SQL

first exercise will only involve the **Movies** table, and the default query below currently shows all the properties of each movie. To continue onto the next lesson, alter the query to find the exact information we need for each task.

Table: Movies

<b>Id</b>	<b>Title</b>	<b>Director</b>	<b>Year</b>	<b>Length_minutes</b>
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116
7	Cars	John Lasseter	2006	117
8	Ratatouille	Brad Bird	2007	115
9	WALL-E	Andrew Stanton	2008	104
10	Up	Pete Docter	2009	101

```
SELECT * FROM movies;
```

Exercise 1 — Tasks

- Find the title of each film ✓
- Find the director of each film ✓
- Find the title and director of each film ✓
- Find the title and year of each film ✓
- Find all the information about each film ✓

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

Continue >

## SQL Lesson 2: Queries with constraints (Pt. 1)

Using the right constraints, find the information we need from the **Movies** table for each task below.

Table: Movies

<b>Id</b>	<b>Title</b>
6	The Incredibles

Exercise 2 — Tasks

- Find the movie with a row id of 6 ✓
- Find the movies released in the year between 2000 and 2010
- Find the movies not released in the years between 2000 and 2010
- Find the first 5 Pixar movies and their release year

```
SELECT id, title FROM movies
WHERE id = 6;
```

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

Next - SQL Lesson 3: Queries with constraints (Pt. 2)

Find SQLBolt useful? Please consider [Donating \(\\$4\) via PayPal](#) to support our site.

Using the right constraints, find the information we need from the **Movies** table for each task below.

Table: Movies

<b>Title</b>	<b>Year</b>
Toy Story	1995
A Bug's Life	1998
Toy Story 2	1999
Cars	2001
Brave	2012
Monsters University	2013

Exercise 2 — Tasks

- Find the movie with a row id of 6 ✓
- Find the movies released in the years between 2000 and 2010 ✓
- Find the movies not released in the years between 2000 and 2010 ✓
- Find the first 5 Pixar movies and their release year

```
SELECT title, year FROM movies
WHERE year < 2000 OR year > 2010;
```

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

Find SQLBolt useful? Please consider [Donating \(\\$4\) via PayPal](#) to support our site.

Using the right constraints, find the information we need from the **Movies** table for each task below.

Table: Movies

<b>Id</b>	<b>Title</b>	<b>Director</b>	<b>Year</b>	<b>Length_minutes</b>
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107

Exercise 2 — Tasks

- Find the movie with a row id of 6 ✓
- Find the movies released in the years between 2000 and 2010 ✓
- Find the movies not released in the years between 2000 and 2010 ✓
- Find the first 5 Pixar movies and their release year ✓

```
SELECT * FROM movies
WHERE id < 6;
```

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

Continue >

RESET

Next - SQL Lesson 3: Queries with constraints (Pt. 2)

Previous - SQL Lesson 1: SELECT queries 101

Find SQLBolt useful? Please consider [Donating \(\\$4\) via PayPal](#) to support our site.

## SQL Lesson 3: Queries with constraints (Pt. 2)

Table: Movies

<b>Title</b>	<b>Director</b>
Toy Story	John Lasseter
Toy Story 2	John Lasseter
Toy Story 3	Lee Unkrich

Exercise 3 — Tasks

- Find all the Toy Story movies ✓
- Find all the movies directed by John Lasseter
- Find all the movies (and director) not directed by John Lasseter
- Find all the WALL-E movies

```
SELECT title, director FROM movies
WHERE title LIKE "Toy Story%";
```

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

Find SQLBolt useful? Please consider [Donating \(\\$4\) via PayPal](#) to support our site.

Table: Movies

<b>Title</b>	<b>Director</b>
Toy Story	John Lasseter
A Bug's Life	John Lasseter
Toy Story 2	John Lasseter
Cars	John Lasseter
Cars 2	John Lasseter

Exercise 3 — Tasks

- Find all the Toy Story movies ✓
- Find all the movies directed by John Lasseter
- Find all the movies (and director) not directed by John Lasseter
- Find all the WALL-E movies

```
SELECT title, director FROM movies
WHERE director = "John Lasseter";
```

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

Find SQLBolt useful? Please consider [Donating \(\\$4\) via PayPal](#) to support our site.

Table: Movies

Title	Director
Monsters, Inc.	Pete Docter
Finding Nemo	Andrew Stanton
The Incredibles	Brad Bird
Toy Story	Brad Bird
WALL-E	Andrew Stanton
Up	Pete Docter
Toy Story 3	Lee Unkrich
Birds	Brenda Chapman
Monsters University	Dan Scanlon
WALL-E	Brenda Chapman

```
SELECT title, director FROM movies
WHERE director = "John Lasseter";
```

Exercise 3 — Tasks

- Find all the Toy Story movies ✓
- List all the movies directed by John Lasseter ✓
- Find all the movies (and director) not directed by John Lasseter ✓
- Find all the WALL-E movies

Start? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

Next → SQL Lesson 4: Filtering and sorting Query results  
Previous ← SQL Lesson 2: Queries with constraints (Pt. 1)

Find SQL Bolt useful? Please consider [Donating \(\\$4\)](#) via PayPal to support our site.

Table: Movies

Id	Title	Director	Year	Length_minutes
9	WALL-E	Andrew Stanton	2008	101
87	WALL-E	Brenda Chapman	2012	97

```
SELECT * FROM movies
WHERE title = "Toy Story";
```

Exercise 1 — Tasks

- Find all the Toy Story movies. ✓
- Find all the movies directed by John Lasseter. ✓
- Find all the movies (and director) not directed by John Lasseter. ✓
- Find all the WALL-E movies. ✓

Start? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

Continue >

RESET

Next → SQL Lesson 1: Filtering and sorting Query results  
Previous ← SQL Lesson 2: Queries with constraints (Pt. 1)

Find SQL Bolt useful? Please consider [Donating \(\\$4\)](#) via PayPal to support our site.

## SQL Lesson 4: Filtering and sorting Query results

Table: Movies

Director
Andrew Stanton
Brad Bird
Brenda Chapman
Dan Scanlon
John Lasseter
Lee Unkrich
Pete Docter

```
SELECT DISTINCT director FROM movies
ORDER BY director ASC;
```

Exercise 4 — Tasks

- List all directors of Pixar movies (alphabetically), without duplicates ✓
- List the last four Pixar movies released (ordered from most recent to least) ✓
- List the first five Pixar movies sorted alphabetically ✓
- List the next five Pixar movies sorted alphabetically

Start? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

Next → SQL Review: Simple SELECT Queries  
Previous ← SQL Lesson 3: Queries with constraints (Pt. 2)

Find SQL Bolt useful? Please consider [Donating \(\\$4\)](#) via PayPal to support our site.

Table: Movies

Title
A Dog's Life
Birds
Cars
Cars 2
Finding Nemo

```
SELECT title FROM movies
ORDER BY title ASC
LIMIT 5;
```

Exercise 4 — Tasks

- List all directors of Pixar movies (alphabetically), without duplicates ✓
- List the last four Pixar movies released (ordered from most recent to least) ✓
- List the first five Pixar movies sorted alphabetically ✓
- List the next five Pixar movies sorted alphabetically

Start? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

Next → SQL Review: Simple SELECT Queries  
Previous ← SQL Lesson 3: Queries with constraints (Pt. 2)

Find SQL Bolt useful? Please consider [Donating \(\\$4\)](#) via PayPal to support our site.

## SQL Review: Simple SELECT Queries

The cities in the list have positive latitudes and negative longitudes.

To try and write some queries to find the information requested in the tasks you know. You may have to use a different combination of clauses in your query for each task. Once you're done, continue onto the next lesson to learn about queries that span multiple tables.

Table: North\_american\_cities

City	Population
Toronto	299660
Montreal	171757

```
SELECT city, population FROM north_american_cities
WHERE country = "Canada";
```

Review 1 — Tasks

- List all the Canadian cities and their populations. ✓
- Order all the cities in the United States by their latitude from north to south. ✓
- List all the cities west of Chicago, ordered from west to east. ✓
- List the two largest cities in Mexico (by population). ✓
- List the third and fourth largest cities (by population) in the United States and their population

Start? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

Table: Movies

Title	Year
Monsters University	2013
Monsters, Inc.	2001
Toy Story	1995
Cars 2	2011

```
SELECT title, year FROM movies
ORDER BY year DESC
LIMIT 4;
```

Exercise 4 — Tasks

- List all versions of Pixar movies (alphabetical), without duplicates. ✓
- List the last four Pixar movies released (ordered from most recent to least). ✓
- List the first five Pixar movies sorted alphabetically. ✓
- List the next five Pixar movies sorted alphabetically

Start? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

Next → SQL Review: Simple SELECT Queries  
Previous ← SQL Lesson 3: Queries with constraints (Pt. 2)

Find SQL Bolt useful? Please consider [Donating \(\\$4\)](#) via PayPal to support our site.

Table: North\_american\_cities

City	Latitude
Chicago	41.878114
New York	40.712784
Philadelphia	39.923584
Los Angeles	34.052334
Phoenix	33.448177
Houston	29.704247

```
SELECT city, latitude FROM north_american_cities
WHERE country = "United States"
ORDER BY latitude DESC;
```

Review 1 — Tasks

- List all the Canadian cities and their populations. ✓
- Order all the cities in the United States by their latitude from north to south. ✓
- List all the cities west of Chicago, ordered from west to east. ✓
- List the two largest cities in Mexico (by population). ✓
- List the third and fourth largest cities (by population) in the United States and their population

Start? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

Finish above Tasks

RESET

The cities in the list have positive latitudes and negative longitudes.

Try and write some queries to find the information requested in the tasks you know. You may have to use a different combination of clauses in your query for each task. Once you're done, continue onto the next lesson to learn about queries that span multiple tables.

Table: North\_american\_cities

City	Longitude
Los Angeles	-118.243565
Phoenix	-112.04207
Guadalajara	-103.39009
Mexico City	99.132206
Ecatepec de Morelos	-99.050614
Chicago	-87.590003

Review 1 — Tasks

1. List all the Canadian cities and their populations ✓
2. Order all the cities in the United States by their latitude from north to south ✓
3. List all the cities west of Chicago, ordered from west to east ✓
4. List the two largest cities in Mexico (by population) ✓
5. List the third and fourth largest cities (by population) in the United States and their populations

Stack? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

RESET Finish above Tasks

SELECT city, longitude FROM north\_american\_cities WHERE longitude < -87.69298 ORDER BY longitude ASC;

The dates in the list have positive latitudes and negative longitudes.

Try and write some queries to find the information requested in the tasks you know. You may have to use a different combination of clauses in your query for each task. Once you're done, continue onto the next lesson to learn about queries that span multiple tables.

Table: North\_american\_cities

City	Population
Mexico City	8555000
Ecatepec de Morelos	1742000

Review 1 — Tasks

1. List all the Canadian cities and their populations ✓
2. Order all the cities in the United States by their latitude from north to south ✓
3. List all the cities west of Chicago, ordered from west to east ✓
4. List the two largest cities in Mexico (by population) ✓
5. For the third and fourth largest cities (by population) in the United States and their populations

Stack? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

RESET Finish above Tasks

SELECT city, population FROM north\_american\_cities WHERE country LIKE "Mexico" ORDER BY population DESC LIMIT 2;

Lesson to learn about queries that span multiple tables.

Table: North\_american\_cities

City	Population
Chicago	2718782
Houston	2199514

Review 1 — Tasks

1. List all the Canadian cities and their populations ✓
2. Order all the cities in the United States by their latitude from north to south ✓
3. List all the cities west of Chicago, ordered from west to east ✓
4. List the two largest cities in Mexico (by population) ✓
5. List the third and fourth largest cities (by population) in the United States and their populations

Stack? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

RESET Continue >

SELECT city, population FROM north\_american\_cities WHERE country = "United States" ORDER BY population DESC LIMIT 2 OFFSET 2;

## SQL Lesson 6: Multi-table queries with JOINS

Query Results

Title	Domestic sales	International sales
Finding Nemo	360913261	555000000
Monsters University	265462764	475566403
Ratatouille	266445554	417277164
Cars 2	19452286	864800000
Toy Story 3	24869119	299163000
The Incredibles	26114102	170301200
WALL-E	235956164	207059595
Toy Story 2	415024080	640757231
Toy Story	1976023	7012563
Cars	24000000	27700000

Exercise 6 — Tasks

1. Find the domestic and international sales for each movie ✓
2. Show the sales numbers for each movie that did better internationally rather than domestically ✓
3. List all the movies by their ratings in descending order ✓

Stack? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

RESET Continue >

SELECT title, domestic\_sales, international\_sales FROM movies JOIN boxoffices ON movies.id = boxoffices.movie\_id;

Query Results

Title	Domestic sales	International sales
Finding Nemo	360913261	555000000
Monsters University	265462764	475566403
Ratatouille	266445554	417277164
Cars 2	19452286	864800000
The Incredibles	26114102	170301200
WALL-E	235956164	207059595
Toy Story 3	415024080	640757231
Up	29304164	40033500
A Bug's Life	16709864	209800000
Bear	237283207	301700000

Exercise 6 — Tasks

1. Find the domestic and international sales for each movie ✓
2. Show the sales numbers for each movie that did better internationally rather than domestically ✓
3. List all the movies by their ratings in descending order ✓

Stack? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

RESET Continue >

SELECT title, domestic\_sales, international\_sales FROM movies JOIN boxoffices ON movies.id = boxoffices.movie\_id WHERE international\_sales > domestic\_sales;

Note: SQL Lesson 7: OUTER JOINs  
Previous: SQL Review: Simple SELECT Queries

Find SQLBot useful? Please consider Donating (\$4+) via PayPal to support our site.

Query Results

Title	Rating
WALL-E	8.5
Toy Story 3	6.4
Toy Story	6.3
Up	8.8
Finding Nemo	6.2
Monsters, Inc.	6.1
Ratatouille	6
The Incredibles	8
Toy Story 2	7.9
Monsters University	7.4

Exercise 6 — Tasks

1. Find the domestic and international sales for each movie ✓
2. Show the sales numbers for each movie that did better internationally rather than domestically ✓
3. List all the movies by their ratings in descending order ✓

Stack? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

RESET Continue >

SELECT title, rating FROM movies INNER JOIN boxoffices ON movies.id = boxoffices.movie\_id ORDER BY rating DESC;

Query Results

Title	Rating
WALL-E	8.5
Toy Story 3	6.4
Toy Story	6.3
Up	8.8
Finding Nemo	6.2
Monsters, Inc.	6.1
Ratatouille	6
The Incredibles	8
Toy Story 2	7.9
Monsters University	7.4

Exercise 6 — Tasks

1. Find the domestic and international sales for each movie ✓
2. Show the sales numbers for each movie that did better internationally rather than domestically ✓
3. List all the movies by their ratings in descending order ✓

Stack? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

RESET Continue >

SELECT title, rating FROM movies INNER JOIN boxoffices ON movies.id = boxoffices.movie\_id ORDER BY rating DESC;

Note: SQL Lesson 7: OUTER JOINs  
Previous: SQL Review: Simple SELECT Queries

Find SQLBot useful? Please consider Donating (\$4+) via PayPal to support our site.

## SOL Lesson 7: OUTER JOINS

**Query Results**

Building
1e
2w

```
SELECT DISTINCT building FROM employees;
```

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

**Exercise 7 — Tasks**

- Find the list of all buildings that have employees ✓
- Find the list of all buildings and their capacity ✓
- List all buildings and the distinct employee roles in each building (including empty buildings)

**Finish above Tasks**

RESET

Next → SQL Lesson 8: A short note on NULLs  
Previous → SQL Lesson 6: Multi-table queries with JOINs

**Query Results**

Building_name	Capacity
1e	21
1w	52
2e	16
2w	20

```
SELECT * FROM buildings;
```

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

**Exercise 7 — Tasks**

- Find the list of all buildings that have employees ✓
- Find the list of all buildings and their capacity ✓
- List all buildings and the distinct employee roles in each building (including empty buildings)

**Finish above Tasks**

RESET

Find SQLJedi useful? Please consider Donating (\$4) via PayPal to support our site.

**Query Results**

Building_name	Role
1e	Engineer
1e	Manager
1w	
2e	
2e	Artist
2w	Manager

```
SELECT DISTINCT building_name, role
FROM buildings
LEFT JOIN employees
ON building_name = building;
```

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

**Exercise 7 — Tasks**

- Find the list of all buildings that have employees ✓
- Find the list of all buildings and their capacity ✓
- List all buildings and the distinct employee roles in each building (including empty buildings) ✓

**Continue >**

RESET

## SQL Lesson 8: A short note on NULLs

**Query Results**

Building_name	Capacity
1e	24
1w	22
2e	16
2w	20

```
SELECT * FROM buildings;
```

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

**Exercise 7 — Tasks**

- Find the list of all buildings that have employees ✓
- Find the list of all buildings and their capacity ✓
- List all buildings and the distinct employee roles in each building (including empty buildings)

**Finish above Tasks**

RESET

Next → SQL Lesson 8: A short note on NULLs  
Previous → SQL Lesson 6: Multi-table queries with JOINs

**Query Results**

Building_name	Role	Name	Role	Name
2e	Engineer	Sharon F.	1e	6
2w	Engineer	Dan M.	1e	4
2e	Engineer	Melissa S.	1e	1
2e	Artist	Tyler S.	2w	2

```
SELECT DISTINCT building_name, role
FROM buildings
LEFT JOIN employees
ON building_name = building
WHERE role IS NULL;
```

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

**Exercise 8 — Tasks**

- Find the name and role of all employees who have not been assigned to a building ✓
- Find the names of the buildings that hold no employees ✓

**Continue >**

## SQL Lesson 9: Queries with expressions

**Query Results**

Title	Gross_sales_millions
Hiding Nemo	556.44061
Monsters University	743.559607
Platzilla	623.722918
Cars 2	504.857466
Toy Story 2	485.05179
The Incredibles	831.422092
WALL-E	521.7106
Toy Story 3	1005.771911
Toy Story	943.96745
Cars	401.363149

```
SELECT title, (domestic_sales + international_sales) / 1000000 AS gross_sales_millions
FROM movies
JOIN boxoffice
ON movies.id = boxoffice.movie_id;
```

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

**Exercise 9 — Tasks**

- List all movies and their combined sales in millions of dollars ✓
- List all movies and their ratings in percent ✓
- List all movies that were released on even-number years

**Finish above Tasks**

RESET

**Query Results**

Title	Rating_percent
Finding Nemo	62
Monsters University	71
Platzilla	40
Cars 2	64
Toy Story 2	76
The Incredibles	40
WALL-E	85
Toy Story 3	84
Toy Story	83
Cars	72

```
SELECT title, rating * 10 AS rating_percent
FROM movies
JOIN boxoffice
ON movies.id = boxoffice.movie_id;
```

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

**Exercise 9 — Tasks**

- List all movies and their combined sales in millions of dollars ✓
- List all movies and their ratings in percent ✓
- List all movies that were released on even-number years

**Finish above Tasks**

RESET

Query Results

Rank	Title	Year	Revenue	Gross Profit	Profit Margin (%)
4	Movies, Inc.	2001	\$2	\$1	50%
5	Training Nemo	2003	\$3	\$2	66.7%
6	The Incredibles	2004	\$6	\$4	66.7%
7	Cars	2006	\$8	\$6	75%
8	Wall-E	2008	\$10	\$8	80%
9	Toy Story 3	2010	\$12	\$10	83.3%
10	Brave	2012	\$15	\$12	80%

`SELECT title, year  
FROM movies  
WHERE year < 2010;`

[Stuck? Read this task's Solution.](#)  
[Solve all tasks to continue to the next lesson.](#)

[Continue >](#)

## SQL Lesson 10: Queries with aggregates (Pt. 1)

### Exercise

For this exercise, we are going to work with our **Employees** table. Notice how the rows in this table have shared data, which will give us an opportunity to use aggregate functions to summarize some high-level metrics about the teams. Go ahead and give it a shot.

Table: Employees

**Max\_years\_employed**

9

`SELECT MAX(years_employed) as Max_years_employed  
FROM employees;`

Exercise 10 — Tasks

- Find the longest time that an employee has been at the studio ✓
- For each role, find the average number of years employed by employees in that role ✓
- Find the total number of employee years worked in each building

[Stuck? Read this task's Solution.](#)  
[Solve all tasks to continue to the next lesson.](#)

[Finish above Tasks](#)

### Exercise

For this exercise, we are going to work with our **Employees** table. Notice how the rows in this table have shared data, which will give us an opportunity to use aggregate functions to summarize some high-level metrics about the teams. Go ahead and give it a shot.

Table: Employees

**Role**      **Average\_years\_employed**

Artist      6

Engineer      3.4

Manager      6

`SELECT role, AVG(years_employed) as Average_years_employed  
FROM employees  
GROUP BY role;`

Exercise 10 — Tasks

- Find the longest time that an employee has been at the studio ✓
- For each role, find the average number of years employed by employees in that role ✓
- Find the total number of employee years worked in each building

[Stuck? Read this task's Solution.](#)  
[Solve all tasks to continue to the next lesson.](#)

[Finish above Tasks](#)

### Exercise

For this exercise, we are going to work with our **Employees** table. Notice how the rows in this table have shared data, which will give us an opportunity to use aggregate functions to summarize some high-level metrics about the teams. Go ahead and give it a shot.

Table: Employees

**Building**      **SUM(years\_employed)**

1e      29

2w      36

`SELECT building, SUM(years_employed)  
FROM employees  
GROUP BY building;`

Exercise 10 — Tasks

- Find the longest time that an employee has been at the studio ✓
- For each role, find the average number of years employed by employees in that role ✓
- Find the total number of employee years worked in each building ✓

[Stuck? Read this task's Solution.](#)  
[Solve all tasks to continue to the next lesson.](#)

[Continue >](#)

## SQL Lesson 11: Queries with aggregates (Pt. 2)

For this exercise, you are going to dive deeper into **Employee** data at the film studio. Think about the different clauses you want to apply for each task.

Table: Employees

**Role**      **Number\_of\_artists**

Artist      5

`SELECT role, COUNT(*) as Number_of_artists  
FROM employees  
WHERE role = "Artist";`

Exercise 11 — Tasks

- Find the number of Artists in the studio (without a HAVING clause) ✓
- Find the number of Employees of each role in the studio
- Find the total number of years employed by all Engineers

[Finish above Tasks](#)

For this exercise, you are going to dive deeper into **Employee** data at the film studio. Think about the different clauses you want to apply for each task.

Table: Employees

**Role**      **COUNT(\*)**

Artist      5

Engineer      5

Manager      3

Exercise 11 — Tasks

- Find the number of Artists in the studio (without a HAVING clause) ✓
- Find the number of Employees of each role in the studio ✓
- Find the total number of years employed by all Engineers

[Finish above Tasks](#)

Next: [SQL Lesson 12: Order of execution of a query](#)

Find SQL Bolt useful? Please consider:

[Find SQL Bolt useful? Please consider](#)

Next: [SQL Lesson 12: Order of execution of a query](#)

Find SQL Bolt useful? Please consider

**EXERCISE**

For this exercise, you are going to dive deeper into **Employee** data at the film studio. Think about the different clauses you want to apply for each task.

Table: Employees

Role	SUM(years_employed)
Engineer	17

**Exercise 11 — Tasks**

- Find the number of Artists in the studio (without a HAVING clause) ✓
- Find the number of Employees of each role in the studio ✓
- Find the total number of years employed by all Engineers ✓

Stuck? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

**Query Results**

```
SELECT role, SUM(years_employed)
FROM employee
GROUP BY role
HAVING role = "Engineer";
```

**Continue >**

## SQL Lesson 12: Order of execution of a Query

**Query Results**

Director	Num_movies_directed
Andrew Stanton	2
Brad Bird	2
Brenda Chapman	1
Don Soslon	1
John Lasseter	5
Lee Unkrich	1
Pete Docter	2

**Exercise 12 — Tasks**

- Find the number of movies each director has directed ✓
- Find the total domestic and international sales that can be attributed to each director

Stuck? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

**Query Results**

Director	Total
Andrew Stanton	1450055121
Brad Bird	125564910
Brenda Chapman	538983207
Don Soslon	74555907
John Lasseter	22320805
Lee Unkrich	106371911
Pete Docter	129415800

**Exercise 12 — Tasks**

- Find the number of movies each director has directed ✓
- Find the total domestic and international sales that can be attributed to each director ✓

Stuck? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

**Finish above Tasks**

**Query Results**

Movie	Year	Length	Rating	Domestic sales	International sales
Monsters, Inc.	2001	92	7	6.4	51457396
Finding Nemo	2003	107	3	7.9	24052179
The Incredibles	2004	116	6	6	71441092

**Exercise 12 — Tasks**

- Find the number of movies each director has directed ✓
- Find the total domestic and international sales that can be attributed to each director ✓

Stuck? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

**Continue >**

## SQL Lesson 13: Inserting rows

**Query Results**

Id	Title	Director	Year	Length	minutes
1	Toy Story	John Lasseter	1995	81	
2	A Bug's Life	John Lasseter	1998	85	
3	Toy Story 2	John Lasseter	1999	93	
4	Toy Story 4	Pete Docter	2025	60	

**Exercise 13 — Tasks**

- Add the studio's new production, Toy Story 4 to the list of movies you can use any director
- Toy Story 4 has been released to critical acclaim! It had a rating of 8.7, and made \$40 million domestically and 270 million internationally. Add the record to the BoxOffice table.

Stuck? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

**Query Results**

Movie_id	Rating	Domestic sales	International sales
3	2.5	244851719	270167002
1	8.7	15170915	170167002
2	2.9	162708545	209200002
4	8.7	48XXXXXX	270167002

**Exercise 13 — Tasks**

- Add the studio's new production, Toy Story 4 to the list of movies you can use any director
- Toy Story 4 has been released to critical acclaim! It had a rating of 8.7, and made \$40 million domestically and 270 million internationally. Add the record to the BoxOffice table ✓

Stuck? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

**Finish above Tasks**

**Note:** SQL Lesson 14: Updating rows  
Previous: SQL Lesson 13: Inserting rows

**Note:** SQL Lesson 14: Updating rows  
Previous: SQL Lesson 12: Order of execution of a Query

**Run Query** **Reset**

## SQL Lesson 14: Updating rows

Through the exercises below

**Table: Movies**

Id	Title	Director	Year	Length	minutes
1	Toy Story	John Lasseter	1995	81	
2	A Bug's Life	John Lasseter	1998	85	
3	Toy Story 2	John Lasseter	1999	93	
4	Monsters, Inc.	Pete Docter	2001	90	
5	Hunting Nemo	Andrew Stanton	2003	107	
6	The Incredibles	Brad Bird	2004	116	
7	Cars	John Lasseter	2006	117	
8	Robotnik	Brad Bird	2007	115	
9	WALL-E	Andrew Stanton	2008	104	
10	Brave	Pete Docter	2012	101	

**Exercise 14 — Tasks**

- The director for A Bug's Life is incorrect. It was actually directed by John Lasseter ✓
- The year that Toy Story 2 was released is incorrect; it was actually released in 1999 ✓
- Both the title and director for Toy Story 4 is incorrect; the title should be "Toy Story 3" and it was directed by Lee Unkrich

Stuck? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

**Query Results**

```
UPDATE movies
SET director = "John Lasseter"
WHERE id = 2;
```

**Exercise 14 — Tasks**

- The director for A Bug's Life is incorrect; it was actually directed by John Lasseter ✓
- The year that Toy Story 2 was released is incorrect; it was actually released in 1999 ✓
- Both the title and director for Toy Story 4 is incorrect; the title should be "Toy Story 3" and it was directed by Lee Unkrich

Stuck? Read this task's [Solution](#).  
Solve all tasks to continue to the next lesson.

**Finish above Tasks**

**Note:** SQL Lesson 15: Deleting rows  
Previous: SQL Lesson 14: Updating rows

**Note:** SQL Lesson 16: Deleting rows  
Previous: SQL Lesson 15: Updating rows

**Run Query** **Reset**

through the exercises below.

**Table: Movies**

ID	Title	Director	Year	Length (minutes)
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116
7	Cars	John Lasseter	2006	117
8	Ratatouille	Brad Bird	2007	115
9	WALL-E	Andrew Stanton	2008	104
10	Up	Pete Docter	2009	101
11	Toy Story 3	Lee Unkrich	2010	103
12	Cars 2	John Lasseter	2011	120
13	Birds	Brenda Chapman	2012	102
14	Monsters University	Dan Scanlon	2013	110

**Run(s) deleted:**  
SET title = "Toy Story 4", director = "Lee Unkrich"  
WHERE id = 11

**RUN QUERY** **RESET**

**Exercise 14 — Tasks**

- The director for A Bug's Life is incorrect, it was actually directed by John Lasseter. ✓
- The year that Toy Story 2 was released is incorrect. It was actually released in 1999. ✓
- Both the title and director for Toy Story 3 is incorrect. The title should be "Toy Story 3" and it was directed by Lee Unkrich. ✓

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

**Continue >**

Next - SQL Lesson 13: Deleting rows  
Previous - SQL Lesson 12: Inserting rows

## SQL Lesson 15: Deleting rows

**Table: Movies**

ID	Title	Director	Year	Length (minutes)
7	Cars	John Lasseter	2006	117
8	Ratatouille	Brad Bird	2007	115
9	WALL-E	Andrew Stanton	2008	104
10	Up	Pete Docter	2009	101
11	Toy Story 3	Lee Unkrich	2010	103
12	Cars 2	John Lasseter	2011	120
13	Birds	Brenda Chapman	2012	102
14	Monsters University	Dan Scanlon	2013	110

**Run(s) deleted:**  
DELETE FROM movies  
WHERE year < 2005;

**RUN QUERY** **RESET**

**Exercise 15 — Tasks**

- This database is getting too big, let's remove all movies that were released before 2005. ✓
- Andrew Stanton has also left the studio, so please remove all movies directed by him. ✓

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

**Finish above Tasks**

Find SQLBolt useful? Please consider Donating (\$1) via PayPal to support our site.

Next - SQL Lesson 16: Creating tables  
Previous - SQL Lesson 14: Updating rows

**Table: Movies**

ID	Title	Director	Year	Length (minutes)
7	Cars	John Lasseter	2006	117
8	Ratatouille	Brad Bird	2007	115
10	Up	Pete Docter	2009	101
11	Toy Story 3	Lee Unkrich	2010	103
12	Cars 2	John Lasseter	2011	120
13	Birds	Brenda Chapman	2012	102
14	Monsters University	Dan Scanlon	2013	110

**Run(s) deleted:**  
DELETE FROM movies  
WHERE director = "Andrew Stanton";

**RUN QUERY** **RESET**

**Exercise 15 — Tasks**

- This database is getting too big, let's remove all movies that were released before 2005. ✓
- Andrew Stanton has also left the studio, so please remove all movies directed by him. ✓

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

**Continue >**

Find SQLBolt useful? Please consider Donating (\$1) via PayPal to support our site.

Next - SQL Lesson 16: Creating tables  
Previous - SQL Lesson 14: Updating rows

## SQL Lesson 16: Creating tables

**Exercise**  
In this exercise, you'll need to create a new table for us to insert some new rows into.

**Table: Database**

Name	Version	Download count
SQLite	3.9	92000000
MySQL	5.5	512000000
Postgres	9.4	384000000

**table created**  
CREATE TABLE Database (  
 Name TEXT,  
 Version FLOAT,  
 Download\_count INTEGER  
)

**RUN QUERY** **RESET**

**Exercise 16 — Tasks**

- Create a new table named `Database` with the following columns:  
  - `Name`: A string (text) describing the name of the database.
  - `Version`: A number (floating point) of the latest version of this database.
  - `Download_count`: An integer count of the number of times this database was downloaded.
- This table has no constraints. ✓

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

**Continue >**

## SQL Lesson 17: Altering tables

**Exercise**  
Our exercises use an implementation that only supports adding new columns, so give that a try below.

**Table: Movies**

ID	Title	Director	Year	Length (minutes)	Aspect ratio
1	Toy Story	John Lasseter	1995	81	2.39
2	A Bug's Life	John Lasseter	1998	95	2.39
3	Toy Story 2	John Lasseter	1999	93	2.39
4	Monsters, Inc.	Pete Docter	2001	92	2.39
5	Finding Nemo	Andrew Stanton	2003	107	2.39
6	The Incredibles	Brad Bird	2004	116	2.39
7	Cars	John Lasseter	2006	117	2.39
8	Ratatouille	Brad Bird	2007	115	2.39
9	WALL-E	Andrew Stanton	2008	104	2.39
10	Up	Pete Docter	2009	101	2.39

**New column added:**  
ALTER TABLE Movies  
ADD COLUMN Aspect\_ratio FLOAT DEFAULT 2.39;

**RUN QUERY** **RESET**

**Exercise 17 — Tasks**

- Add a column named `Aspect_ratio` with a `FLOAT` data type to store the aspect ratio of each movie. ✓
- Add another column named `Language` with a `TEXT` data type to store the language that the movie was released in. Ensure that the default for this language is English. ✓

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

**Finish above Tasks**

**Exercise**  
Our exercises use an implementation that only supports adding new columns, so give that a try below.

**Table: Movies**

ID	Title	Director	Year	Length (minutes)	Aspect_ratio	Language
1	Toy Story	John Lasseter	1995	81	2.39	English
2	A Bug's Life	John Lasseter	1998	95	2.39	English
3	Toy Story 2	John Lasseter	1999	93	2.39	English
4	Monsters, Inc.	Pete Docter	2001	92	2.39	English
5	Finding Nemo	Andrew Stanton	2003	107	2.39	English
6	The Incredibles	Brad Bird	2004	116	2.39	English
7	Cars	John Lasseter	2006	117	2.39	English
8	Ratatouille	Brad Bird	2007	115	2.39	English
9	WALL-E	Andrew Stanton	2008	104	2.39	English
10	Up	Pete Docter	2009	101	2.39	English

**New column added:**  
ALTER TABLE Movies  
ADD COLUMN Language TEXT DEFAULT "English";

**RUN QUERY** **RESET**

**Exercise 17 — Tasks**

- Add a column named `Aspect_ratio` with a `FLOAT` data type to store the aspect ratio each movie was released in. ✓
- Add another column named `Language` with a `TEXT` data type to store the language that the movie was released in. Ensure that the default for this language is English. ✓

Stuck? Read this task's Solution.  
Solve all tasks to continue to the next lesson.

**Continue >**

## SQL Lesson 18: Dropping tables

The left screenshot shows the completion of Task 1: "And drop the BoxOffice table as well". The right screenshot shows the completion of Task 2: "And drop the BoxOffice table as well". Both screenshots include a "Finish above Tasks" button at the bottom.

The page displays a graduation cap icon with a lightning bolt, indicating completion. It includes a message of thanks and links to other resources like "Interactive Tutorial" and "More Topics".

## EJERCICIO # 2

En esta segunda parte de la tarea se le solicitará completar un segundo tutorial que gira alrededor del análisis de un caso de uso más real, con el fin de ver aplicados los conceptos que se van aprendiendo:

<https://selectstarsql.com/>

Debe completar los cinco capítulos del tutorial, registrando pantallazos de cada uno de los ejercicios realizados dentro de las herramientas del sitio web.

# THE BEAZLEY CASE

## A First SQL Query

that an eye for an eye does not constitute justice. Our task is to retrieve his statement from the database.

### 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.

The SQL query may look like an ordinary sentence, but you should view it as three Lego blocks: `SELECT * FROM executions` `LIMIT 3`. As with Lego, each block has a fixed format and the different blocks have to fit together in particular ways.

## 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.

```
1 SELECT first_name, last_name, last_statement  
2 FROM executions  
3 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.

### SQL Comments

Notice that clicking "Show Solution" displays the solution in the editor preceded by `/*`. The contents between `/*` and `*/` are taken as

## 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

In the next example, observe that we don't need the `FROM` block if we're not using anything from a table.

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

```
1 SELECT 50 / 2, 51 / 2
```

In the next example, observe that we don't need the `FROM` block if we're not using anything from a table.

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

Correct

```
50 / 2 51 / 2
```

```
25 25
```

Isn't it strange that `51 / 2` gives `25` rather than `25.5`? This is because SQL is doing integer division. To do decimal division, at least one of the operands must be a decimal, for instance `51.0 / 2`. A common trick is to multiply one number by `1.0` to convert it into a decimal. This will come in useful in the later chapters.

### Capitalization

Even though we've capitalized `SELECT`, `FROM` and `LIMIT`, SQL commands are not case sensitive. You can see that the code editor recognizes them and formats them as a command no matter the capitalization. Nevertheless, I recommend capitalizing them to differentiate them from column names, table names and variables.

Column names, table names and variables are also not case-sensitive in this version of SQL, though they are case-sensitive in many other



# CHAPTER # 2, POSSIBLE INNOCENCE

## The COUNT function

### The COUNT Function

COUNT is probably the most widely-used aggregate function. As the name suggests, it counts things! For instance, COUNT(<column>) returns the number of non-null rows in the column.

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

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

Run | Show Solution | Reset |

Correct

COUNT(last_statement)
443

As you can tell, the COUNT function is intrinsically tied to the concept of NULLs. Let's make a small digression to learn about NULLs.

#### Nulls

In SQL, NULL is the value of an empty entry. This is different from the

!=.

**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

With this, we can find the denominator for our proportion:

#### 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 NULLs and count it.

```
1 SELECT COUNT(ex_number) AS total_executions FROM execut
```

Run | Show Solution | Reset |

Correct

total_executions
553

### Variations on COUNT

So far so good. But what if we don't know which columns are NULL-free? Worse still, what if none of the columns are NULL-free?

## Variations on COUNT

### Variations on COUNT

So far so good. But what if we don't know which columns are NULL-free? Worse still, what if none of the columns are NULL-free? Surely there must still be a way to find the length of the table!

The solution is COUNT(\*). This is reminiscent of SELECT \* where the \* represents all columns. In practice COUNT(\*) counts rows as long as *any one* of their columns is non-null. This helps us find table lengths because a table shouldn't have rows that are completely null.

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

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

Run | Reset |

COUNT(*)
553

Another common variation is to count a subset of the table. For instance, counting Harris county executions. We could run SELECT COUNT(\*) FROM executions WHERE county='Harris' which filters down to a smaller dataset consisting of Harris executions

Correct  
COUNT(last\_statement)  
443

As you can tell, the COUNT function is intrinsically tied to the concept of NULLs. Let's make a small digression to learn about NULLs.

#### 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

With this, we can find the denominator for our proportion:

#### 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 NULLs and count it.

!=.

**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

With this, we can find the denominator for our proportion:

#### 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 NULLs and count it.

```
1 SELECT COUNT(ex_number) AS total_executions FROM execut
```

Run | Show Solution | Reset |

Correct

total_executions
553

This is admittedly one of the clunkier parts of SQL. A common mistake is to miss out the END command and the ELSE condition which is a catchall in case all the prior clauses are false. Also recall from the previous chapter that clauses are expressions that can be evaluated to be true or false. This makes it important to think about the boolean value of whatever you stuff in there.

**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.

```
1 SELECT  
2   SUM(CASE WHEN county='Harris' THEN 1  
3           ELSE 0 END) AS Harris,  
4   SUM(CASE WHEN county='Bexar' THEN 1  
5           ELSE 0 END) AS Bexar  
6 FROM executions
```

Run | Show Solution | Reset |

Correct

Harris	Bexar
128	46

```

4 SUM(CASE WHEN county='Bexar' THEN 1
5 ELSE 0 END) AS Bexar
6 FROM executions

```

Run Show Solution Reset

Correct

Harris	Bexar
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 T SUM(CASE WHEN ex_age > 50 THEN 1 ELSE 0 END) AS inmates;
2 executions;

```

Run Show Solution Reset

Correct

inmates
68

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.

Run Show Solution Reset

Correct

Inmates
68

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(CASE WHEN last_statement IS NULL THEN 1 ELSE 0 END) AS total;
2 FROM executions;

```

Run Show Solution Reset

total
553

It is worthwhile to step back and think about the different ways the computer handled these three queries. The WHERE version had it filter down to a small table first before aggregating while in the other two, it had to look through the full table. In the COUNT + CASE WHEN version, it only had to go through once, while the double COUNT version made it go through twice. So even though the output was identical, the performance was probably best in the first and worst in the third version.

And it filter down to a small table first before aggregating while in the other two, it had to look through the full table. In the COUNT + CASE WHEN version, it only had to go through once, while the double COUNT version made it go through twice. So even though the output was identical, the performance was probably best in the first and worst in the third version.

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

Use the MIN, MAX, and AVG aggregate functions.

```

1 SELECT MIN(ex_age) AS minimum,
2      MAX(ex_age) AS maximum,
3      AVG(ex_age) AS average
4 FROM executions

```

Run Show Solution Reset

Correct

minimum	maximum	average
24	67	39.47016274864376

### Looking Up Documentation

This book was never intended to be a comprehensive reference for the SQL language. For that, you will have to look up other online resources. This is a skill in itself, and one that is worth mastering because you will be looking up documentation years after you've achieved familiarity with the language.

The good news is that with the mental models you will learn in this

achieved familiarity with the language.

The good news is that with the mental models you will learn in this book, lookups should be quick and painless because you will just be checking details like whether the function is called AVERAGE or AVG instead of figuring out what approach to take.

For lookups, I often use [W3 Schools](#), Stack Overflow and the [official SQLite documentation](#).

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)) AS average
2 FROM executions;

```

Run Show Solution Reset

Correct

average
537.492099322799

List all the counties in the dataset without duplication.

We can get unique entries by using SELECT DISTINCT. See documentation.

```

1

```

Run Show Solution Reset

Run Show Solution Reset

Correct

average
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

SELECT DISTINCT isn't really an aggregate function because it doesn't return a single number and because it operates on the output of the query rather than the underlying table. Nevertheless, I've included it here because it shares a common characteristic of operating on multiple rows.

## A Strange Query

Before we wrap up, let's take a look at this query:

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

Doesn't it look strange? If you have a good mental model of aggregations, it should! `COUNT(*)` is trying to return a single entry consisting the length of the execution table. `first_name` is trying to return one entry for each row. Should the computer return one or multiple rows? If it returns one, which `first_name` should it pick? If it returns multiple, is it supposed to replicate the `COUNT(*)` result across all the rows? The shapes of the output just don't match!

Let's try it anyway and see what happens.

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

Run | Reset

first_name	COUNT(*)
Charlie	553

In practice, databases try to return something sensible even though you pass in nonsense. In this case, our database picks the first name from the last entry in our table. Since our table is in reverse chronological order, the last entry is Charlie Brooks Jr., the first person executed since the Supreme Court lifted the ban on the death penalty. Different databases will handle this case

### 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.

```
1 SELECT 1.0 * COUNT(CASE WHEN last_statement LIKE '%innocent%' THEN 1 ELSE NULL END) / COUNT(*)
2
3   AS inmates_proportion
4
5   FROM executions;
```

Run | Show Solution | Reset

Correct

inmates_proportion
0.0560578661844846

This method of finding claims of innocence is admittedly rather inaccurate because innocence can be expressed in other terms like "not guilty". Nevertheless, I suspect it underestimates the real number, and is probably of the right order of magnitude. The question we are left with then, is whether we are willing to accept the possibility that up to 5% percent of people we execute

## CHAPTER # 3. LONG TAILS

### The GROUP BY block

#### The GROUP BY Block

This is where the `GROUP BY` block comes in. It allows us to split up the dataset and apply aggregate functions within each group, resulting in one row per group. Its most basic form is `GROUP BY <column>, <column>, ...` and comes after the `WHERE` block.

This query pulls the execution counts per county.

```
1 SELECT
2   county,
3   COUNT(*) AS county_executions
4   FROM executions
5   GROUP BY county
```

Run | Reset

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

If you recall [A Strange Query](#), alarm bells would be going off in

provides all alias that can be referred to later in the query. This saves us from rewriting long expressions, and allows us to clarify the purpose of the expression.

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.

```
1 SELECT
2   last_statement IS NOT NULL AS has_last_statement,
3   county,
4   COUNT(*)
5 FROM executions
6 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

### The HAVING Block

This next exercise illustrates that filtering via the `WHERE` block happens before grouping and aggregation. This is reflected in the order of syntax since the `WHERE` block always precedes the `GROUP BY` 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.

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

[Run](#) [Show Solution](#) [Reset](#)

Correct

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

forward into the future and grab information from there. To solve this problem, we use `HAVING`.

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;
6
```

[Run](#) [Show Solution](#) [Reset](#)

Correct

county
Dallas
Harris
Lubbock
Montgomery
Tarrant

## Practice

### Practice

This quiz is designed to challenge your understanding. Read the explanations even if you get everything correct.

**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
```

- The query is valid (ie. won't throw an error when run).  
Were you thrown off by `ex_age/10?` Grouping by transformed columns is fine too.
- The query would return more rows if we were to use `ex_age` instead of `ex_age/10`.  
Remember that `ex_age/10` does integer division which rounds all the ages. This produces fewer unique groups.
- The output will have as many rows as there are unique combinations of counties and decade\_ages in the dataset.  
This is correct.
- The output will have a group ('Bexar', 6) even though no Bexar county inmates were between 60 and 69 at execution time.

Remember that `ex_age/10` does integer division which rounds all the ages. This produces fewer unique groups.

- The output will have as many rows as there are unique combinations of counties and decade\_ages in the dataset.  
This is correct.
- The output will have a group ('Bexar', 6) even though no Bexar county inmates were between 60 and 69 at execution time.  
The `GROUP BY` block finds all combinations *in the dataset* rather than all theoretically possible combinations.
- The output will have a different value of county for every row it returns.  
This would be true only if `county` were the only grouping column. Here, we can have many groups with the same county but different `decade_ages`.
- The output can have groups where the count is 0.  
This is similar to the ('Bexar', 6) question. If there are no rows with ('Bexar', 6), the group won't even show up.
- The query would be valid even if we don't specify `county` in the `SELECT` block.  
The grouping columns don't necessarily have to be in the `SELECT` block. It would be valid, but not make much sense because we wouldn't know which counts are for which county.
- It is reasonable to add `last_name` to the `SELECT` block even without grouping by it.  
Even though it would be valid (in SQLite) for the reasons set forth in A Strange Query, it is poor form to have non-aggregate, non-grouping columns in the `SELECT` block.

[Check Answers](#) [Show Explanations](#)

It is reasonable to add `last_name` to the `SELECT` block even without grouping by it.  
 Even though it would be valid (in SQLite) for the reasons set forth in A Strange Query, it is poor form to have non-aggregate, non-grouping columns in the `SELECT` block.

[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
2 FROM executions
3 GROUP BY county;
```

[Run](#) [Show Solution](#) [Reset](#)

Correct

county
Anderson
Aansas
Atascosa
Bailey
Bastrop
Bee

## Nested Queries

query and the outer one:

**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)
6 /* SELECT first_name, last_name
7 FROM executions
8 WHERE LENGTH(last_statement) =
9   (SELECT MAX(LENGTH(last_statement))
10  FROM executions)
```

[Run](#) [Show Solution](#) [Reset](#)

Correct

first_name	last_name
Gary	Graham

To reiterate, nesting is necessary here because in the `WHERE` clause, as the computer is inspecting a row to decide if its last statement is the right length, it can't look outside to figure out the maximum length across the entire dataset. We have to find the maximum length separately and feed it into the clause. Now let's apply the same concept to find the percentage of 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.

```
1 SELECT
2   county,
3   100.0 * COUNT(*) / (SELECT COUNT(*)
4                           FROM executions)
5   AS percentage
6 FROM executions
7 GROUP BY county
8 ORDER BY percentage DESC
```

[Run](#) [Show Solution](#) [Reset](#)

Correct

county	percentage
Harris	23.146473779385172
Dallas	10.488245931283906
Bexar	8.318264014466546
Tarrant	7.41410482459313
Nueces	2.893309224231465
Jefferson	2.7124773960216997

I've quietly slipped in an `ORDER BY` block. Its format is `ORDER BY <column>, <column>, ...` and it can be modified by appending `DESC` if you don't want the default ascending order.

## CHAPTER # 4, HIATUSES

### Thinking about JOINS

#### Types of JOINS

...	2018-07-17	553	553	2018-06-27
:	:	:	:	:
:	:	:	:	:

#### Mark the true statements.

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

- `tableA JOIN tableB ON 1` returns 15 rows.  
 The `ON 1` clause is always true, so every row of tableA is matched against every row of tableB.
- `tableA JOIN tableB ON 0` returns 0 rows.  
 For the same reason that `ON 1` returns 15 rows.
- `tableA LEFT JOIN tableB ON 0` returns 3 rows.  
 The left join preserves all the rows of tableA even though no rows of tableB match.
- `tableA OUTER JOIN tableB ON 0` returns 8 rows.  
 The outer join preserves all the rows of tableA and tableB even though none of them are paired.
- `tableA OUTER JOIN tableB ON 1` returns 15 rows.  
 All the rows of tableA match all the rows of tableB because of the `on 1` clause, so any join will return 15 rows. The different joins only differ in how they handle unmatched rows.

[Check Answers](#) [Show Explanations](#)

All correct!

## DATES

We've made a *big assumption* that we can subtract dates from one another. But imagine you're the computer receiving a line like this. Do you return the number of days between the dates? Why not hours or seconds? To make things worse, SQLite doesn't actually have date or time types (unlike most other SQL dialects) so the `ex_date` and `last_ex_date` columns look like ordinary strings to you. You're effectively being asked to do '`'hello'` - '`'world'`'. What does that even mean?

Fortunately, SQLite contains a bunch of functions to tell the computer: "Hey, these strings that I'm passing you actually contain dates or times. Act on them as you would a date."

Look up [the documentation](#) to fix the query so that it returns the number of days between the dates.

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

Run | Show Solution | Reset

Correct

day_difference
1495

## Self Joins

With what we learned about dates, we can connect our template.

## Self JOINS

The next step is to build out the `previous` table.

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!

```
1 SELECT
2   ex_number + 1 AS ex_number,
3   ex_date AS last_ex_date
4 FROM executions
5 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

Now we can nest this query into our template above:

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".

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".

```
1 SELECT
2   last_ex_date AS start,
3   ex_date AS end,
4   JULIANDAY(ex_date) - JULIANDAY(last_ex_date)
5   AS day_difference
6 FROM executions
7 JOIN (SELECT
8   ex_number + 1 AS ex_number,
9   ex_date AS last_ex_date
10  FROM executions
11 WHERE ex_number < 553) previous
12 ON executions.ex_number = previous.ex_number
13 ORDER BY day_difference DESC
14 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

`previous` is derived from `executions`, so we're effectively joining

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

```
1 SELECT
2   previous.ex_date AS start,
3   executions.ex_date AS end,
4   JULIANDAY(executions.ex_date) - JULIANDAY(previous.ex_
5   AS day_difference
6 FROM executions
7 JOIN executions previous
8   ON executions.ex_number = previous.ex_number + 1
9 ORDER BY day_difference DESC
10 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

We can now use the precise dates of the hiatuses to research what happened over each period. In the years immediately after the ban on capital punishment was lifted, there were long periods without executions due to the low number of death

## CHAPTER # 5. CLOSING REMARKS

15894

**Find the most networked senator. That is, the one with the most mutual cosponsorships.**

A mutual cosponsorship refers to two senators who have each cosponsored a bill sponsored by the other. Even if a pair of senators have cooperated on many bills, the relationship still counts as one.

```

1 WITH mutuals AS (
2   SELECT DISTINCT
3     c1.sponsor_name AS senator,
4     c2.sponsor_name AS senator2
5   FROM cosponsors c1
6   JOIN cosponsors c2
7     ON c1.sponsor_name = c2.cosponsor_name
8     AND c2.sponsor_name = c1.cosponsor_name
9 )
10
11 SELECT senator, COUNT(*) AS mutual_count
12 FROM mutuals
13 GROUP BY senator
14 ORDER BY mutual_count DESC
15 LIMIT 1

```

Run Show Solution Reset

Correct

senator	mutual_count
Brown, Sherrod	65

Now find the most networked senator from each state.

```

1 WITH mutual_counts AS (
2   SELECT
3     senator, state, COUNT(*) AS mutual_count
4   FROM (
5     SELECT DISTINCT
6       c1.sponsor_name AS senator,
7       c1.sponsor_state AS state,
8       c2.sponsor_name AS senator2
9     FROM cosponsors c1
10    JOIN cosponsors c2
11      ON c1.sponsor_name = c2.cosponsor_name
12      AND c2.sponsor_name = c1.cosponsor_name
13  )
14 GROUP BY senator, state
15 ),
16 state_max AS (
17   SELECT
18     state,
19     MAX(mutual_count) AS max_mutual_count
20   FROM mutual_counts
21   GROUP BY state
22 )
23
24 SELECT
25   mutual_counts.state,
26   mutual_counts.senator,
27   mutual_counts.mutual_count
28 FROM mutual_counts
29 JOIN state_max
30   ON mutual_counts.state = state_max.state
31   AND mutual_counts.mutual_count = state_max.max_mutual
32
33

```

CO	Bennet, Michael F.	31
CT	Blumenthal, Richard	55
MO	Blunt, Roy	57
AR	Boozman, John	54

Find the senators who cosponsored but didn't sponsor bills.

```

1 SELECT DISTINCT c1.cosponsor_name
2 FROM cosponsors c1
3 LEFT JOIN cosponsors c2
4   ON c1.cosponsor_name = c2.sponsor_name
5   -- This join identifies cosponsors
6   -- who have sponsored bills
7 WHERE c2.sponsor_name IS NULL
8   -- LEFT JOIN + NULL is a standard trick for excluding
9   -- rows. It's more efficient than WHERE ... NOT IN.
10

```

Run Show Solution Reset

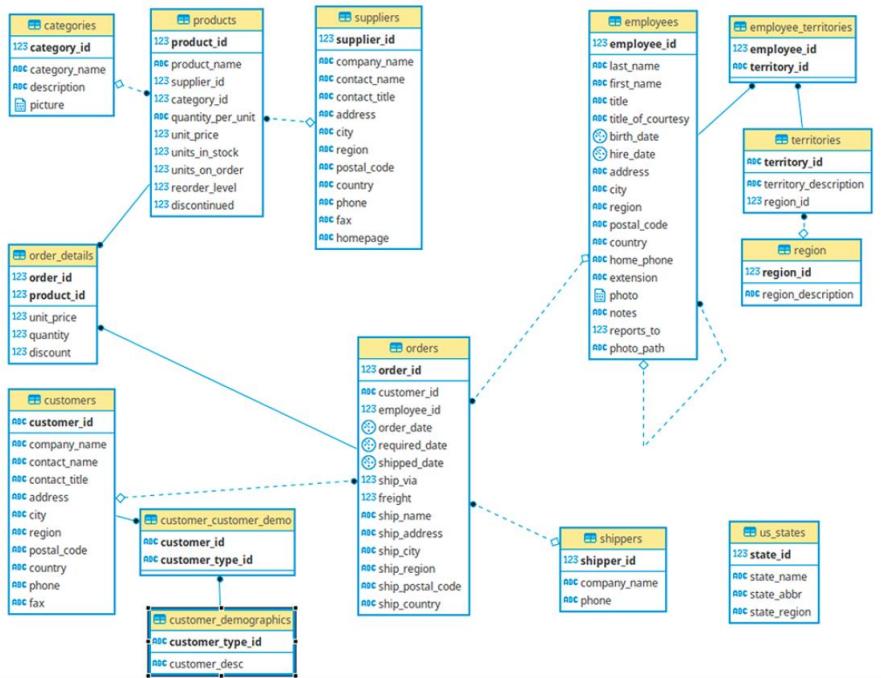
Correct

cosponsor_name
Durbin, Richard
Murphy, Christopher S.

Previous: [Execution Hiatuses](#)

## Ejercicio 3: Consultas en SQL (40 puntos)

Se le ha compartido el archivo `tarea1.sql` que contiene los *queries* necesarios para levantar la base de datos de una tienda distribuidora de productos alimenticios con sus ventas y proveedores de productos. La base de datos tiene una estructura descrita por el siguiente diagrama:



En esta base de datos se registran órdenes de venta (*orders*) en donde una orden de venta puede incluir múltiples productos (registrados por medio de la tabla *order\_details*) y una orden está asociada a un solo cliente (*customers*) y vendida por un solo vendedor (*employees*).

1. Liste los productos (id, nombre y precio unitario) cuyo precio unitario está entre \$15.00 y \$20.00.

```

3912
3913  SELECT product_id, product_name, unit_price
3914  FROM products
3915  WHERE unit_price >= 15 AND unit_price <= 20;
    
```

Data Output		
	product_id [PK] smallint	product_name character varying (40)
1	1	Chai
2	2	Chang
3	16	Pavlova
4	35	Steeleye Stout
5	36	Inlagd Sill
6	39	Chartreuse verte
7	40	Boston Crab Meat
8	44	Gula Malacca
9	49	Maxilaku
10	50	Valkoinen suklaa

2. ¿Cuáles son los productos descontinuados? Para cada producto muestre su ID, nombre del producto y nombre de su categoría?

```

3916
3917  SELECT product_id, product_name, category_name
3918  FROM products
3919  INNER JOIN categories
3920  ON products.category_id = categories.category_id
3921  WHERE discontinued = 1;
3922

```

Data Output Messages Notifications

product_id	product_name	category_name
1	Chai	Beverages
2	Chang	Beverages
5	Chef Anton's Gumbo Mix	Condiments
9	Mishi Kobe Niku	Meat/Poultry
17	Alice Mutton	Meat/Poultry
24	Guaraná Fantástica	Beverages
28	Rössle Sauerkraut	Produce
29	Thüringer Rostbratwurst	Meat/Poultry
42	Singaporean Hokkien Fried Mee	Grains/Cereals
53	Perth Pasties	Meat/Poultry

Total rows: 10 of 10 Query complete 00:00:00.075

3. Liste los productos (id, nombre y precio unitario) cuyo precio está por encima del promedio de precios

```

3924  SELECT product_id, product_name, unit_price
3925  FROM products
3926  WHERE unit_price > (
3927      SELECT AVG(unit_price) FROM products);
3928

```

Data Output Messages Notifications

product_id	product_name	unit_price
1	Uncle Bob's Organic Dried Pears	30
2	Northwoods Cranberry Sauce	40
3	Mishi Kobe Niku	97
4	Ikura	31
5	Queso Manchego La Pastora	38
6	Alice Mutton	39
7	Carnarvon Tigers	62.5
8	Sir Rodney's Marmalade	81
9	Gumbär Gummibärchen	31.23
10	Schoggi Schokolade	43.9

Total rows: 25 of 25 Query complete 00:00:00.080

4. Genere un listado de las órdenes que aún no han sido entregadas (*shipped*), incluyendo el monto total de la orden (considerando descuentos)

```

3929  SELECT orders.order_id, (order_details.unit_price - order_details.unit_price * order_details.discount) AS total
3930  FROM orders
3931  INNER JOIN order_details ON orders.order_id = order_details.order_id
3932  WHERE orders.shipped_date IS NULL;

```

Data Output Messages Notifications

order_id	total
1	43.32
2	13.3
3	21.5
4	12
5	20
6	45.6
7	18
8	20
9	19.5
10	10

Total rows: 73 of 73 Query complete 00:00:00.115 Ln 3931, C

5. ¿Cuál es el rango de precios de productos que maneja la tienda? Elabore un query que muestre el producto más caro y el mas barato manejado en el catálogo de productos de la tienda, considerando solo los productos que no estén descontinuados.

```

3936 SELECT product_name, unit_price
3937 FROM products
3938 WHERE unit_price = (SELECT MAX(unit_price) FROM products)
3939 OR unit_price = (SELECT MIN(unit_price) FROM products)
3940 AND discontinued != 1;

```

Data Output Messages Notifications

	product_name character varying (40)	unit_price real
1	Geitost	2.5
2	Côte de Blaye	263.5

6. Encuentre el cliente que hay registrado el mayor número de órdenes, mostrando su nombre y la cantidad de órdenes registradas

```

3947 SELECT orders.order_id, orders.customer_id, company_name, quantity FROM orders
3948 INNER JOIN customers
3949 ON orders.customer_id = customers.customer_id
3950 INNER JOIN order_details
3951 ON orders.order_id = order_details.order_id
3952 WHERE quantity = (SELECT MAX(quantity) FROM order_details);
3953

```

3954 SELECT customer\_id, company\_name, COUNT(customers.company\_name) AS order\_amount

Data Output Messages Notifications

	order_id smallint	customer_id character	company_name character varying (40)	quantity smallint
1	10764	ERNSH	Ernst Handel	130
2	11072	ERNSH	Ernst Handel	130

7. Liste los cinco productos que han generado el mayor volumen de dinero en ventas (registradas por medio órdenes). Para cada uno liste su el nombre del producto y el volumen monetario generado en órdenes.

```

3959 SELECT products.product_name, SUM(order_details.quantity * order_details.unit_price) AS total
3960 FROM order_details
3961 JOIN products
3962 ON products.product_id = order_details.product_id
3963 GROUP BY products.product_name
3964 ORDER BY total DESC
3965 LIMIT 5

```

Data Output Messages Notifications

	product_name character varying (40)	total double precision
1	Côte de Blaye	149984.20082092285
2	Thüringer Rostbratwurst	87736.40051269531
3	Raclette Courdavault	76296
4	Camembert Pierrot	50286.00037384033
5	Tarte au sucre	49827.89999771118

## Ejercicio 4: Cálculo de covarianza (20 puntos)

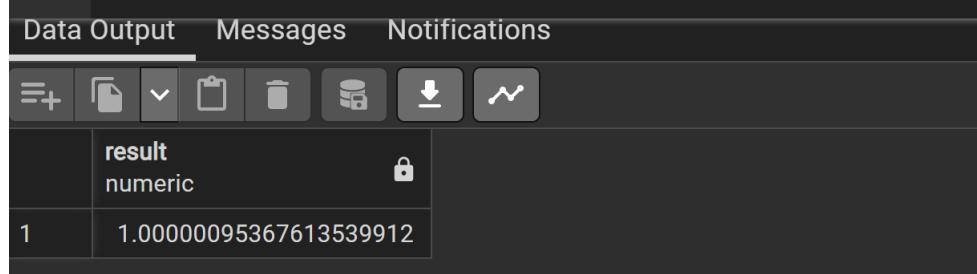
En este último ejercicio utilizaremos la base de datos **covid** que se encuentra disponible en CANVAS, con el objetivo de responder a la pregunta *¿cómo se relacionan el tabaco con respecto de la hipertensión en dicha base de datos?* Esto quiere decir que si un paciente tiene hipertensión, si esto está íntimamente relacionado con que fuma.

Una forma de estudiar la variación conjunta de dos variables es utilizar la *covariancia muestral* (si lo necesita, puede refrescar dicho concepto [aquí](#)). Mientras más alto sea el valor de la covarianza muestral entre dos variables, mayor será la tendencia de estas a comportarse de la misma forma. Por el contrario, una covarianza negativa indica que las variables tienden a estar inversamente relacionadas.

Una forma de computar la covarianza entre dos variables  $X$  y  $Y$  es la siguiente:

4.1 Escriba un query para computar el término  $1/(n-1)$  de la tabla de covid, en donde  $n$  representa la cantidad de casos de covid.

```
3 SELECT 1.0 * COUNT(*) / (COUNT(*) - 1) AS result  
4 FROM covid;
```



The screenshot shows a database interface with a query editor and a results viewer. The query editor contains the code provided above. The results viewer has tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab shows a table with one row. The table has two columns: 'result' (type: numeric) and a lock icon. The value in the 'result' column is 1.00000095367613539912.

	result numeric	🔒
1	1.00000095367613539912	