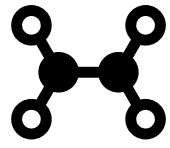
Cracking the SQL Interview for DATA SCIENTISTS

Nervous about your SQL Interview?
Anxiety ends here.

90 SQL CODING QUESTIONS & SOLUTIONS

LEON WEI

"SQLPad Helped me gain mastery of all the core SQL concepts in a structured manner with a thoughtfully designed business schema"
- Nisheeth, Data Scientist



INTRODUCTION

Hi, thanks for purchasing this SQL ebook!

SQL is a must-know programming language for any data analytics professionals.

However, many college graduates or young professionals start their job searches without solid SQL coding skills, - which ultimately costs them their dream jobs.

This ebook includes **90 SQL interview questions and solutions,** which will help data candidates learn or refresh their SQL coding skills and get ready for a SQL interview.

This book is also accompanied by the **Cracking the SQL Interview for Data Scientists Course (sold separately),** which includes 21 video lectures with tips and techniques to ace a SQL interview.

If there are any questions, feedback, please feel free to send me an email at hello@sqlpad.io.

Leon Wei,
March 2021.

ABOUT THE AUTHOR

Leon Ice Climbing at Matanuska Glacier, Alaska 2019

Leon Wei is the founder of sqlpad.io and instamentor.com, two interview preparation sites for data professionals.

Most recently, he was a senior manager of machine learning at **Apple**, where he lead a team of data scientists and engineers building large-scale machine learning systems for Apple's billion dollars businesses.

Before that, he ran the machine learning and data science team at **Chegg**, a leading education technology company. He also worked as a research scientist at **Amazon** developing its real-time pricing engine using machine learning.

He has interviewed thousands of candidates as a hiring manager or part of a hiring committee.

If you need any help with your job search or career, you can hire him as your coach/mentor on instamentor.com

SELECTED CUSTOMER REVIEWS

“Worked through all the problems at sqlpad.io - Great teaching / refresher tool and highly recommended!”

— Mike Metzger

“I have been relearning SQL and sqlpad.io has been a great resource. There is a good ratio of a new concept to practice questions! Highly recommended.”

— Anup Kalburgi

“I signed up on SQLPad and was pleasantly surprised when Leon helped me out personally on the website. In two mock interviews with him from instamentor.com, he was meticulously prepared and very professional.”

— Rahul Nayak

“SQLPad is the best website I have used for practicing SQL.The databases and practice problems resemble real-world data and daily tasks in a data scientist / data analyst role.”

— William

“SQLPad helped me gain mastery of all the core SQL concepts in a structured manner with a thoughtfully designed business schema. ”

— Nisheeth

“This course helped me get into a final round of a data scientist interview at Facebook. ”

— Justin

“I feel so confident with SQL after practicing here. ”

— Juhi

“I am currently in question 30, I really like some of the questions you posted here. Also thank you for your excellent customer service !!!! ”

— Mamath

“Enjoying it so far, love the mix with practical exercises and the focus to land a job, that's really important because normal courses do not prepare you for interviews. Also, Leon is really kind and helpful.”

— Jose

“This site is a great resource for SQL interview practice questions. The interface is excellent! And even as someone who currently uses SQL for their day job, I have definitely improved my skills by working through these problems.”

— Kyle

WHAT A SQL INTERVIEW MAY LOOK LIKE AT A TOP TECH COMPANY

Summary

Let me walk you through a typical SQL interview at one of the top tech firms, so you can get a sense of how a SQL interview is conducted and what the main criteria are to determine your performance. I will also share a few tips and techniques to give you an edge for your next interview.

Introduction

SQL and its related database management system, such as **Hive**, **BigQuery**, **Spark SQL**, are the #1 toolset in managing and processing data in the industry. If you are interviewing for a data-related job, you need to prove your SQL proficiency.

Here is a list of typical roles that will require at least one round of SQL interviews:

1. Data analyst
2. Data scientist
3. Data engineer
4. Business intelligence engineer
5. Product analyst
6. Decision scientist
7. Research scientist
8. Software engineer (especially those on the backend side)

Why is SQL interview important for a candidate?

SQL Interview can sometimes bear other names such as **Technical Data Interview** or **Data Processing Interview.** The primary purpose is to make sure a candidate is hands-on with data and can contribute immediately after joining the company.

You will be asked to perform a series of SQL coding exercises to extract facts or insights from the given data and answer follow-up questions based on different scenarios.

I have been using SQL for 10+ years, and it is still my #1 choice in preparing data.

However, I have interviewed so many candidates who started their job searches without a solid coding skill in SQL. In the end, the SQL skills gap cost their dream jobs.

How is a SQL interview conducted?

Before COVID-19, a typical interview process included 2 or 3 rounds of phone screens and a final round of onsite interviews.

Nowadays, most of the interviews are online, often assisted by video conferencing tools such as **Zoom**, **Webex**, or **Google Hangout**.

In the old days, during a final interview. a candidate would be asked to write their SQL code on a whiteboard without any help from a modern IDE for syntax highlighting or auto-completion.

Nowadays, the whiteboard has been replaced by interactive online coding tools such as **coderpad** or **google docs**.

Tips: Before your interview, make sure you familiarize yourself with those online coding environments. If you don't know what tool they use, ask your recruiter.

Right before the interview, you will receive a link that points to an online coding environment, where you will code up solutions in SQL.

An interviewer will be able to see everything you type in real-time.

A typical SQL interview process

Let's assume you are interviewing for a data scientist role at a major music streaming company (e.g., Spotify).

You are given 3 (simplified) tables:

TABLE 1: song

```
-- id: BIGINT
-- name: VARCHAR(255)
-- artist_id: BIGINT
```

The first table is a list of songs and their metadata, song names, and associated artists.

TABLE 2: artist

```
-- id: BIGINT
-- name: VARCHAR(255)
```

The second table is simply an artist table with only 2 columns: artist id and the artist name.

TABLE 3: daily_plays

```
-- date: DATE
-- country: VARCHAR(2) ('uk','us','in','jp','cn')
-- song_id BIGINT
-- plays BIGINT
```

The third table is a daily aggregate table that counts the number of times a song is played in different countries.

And you are asked the following questions:

Question 1:

Write a query to return the top 5 songs in the UK yesterday.

Tips: a common mistake a beginner usually makes is diving into coding right away without fully understanding the question.

An interviewer often start with a vague question to see if a candidate is good at communicating.

It is crucial to clarify the following key points before diving into coding.

1. What does it mean '**top**'? What is **top's** definition? Do you mean number of plays, number of unique customers, or perhaps other metrics?

2. What columns should I return, a song id, or include a song's name?

3. What if there are ties? Should I return only 1 row or all of them? (optionally)

4. Should we care about the timezone?

After clarifying this question, you can go ahead and start coding, and here is a sample solution:

```
SELECT
    S.song_id,
    S.name
FROM daily_plays P
INNER JOIN song S
ON P.song_id = S.id
WHERE P.country = 'UK'
AND P.date = CURRENT_DATE - 1
ORDER BY daily_plays DESC
LIMIT 5;
```

Question 2:

Write a query to return the top 5 artists in the **US** and **UK** yesterday.

This question tries to evaluate whether a candidate knows window functions.

Window functions are asked a lot, especially if the hiring company has a global business. To identify top artists/songs/albums across different countries, you need to know how to use **ROW_NUMBER** and **RANK**. We will dive into Window functions in our next section.

Again, before start coding, try to clarify a few things with the interviewer:

1. The definition of **'top.'**

2. Columns to return.

3. Situations when there are ties.

4. This is probably subtle, but because multiple artists can perform a song, you need to talk to the interviewer how to '**count**' the plays properly. For simplicity, let's assume a song only has one artist.

And here is a sample solution.

```
WITH artist_ranking AS (
  SELECT
    A.artist_id,
    MAX(A.artist_name) AS artist_name,
    MAX(P.country) AS country
```

```
    ROW_NUMBER() OVER(PARTITION BY country ORDER BY SUM(plays) DESC) AS
ranking
  FROM daily_plays P
  INNER JOIN song S
  ON P.song_id = S.id
  INNER JOIN artist A ON
    A.artist_id = S.artist_id
  WHERE P.country IN ('UK', 'US')
  AND P.date = CURRENT_DATE - 1
  GROUP BY A.artist_id
)
SELECT artist_id, artist_name, country, ranking
FROM artist_ranking
WHERE ranking <= 5
LIMIT 5;
```

At this step, if you completed the above 2 questions, the interview would have a good sense that you are very good with SQL coding. He would likely pause SQL coding and transition to follow-up analytics questions.

Sample analytics question #1: Taylor Swift is a popular artist, but her songs' plays dropped yesterday. How would you analyze this, what data would you use, and what would be your process?

For this kind of question (**data change: drop/increase**), I have created a framework that you can borrow:

1. Clarify the question.

1. What do you mean dropped, comparing today's data with yesterday's or same day last week?
2. Did it drop a lot? If it only dropped a little, maybe we don't have to worry about it?
3. And you were told by the interviewer, it's comparing yesterday to the day before, and yes, it indeed dropped a lot significantly.

2. Rule out technical issues first.

In my 15 years of career, the #1 cause of poor data quality often comes from a technical issue, e.g., power outage, a bug in our logging code, someone checked in a code without testing, forgot to pay a license fee, and the third-party software stopped working.

3. Whether the drop is a big deal.

Even it is a significant drop. We still don't know whether we should treat it as a big problem, e.g., it could be because of seasonality, the day before yesterday was **New Years' Day**, so of course the number will drop, and all other artists' songs are also dropped significantly as well.

And you were told by the interviewer, yes, it is a big problem, and it is not due to seasonality, and it only happened to Taylor.

4. Dive into data

After ruling out those potential issues, now it's time to dive into data.

4.1 Start with internal data (data that you already have full access to).

We can slice Taylor's song plays by different countries, genres, devices, iOS vs. Android, Web vs. Mobile, etc. And it could be because of a particular country, we just lost a popular song's license in that country to a competitor, for example.

4.2 external data (might take extra time/resource to obtain)

Finally, you can also mention external data impact, e.g., a competitor such as Amazon Music just had a new product launch. They are giving out free listening to all Taylor's songs. Therefore, we lost a bunch of users.

Maybe it is because this artist dropped out of a top chart on Billboard or it could also because the artist lost a lot of followers after posting something controversial on social media.

A few follow-up questions for your reference.

This round of interviews is finished at this step, and the next interviewer is waiting to join the conversation, good job!

Here are a few more questions that could be asked during interview, in case you want to continue practicing.

1. Write a query to find the top trending artist.
2. How would you design a system to identify trending artists?
3. We want to expand our music streaming services to a few more countries. Can you help find which new countries we should add our services to? What data will you use, and why?
4. How would you design a HOT 100 artist chart?

First, clarify the interviewer's requirements and the goals, why do we need to build this, what problems are we trying to solve, who are our customers we are serving, why are you not happy with the existing solution?

Always start small, define a handful of critical product features first, and only launch it in one or two countries to test it out. And don't forget to mention you will always keep iterating and fine-tuning the algorithms/product features to expand its scope and bring it to more markets/countries.

Tips: Many tech companies really care about how quickly they can launch the first version for system design questions. Bias towards action, and as Reid Hoffman famously put it: if you are not embarrassed about your first product, you launched too late.

Conclusion

I have shown you what a typical SQL interview (sometimes called technical data interview or data processing interview) process may look like. I also gave you some tips and techniques and a framework to answer that standard interview questions.

I hope you learned something today. If you have any questions or want some help with your job search, please feel free to reach out to me at instamentor.

WINDOW FUNCTIONS

Introduction

WINDOW functions are a family of SQL functions that are frequently asked during a data scientist job interview.

However, writing a bug-free WINDOW function-based SQL query could be quite challenging for any job candidates, especially those who just get started with SQL.

In this article, I will share some typical WINDOW functions specific interview questions, their patterns, and step-by-step solutions.

Outline

I am going to break down this article into 4 sections:

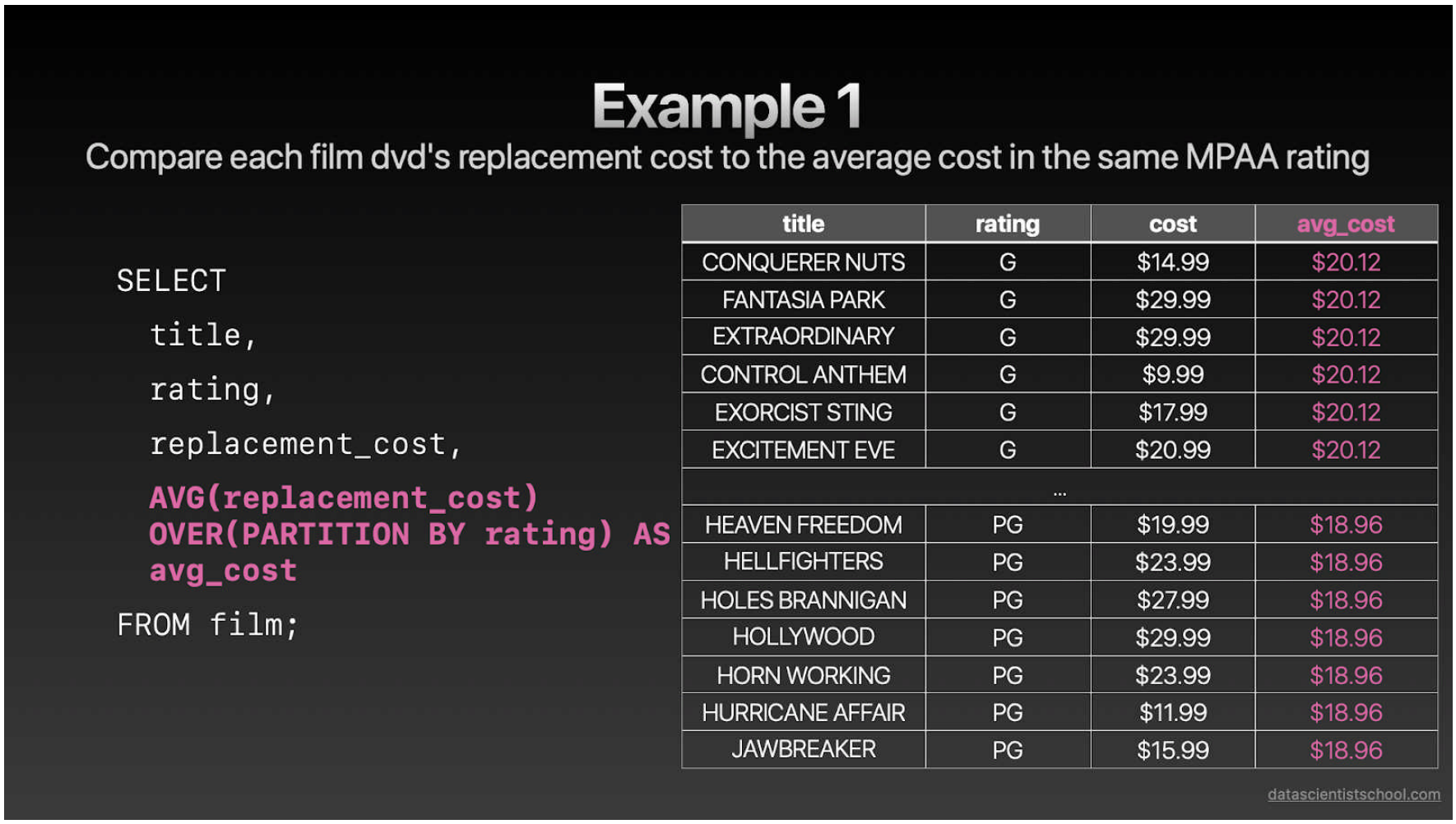
1. In the first section, I will go through a few WINDOW functions based on regular aggregate functions, such as **AVG**, **MIN/MAX**, **COUNT, SUM.**
2. In section 2, I will focus on rank-related functions, such as **ROW_NUMBER**, **RANK**, and **RANK_DENSE**. Those functions are handy when generating ranking indexes, and you need to be fluent in those functions before entering a data scientist SQL interview.
3. In the third section, I will talk about generating statistics (e.g., percentiles, quartiles, median, etc .) with the **NTILE** function, a common task for a data scientist.
4. In the last section, let's focus on **LAG** and **LEAD**, two functions that are super important if you are interviewing for a role that requires dealing with time-series data.

Section 1. WINDOW functions based on regular aggregate functions: AVG, MIN/MAX, SUM, COUNT

Window functions are functions that perform calculations across a set of rows related to the current row.

It is comparable to the type of calculation done with an aggregate function, but unlike regular aggregate functions, window functions do not group several rows into a single output row — the rows retain their own identities.

Behind the scenes, the window functions process more than just the query results' current row.



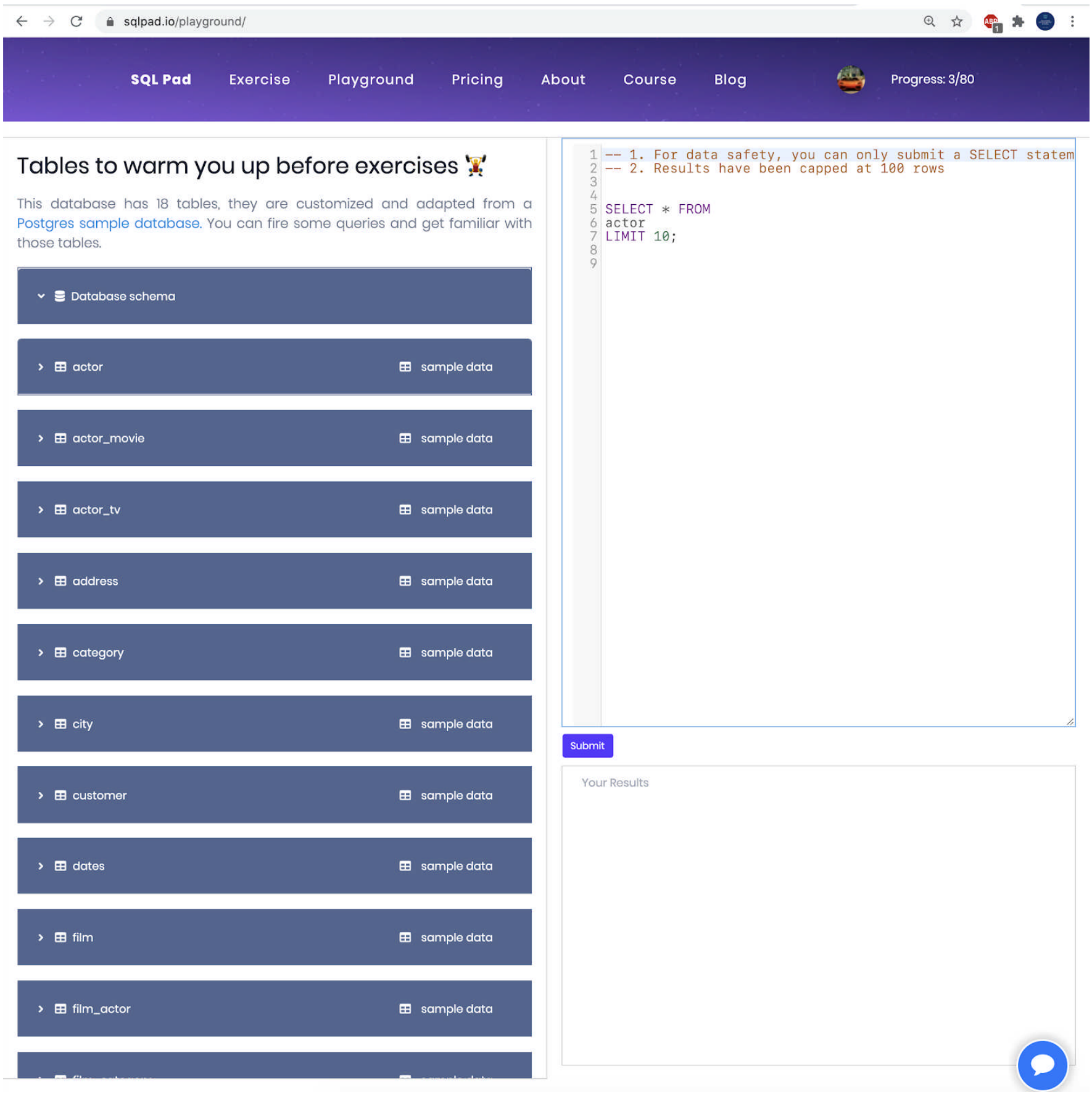
All examples in this article are based on **movie DVD rental business data**. In this first example, our goal is to compare each movie DVDs replacement cost to the average cost of movies sharing the same MPAA ratings.

```sql
SELECT
    title,
    rating,
    replacement_cost,
    AVG(replacement_cost) OVER(PARTITION BY rating) AS avg_cost
FROM film;
```

For those of you not based in the United States, an **MPAA rating** is a film rating system that decides a film's suitability for specific audiences, based on a film's content. For example, **G** means it's appropriate for all ages, while **PG-13** contains materials that could be inappropriate for children under 13.

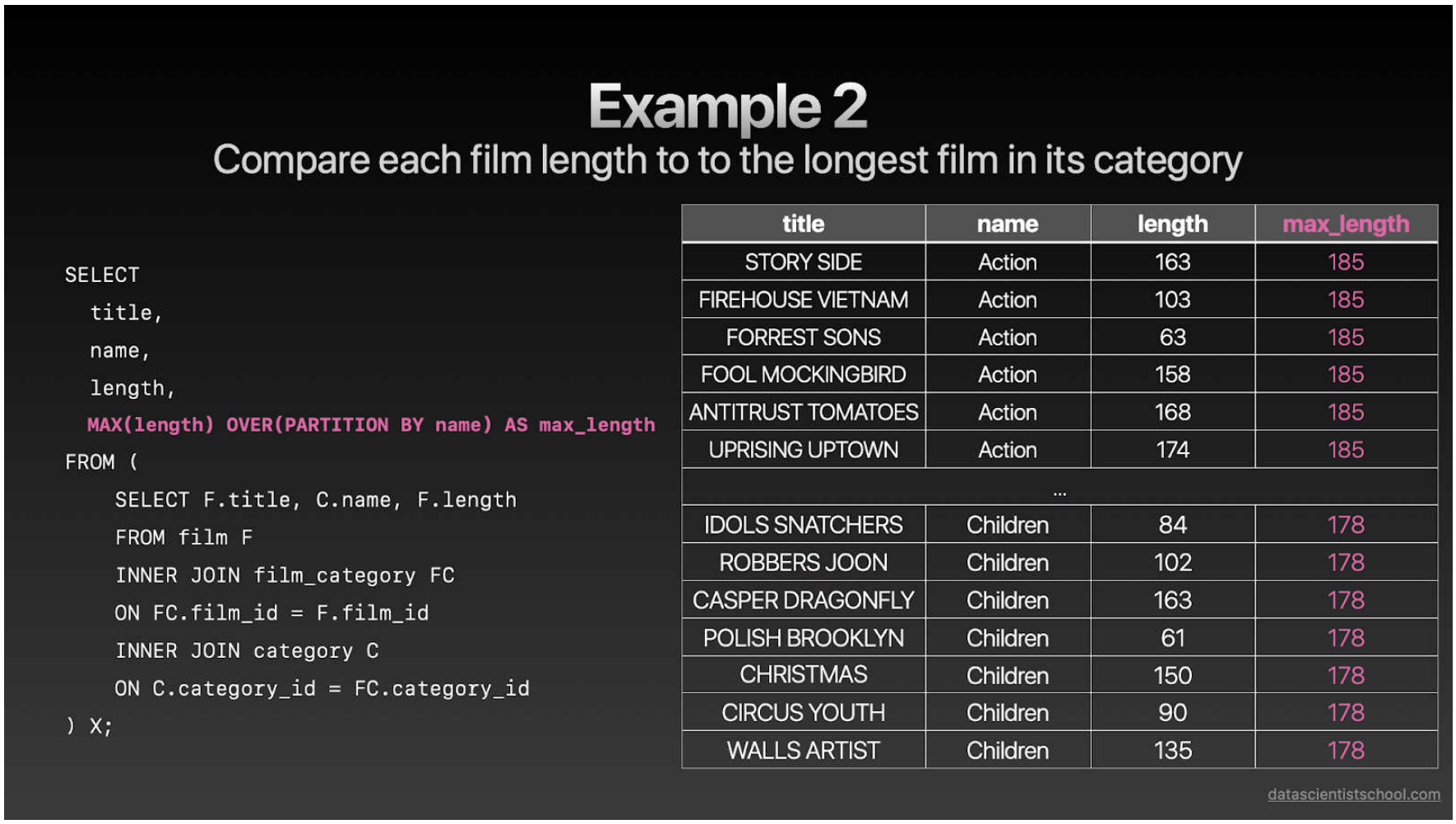
There is no **GROUP BY** clause for the **AVG** function, but how does the SQL engine know which rows to use to compute the average? The answer is the **PARTITION BY** clause inside the **OVER()** utility, and we are calculating the average based on a unique value of rating.

In the final output, every row has the average cost from the same rating. You can perform other analyses such as dividing a movie's **cost** to **avg_cost** and find out its expense relative to similar movies.



https://sqlpad.io/playground/

All tables in this article are available on SQLPad's online SQL playground. If you want to follow along and submit queries against those tables, please feel free to go to sqlpad.io/playground and have some fun.



Let's take a look at another example. In this example, I want to compare every movie's length (in minutes) to the maximum length of movies from the same category.

```sql
SELECT
  title,
  name,
  length,
  MAX(length) OVER(PARTITION BY name) AS max_length
FROM (
  SELECT F.title, C.name, F.length
  FROM film F
  INNER JOIN film_category FC
  ON FC.film_id = F.film_id
  INNER JOIN category C
  ON C.category_id = FC.category_id
) X;
```

It's very similar to the first example. Still, I combined a **MAX** function with **OVER** and **PARTITION BY** to create a window function, which returned the maximum movie length inside the same movie category.

For the first film: **story side**, its length is 163 minutes, and the maximum length of an **action** film (same category) is 185. If I compare every action movie's length to 185, I can get a sense of how long this specific movie is, relative to its category, as films from different genres tend to have different durations.

Example 3

Overall progress for binge watching all films

```
SELECT
  film_id,
  title,
  length,
  SUM(length) OVER(ORDER BY film_id)
AS running_total,
  SUM(length) OVER() AS overall,
  SUM(length) OVER(ORDER BY film_id)
* 100.0 /SUM(length) OVER() AS
running_percentage
FROM film
ORDER BY film_id;
```

film_id	title	length	running_total	overall	running %
1	ACADEMY	86	86	115,267	0.07%
2	ACE	48	134	115,267	0.12%
3	ADAPTATION	50	184	115,267	0.16%
4	AFFAIR	117	301	115,267	0.26%
5	AFRICAN EGG	130	431	115,267	0.37%
6	AGENT TRUMAN	169	600	115,267	0.52%
...					
994	WYOMING	100	114,563	115,267	99.39%
995	YENTL IDAHO	86	114,649	115,267	99.46%
996	YOUNG	183	114,832	115,267	99.62%
997	YOUTH KICK	179	115,011	115,267	99.78%
998	ZHIVAGO CORE	105	115,116	115,267	99.87%
999	ZOOLANDER	101	115,217	115,267	99.96%
1000	ZORRO ARK	50	115,267	115,267	100.00%

datascientistschool.com

```sql
SELECT
  film_id,
  title,
  length,
  SUM(length) OVER(ORDER BY film_id) AS running_total,
  SUM(length) OVER() AS overall,
  SUM(length) OVER(ORDER BY film_id) * 100.0 /SUM(length) OVER() AS
running_percentage
FROM film
ORDER BY film_id;
```

Let's take a look at a more complicated example, where we calculated a **running sum** with a window function.

Assuming it's the holiday season, I want to binge-watch all 1000 movies, starting from **movie id=1**. After finishing each film, I want to know what my overall progress is. I can use **SUM** and **OVER** to calculate a running total of time to get my progress.

Notice that there is no **PARTITION BY** clause because I am not grouping those movies into any sub-categories. I want to compute my overall progress but not based on any subgroups or categories.

Another thing to notice is that if I don't add anything inside the **OVER**() function, I get the total number of minutes from the entire movie catalog. As you can see from the second last column: they all have the same value of **115267**, but after I add the **ORDER BY** clause, I get the **running total** of the minutes up to that specific row (**running_total** column).

Again, please feel free to go to sqlpad's playground and play with this film table until you become comfortable with the syntax.

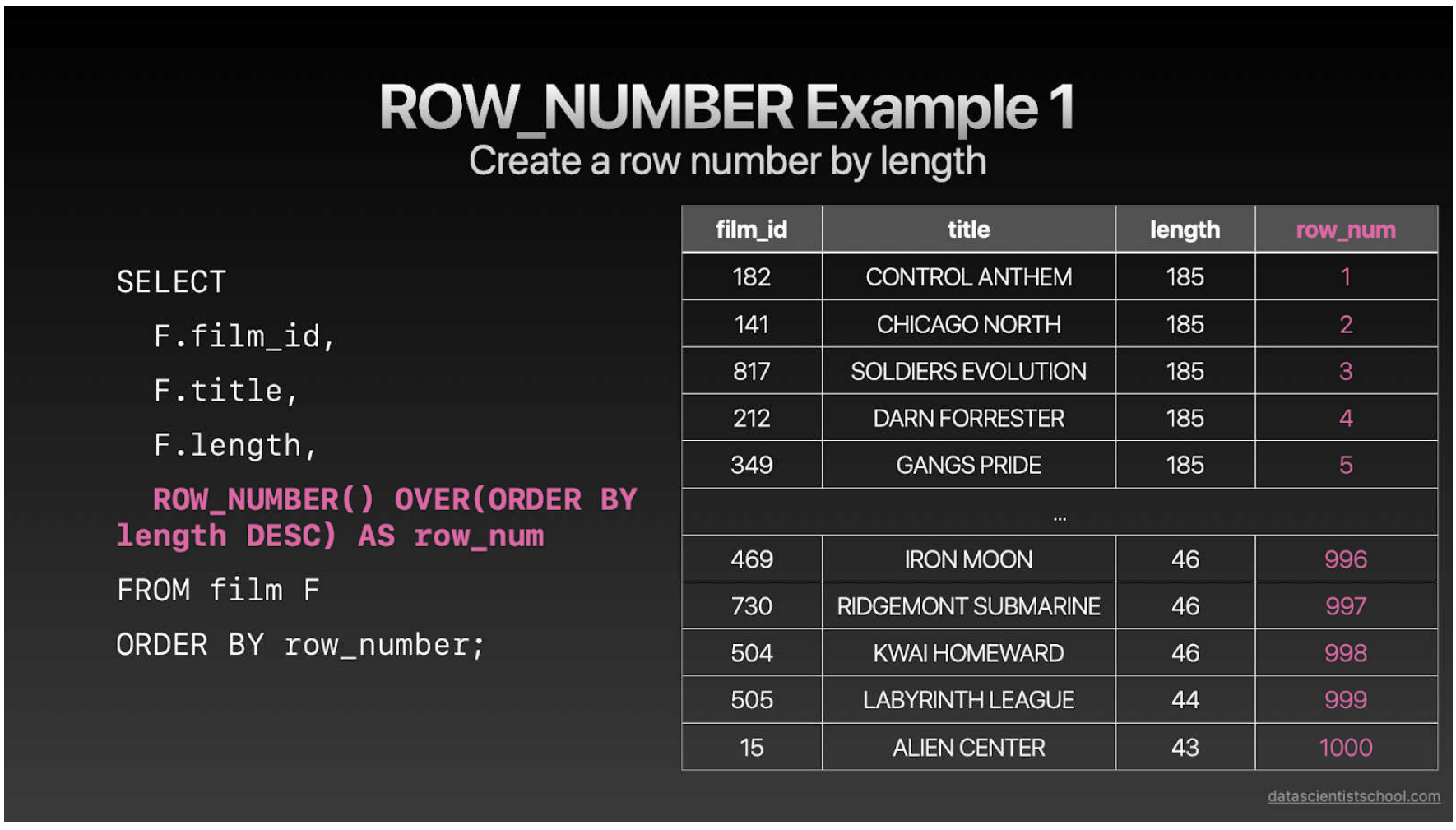
If you are interested in practicing a few more WINDOW functions that we just covered, here are 4 exercises for you to reinforce your learning.

Time to complete: ~ 30 -45 mins.

- #58: Percentage of revenue per movie
- #59: Relative percentage of a movie's revenue per category
- #60: Compare each film rentals with the average rental per category
- #61: Customer spend vs. average customer spend in the same store

Section 2: ROW_NUMBER, RANK, DENSE_RANK

Let's go through some of the essential WINDOW functions: **ROW_NUMBER** and **RANK**.




```sql
SELECT
  F.film_id,
  F.title,
  F.length,
  ROW_NUMBER() OVER(ORDER BY length DESC) AS row_num
FROM film F
ORDER BY row_number;
```

In this example, our goal is to create a ranking index based on the movie's length for the entire movie catalog.

As you can see, the **ROW_NUMBER** function generates a sequence of integers, starting from 1, for each row.

But movies with the same lengths were given a **DIFFERENT** row number, as the database randomly assigned a unique number when there was a tie.

ROW_NUMBER Example 2

Create a row number by length in a category

```
SELECT
  F.film_id,
  F.title,
  F.length,
  C.name AS category,
  ROW_NUMBER() OVER(PARTITION BY C.name
ORDER BY F.length DESC) row_num
FROM film F
INNER JOIN film_category FC
ON FC.film_id = F.film_id
INNER JOIN category C
ON C.category_id = FC.category_id
ORDER BY C.name, row_number;
```

film_id	title	length	name	row_num
991	WORST BANGER	185	Action	1
212	DARN FORRESTER	185	Action	2
126	CASUALTIES ENCINO	179	Action	3
707	QUEST MUSSOLINI	177	Action	4
287	ENTRAPMENT SATISFACTION	176	Action	5
...				
690	POND SEATTLE	185	Animation	1
349	GANGS PRIDE	185	Animation	2
886	THEORY MERMAID	184	Animation	3
820	SONS INTERVIEW	184	Animation	4
510	LAWLESS VISION	181	Animation	5

datascientistschool.com

```sql
SELECT
  F.film_id,
  F.title,
  F.length,
  C.name AS category,
  ROW_NUMBER() OVER(PARTITION BY C.name ORDER BY F.length DESC) row_num
FROM film F
INNER JOIN film_category FC
ON FC.film_id = F.film_id
INNER JOIN category C
ON C.category_id = FC.category_id
ORDER BY C.name, row_number;
```

Let's take a look at another example. Instead of comparing a movie's length to all other films from the entire catalog, we can rank them within each movie category using PARTITION BY.

ROW_NUMBER with **OVER** and **PARTITION BY** is a regular pattern that is frequently used in advanced SQL. Mastering this pattern will make your life much easier.

For example, imagine you are working at an e-commerce company, and it has a global business. Your boss asks you to send her a list of best sellers for each country. You can use **ROW_NUMBER** and **PARTITION BY** to generate this list quickly.

RANK Example 1

Create a ranking index by length

```
SELECT
  F.film_id,
  F.title,
  F.length,
  RANK() OVER(ORDER BY length
DESC) AS ranking
FROM film F
ORDER BY ranking;
```

film_id	title	length	ranking
182	CONTROL ANTHEM	185	1
141	CHICAGO NORTH	185	1
817	SOLDIERS EVOLUTION	185	1
212	DARN FORRESTER	185	1
349	GANGS PRIDE	185	1
609	MUSCLE BRIGHT	185	1
690	POND SEATTLE	185	1
872	SWEET BROTHERHOOD	185	1
991	WORST BANGER	185	1
426	HOME PITY	185	1
886	THEORY MERMAID	184	11
597	MOONWALKER FOOL	184	11
821	SORORITY QUEEN	184	11
813	SMOOCHY CONTROL	184	11
820	SONS INTERVIEW	184	11
198	CRYSTAL BREAKING	184	11
180	CONSPIRACY SPIRIT	184	11
499	KING EVOLUTION	184	11
996	YOUNG LANGUAGE	183	19
469	IRON MOON	46	996
730	RIDGEMONT SUBMARINE	46	996
504	KWAI HOMEWARD	46	996
505	LABYRINTH LEAGUE	44	999
15	ALIEN CENTER	43	1000

datascientistschool.com

```sql
SELECT
  F.film_id,
  F.title,
  F.length,
  RANK() OVER(ORDER BY length DESC) AS ranking
FROM film F
ORDER BY ranking;
```

Let's take a look at the **RANK** function, which is very similar to **ROW_NUMBER**. The difference between **RANK** and **ROW_NUMBER** is that **RANK** assigns the same unique values if there is a tie and restarts the next value with the total number of rows up to that row. Notice how it jumps from 1 to 11.

RANK Example 2

Create a ranking index by length in a category

```
SELECT
  F.film_id,
  F.title,
  F.length,
  C.name AS category,
  RANK() OVER(PARTITION BY C.name ORDER BY
F.length DESC) ranking
FROM film F
INNER JOIN film_category FC
ON FC.film_id = F.film_id
INNER JOIN category C
ON C.category_id = FC.category_id
ORDER BY C.name, ranking;
```

film_id	title	length	category	ranking
991	WORST BANGER	185	Action	1
212	DARN FORRESTER	185	Action	1
126	CASUALTIES ENCINO	179	Action	3
707	QUEST MUSSOLINI	177	Action	4
287	ENTRAPMENT	176	Action	5
...				
690	POND SEATTLE	185	Animation	1
349	GANGS PRIDE	185	Animation	1
886	THEORY MERMAID	184	Animation	3
820	SONS INTERVIEW	184	Animation	3
510	LAWLESS VISION	181	Animation	5

datascientistschool.com

```
SELECT
  F.film_id,
  F.title,
  F.length,
  C.name AS category,
  RANK() OVER(PARTITION BY C.name ORDER BY F.length DESC) ranking
FROM film F
INNER JOIN film_category FC
ON FC.film_id = F.film_id
INNER JOIN category C
ON C.category_id = FC.category_id
ORDER BY C.name, ranking;
```

Similarly, we can also generate rankings within a subgroup with the help of **PARTITION BY**.

DENSE_RANK Example 1

Create a dense ranking index by length

```
SELECT
  F.film_id,
  F.title,
  F.length,
  DENSE_RANK() OVER(ORDER BY length DESC) AS ranking
FROM film F
ORDER BY ranking;
```

film_id	title	length	ranking
182	CONTROL ANTHEM	185	1
141	CHICAGO NORTH	185	1
817	SOLDIERS EVOLUTION	185	1
212	DARN FORRESTER	185	1
349	GANGS PRIDE	185	1
609	MUSCLE BRIGHT	185	1
690	POND SEATTLE	185	1
872	SWEET BROTHERHOOD	185	1
991	WORST BANGER	185	1
426	HOME PITY	185	1
886	THEORY MERMAID	184	2
597	MOONWALKER FOOL	184	2
821	SORORITY QUEEN	184	2
	...		
398	HANOVER GALAXY	47	139
869	SUSPECTS QUILLS	47	139
469	IRON MOON	46	140
730	RIDGEMONT SUBMARINE	46	140
504	KWAI HOMEWARD	46	140
505	LABYRINTH LEAGUE	44	141
15	ALIEN CENTER	43	142

datascientistschool.com

```sql
SELECT
  F.film_id,
  F.title,
  F.length,
  DENSE_RANK() OVER(ORDER BY length DESC) AS ranking
FROM film F
ORDER BY ranking;
```

The last function I want to show you is **DENSE_RANK**. It is very similar to **RANK** but differs in how it handles **ties**. It restarts with the following immediate consecutive value rather than creating a gap.

As you can see here, for the first 2 rows, two movies both have a value of 1. Instead of restarting from 3, the next **dense_rank** value starts as 2.

DENSE_RANK Example 2

Create a dense rank index by length in a category

```
SELECT
  F.film_id,
  F.title,
  F.length,
  C.name AS category,
  DENSE_RANK() OVER(PARTITION BY C.name
ORDER BY F.length DESC) ranking
FROM film F
INNER JOIN film_category FC
ON FC.film_id = F.film_id
INNER JOIN category C
ON C.category_id = FC.category_id
ORDER BY C.name, ranking;
```

film_id	title	length	category	ranking
991	WORST BANGER	185	Action	1
212	DARN FORRESTER	185	Action	1
126	CASUALTIES ENCINO	179	Action	2
707	QUEST MUSSOLINI	177	Action	3
287	ENTRAPMENT	176	Action	4
511	LAWRENCE LOVE	175	Action	5
927	UPRISING UPTOWN	174	Action	6
549	MAGNOLIA	171	Action	7
250	DRAGON SQUAD	170	Action	8
...				
811	SMILE EARRING	60	Travel	43
981	WOLVES DESIRE	55	Travel	44
386	GUMP DATE	53	Travel	45
931	VALENTINE	48	Travel	46
784	SHANGHAI TYCOON	47	Travel	47

datascientistschool.com

```sql
SELECT
  F.film_id,
  F.title,
  F.length,
  C.name AS category,
  DENSE_RANK() OVER(PARTITION BY C.name ORDER BY F.length DESC) ranking
FROM film F
INNER JOIN film_category FC
ON FC.film_id = F.film_id
INNER JOIN category C
ON C.category_id = FC.category_id
ORDER BY C.name, ranking;
```

Similarly, if we combine **PARTITION BY** with **DENSE_RANK**, we can get consecutive rankings inside a category. The most significant value for a **dense_rank** is the total number of unique values inside of a partition.

In summary, **ROW_NUMBER**, **RANK,** and **DENSE_RANK** are 3 functions that are very helpful to generate rankings. I have used **ROW_NUMBER** quite often at work as a data scientist, and I have also used **RANK** occasionally when dealing with ties (rare).

Time for some exercises. I have prepared 3 exercises to help with your understanding.

Time to complete: ~30 -45 mins.

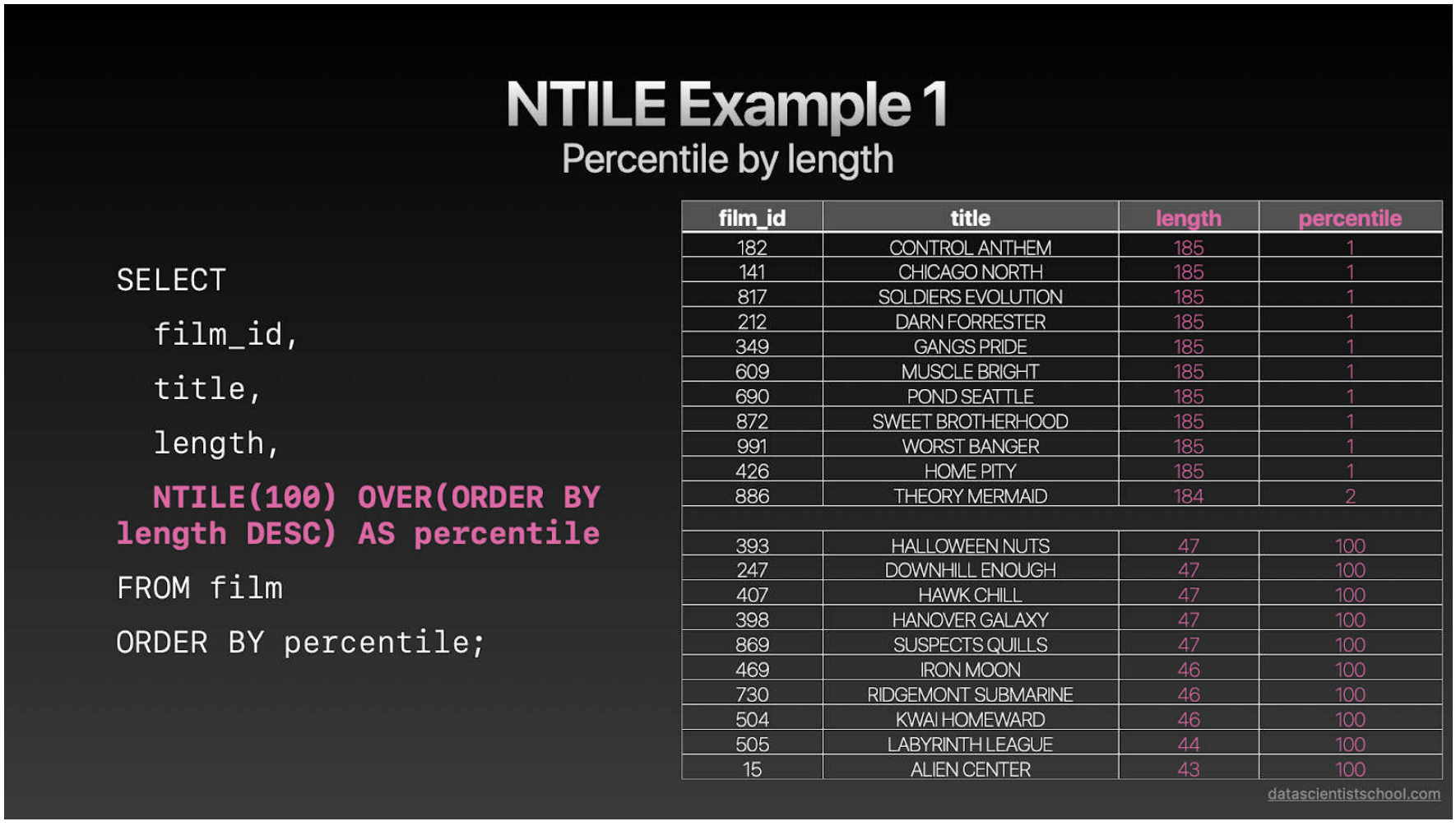
- #62: Shortest film by category
- #63: Top 5 customers by store
- #64: Top 2 films by category

Section 3: NTILE

In this section, I am going to show you how to create statistics using **NTILE**.

NTILE is a handy function, especially for data analytics professionals. For example, as a data scientist, you probably need to create robust statistics such as quartile, quintile, median, etc., in your daily job, and **NTILE** makes it very easy to generate those numbers.

NTILE takes an argument of the number of buckets and then creates this number of buckets as equally as possible, based on how the rows are partitioned and ordered inside the **OVER** function.




```sql
SELECT
  film_id,
  title,
  length,
  NTILE(100) OVER(ORDER BY length DESC) AS percentile
FROM film
ORDER BY percentile;
```

Let's take a look at example 1, where we created 100 buckets, and we ordered all of the movies by their length descendingly. Therefore, the longest ones are assigned to bucket #1, and the shortest ones #100.

NTILE Example 2

Percentile, decile, quartile by MPAA rating

```
SELECT
  title,
  replacement_cost AS cost,
  rating,
  NTILE(100) OVER(PARTITION BY rating
ORDER BY replacement_cost) AS
percentile,
  NTILE(10) OVER(PARTITION BY rating
ORDER BY replacement_cost) AS decile,
  NTILE(4) OVER(PARTITION BY rating
ORDER BY replacement_cost) AS quartile
FROM film
ORDER BY rating, replacement_cost;
```

title	cost	rating	percentile	decile	quartile
YOUNG LANGUAGE	$9.99	G	1	1	1
TRUMAN CRAZY	$9.99	G	1	1	1
HEARTBREAKERS BRIGHT	$9.99	G	2	1	1
CONTROL ANTHEM	$9.99	G	2	1	1
DAISY MENAGERIE	$9.99	G	3	1	1
DUDE BLINDNESS	$9.99	G	3	1	1
ENCINO ELF	$9.99	G	4	1	1
INTRIGUE WORST	$10.99	G	4	1	1
...					
LAWLESS VISION	$29.99	G	97	10	4
SASSY PACKER	$29.99	G	98	10	4
SMILE EARRING	$29.99	G	99	10	4
CRUELTY UNFORGIVEN	$29.99	G	100	10	4
PATHS CONTROL	$9.99	PG	1	1	1
...					
DIRTY ACE	$29.99	NC-17	99	10	4
RANDOM GO	$29.99	NC-17	99	10	4
ARABIA DOGMA	$29.99	NC-17	100	10	4
PATIENT SISTER	$29.99	NC-17	100	10	4

datascientistschool.com

For the second example, we created a few more statistics, such as **DECILES** (10 buckets) and **QUARTILES** (4 buckets). We also partitioned them by MPAA ratings, so the statistics are relative to each unique MPAA rating.

NTILE is a very straightforward window function that can be very useful for your daily job as a data scientist. Let's do some exercises to help you remember its syntax and reinforce your learning in this lecture.

Some exercises for fun.

Time to complete: ~30 -45 mins.

- #65: Movie percentiles by revenue
- #66: Movie percentiles by revenue by category
- #67: Quartile buckets by number of rentals

Section 4: LAG, Lead

In the last section, I will walk you through two WINDOW functions: **LAG** and **LEAD**, which are extremely useful for dealing with time-related data.

LAG and LEAD's main difference is that **LAG** gets data from *previous rows,* while **LEAD** is the opposite, which fetches data from the following *rows.*

We can use either one of the two functions to compare month-over-month growth. As a data analytics professional, you are very likely to work on time-related data. If you can use **LAG** or **LEAD** efficiently, you will be a very productive data scientist.

Their syntax is very similar to other window functions. Instead of focusing on the format of the syntax, let me show you a couple of examples.

LAG Example

Previous day's revenue

```
WITH daily_revenue AS (
  SELECT
    DATE(payment_ts) date,
    SUM(amount) revenue
  FROM payment
  WHERE DATE(payment_ts) >= '2020-05-24'
  AND   DATE(payment_ts) <= '2020-05-31'
  GROUP BY DATE(payment_ts)
)
SELECT
  date,
  revenue,
  LAG(revenue, 1) OVER (ORDER BY date) prev_day_sales,
  revenue *1.0/LAG(revenue,1) OVER (ORDER BY date) AS dod
FROM
  daily_revenue
ORDER BY date;
```

date	revenue	prev_day_sales	dod
2020-05-24	$29.92	NULL	NULL
2020-05-25	$573.63	$29.92	1,917%
2020-05-26	$754.26	$573.63	131%
2020-05-27	$684.34	$754.26	91%
2020-05-28	$804.04	$684.34	117%
2020-05-29	$648.46	$804.04	81%
2020-05-30	$628.42	$648.46	97%
2020-05-31	$700.37	$628.42	111%

datascientistschool.com

```sql
WITH daily_revenue AS (
  SELECT
    DATE(payment_ts) date,
    SUM(amount) revenue
  FROM payment
  WHERE DATE(payment_ts) >= '2020-05-24'
  AND DATE(payment_ts) <= '2020-05-31'
  GROUP BY DATE(payment_ts)
)
SELECT
  date,
  revenue,
  LAG(revenue, 1) OVER (ORDER BY date) prev_day_sales,
  revenue *1.0/LAG(revenue,1) OVER (ORDER BY date) AS dod
FROM daily_revenue
ORDER BY date;
```

1. In the first step, we created daily movie rental revenue with **CTE** (common table expression).
2. And in the second step, we appended the previous day's revenue to the current day's using the LAG function.
3. Notice that last 2 columns of the first row are empty. It's simply because May 24th is the first available day.
4. We also specified the offset, which is 1, so we fetch the next row. If you change this number to 2, then you compare the current day's revenue to the day before the previous day.
5. Finally, we divided the current day's revenue by the previous day's to create our daily revenue growth.

LEAD Example

Next day's revenue

```
WITH daily_revenue AS (
  SELECT
    DATE(payment_ts) date,
    SUM(amount) revenue
  FROM payment
  WHERE DATE(payment_ts) >= '2020-05-24'
  AND   DATE(payment_ts) <= '2020-05-31'
  GROUP BY DATE(payment_ts)
)
SELECT
  date,
  revenue,
  LEAD(revenue, 1) OVER (ORDER BY date) next_day_sales,
  LEAD(revenue,1) OVER (ORDER BY date) *1.0/revenue AS dod
FROM
  daily_revenue
ORDER BY date;
```

date	revenue	next_day_sales	dod
2020-05-24	$30	$574	1,917%
2020-05-25	$574	$754	131%
2020-05-26	$754	$684	91%
2020-05-27	$684	$804	117%
2020-05-28	$804	$648	81%
2020-05-29	$648	$628	97%
2020-05-30	$628	$700	111%
2020-05-31	$700	NULL	NULL

datascientistschool.com

```sql
WITH daily_revenue AS (
  SELECT
    DATE(payment_ts) date,
    SUM(amount) revenue
  FROM payment
  WHERE DATE(payment_ts) >= '2020-05-24'
  AND DATE(payment_ts) <= '2020-05-31'
  GROUP BY DATE(payment_ts)
)
SELECT
  date,
  revenue,
  LAG(revenue, 2) OVER (ORDER BY date) prev_day_sales,
  revenue *1.0/LAG(revenue,2) OVER (ORDER BY date) AS dod
FROM daily_revenue
ORDER BY date;
```

Let's take a look at another example. It's very similar to the previous one, but instead of appending the previous day's revenue, we used the **LEAD** function with an offset of 1 to get the next day's movie rental revenue.

We then divided the next day's revenue by the current day's revenue to get the day-over-day growth.

LEAD Example

Next day's revenue

```
WITH daily_revenue AS (
  SELECT
    DATE(payment_ts) date,
    SUM(amount) revenue
  FROM payment
  WHERE DATE(payment_ts) >= '2020-05-24'
  AND   DATE(payment_ts) <= '2020-05-31'
  GROUP BY DATE(payment_ts)
)
SELECT
  date,
  revenue,
  LEAD(revenue, 1) OVER (ORDER BY date) next_day_sales,
  LEAD(revenue,1) OVER (ORDER BY date) *1.0/revenue AS dod
FROM
  daily_revenue
ORDER BY date;
```

date	revenue	next_day_sales	dod
2020-05-24	$30	$574	1,917%
2020-05-25	$574	$754	131%
2020-05-26	$754	$684	91%
2020-05-27	$684	$804	117%
2020-05-28	$804	$648	81%
2020-05-29	$648	$628	97%
2020-05-30	$628	$700	111%
2020-05-31	$700	NULL	NULL

datascientistschool.com

For this lecture, you can try the following 2 exercises to help you get familiar with the syntax.

Time to complete: ~45 mins — 1 hour.

- #68: Spend difference of first and second rentals
- #69: Number of happy customers

Summary

Great job. If you have followed through all the examples, you have seen most of the common **WINDOW functions/patterns**.

WINDOW functions are a family of SQL utilities that are often asked during a data scientist job interview.

Writing a bug-free WINDOW function query could be quite challenging. It takes time and practice to become a master, and you are getting there soon, once you finish this book. 🏆

QUESTIONS

SINGLE TABLE OPERATIONS

Question 1. Top store for movie sales easy

Instruction

Write a query to return the name of the store and its manager, that generated the most sales.

Table: sales_by_store

Movie sales by store

```
  col_name   | col_type
-------------+----------
 store       | text
 manager     | text
 total_sales | numeric
```

Sample results

```
   store    |   manager
------------+-------------
 Woodridge | Jon Stephens
```

Question 2. Top 3 movie categories by sales easy

Instruction

- Write a query to find the top 3 film categories that generated the most sales.
- The order of your results doesn't matter.

Table: sales_by_film_category

Total sales by movie categories.

```
  col_name   | col_type
-------------+----------
 category    | text
 total_sales | numeric
```

Sample results

```
category
-----------
Category 1
Category 2
Category 3
```

Question 3. Top 5 shortest movies easy

Instruction

- Write a query to return the titles of the 5 shortest movies by duration.
- The order of your results doesn't matter.

Table: film

col_name	col_type
film_id	integer
title	text
description	text
release_year	integer
language_id	smallint
original_language_id	smallint
rental_duration	smallint
rental_rate	numeric
length	smallint
replacement_cost	numeric
rating	text

Sample results

title
MOVIE 1
MOVIE 2
MOVIE 3
MOVIE 4
MOVIE 5

Question 4. Staff without a profile image easy

Instruction

- Write a SQL query to return this staff's first name and last name.
- Picture field contains the link that points to a staff's profile image.
- There is only one staff who doesn't have a profile picture.
- Use `colname IS NULL` to identify data that are missing.

Table: staff

col_name	col_type
staff_id	integer
first_name	text
last_name	text
address_id	smallint
email	text
store_id	smallint
active	boolean
username	text
picture	character varying

Sample results

```
 first_name | last_name
------------+-----------
 Jon        | Snow
```

Question 5. Monthly revenue easy

Instruction

- Write a query to return the total movie rental revenue for each month.
- You can use `EXTRACT(MONTH FROM colname)` and `EXTRACT(YEAR FROM colname)` to extract month and year from a timestamp column.

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+--------------------------
 payment_id    | integer
 customer_id   | smallint
 staff_id      | smallint
 rental_id     | integer
 amount        | numeric
 payment_ts    | timestamp with time zone
```

Sample results

```
 year | mon |   rev
------+-----+---------
 2020 |   1 |  123.45
 2020 |   2 |  234.56
 2020 |   3 |  345.67
```

Question 6. Daily revenue in June, 2020 easy

Instruction

- Write a query to return daily revenue in June, 2020.
- Use `DATE(colname)` to extract the date from a timestamp column.
- `payment_ts`'s data type is timestamp, convert it to date before grouping.
- No dates need to be included if there was no revenue on that day.

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+--------------------------
 payment_id    | integer
 customer_id   | smallint
 staff_id      | smallint
 rental_id     | integer
```

```
 amount         | numeric
 payment_ts     | timestamp with time zone
```

Sample results

```
     dt      |   sum
-------------+---------
 2020-06-14 |   57.84
 2020-06-15 | 1376.52
```

Question 7. Unique customers count by month easy

Instruction

- Write a query to return the total number of unique customers for each month
- Use `EXTRACT(YEAR from ts_field)` and `EXTRACT(MONTH from ts_field)` to get year and month from a timestamp column.
- The order of your results doesn't matter.

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Sample results

```
 year | mon | uu_cnt
------+-----+--------
 2020 |   1 |    123
 2020 |   2 |    456
 2020 |   3 |    789
```

Question 8. Average customer spend by month medium

Instruction

- Write a query to return the average customer spend by month.
- Definition: average customer spend: total customer spend divided by the unique number of customers for that month.
- Use `EXTRACT(YEAR from ts_field)` and `EXTRACT(MONTH from ts_field)` to get year and month from a timestamp column.
- The order of your results doesn't matter.

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+--------------------------
 payment_id    | integer
 customer_id   | smallint
 staff_id      | smallint
 rental_id     | integer
 amount        | numeric
 payment_ts    | timestamp with time zone
```

Sample results

```
yearmon avg_spend
20202   3.2543037974683544
20205   9.1301559454191033
```

Question 9. Number of high spend customers by month **medium**

Instruction

- Write a query to count the number of customers who spend more than (>) $20 by month
- Use `EXTRACT(YEAR from ts_field)` and `EXTRACT(MONTH from ts_field)` to get year and month from a timestamp column.
- The order of your results doesn't matter.
- **Hint:** a customer's spend varies every month.

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+--------------------------
 payment_id    | integer
 customer_id   | smallint
 staff_id      | smallint
 rental_id     | integer
 amount        | numeric
 payment_ts    | timestamp with time zone
```

Sample results

```
yearmon num_hp_customers
20202   158
20205   520
```

Question 10. Min and max spend **medium**

Instruction

- Write a query to return the minimum and maximum customer total spend in June 2020.
- For each customer, first calculate their total spend in June 2020.
- Then use `MIN`, and `MAX` function to return the min and max customer spend .

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+--------------------------
 payment_id    | integer
 customer_id   | smallint
 staff_id      | smallint
 rental_id     | integer
 amount        | numeric
 payment_ts    | timestamp with time zone
```

Sample results

```
min_spend | max_spend
----------+----------
      0.99 |     52.90
```

Question 11. Actors' last name easy

Instruction

Find the number of actors whose last name is one of the following: `'DAVIS', 'BRODY', 'ALLEN', 'BERRY'`

Table: actor

```
  col_name    | col_type
--------------+--------------------------
 actor_id     | integer
 first_name   | text
 last_name    | text
```

Sample results

```
last_name | count
----------+-------
 ALLEN    |     3
 DAVIS    |     3
```

Question 12. Actors' last name ending in 'EN' or 'RY' easy

Instruction

- Identify all actors whose last name ends in `'EN'` or `'RY'`.
- Group and count them by their last name.

Table: actor

```
  col_name    | col_type
--------------+--------------------------
 actor_id     | integer
```

```
 first_name  | text
 last_name   | text
```

Sample results

```
last_name | count
----------+-------
 ALLEN    |     3
 BERGEN   |     1
```

Question 13. Actors' first name medium

Instruction

- Write a query to return the number of actors whose first name starts with 'A', 'B', 'C', or others.
- The order of your results doesn't matter.
- You need to return 2 columns:
- The first column is the group of actors based on the first letter of their `first_name`, use the following: `'a_actors', 'b_actors', 'c_actors', 'other_actors'` to represent their groups.
- Second column is the number of actors whose first name matches the pattern.

Table: actor

```
  col_name   | col_type
-------------+--------------------------
 actor_id    | integer
 first_name  | text
 last_name   | text
```

Sample results

```
actor_category | count
---------------+-------
 a_actors      |    13
 b_actors      |     8
```

Question 14. Good days and bad days difficult

Instruction

- Write a query to return the number of good days and bad days in May 2020 based on number of daily rentals.
- Return the results in one row with 2 columns from left to right: good_days, bad_days.
- `good day`: `> 100` rentals.
- `bad day`: `<= 100` rentals.
- **Hint** (For users already know `OUTER JOIN`), you can use `dates` table
- **Hint**: be super careful about datetime columns.

- **Hint**: this problem could be tricky, feel free to explore the `rental` table and take a look at some data.

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Sample results

```
good_days | bad_days
----------+----------
        7 |       24
```

Question 15. Fast movie watchers vs slow watchers **difficult**

Instruction

- Write a query to return the number of fast movie watchers vs slow movie watchers.
- `fast movie watcher`: by average return their rentals within 5 days.
- `slow movie watcher`: takes an average of `>5` days to return their rentals.
- Most customers have multiple rentals over time, you need to first compute the number of days for each rental transaction, then compute the average on the rounded up days. e.g., if the rental period is 1 day and 10 hours, count it as 2 days.
- Skip the rentals that have not been returned yet, e.g., `rental_ts IS NULL`.
- The orders of your results doesn't matter.
- A customer can only rent one movie per transaction.

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Sample results

```
watcher_category | count
-----------------+-------
 fast_watcher    |   112
 slow_watcher    |   487
```

Question 16. Staff who live in Woodridge easy

Instruction

- Write a query to return the names of the staff who live in the city of 'Woodridge'

Table: staff_list

```
 col_name | col_type
----------+----------
 id       | integer
 name     | text
 address  | text
 zip code | text
 phone    | text
 city     | text
 country  | text
 sid      | smallint
```

Sample results

```
     name
--------------
 Jon Stephens
```

Question 17. GROUCHO WILLIAMS' actor_id easy

Instruction

- Write a query to return GROUCHO WILLIAMS' `actor_id`.
- Actor's first_name and last_name are all stored as UPPER case in our database, and the database is case sensitive.

Table: actor

```
  col_name   | col_type
-------------+--------------------------
 actor_id    | integer
 first_name  | text
 last_name   | text
```

Sample results

```
 actor_id
----------
      1
```

Question 18. Top film category easy

Instruction

- Write a query to return the film category id with the most films, as well as the number films in that category.

Table: film_category

A film can only belong to one category

```
  col_name   | col_type
-------------+-------------------------
 film_id     | smallint
 category_id | smallint
```

Sample results

```
category_id | film_cnt
------------+----------
          1 |        2
```

Question 19. Most productive actor medium

Instruction

- Write a query to return the first name and the last name of the actor who appeared in the most films.

Table: actor

```
  col_name   | col_type
-------------+-------------------------
 actor_id    | integer
 first_name  | text
 last_name   | text
```

Table: film_actor

Films and their casts

```
  col_name   | col_type
-------------+-------------------------
 actor_id    | smallint
 film_id     | smallint
```

Sample results

```
 first_name | last_name
------------+-----------
 MICHAEL    | JACKSON
```

Question 20. Customer who spent the most medium

Instruction

- Write a query to return the first and last name of the customer who spent the most on movie rentals in Feb 2020.

Table: payment

Movie rental payment transactions table

```
   col_name   | col_type
--------------+--------------------------
 payment_id   | integer
 customer_id  | smallint
 staff_id     | smallint
 rental_id    | integer
 amount       | numeric
 payment_ts   | timestamp with time zone
```

Table: customer

```
  col_name   | col_type
-------------+--------------------------
 customer_id | integer
 store_id    | smallint
 first_name  | text
 last_name   | text
 email       | text
 address_id  | smallint
 activebool  | boolean
 create_date | date
 active      | integer
```

Sample results

```
first_name | last_name
-----------+-----------
 JAMES     | BOND
```

Question 21. Customer who rented the most **medium**

Instruction

- Write a query to return the first and last name of the customer who made the most rental transactions in May 2020.

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Table: customer

```
  col_name   | col_type
-------------+--------------------------
 customer_id | integer
 store_id    | smallint
 first_name  | text
 last_name   | text
 email       | text
 address_id  | smallint
 activebool  | boolean
 create_date | date
 active      | integer
```

Sample results

```
 first_name | last_name
------------+-----------
 JENNIFER   | ANISTON
```

Question 22. Average cost per rental transaction easy

Instruction

- Write a query to return the average cost on movie rentals in May 2020 per transaction.

Table: payment

Movie rental payment transactions table

```
   col_name   | col_type
--------------+--------------------------
 payment_id   | integer
 customer_id  | smallint
 staff_id     | smallint
 rental_id    | integer
 amount       | numeric
 payment_ts   | timestamp with time zone
```

Sample results

```
        avg
--------------------
 1.234567
```

Question 23. Average spend per customer in Feb 2020 easy

Instruction

- Write a query to return the average movie rental spend per customer in Feb 2020.

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+--------------------------
 payment_id    | integer
 customer_id   | smallint
 staff_id      | smallint
 rental_id     | integer
 amount        | numeric
 payment_ts    | timestamp with time zone
```

Sample results

```
        avg
--------------------
1.23456789
```

Question 24. Films with more than 10 actors **medium**

Instruction

- Write a query to return the titles of the films with `>= 10` actors.

Table: film_actor

Films and their casts

```
  col_name    | col_type
--------------+--------------------------
 actor_id     | smallint
 film_id      | smallint
```

Table: film

```
      col_name       |  col_type
---------------------+--------------------------
 film_id             | integer
 title               | text
 description         | text
 release_year        | integer
 language_id         | smallint
 original_language_id | smallint
 rental_duration     | smallint
 rental_rate         | numeric
 length              | smallint
 replacement_cost    | numeric
 rating              | text
```

Sample results

```
         title
----------------------
 ACADEMY DINOSAUR
 ARABIA DOGMA
```

Question 25. Shortest film easy

Instruction

- Write a query to return the title of the film with the minimum duration.
- A movie's duration can be found using the **length** column.
- If there are ties, e.g., two movies have the same **length**, return either one of them.

Table: film

col_name	col_type
film_id	integer
title	text
description	text
release_year	integer
language_id	smallint
original_language_id	smallint
rental_duration	smallint
rental_rate	numeric
length	smallint
replacement_cost	numeric
rating	text

Sample results

title
SHORT FILM

Question 26. Second shortest film medium

Instruction

- Write a query to return the title of the second shortest film based on its duration/length.
- A movie's duration can be found using the **length** column.
- If there are ties, return just one of them.

Table: film

col_name	col_type
film_id	integer
title	text
description	text
release_year	integer
language_id	smallint
original_language_id	smallint
rental_duration	smallint
rental_rate	numeric

```
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Sample results

```
    title
--------------
 SECOND SHORTEST
```

Question 27. Film with the largest cast easy

Instruction

- Write a query to return the title of the film with the largest cast (most actors).
- If there are ties, return just one of them.

Table: film_actor

Films and their casts

```
  col_name    | col_type
--------------+--------------------------
 actor_id     | smallint
 film_id      | smallint
```

Table: film

```
       col_name        |  col_type
-----------------------+---------------------------
 film_id               | integer
 title                 | text
 description           | text
 release_year          | integer
 language_id           | smallint
 original_language_id  | smallint
 rental_duration       | smallint
 rental_rate           | numeric
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Sample results

```
      title
------------------
 LARGEST MOVIE
```

Question 28. Film with the second largest cast medium

Instruction

- Write a query to return the title of the film with the second largest cast.
- If there are ties, e.g., two movies have the same number of actors, return either one of the movie.

Table: film_actor

Films and their casts

```
  col_name   | col_type
-------------+-------------------------
 actor_id    | smallint
 film_id     | smallint
```

Table: film

```
       col_name        |  col_type
-----------------------+-------------------------
 film_id               | integer
 title                 | text
 description           | text
 release_year          | integer
 language_id           | smallint
 original_language_id  | smallint
 rental_duration       | smallint
 rental_rate           | numeric
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Sample results

```
  title
-----------
 SECOND LARGEST
```

Question 29. Second highest spend customer **medium**

Instruction

- Write a query to return the name of the customer who spent the second highest for movie rentals in May 2020.
- If there are ties, return any one of them.

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+-------------------------
 payment_id    | integer
 customer_id   | smallint
 staff_id      | smallint
 rental_id     | integer
 amount        | numeric
```

```
 payment_ts   | timestamp with time zone
```

Table: customer

```
  col_name   | col_type
-------------+--------------------------
 customer_id | integer
 store_id    | smallint
 first_name  | text
 last_name   | text
 email       | text
 address_id  | smallint
 activebool  | boolean
 create_date | date
 active      | integer
```

Sample results

```
 first_name | last_name
------------+-----------
 MARK       | ZUCKERBERG
```

Question 30. Inactive customers in May easy

Instruction

- Write a query to return the total number of customers who didn't rent any movies in May 2020.

Hint

- You can use `NOT IN` to exclude customers who have rented movies in May 2020.

Table: rental

```
   col_name   | col_type
--------------+--------------------------
 rental_id    | integer
 rental_ts    | timestamp with time zone
 inventory_id | integer
 customer_id  | smallint
 return_ts    | timestamp with time zone
 staff_id     | smallint
```

Table: customer

```
  col_name   | col_type
-------------+--------------------------
 customer_id | integer
 store_id    | smallint
 first_name  | text
 last_name   | text
```

```
 email       | text
 address_id  | smallint
 activebool  | boolean
 create_date | date
 active      | integer
```

Sample results

```
 count
-------
 1234
```

Question 31. Movies that have not been returned easy

Instruction

- Write a query to return the titles of the films that have not been returned by our customers.

Hint

- If a movie is not returned, the `return_ts` will be `NULL` in the rental table.

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+-------------------------
 inventory_id | integer
 film_id      | smallint
 store_id     | smallint
```

Table: film

```
       col_name        |  col_type
-----------------------+-------------------------
 film_id               | integer
 title                 | text
 description           | text
 release_year          | integer
 language_id           | smallint
 original_language_id | smallint
 rental_duration       | smallint
 rental_rate           | numeric
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Table: rental

```
   col_name    | col_type
---------------+-------------------------
 rental_id     | integer
```

```
 rental_ts    | timestamp with time zone
 inventory_id | integer
 customer_id  | smallint
 return_ts    | timestamp with time zone
 staff_id     | smallint
```

Sample results

```
          title
------------------------
 AGENT TRUMAN
 ALABAMA DEVIL
 AMERICAN CIRCUS
 ANGELS LIFE
```

Question 32. Unpopular movies difficult

Instruction

- Write a query to return the number of films with no rentals in Feb 2020.
- Count the entire movie catalog from the `film` table.

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+-------------------------
 inventory_id | integer
 film_id      | smallint
 store_id     | smallint
```

Table: film

```
       col_name        |  col_type
-----------------------+-------------------------
 film_id               | integer
 title                 | text
 description           | text
 release_year          | integer
 language_id           | smallint
 original_language_id | smallint
 rental_duration       | smallint
 rental_rate           | numeric
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Table: rental

```
   col_name    | col_type
---------------+-------------------------
```

```
 rental_id    | integer
 rental_ts    | timestamp with time zone
 inventory_id | integer
 customer_id  | smallint
 return_ts    | timestamp with time zone
 staff_id     | smallint
```

Sample results

```
 count
-------
   123
```

Question 33. Returning customers medium

Instruction

- Write a query to return the number of customers who rented at least one movie in both May 2020 and June 2020.

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Sample results

```
 count
-------
   123
```

Question 34. Stocked up movies easy

Instruction

- Write a query to return the titles of movies with more than `>7` dvd copies in the inventory.

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+--------------------------
 inventory_id  | integer
 film_id       | smallint
 store_id      | smallint
```

Table: film

col_name	col_type
film_id	integer
title	text
description	text
release_year	integer
language_id	smallint
original_language_id	smallint
rental_duration	smallint
rental_rate	numeric
length	smallint
replacement_cost	numeric
rating	text

Sample results

title
ACADEMY DINOSAUR
APACHE DIVINE

Question 35. Film length report easy

Instruction

- Write a query to return the number of films in the following categories: short, medium, and long.
- The order of your results doesn't matter.

Definition

- short: less `<60` minutes.
- medium: `>=60` minutes, but `<100` minutes.
- long: `>=100` minutes

Table: film

col_name	col_type
film_id	integer
title	text
description	text
release_year	integer
language_id	smallint
original_language_id	smallint
rental_duration	smallint
rental_rate	numeric
length	smallint
replacement_cost	numeric
rating	text

Sample results

```
 film_category | count
---------------+-------
 medium        |   1
 long          |   2
 short         |   3
```

Question 81. How many people searched on new year's day easy

Write a query to return the total number of users who have searched on new year's day: 2021-01-01.

Table: search

```
  col_name| col_type
----------+------------
country   | varchar(2)
date      | date
user_id   | integer
search_id | integer
query     | text
```

Sample results

```
num_searches
--------------------
1234567899
```

Question 82. The top search query on new year's day easy

Write a query to return the top search term on new year's day: 2021-01-01

Table: search

```
  col_name| col_type
----------+------------
country   | varchar(2)
date      | date
user_id   | integer
search_id | integer
query     | text
```

Sample results

```
top_search_term
-------------------
Joe Biden
```

Question 84. Click through rate on new year's day easy

- Write a query to compute the click through rate for the search results on new year's day (2021-01-01).
- `Click through rate:` number of searches end up with at least one click.
- Convert your result into a percentage (* 100.0).

Table: search_result

```
  col_name   | col_type
-------------+-------------------------
 date        | date
 search_id   | bigint
 result_id   | bigint
 result_type | varchar(20)
 action      | varchar(20)
```

Sample results

```
 ctr
----------
2.34
```

Question 85. Top 5 queries based on click through rate on new year's day difficult

Instruction

- Write a query to return the top 5 search terms with the highest click through rate on new year's day (2021-01-01)
- The search term has to be searched by more than 2 `(>2)` distinct users.
- `Click through rate:` number of searches end up with at least one click.

Table: search

```
  col_name| col_type
----------+------------
country   | varchar(2)
date      | date
user_id   | integer
search_id | integer
query     | text
```

Table: search_result

col_name	col_type
date	date
search_id	bigint
result_id	bigint
result_type	varchar(20)
action	varchar(20)

Sample results

query
covid
exit
biden

Question 86. Top song in the US easy

Write a query to return the name of the top song in the US yesterday.

Table: song

col_name	col_type
song_id	bigint
title	varchar(1000)
artist_id	bigint

Table: song_plays

Number of times a song is played (streamed), aggregated on daily basis.

col_name	col_type
date	date
country	varchar(2)
song_id	bigint
num_plays	bigint

Sample results

title
Eminence Front

Question 89. Top 5 artists in the US **medium**

Instruction

- Write a query to return the top 5 artists id for US yesterday.
- Return a unique row for each country
- For simplicity, let's assume there is no tie.
- The order of your results doesn't matter.

Table: artist

```
  col_name    | col_type
--------------+-------------------------
 artist_id     | bigint
 name      | VARCHAR(255)
```

MULTI-TABLE OPERATIONS

Question 36. Actors from film 'AFRICAN EGG' easy

Instruction

- Write a query to return the first name and last name of all actors in the film 'AFRICAN EGG'.
- The order of your results doesn't matter.

Table: actor

col_name	col_type
actor_id	integer
first_name	text
last_name	text

Table: film_actor

Films and their casts

col_name	col_type
actor_id	smallint
film_id	smallint

Table: film

col_name	col_type
film_id	integer
title	text
description	text
release_year	integer
language_id	smallint
original_language_id	smallint
rental_duration	smallint
rental_rate	numeric
length	smallint
replacement_cost	numeric
rating	text

Sample results

first_name	last_name
GARY	PHOENIX
DUSTIN	TAUTOU

Question 37. Most popular movie category easy

Instruction

- Return the name of the category that has the most films.
- If there are ties, return just one of them.

Table: film_category

A film can only belong to one category

```
  col_name   | col_type
-------------+--------------------------
 film_id     | smallint
 category_id | smallint
```

Table: category

Movie categories.

```
  col_name   | col_type
-------------+--------------------------
 category_id | integer
 name        | text
```

Sample results

```
  name
--------
Category Name
```

Question 38. Most popular movie category (name and id) medium

Instruction

- Write a query to return the name of the most popular film category and its category id
- If there are ties, return just one of them.

Table: film_category

A film can only belong to one category

```
  col_name   | col_type
-------------+--------------------------
 film_id     | smallint
 category_id | smallint
```

Table: category

Movie categories.

```
  col_name   | col_type
-------------+--------------------------
 category_id | integer
 name        | text
```

Sample results

```
 category_id |  name
```

```
-------------+--------
           123 | Category
```

Question 39. Most productive actor with inner join **easy**

Instruction

- Write a query to return the name of the actor who appears in the most films.
- You have to use INNER JOIN in your query.

Table: actor

```
  col_name    | col_type
--------------+--------------------------
 actor_id     | integer
 first_name   | text
 last_name    | text
```

Table: film_actor

Films and their casts

```
  col_name    | col_type
--------------+--------------------------
 actor_id     | smallint
 film_id      | smallint
```

Sample results

```
 actor_id | first_name | last_name
----------+------------+-----------
     1234 | FIRST_NAME | LAST_NAME
```

Question 40. Top 5 most rented movie in June 2020 **medium**

Instruction

- Write a query to return the `film_id` and `title` of the top 5 movies that were rented the most times in June 2020
- Use the `rental_ts` column from the `rental` for the transaction time.
- The order of your results doesn't matter.
- If there are ties, return any 5 of them.

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+--------------------------
 inventory_id  | integer
 film_id       | smallint
 store_id      | smallint
```

Table: film

col_name	col_type
film_id	integer
title	text
description	text
release_year	integer
language_id	smallint
original_language_id	smallint
rental_duration	smallint
rental_rate	numeric
length	smallint
replacement_cost	numeric
rating	text

Table: rental

col_name	col_type
rental_id	integer
rental_ts	timestamp with time zone
inventory_id	integer
customer_id	smallint
return_ts	timestamp with time zone
staff_id	smallint

Sample results

film_id	title
12345	MOVIE TITLE 1
12346	MOVIE TITLE 2
12347	MOVIE TITLE 3
12348	MOVIE TITLE 4
12349	MOVIE TITLE 5

Question 41. Productive actors vs less-productive actors **medium**

Instruction

- Write a query to return the number of `productive` and `less-productive` actors.
- The order of your results doesn't matter.

Definition

- `productive`: appeared in `>= 30` films.
- `less-productive`: appeared in `<30` films.

Table: actor

col_name	col_type

```
-------------+--------------------------
 actor_id    | integer
 first_name  | text
 last_name   | text
```

Table: film_actor

Films and their casts

```
  col_name   | col_type
-------------+--------------------------
 actor_id    | smallint
 film_id     | smallint
```

Sample results

```
 actor_category  | count
-----------------+-------
 less productive |   123
 productive      |    456
```

Question 42. Films that are in stock vs not in stock **medium**

Instruction

- Write a query to return the number of films that we have inventory vs no inventory.
- A film can have multiple inventory ids
- Each film dvd copy has a unique inventory ids

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name   | col_type
--------------+--------------------------
 inventory_id | integer
 film_id      | smallint
 store_id     | smallint
```

Table: film

```
       col_name        |  col_type
-----------------------+--------------------------
 film_id               | integer
 title                 | text
 description           | text
 release_year          | integer
 language_id           | smallint
 original_language_id  | smallint
 rental_duration       | smallint
 rental_rate           | numeric
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Sample results

```
   in_stock    | count
---------------+-------
 in stock      |   123
 not in stock  |    456
```

Question 43. Customers who rented vs. those who did not **medium**

Instruction

- Write a query to return the number of customers who rented at least one movie vs. those who didn't in **May 2020**.
- The order of your results doesn't matter.
- Use `customer` table as the base table for all customers (assuming all customers have signed up before May 2020)
- `Rented`: if a customer rented at least one movie.
- **Bonus:** Develop a `LEFT JOIN` as well as a `RIGHT JOIN` solution

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Table: customer

```
  col_name    | col_type
--------------+--------------------------
 customer_id  | integer
 store_id     | smallint
 first_name   | text
 last_name    | text
 email        | text
 address_id   | smallint
 activebool   | boolean
 create_date  | date
 active       | integer
```

Sample results

```
   hass_rented | count
---------------+-------
 rented        |   123
 never-rented  |   456
```

Question 44. In-demand vs not-in-demand movies **medium**

Instruction

- Write a query to return the number of `in demand` and `not in demand` movies in May 2020.
- Assumptions (great to clarify in your interview): all films are available for rent before May.
- But if a film is not in stock, it is not in demand.
- The order of your results doesn't matter.

Definition

- `in-demand`: rented `>1 times` in May 2020.
- `not-in-demand`: rented `<= 1 time` in May 2020.

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+--------------------------
 inventory_id | integer
 film_id      | smallint
 store_id     | smallint
```

Table: film

```
       col_name        |  col_type
-----------------------+---------------------------
 film_id               | integer
 title                 | text
 description           | text
 release_year          | integer
 language_id           | smallint
 original_language_id | smallint
 rental_duration       | smallint
 rental_rate           | numeric
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Sample results

```
 demand_category | count
-----------------+-------
 in-demand       |   123
 not-in-demand   |   456
```

Question 45. Movie inventory optimization **difficult**

Instruction

- For movies that are not in demand (rentals = 0 in May 2020), we want to remove them from our inventory.
- Write a query to return the number of unique `inventory_id` from those movies with **0 demand**.
- **Hint**: a movie can have multiple `inventory_id`.

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+-------------------------
 inventory_id | integer
 film_id      | smallint
 store_id     | smallint
```

Table: film

```
       col_name        |  col_type
-----------------------+---------------------------
 film_id               | integer
 title                 | text
 description           | text
 release_year          | integer
 language_id           | smallint
 original_language_id | smallint
 rental_duration       | smallint
 rental_rate           | numeric
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Table: rental

```
   col_name    | col_type
---------------+-------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id | integer
```

```
 customer_id  | smallint
 return_ts    | timestamp with time zone
 staff_id     | smallint
```

Sample results

```
count
-------
12345
```

Question 46. Actors and customers whose last name starts with 'A' easy

Instruction

- Write a query to return unique names (first_name, last_name) of our `customers` and `actors` whose last name starts with letter 'A'.

Table: actor

```
  col_name   | col_type
-------------+--------------------------
 actor_id    | integer
 first_name  | text
 last_name   | text
```

Table: customer

```
  col_name   | col_type
-------------+--------------------------
 customer_id | integer
 store_id    | smallint
 first_name  | text
 last_name   | text
 email       | text
 address_id  | smallint
 activebool  | boolean
 create_date | date
 active      | integer
```

Sample results

```
 first_name | last_name
------------+-----------
 KENT       | ARSENAULT
 JOSE       | ANDREW
```

Question 47. Actors and customers whose first names end in 'D'. easy

Instruction

- Write a query to return all actors and customers whose first names ends in 'D'.
- Return their ids (for actor: use `actor_id`, customer: `customer_id`), `first_name` and `last_name`.
- The order of your results doesn't matter.

Table: actor

```
  col_name    | col_type
--------------+--------------------------
 actor_id     | integer
 first_name   | text
 last_name    | text
```

Table: customer

```
  col_name    | col_type
--------------+--------------------------
 customer_id  | integer
 store_id     | smallint
 first_name   | text
 last_name    | text
 email        | text
 address_id   | smallint
 activebool   | boolean
 create_date  | date
 active       | integer
```

Sample results

```
customer_id   | first_name |  last_name
--------------+------------+--------------
          55  | DORIS      | REED
          65  | ROSE       | HOWARD
```

Question 48. Movie and TV actors easy

Instruction

- Write a query to return actors who appeared in both tv and movies
- The order of your results doesn't matter.
- You need to use `INNER JOIN`.

Table: actor_movie

Actors who appeared in a movie.

```
  col_name   | col_type
-------------+-------------------
 actor_id    | integer
 first_name  | character varying
 last_name   | character varying
```

Table: actor_tv

Actors who appeared in a TV show.

```
  col_name   | col_type
-------------+-------------------
 actor_id    | integer
 first_name  | character varying
 last_name   | character varying
```

Sample results

```
 actor_id | first_name  |  last_name
----------+-------------+-------------
        1 | PENELOPE    | GUINESS
        4 | JENNIFER    | DAVIS
```

Question 49. Top 3 money making movie categories **medium**

Instruction

- Write a query to return the `name` of the 3 movie categories that generated the most rental revenue
- And rental revenue from each of the category.
- The order of your results doesn't matter.
- If there are ties, return just one of them.

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+--------------------------
 inventory_id  | integer
 film_id       | smallint
 store_id      | smallint
```

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+--------------------------
 payment_id    | integer
 customer_id   | smallint
 staff_id      | smallint
 rental_id     | integer
 amount        | numeric
 payment_ts    | timestamp with time zone
```

Table: film_category

A film can only belong to one category

```
  col_name    | col_type
--------------+--------------------------
```

```
 film_id     | smallint
 category_id | smallint
```

Table: film

```
      col_name       |  col_type
---------------------+---------------------------
 film_id              | integer
 title                | text
 description          | text
 release_year         | integer
 language_id          | smallint
 original_language_id | smallint
 rental_duration      | smallint
 rental_rate          | numeric
 length               | smallint
 replacement_cost     | numeric
 rating               | text
```

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Table: category

Movie categories.

```
  col_name    | col_type
--------------+--------------------------
 category_id  | integer
 name         | text
```

Sample results

```
    name      |   revenue
--------------+---------
 Sports       | 123
 Sci-Fi       | 456
 Animation    | 789
```

Question 50. Top 5 cities for movie rentals easy

Instruction

- Write a query to return the names of the top 5 cities with the most rental revenues in 2020.
- Include each city's revenue in the second column.
- The order of your results doesn't matter.
- If there are ties, return any one of them.
- Yours results should have exactly 5 rows.

Table: payment

Movie rental payment transactions table

```
   col_name   | col_type
--------------+--------------------------
 payment_id   | integer
 customer_id  | smallint
 staff_id     | smallint
 rental_id    | integer
 amount       | numeric
 payment_ts   | timestamp with time zone
```

Table: address

```
  col_name   | col_type
-------------+--------------------------
 address_id  | integer
 address     | text
 address2    | text
 district    | text
 city_id     | smallint
 postal_code | text
 phone       | text
```

Table: city

```
  col_name   | col_type
-------------+--------------------------
 city_id     | integer
 city        | text
 country_id  | smallint
```

Table: customer

```
  col_name   | col_type
-------------+--------------------------
 customer_id | integer
 store_id    | smallint
 first_name  | text
 last_name   | text
 email       | text
 address_id  | smallint
 activebool  | boolean
```

```
 create_date | date
 active      | integer
```

Sample results

```
            city              |  sum
------------------------------+--------
 Cape Coral                   | 221.55
 Saint-Denis                  | 216.54
 Aurora                       | 198.50
 City 4                       | 12.34
 City 5                       | 1.23
```

Question 51. Movie only actor easy

Instruction

- Write a query to return the first name and last name of actors who only appeared in movies.
- Actor appeared in tv should not be included .
- The order of your results doesn't matter.

Table: actor_movie

Actors who appeared in a movie.

```
  col_name   | col_type
-------------+------------------
 actor_id    | integer
 first_name  | character varying
 last_name   | character varying
```

Table: actor_tv

Actors who appeared in a TV show.

```
  col_name   | col_type
-------------+------------------
 actor_id    | integer
 first_name  | character varying
 last_name   | character varying
```

Sample results

```
 first_name  |  last_name
-------------+-------------
 ED          | CHASE
 ZERO        | CAGE
 CUBA        | OLIVIER
```

Question 52. Movies cast by movie only actors difficult

Instruction

- Write a query to return the `film_id` with movie only casts (actors who never appeared in tv).
- The order of your results doesn't matter.
- You should exclude movies with one or more tv actors

Table: film_actor

Films and their casts

```
  col_name   | col_type
-------------+--------------------------
 actor_id    | smallint
 film_id     | smallint
```

Table: actor_tv

Actors who appeared in a TV show.

```
  col_name  | col_type
------------+-------------------
 actor_id   | integer
 first_name | character varying
 last_name  | character varying
```

Table: film

```
      col_name       |  col_type
---------------------+---------------------------
 film_id             | integer
 title               | text
 description         | text
 release_year        | integer
 language_id         | smallint
 original_language_id | smallint
 rental_duration     | smallint
 rental_rate         | numeric
 length              | smallint
 replacement_cost    | numeric
 rating              | text
```

Sample results

```
 film_id
---------
     174
     201
```

Question 53. Movie groups by rental income **difficult**

Instruction

- Write a query to return the number of films in 3 separate groups: high, medium, low.
- The order of your results doesn't matter.

Definition

- high: revenue `>= $100`.
- medium: revenue `>= $20, <$100` .
- low: revenue `<$20`.

Hint

- If a movie has no rental revenue, it belongs to the `low` group

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+--------------------------
 inventory_id | integer
 film_id      | smallint
 store_id     | smallint
```

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+--------------------------
 payment_id   | integer
 customer_id  | smallint
 staff_id     | smallint
 rental_id    | integer
 amount       | numeric
 payment_ts   | timestamp with time zone
```

Table: film

```
       col_name        |  col_type
-----------------------+---------------------------
 film_id               | integer
 title                 | text
 description           | text
 release_year          | integer
 language_id           | smallint
 original_language_id | smallint
 rental_duration       | smallint
 rental_rate           | numeric
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id    | integer
 rental_ts    | timestamp with time zone
 inventory_id | integer
```

```
 customer_id  | smallint
 return_ts    | timestamp with time zone
 staff_id     | smallint
```

Sample results

```
 film_group | count
------------+-------
 medium     |   123
 high       |   456
 low        |   789
```

Question 54. Customer groups by movie rental spend medium

Instruction

- Write a query to return the number of customers in 3 separate groups: high, medium, low.
- The order of your results doesn't matter.

Definition

- high: movie rental spend `>= $150`.
- medium: movie rental spend `>= $100, <$150`.
- low: movie rental spend `<$100`.

Hint

- If a customer spend 0 in movie rentals, he/she belongs to the `low` group.

Table: payment

Movie rental payment transactions table

```
   col_name   | col_type
--------------+--------------------------
 payment_id   | integer
 customer_id  | smallint
 staff_id     | smallint
 rental_id    | integer
 amount       | numeric
 payment_ts   | timestamp with time zone
```

Table: customer

```
  col_name   | col_type
-------------+-------------------------
 customer_id | integer
 store_id    | smallint
 first_name  | text
 last_name   | text
 email       | text
 address_id  | smallint
 activebool  | boolean
 create_date | date
 active      | integer
```

Sample results

```
customer_group | count
---------------+-------
 high          |   123
 medium        |   456
 low           |   789
```

Question 55. Busy days and slow days **medium**

Instruction

- Write a query to return the number of busy days and slow days in **May 2020** based on the number of movie rentals.
- The order of your results doesn't matter.
- If there are ties, return just one of them.

Definition

- busy: rentals `>= 100`.
- slow: rentals `< 100`.

Table: dates

Dates from 01/01/2019 to 12/31/2021.

```
 col_name | col_type
----------+----------
 year     | smallint
 month    | smallint
 date     | date
```

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Sample results

```
date_category  | count
---------------+-------
 busy          |    10
 slow          |    21
```

Question 56. Total number of actors **easy**

Instruction

- Write a query to return the total number of actors from actor_tv, actor_movie with `FULL OUTER JOIN`.
- Use `COALESCE` to return the first non-null value from a list.
- Actors who appear in both tv and movie share the same value of `actor_id` in both actor_tv and actor_movie tables.

Table: actor_movie

Actors who appeared in a movie.

```
  col_name   | col_type
-------------+------------------
 actor_id    | integer
 first_name  | character varying
 last_name   | character varying
```

Table: actor_tv

Actors who appeared in a TV show.

```
  col_name   | col_type
-------------+------------------
 actor_id    | integer
 first_name  | character varying
 last_name   | character varying
```

Sample results

```
 count
-------
   123
```

Question 57. Total number of actors (with UNION) easy

Instruction

- Write a query to return the total number of actors using `UNION`.
- Actor who appeared in both tv and movie has the same value of `actor_id` in both actor_tv and actor_movie tables.

Table: actor_movie

Actors who appeared in a movie.

```
  col_name   | col_type
-------------+------------------
 actor_id    | integer
 first_name  | character varying
 last_name   | character varying
```

Table: actor_tv

Actors who appeared in a TV show.

```
  col_name   | col_type
-------------+------------------
```

```
 actor_id   | integer
 first_name | character varying
 last_name  | character varying
```

Sample results

```
 count
-------
   123
```

Question 87. Top song in the US and UK **medium**

Write a query to return the name of the top song in the US and UK yesterday, respectively.
Table: song

```
  col_name   | col_type
-------------+--------------------------
   song_id   | bigint
   title     | varchar(1000)
   artist_id | bigint
```

Table: song_plays

Number of times a song is played (streamed), aggregated on daily basis.

```
  col_name   | col_type
-------------+--------------------------
 date        | date
 country     | varchar(2)
 song_id     | bigint
 num_plays   | bigint
```

Sample results

```
 country      | song_name
--------------+-----------------
           US |  Superhero
           UK |  Eminence Front
```

Question 58. Percentage of revenue per movie **medium**

Instruction

- Write a query to return the percentage of revenue for each of the following films: `film_id <= 10`.
- Formula: `revenue (film_id x) * 100.0/ revenue of all movies`.
- The order of your results doesn't matter.

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+--------------------------
 inventory_id | integer
 film_id      | smallint
 store_id     | smallint
```

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+--------------------------
 payment_id   | integer
 customer_id  | smallint
 staff_id     | smallint
 rental_id    | integer
 amount       | numeric
 payment_ts   | timestamp with time zone
```

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id    | integer
 rental_ts    | timestamp with time zone
 inventory_id | integer
 customer_id  | smallint
 return_ts    | timestamp with time zone
 staff_id     | smallint
```

Sample results

```
film_id  revenue_percentage
1    0.05454153589380405482
2    0.07851192534291674250
3    0.05618801685225177037
4    0.13612392572679896957
```

Question 59. Percentage of revenue per movie by category **medium**

Instruction

- Write a query to return the percentage of revenue for each of the following films: `film_id <= 10` by its category.
- Formula: `revenue (film_id x) * 100.0/ revenue of all movies in the same category`.
- The order of your results doesn't matter.
- Return 3 columns: film_id, category name, and percentage.

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+--------------------------
 inventory_id | integer
 film_id      | smallint
 store_id     | smallint
```

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+--------------------------
 payment_id    | integer
 customer_id   | smallint
 staff_id      | smallint
 rental_id     | integer
 amount        | numeric
 payment_ts    | timestamp with time zone
```

Table: film_category

A film can only belong to one category

```
  col_name    | col_type
--------------+--------------------------
 film_id      | smallint
 category_id | smallint
```

Table: film

```
       col_name        |  col_type
-----------------------+--------------------------
 film_id               | integer
 title                 | text
 description           | text
 release_year          | integer
 language_id           | smallint
 original_language_id | smallint
 rental_duration       | smallint
 rental_rate           | numeric
```

```
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Table: category

Movie categories.

```
  col_name    | col_type
--------------+--------------------------
 category_id  | integer
 name         | text
```

Sample results

```
 film_id | category_name | revenue_percent_category
---------+---------------+--------------------------
       1 | Documentary   |   0.8718393747984597583 4
       2 | Horror        |       1.42187860976644 98
       3 | Documentary   |   0.8981581592974070 0696
       4 | Horror        |       2.46525222025821 08
       5 | Family        |       1.22761809435243 62
```

Question 60. Movie rentals and average rentals in the same category **medium**

Instruction

- Write a query to return the number of rentals per movie, and the average number of rentals in its same category.
- You only need to return results for `film_id <= 10`.
- Return 4 columns: film_id, category name, number of rentals, and the average number of rentals from its category.

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+--------------------------
 inventory_id  | integer
 film_id       | smallint
 store_id      | smallint
```

Table: film_category

A film can only belong to one category

```
  col_name   | col_type
-------------+-------------------------
 film_id     | smallint
 category_id | smallint
```

Table: rental

```
   col_name   | col_type
--------------+--------------------------
 rental_id    | integer
 rental_ts    | timestamp with time zone
 inventory_id | integer
 customer_id  | smallint
 return_ts    | timestamp with time zone
 staff_id     | smallint
```

Table: category

Movie categories.

```
  col_name   | col_type
-------------+-------------------------
 category_id | integer
 name        | text
```

Sample results

```
 film_id | category_name | rentals | avg_rentals_category
---------+---------------+---------+----------------------
       1 | Documentary   |      23 |  16.6666666666666667
       2 | Horror        |       7 |  15.9622641509433962
       3 | Documentary   |      12 |  16.6666666666666667
       4 | Horror        |      23 |  15.9622641509433962
```

Question 61. Customer spend vs average spend in the same store **difficult**

Instruction

- Write a query to return a customer's life time value for the following: `customer_id IN (1, 100, 101, 200, 201, 300, 301, 400, 401, 500)`.
- Add a column to compute the average LTV of all customers from the same store.
- Return 4 columns: customer_id, store_id, customer total spend, average customer spend from the same store.
- The order of your results doesn't matter.

Hint

- Assumptions: a customer can only be associated with one store.

Table: payment

Movie rental payment transactions table

```
   col_name   | col_type
--------------+--------------------------
 payment_id   | integer
 customer_id  | smallint
 staff_id     | smallint
 rental_id    | integer
 amount       | numeric
 payment_ts   | timestamp with time zone
```

Table: customer

```
  col_name   | col_type
-------------+--------------------------
 customer_id | integer
 store_id    | smallint
 first_name  | text
 last_name   | text
 email       | text
 address_id  | smallint
 activebool  | boolean
 create_date | date
 active      | integer
```

Sample results

```
 customer_id | store_id | ltd_spend |         avg
-------------+----------+-----------+----------------------
         101 |        1 |     96.76 | 113.3933333333333333
         500 |        1 |    115.72 | 113.3933333333333333
         201 |        1 |    108.75 | 113.3933333333333333
         300 |        1 |    137.69 | 113.3933333333333333
         100 |        1 |    102.76 | 113.3933333333333333
           1 |        1 |    118.68 | 113.3933333333333333
         200 |        2 |    136.73 | 107.2550000000000000
```

Question 62. Shortest film by category medium

Instruction

- Write a query to return the shortest movie from each category.
- The order of your results doesn't matter.
- If there are ties, return just one of them.
- Return the following columns: `film_id, title, length, category, row_num`

Table: film_category

A film can only belong to one category

```
  col_name   | col_type
-------------+--------------------------
```

```
 film_id     | smallint
 category_id | smallint
```

Table: film

```
      col_name       |  col_type
---------------------+---------------------------
 film_id             | integer
 title               | text
 description         | text
 release_year        | integer
 language_id         | smallint
 original_language_id | smallint
 rental_duration     | smallint
 rental_rate         | numeric
 length              | smallint
 replacement_cost    | numeric
 rating              | text
```

Table: category

Movie categories.

```
  col_name   | col_type
-------------+-------------------------
 category_id | integer
 name        | text
```

Sample results

```
 film_id |        title         | length |    category     | row_num
---------+----------------------+--------+-----------------+---------
     869 | SUSPECTS QUILLS      |     47 | Action          |       1
     243 | DOORS PRESIDENT      |     49 | Animation       |       1
     505 | LABYRINTH LEAGUE     |     44 | Children        |       1
```

Question 63. Top 5 customers by store **medium**

Instruction

- Write a query to return the top 5 customer ids and their rankings based on their spend for each store.
- The order of your results doesn't matter.
- If there are ties, return just one of them.

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+-------------------------
 payment_id    | integer
 customer_id   | smallint
 staff_id      | smallint
```

```
 rental_id    | integer
 amount       | numeric
 payment_ts   | timestamp with time zone
```

Table: customer

```
  col_name   | col_type
-------------+--------------------------
 customer_id | integer
 store_id    | smallint
 first_name  | text
 last_name   | text
 email       | text
 address_id  | smallint
 activebool  | boolean
 create_date | date
 active      | integer
```

Sample results

```
 store_id | customer_id | revenue | ranking
----------+-------------+---------+---------
        1 |         148 |  216.54 |       1
        1 |         144 |  195.58 |       2
        1 |         459 |  186.62 |       3
        1 |         468 |  175.61 |       4
        1 |         236 |  175.58 |       5
        2 |         526 |  221.55 |       1
        2 |         178 |  194.61 |       2
        2 |         137 |  194.61 |       2
        2 |         469 |  177.60 |       3
        2 |         181 |  174.66 |       4
        2 |         259 |  170.67 |       5
```

Question 64. Top 2 films by category difficult

Instruction

- Write a query to return top 2 films based on their rental revenues in their category.
- A film can only belong to one category.
- The order of your results doesn't matter.
- If there are ties, return just one of them.
- Return the following columns: `category, film_id, revenue, row_num`

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+--------------------------
 inventory_id | integer
 film_id      | smallint
 store_id     | smallint
```

Table: payment

Movie rental payment transactions table

```
   col_name   | col_type
--------------+--------------------------
 payment_id   | integer
 customer_id  | smallint
 staff_id     | smallint
 rental_id    | integer
 amount       | numeric
 payment_ts   | timestamp with time zone
```

Table: film_category

A film can only belong to one category

```
  col_name   | col_type
-------------+--------------------------
 film_id     | smallint
 category_id | smallint
```

Table: film

```
       col_name        |  col_type
-----------------------+---------------------------
 film_id               | integer
 title                 | text
 description           | text
 release_year          | integer
 language_id           | smallint
 original_language_id  | smallint
 rental_duration       | smallint
 rental_rate           | numeric
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Table: rental

```
   col_name   | col_type
--------------+--------------------------
 rental_id    | integer
 rental_ts    | timestamp with time zone
 inventory_id | integer
 customer_id  | smallint
 return_ts    | timestamp with time zone
 staff_id     | smallint
```

Table: category

Movie categories.

col_name	col_type
category_id	integer
name	text

Sample results

category	film_id	revenue	row_num
Action	327	175.77	1
Action	21	167.78	2
Animation	239	178.70	1
Animation	865	170.76	2
Children	48	158.81	1
Children	409	132.80	2
Classics	843	141.77	1
Classics	131	137.76	2

Question 65. Movie revenue percentiles easy

Instruction

- Write a query to return percentile distribution for the following movies by their total rental revenues in the entire movie catalog.
- `film_id IN (1,10,11,20,21,30)`.
- A film can only belong to one category.
- The order of your results doesn't matter.
- Return the following columns: `film_id, revenue, percentile`

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

col_name	col_type
inventory_id	integer
film_id	smallint
store_id	smallint

Table: payment

Movie rental payment transactions table

col_name	col_type
payment_id	integer
customer_id	smallint
staff_id	smallint
rental_id	integer
amount	numeric
payment_ts	timestamp with time zone

Table: film

col_name	col_type
film_id	integer
title	text
description	text
release_year	integer
language_id	smallint
original_language_id	smallint
rental_duration	smallint
rental_rate	numeric
length	smallint
replacement_cost	numeric
rating	text

Table: rental

col_name	col_type
rental_id	integer
rental_ts	timestamp with time zone
inventory_id	integer
customer_id	smallint
return_ts	timestamp with time zone
staff_id	smallint

Sample results

film_id	revenue	percentile
11	35.76	23
1	36.77	24
30	46.91	35

Question 66. Movie percentiles by revenue by category **medium**

Instruction

- Write a query to generate percentile distribution for the following movies by their total rental revenue in their category.
- `film_id <= 20`.
- Use NTILE(100) to create percentile.
- The order of your results doesn't matter.
- Return the following columns: `category, film_id, revenue, percentile`

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

col_name	col_type
inventory_id	integer

```
 film_id      | smallint
 store_id     | smallint
```

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+--------------------------
 payment_id    | integer
 customer_id   | smallint
 staff_id      | smallint
 rental_id     | integer
 amount        | numeric
 payment_ts    | timestamp with time zone
```

Table: film_category

A film can only belong to one category

```
  col_name    | col_type
--------------+-------------------------
 film_id      | smallint
 category_id  | smallint
```

Table: film

```
       col_name        |  col_type
-----------------------+---------------------------
 film_id               | integer
 title                 | text
 description           | text
 release_year          | integer
 language_id           | smallint
 original_language_id  | smallint
 rental_duration       | smallint
 rental_rate           | numeric
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Table: category

Movie categories.

col_name	col_type
category_id	integer
name	text

Sample results

category	film_id	revenue	row_num
Action	19	33.79	11
Animation	18	32.78	13
Comedy	7	82.85	35
Documentary	1	36.77	17
Documentary	3	37.88	19
Family	5	51.88	30

Question 67. Quartile by number of rentals easy

Instruction

- Write a query to return quartiles for the following movies by number of rentals among all movies.
- `film_id IN (1,10,11,20,21,30)`.
- Use `NTILE(4)` to create quartile buckets.
- The order of your results doesn't matter.
- Return the following columns: `film_id, number of rentals, quartile`.

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

col_name	col_type
inventory_id	integer
film_id	smallint
store_id	smallint

Table: film

col_name	col_type
film_id	integer
title	text
description	text
release_year	integer
language_id	smallint
original_language_id	smallint
rental_duration	smallint
rental_rate	numeric

```
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Sample results

```
 film_id | num_rentals | quartile
---------+-------------+----------
      30 |           9 |        1
      20 |          10 |        1
      21 |          22 |        4
```

Question 68. Spend difference between first and second rentals difficult

Instruction

- Write a query to return the difference of the spend amount between the following customers' first movie rental and their second rental.
- `customer_id in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)`.
- Use `first spend - second spend` to compute the difference.
- Skip users who only rented once.

Hint:

- You can use `ROW_NUMBER` to identify the first and second transactions.
- You can use `LAG or LEAD` to find previous or following transaction amount.

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+--------------------------
 payment_id    | integer
 customer_id   | smallint
 staff_id      | smallint
 rental_id     | integer
 amount        | numeric
 payment_ts    | timestamp with time zone
```

Sample results

```
 customer_id | delta
-------------+-------
```

```
1 |  2.00
2 |  2.00
3 | -1.00
4 |  4.00
5 |  0.00
```

Question 69. Number of happy customers **difficult**

Instructions:

- Write a query to return the number of happy customers from May 24 (inclusive) to May 31 (inclusive).

Definition

- Happy customer: customers who made at least 1 rental in each day of any 2 consecutive days.

Hint

- For customer 1, you can create the following temporary table:
- customer 1, first rental date, second rental date
- customer 1, second rental date, third rental date
- .....
- customer 1, second last rental date, last rental date
- customer 1, last rental date, NULL
- As long as there is at least one row, where the delta of the last 2 columns are not null, and less or equal than 1 day, this customer must be a happy customer.

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Sample results

```
count
-------
   123
(1 row)
```

Question 70. Cumulative spend **easy**

Instructions

- Write a query to return the cumulative daily spend for the following customers:
- `customer_id in (1, 2, 3)`.
- Each day a user has a rental, return their total spent until that day.

- If there is no rental on that day, you can skip that day.

Table: payment

Movie rental payment transactions table

col_name	col_type
payment_id	integer
customer_id	smallint
staff_id	smallint
rental_id	integer
amount	numeric
payment_ts	timestamp with time zone

Sample results

date	customer_id	daily_spend	cumulative_spend
2020-05-25	1	2.99	2.99
2020-05-28	1	0.99	3.98
2020-06-15	1	16.97	20.95
2020-06-16	1	4.99	25.94
2020-06-18	1	5.98	31.92
2020-06-21	1	3.99	35.91
2020-07-08	1	11.98	47.89
2020-07-09	1	9.98	57.87
2020-07-11	1	7.99	65.86
2020-07-27	1	2.99	68.85

Question 71. Cumulative rentals easy

Instructions

- Write a query to return the cumulative daily rentals for the following customers:
- `customer_id in (3, 4, 5)`.
- Each day a user had a rental, return their total spent until that day.
- If there is no rental on that day, ignore that day.

Table: rental

col_name	col_type
rental_id	integer
rental_ts	timestamp with time zone
inventory_id	integer
customer_id	smallint
return_ts	timestamp with time zone
staff_id	smallint

Sample results

date	customer_id	daily_rental	cumulative_rentals
2020-05-27	3	1	1

```
2020-05-29 |           3 |           1 |                  2
2020-06-16 |           3 |           2 |                  4
2020-06-17 |           3 |           1 |                  5
2020-06-19 |           3 |           1 |                  6
2020-07-07 |           3 |           1 |                  7
2020-07-08 |           3 |           1 |                  8
```

Question 72. Days when they became happy customers **easy**

Instructions:

- Any customers who made at least 10 movie rentals are happy customers, write a query to return the dates when the following customers became happy customers:
- `customer_id in (1,2,3,4,5,6,7,8,9,10)`.
- You can skip a customer if he/she never became a 'happy customer'.

Table: rental

```
   col_name   | col_type
--------------+--------------------------
 rental_id    | integer
 rental_ts    | timestamp with time zone
 inventory_id | integer
 customer_id  | smallint
 return_ts    | timestamp with time zone
 staff_id     | smallint
```

Sample results

```
 customer_id |    date
-------------+------------
           1 | 2020-07-08
           2 | 2020-07-29
           3 | 2020-07-27
           4 | 2020-07-30
           5 | 2020-07-06
           6 | 2020-07-10
```

Question 73. Number of days to become a happy customer **medium**

Instructions:

- Any customers who made 10 movie rentals are happy customers
- Write a query to return the average number of days for a customer to make his/her 10th rental.
- If a customer has never become a 'happy' customer, you should skip this customer when computing the average.
- You can use `EXTRACT(DAYS FROM tenth_rental_ts - first_rental_ts)` to get the number of days in between the 1st rental and 10th rental
- Use `ROUND(average_days)` to return an integer

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Sample results

```
 avg
-------
    65
```

Question 74. The most productive actors by category **medium**

Instructions:

- An actor's productivity is defined as the number of movies he/she has played.
- Write a query to return the `category_id, actor_id and number of movies`by the most productive actor in that category.
- For example: John Doe filmed the most **action** movies, your query will return John as the result for action movie category.
- Do this for every movie category.

Table: actor

```
  col_name    | col_type
--------------+--------------------------
 actor_id     | integer
 first_name   | text
 last_name    | text
```

Table: film_actor

Films and their casts

```
  col_name    | col_type
--------------+--------------------------
 actor_id     | smallint
 film_id      | smallint
```

Table: film_category

A film can only belong to one category

```
  col_name    | col_type
--------------+--------------------------
 film_id      | smallint
 category_id  | smallint
```

Sample results

```
 category_id | actor_id | num_movies
-------------+----------+------------
           1 |       50 |          6
           2 |      150 |          6
           3 |       17 |          7
           4 |       86 |          6
           5 |      196 |          6
           6 |       48 |          6
           7 |        7 |          7
```

Question 75. Top customer by movie category **medium**

Instruction

- For each movie category: return the customer id who spend the most in rentals.
- If there are ties, return any customer id.

Hint

- To save you some time, you can use the following CTE (Common table expression) to get each customer's spend by movie category:

```
WITH cust_revenue_by_cat AS (
    SELECT
        P.customer_id,
    FC.category_id,
    SUM(P.amount) AS revenue
    FROM payment P
    INNER JOIN rental R
    ON R.rental_id = P.rental_id
    INNER JOIN inventory I
    ON I.inventory_id = R.inventory_id
    INNER JOIN film F
    ON F.film_id = I.film_id
    INNER JOIN film_category FC
    ON FC.film_id = F.film_id
    GROUP BY P.customer_id, FC.category_id
)
```

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name   | col_type
--------------+-------------------------
 inventory_id | integer
 film_id      | smallint
 store_id     | smallint
```

Table: payment

Movie rental payment transactions table

```
   col_name   | col_type
--------------+-------------------------
```

```
 payment_id   | integer
 customer_id  | smallint
 staff_id     | smallint
 rental_id    | integer
 amount       | numeric
 payment_ts   | timestamp with time zone
```

Table: film_category

A film can only belong to one category

```
  col_name   | col_type
-------------+-------------------------
 film_id     | smallint
 category_id | smallint
```

Table: film

```
       col_name        |  col_type
-----------------------+----------------------------
 film_id               | integer
 title                 | text
 description           | text
 release_year          | integer
 language_id           | smallint
 original_language_id  | smallint
 rental_duration       | smallint
 rental_rate           | numeric
 length                | smallint
 replacement_cost      | numeric
 rating                | text
```

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Sample results

```
 category_id | customer_id
-------------+-------------
           1 |         363
           2 |         526
           3 |         467
           4 |         293
           5 |         459
```

Question 76. Districts with the most and least customers easy

Instructions

- Return the `district` where the most and least number of customers are.
- Append a column to indicate whether this district has the most customers or least customers with **'most'** or **'least'** category.
- **HINT**: it is possible an address is not associated with any customer.

Table: address

```
  col_name   | col_type
-------------+--------------------------
 address_id  | integer
 address     | text
 address2    | text
 district    | text
 city_id     | smallint
 postal_code | text
 phone       | text
```

Table: customer

```
  col_name   | col_type
-------------+--------------------------
 customer_id | integer
 store_id    | smallint
 first_name  | text
 last_name   | text
 email       | text
 address_id  | smallint
 activebool  | boolean
 create_date | date
 active      | integer
```

Sample results

```
district  | cat
---------+----------
   New York | most
     Kamado | least
```

Question 77. Movie revenue percentiles by category easy

Instructions:

- Write a query to return revenue percentiles (ordered ascendingly) of the following movies within their category:
- `film_id IN (1,2,3,4,5)`.

Hint

- Use `NTILE(100)` to create percentiles.
- To save you some time, here is the CTE to create a movie's revenue by category.

```
WITH movie_rev_by_cat AS (
    SELECT
       F.film_id,
       MAX(FC.category_id) AS category_id,
       SUM(P.amount) AS revenue
    FROM film F
    INNER JOIN inventory I
    ON I.film_id = F.film_id
    INNER JOIN rental R
    ON R.inventory_id = I.inventory_id
    INNER JOIN payment P
    ON P.rental_id = R.rental_id
    INNER JOIN film_category FC
    ON FC.film_id = F.film_id
    GROUP BY F.film_id
)
```

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+-------------------------
 inventory_id | integer
 film_id      | smallint
 store_id     | smallint
```

Table: payment

Movie rental payment transactions table

```
   col_name    | col_type
---------------+-------------------------
 payment_id   | integer
 customer_id  | smallint
 staff_id     | smallint
 rental_id    | integer
 amount       | numeric
 payment_ts   | timestamp with time zone
```

Table: film_category

A film can only belong to one category

```
  col_name    | col_type
--------------+-------------------------
 film_id     | smallint
 category_id | smallint
```

Table: film

```
      col_name       |  col_type
```

```
----------------------+--------------------------
 film_id              | integer
 title                | text
 description          | text
 release_year         | integer
 language_id          | smallint
 original_language_id | smallint
 rental_duration      | smallint
 rental_rate          | numeric
 length               | smallint
 replacement_cost     | numeric
 rating               | text
```

Table: rental

```
   col_name    | col_type
---------------+--------------------------
 rental_id     | integer
 rental_ts     | timestamp with time zone
 inventory_id  | integer
 customer_id   | smallint
 return_ts     | timestamp with time zone
 staff_id      | smallint
```

Sample results

```
 film_id | perc_by_cat
---------+-------------
       1 |          17
       3 |          19
       5 |          30
       2 |          22
       4 |          36
```

Question 78. Quartiles buckets by number of rentals **medium**

Instructions:

- Write a query to return the quartile by the number of rentals for the following customers:
- `customer_id IN (1,2,3,4,5,6,7,8,9,10)`

Hint

- USE `NTILE(4)` to create quartiles.
- To save you some time, here is the CTE to create customer rentals by store:

```
WITH cust_rentals AS (
    SELECT C.customer_id,
    MAX(C.store_id) AS store_id, -- one customer can only belong to one
store
    COUNT(*) AS num_rentals FROM
    rental R
    INNER JOIN customer C
    ON C.customer_id = R.customer_id
    GROUP BY C.customer_id
)
```

Table: rental

col_name	col_type
rental_id	integer
rental_ts	timestamp with time zone
inventory_id	integer
customer_id	smallint
return_ts	timestamp with time zone
staff_id	smallint

Table: customer

col_name	col_type
customer_id	integer
store_id	smallint
first_name	text
last_name	text
email	text
address_id	smallint
activebool	boolean
create_date	date
active	integer

Sample results

customer_id	store_id	quartile
1	1	2
2	1	2
3	1	1
4	2	1

Question 79. Spend difference between the last and the second last rentals **difficult**

Instructions

- Write a query to return the spend amount difference between the last and the second last movie rentals for the following customers:
- `customer_id IN (1,2,3,4,5,6,7,8,9,10)`.
- Skip customers if they made less than 2 rentals.

Hint

- Use `ROW_NUMBER` to determine the sequence of movie rental

Table: payment

Movie rental payment transactions table

col_name	col_type

```
 payment_id   | integer
 customer_id  | smallint
 staff_id     | smallint
 rental_id    | integer
 amount       | numeric
 payment_ts   | timestamp with time zone
```

Sample results

```
 customer_id | delta
-------------+-------
           1 |  3.00
           2 |  0.00
           3 |  2.00
           4 | -1.00
           5 | -2.00
```

Question 80. DoD revenue growth for each store difficult

Instruction

- Write a query to return DoD(day over day) growth for each store from May 24 (inclusive) to May 31 (inclusive).
- `DoD:` (current_day/ prev_day -1) * 100.0
- Multiply dod growth to **100.0** to get percentage of growth.
- Use `ROUND` to convert dod growth to the nearest integer.

Hint

- To save you some time, use the following CTE to create a store's daily revenue:

```
WITH store_daily_rev AS (
  SELECT
    I.store_id,
    DATE(P.payment_ts) date,
    SUM(amount) AS daily_rev
  FROM
    payment P
  INNER JOIN rental R
  ON R.rental_id = P.rental_id
  INNER JOIN inventory I
  ON I.inventory_id = R.inventory_id
  WHERE DATE(P.payment_ts) >= '2020-05-01'
  AND DATE(P.payment_ts) <= '2020-05-31'
  GROUP BY I.store_id, DATE(P.payment_ts)
)
```

Table: inventory

Each row is unique; Inventoy_id is the primary key of the table

```
   col_name    | col_type
---------------+--------------------------
 inventory_id | integer
 film_id      | smallint
 store_id     | smallint
```

Table: payment

Movie rental payment transactions table

col_name	col_type
payment_id	integer
customer_id	smallint
staff_id	smallint
rental_id	integer
amount	numeric
payment_ts	timestamp with time zone

Table: rental

col_name	col_type
rental_id	integer
rental_ts	timestamp with time zone
inventory_id	integer
customer_id	smallint
return_ts	timestamp with time zone
staff_id	smallint

Sample results

store_id	date	dod_growth
1	2020-05-24	
1	2020-05-25	2058
1	2020-05-26	137
1	2020-05-27	74
1	2020-05-28	166
1	2020-05-29	62
1	2020-05-30	109
1	2020-05-31	107
2	2020-05-24	
2	2020-05-25	1794

Question 83. Top search_query in US and UK on new year's day **medium**

- Write a query to return the top searched term in the US and UK on new year's day (2021-01-01), separately
- The order of your results doesn't matter.
- Rank them based on search volume.

Table: search

```
  col_name| col_type
----------+------------
country   | varchar(2)
date      | date
user_id   | integer
search_id | integer
query     | text
```

Sample results

```
 country      | query
--------------+-------
           US |  Joe Biden
           UK |  David Beckham
```

Question 88. Top song report medium

Instruction

- Write a query to return the top song id for every country
- Return a unique row for each country
- For simplicity, let's assume there is no tie.
- The order of your results doesn't matter.

Table: song_plays

Number of times a song is played (streamed), aggregated on daily basis.

```
  col_name   | col_type
-------------+--------------------------
 date        | date
 country     | varchar(2)
 song_id     | bigint
 num_plays   | bigint
```

Sample results

```
 country      | song_id
--------------+-------
           US |  12345
           UK |  23456
           JP |  23456
           CA |  12345
           AU |  23456
```

Question 90. Top artist report medium

Instruction

- Write a query to return the top artist id for every country
- Return a unique row for each country
- For simplicity, let's assume there is no tie.
- The order of your results doesn't matter.

Table: artist

```
  col_name    | col_type
--------------+--------------------------
 artist_id     | bigint
 name      | VARCHAR(255)
```

Table: song_plays

Number of times a song is played (streamed), aggregated on daily basis.

```
  col_name    | col_type
--------------+--------------------------
 date         | date
 country      | varchar(2)
 song_id      | bigint
 num_plays    | bigint
```

Sample results

```
 country       | artist_id
---------------+-------
            US |  100
            UK |  200
            JP |  300
            CA |  100
            AU |  200
```

SOLUTIONS

SINGLE TABLE OPERATIONS

Question 1. Top store for movie sales easy

Instruction
Write a query to return the name of the store and its manager, that generated the most sales.

Solution
```
SELECT store, manager
FROM sales_by_store
ORDER BY total_sales DESC
LIMIT 1;
```

Question 2. Top 3 movie categories by sales easy

Instruction
- Write a query to find the top 3 film categories that generated the most sales.
- The order of your results doesn't matter.

Solution
```
SELECT category
FROM sales_by_film_category
ORDER BY total_sales DESC
LIMIT 3;
```

Question 3. Top 5 shortest movies easy

Instruction
- Write a query to return the titles of the 5 shortest movies by duration.
- The order of your results doesn't matter.

Solution
```
SELECT title
FROM film
ORDER BY length
LIMIT 5;
```

Question 4. Staff without a profile image easy

Instruction
- Write a SQL query to return this staff's first name and last name.
- Picture field contains the link that points to a staff's profile image.
- There is only one staff who doesn't have a profile picture.
- Use `colname IS NULL` to identify data that are missing.

Solution
```
SELECT first_name, last_name
FROM staff
WHERE picture IS NULL;
```

Question 5. Monthly revenue easy

Instruction

- Write a query to return the total movie rental revenue for each month.
- You can use `EXTRACT(MONTH FROM colname)` and `EXTRACT(YEAR FROM colname)` to extract month and year from a timestamp column.

Solution

```
SELECT
    EXTRACT(YEAR FROM payment_ts) AS year,
    EXTRACT(MONTH FROM payment_ts) AS mon,
    SUM(amount) as rev
FROM payment
GROUP BY year, mon
ORDER BY year, mon;
```

Question 6. Daily revenue in June, 2020 easy

Instruction

- Write a query to return daily revenue in June, 2020.
- Use `DATE(colname)` to extract the date from a timestamp column.
- `payment_ts`'s data type is timestamp, convert it to date before grouping.
- No dates need to be included if there was no revenue on that day.

Solution

```
-- For course taker
SELECT
    DATE(payment_ts) AS dt,
    SUM(amount)
FROM payment
WHERE EXTRACT(MONTH FROM payment_ts) = '6'
AND EXTRACT(YEAR FROM payment_ts) = '2020'
GROUP BY dt
ORDER BY dt;
```

Solution

```
SELECT
    DATE(payment_ts) AS dt,
    SUM(amount)
FROM payment
WHERE DATE(payment_ts) >= '2020-06-01'
AND DATE(payment_ts) <= '2020-06-30'
GROUP BY dt;
```

Question 7. Unique customers count by month easy

Instruction

- Write a query to return the total number of unique customers for each month
- Use `EXTRACT(YEAR from ts_field)` and `EXTRACT(MONTH from ts_field)` to get year and month from a timestamp column.
- The order of your results doesn't matter.

Solution

```
SELECT
    EXTRACT(YEAR FROM rental_ts) AS year,
    EXTRACT(MONTH FROM rental_ts) AS mon,
    COUNT(DISTINCT customer_id) AS uu_cnt
FROM rental
GROUP BY year, mon;
```

Question 8. Average customer spend by month **medium**

Instruction

- Write a query to return the average customer spend by month.
- Definition: average customer spend: total customer spend divided by the unique number of customers for that month.
- Use `EXTRACT(YEAR from ts_field)` and `EXTRACT(MONTH from ts_field)` to get year and month from a timestamp column.
- The order of your results doesn't matter.

Solution

```
SELECT
    EXTRACT(YEAR FROM payment_ts) AS year,
    EXTRACT(MONTH FROM payment_ts) AS mon,
    SUM(amount)/COUNT(DISTINCT customer_id) AS avg_spend
FROM payment
GROUP BY year, mon
ORDER BY year, mon;
```

Question 9. Number of high spend customers by month **medium**

Instruction

- Write a query to count the number of customers who spend more than (>) $20 by month
- Use `EXTRACT(YEAR from ts_field)` and `EXTRACT(MONTH from ts_field)` to get year and month from a timestamp column.
- The order of your results doesn't matter.
- **Hint:** a customer's spend varies every month.

Solution

```
SELECT
    year,
    mon,
    COUNT(DISTINCT customer_id)
FROM (
    SELECT
        EXTRACT(YEAR FROM payment_ts) AS year,
        EXTRACT(MONTH FROM payment_ts) AS mon,
        customer_id,
        SUM(amount) amt
    FROM payment
    GROUP BY year, mon, customer_id
) X
WHERE amt > 20
GROUP BY 1,2;
```

Question 10. Min and max spend **medium**

Instruction

- Write a query to return the minimum and maximum customer total spend in June 2020.
- For each customer, first calculate their total spend in June 2020.
- Then use `MIN`, and `MAX` function to return the min and max customer spend .

Solution

```
WITH cust_tot_amt AS (
    SELECT
        customer_id,
        SUM(amount) AS tot_amt
    FROM payment
    WHERE DATE(payment_ts) >= '2020-06-01'
    AND DATE(payment_ts) <= '2020-06-30'
    GROUP BY customer_id
)
SELECT
    MIN(tot_amt) AS min_spend,
    MAX(tot_amt) AS max_spend
FROM cust_tot_amt;
```

Question 11. Actors' last name **easy**

Instruction

Find the number of actors whose last name is one of the following: `'DAVIS'`, `'BRODY'`, `'ALLEN'`, `'BERRY'`

Solution

```
SELECT
  last_name,
  COUNT(*)
FROM actor
WHERE last_name IN ('DAVIS', 'BRODY', 'ALLEN', 'BERRY')
GROUP BY last_name;
```

Question 12. Actors' last name ending in 'EN' or 'RY' **easy**

Instruction

- Identify all actors whose last name ends in `'EN'` or `'RY'`.
- Group and count them by their last name.

Solution

```
SELECT
  last_name,
  COUNT(*)
FROM actor
WHERE last_name LIKE ('%RY')
OR last_name LIKE ('%EN')
GROUP BY last_name;
```

Question 13. Actors' first name **medium**

Instruction

- Write a query to return the number of actors whose first name starts with 'A', 'B', 'C', or others.
- The order of your results doesn't matter.
- You need to return 2 columns:
- The first column is the group of actors based on the first letter of their `first_name`, use the following: `'a_actors'`, `'b_actors'`, `'c_actors'`, `'other_actors'` to represent their groups.
- Second column is the number of actors whose first name matches the pattern.

Solution

```
SELECT
 CASE WHEN first_name LIKE 'A%' THEN 'a_actors'
      WHEN first_name LIKE 'B%' THEN 'b_actors'
      WHEN first_name LIKE 'C%' THEN 'c_actors'
      ELSE 'other_actors'
      END AS actor_category,
  COUNT(*)
FROM actor
GROUP BY actor_category;
```

Question 14. Good days and bad days **difficult**

Instruction

- Write a query to return the number of good days and bad days in May 2020 based on number of daily rentals.
- Return the results in one row with 2 columns from left to right: good_days, bad_days.
- `good day`: `> 100` rentals.
- `bad day`: `<= 100` rentals.
- **Hint** (For users already know `OUTER JOIN`), you can use `dates` table
- **Hint**: be super careful about datetime columns.
- **Hint**: this problem could be tricky, feel free to explore the `rental` table and take a look at some data.

Solution

```
-- For people following my course (who have not learned outer join yet)
WITH daily_rentals AS (
    SELECT
     DATE(rental_ts) AS dt,
     COUNT(*) AS num_rentals
    FROM rental
    WHERE DATE(rental_ts) >= '2020-05-01'
        AND DATE(rental_ts) <= '2020-05-31'
    GROUP BY dt
)
SELECT
    SUM(CASE WHEN num_rentals > 100 THEN 1
         ELSE 0
         END) AS good_days,
    31 - SUM(CASE WHEN num_rentals > 100 THEN 1
         ELSE 0
         END) AS bad_days
FROM daily_rentals;
```

Solution

```
-- (For users who already know OUTER JOIN):

WITH daily_rentals AS (
  SELECT
      D.date AS dt,
      COUNT(R.rental_id) AS num_rentals
  FROM dates D
  LEFT JOIN rental R
  ON D.date = DATE(R.rental_ts)
  WHERE D.date >= '2020-05-01'
```

```
  AND D.date <= '2020-05-31'
  GROUP BY D.date
)
SELECT
    SUM(CASE WHEN num_rentals >100 THEN 1 ELSE 0 END) AS good_days,
    SUM(CASE WHEN num_rentals <=100 THEN 1 ELSE 0 END) AS bad_days
FROM daily_rentals;
```

Question 15. Fast movie watchers vs slow watchers **difficult**

Instruction

- Write a query to return the number of fast movie watchers vs slow movie watchers.
- `fast movie watcher`: by average return their rentals within 5 days.
- `slow movie watcher`: takes an average of `>5` days to return their rentals.
- Most customers have multiple rentals over time, you need to first compute the number of days for each rental transaction, then compute the average on the rounded up days. e.g., if the rental period is 1 day and 10 hours, count it as 2 days.
- Skip the rentals that have not been returned yet, e.g., `rental_ts IS NULL`.
- The orders of your results doesn't matter.
- A customer can only rent one movie per transaction.

Solution

```
WITH average_rental_days AS (
    SELECT
        customer_id,
        AVG(EXTRACT(days FROM (return_ts - rental_ts) ) + 1) AS average_days
    FROM rental
    WHERE return_ts IS NOT NULL
    GROUP BY 1
)
SELECT CASE WHEN average_days <= 5 THEN 'fast_watcher'
            WHEN average_days > 5 THEN 'slow_watcher'
            ELSE NULL
            END AS watcher_category,
        COUNT(*)
FROM average_rental_days
GROUP BY watcher_category;
```

Question 16. Staff who live in Woodridge **easy**

Instruction

- Write a query to return the names of the staff who live in the city of 'Woodridge'

Solution

```
SELECT name
FROM staff_list
WHERE city = 'Woodridge';
```

Question 17. GROUCHO WILLIAMS' actor_id **easy**

Instruction

- Write a query to return GROUCHO WILLIAMS' `actor_id`.
- Actor's first_name and last_name are all stored as UPPER case in our database, and the database is case sensitive.

Solution

```
SELECT actor_id
FROM actor
WHERE first_name = 'GROUCHO'
AND last_name = 'WILLIAMS';
```

Question 18. Top film category easy

Instruction

- Write a query to return the film category id with the most films, as well as the number films in that category.

Solution

```
SELECT
    category_id,
    COUNT(*) film_cnt
FROM film_category
GROUP BY category_id
ORDER BY film_cnt DESC
LIMIT 1;
```

Question 19. Most productive actor medium

Instruction

- Write a query to return the first name and the last name of the actor who appeared in the most films.

Solution

```
WITH top_actor AS (
    SELECT
      actor_id,
      COUNT(*) AS film_cnt
    FROM film_actor
    GROUP BY actor_id
    ORDER BY film_cnt DESC
    LIMIT 1
)
SELECT first_name, last_name
FROM actor A
WHERE A.actor_id
IN (
  SELECT actor_id
  FROM top_actor
);
```

Question 20. Customer who spent the most medium

Instruction

- Write a query to return the first and last name of the customer who spent the most on movie rentals in Feb 2020.

Solution

```
WITH cust_feb_spend AS (
    SELECT
        customer_id,
        SUM(amount) AS cust_amt
    FROM payment
    WHERE DATE(payment_ts) >= '2020-02-01'
```

```
    AND DATE(payment_ts) <= '2020-02-29'
    GROUP BY customer_id
    ORDER BY cust_amt DESC
    LIMIT 1
)
SELECT first_name, last_name
FROM customer
WHERE customer_id IN (
    SELECT customer_id
    FROM cust_feb_spend
);
```

Question 21. Customer who rented the most **medium**

Instruction

- Write a query to return the first and last name of the customer who made the most rental transactions in May 2020.

Solution

```
WITH cust_may_rentals AS (
    SELECT
        customer_id,
        COUNT(*) AS cust_rentals
    FROM rental
    WHERE DATE(rental_ts) >= '2020-05-01'
    AND DATE(rental_ts) <= '2020-05-31'
    GROUP BY customer_id
    ORDER BY cust_rentals DESC
    LIMIT 1
)
SELECT first_name, last_name
FROM customer
WHERE customer_id IN (
    SELECT customer_id
    FROM cust_may_rentals
);
```

Question 22. Average cost per rental transaction **easy**

Instruction

- Write a query to return the average cost on movie rentals in May 2020 per transaction.

Solution

```
SELECT AVG(amount)
FROM payment
WHERE  DATE(payment_ts) >= '2020-05-01'
AND DATE(payment_ts) <= '2020-05-31';
```

Question 23. Average spend per customer in Feb 2020 **easy**

Instruction

- Write a query to return the average movie rental spend per customer in Feb 2020.

Solution

```
WITH cust_feb_spend AS (
  SELECT customer_id,
      SUM(amount) cust_spend
```

```
  FROM payment
  WHERE DATE(payment_ts ) >= '2020-02-01'
  AND DATE(payment_ts ) <= '2020-02-28'
  GROUP BY customer_id
)
SELECT AVG(cust_spend)
FROM cust_feb_spend
;
```

Question 24. Films with more than 10 actors **medium**

Instruction

- Write a query to return the titles of the films with `>= 10` actors.

Solution

```
WITH film_casts_cnt AS (
    SELECT
        film_id,
        COUNT(*) AS actors_cnt
    FROM film_actor
    GROUP BY film_id
    HAVING COUNT(*)>=10
)

SELECT title
FROM film
WHERE film_id IN (
    SELECT film_id
    FROM film_casts_cnt
)
```

Question 25. Shortest film **easy**

Instruction

- Write a query to return the title of the film with the minimum duration.
- A movie's duration can be found using the **length** column.
- If there are ties, e.g., two movies have the same **length**, return either one of them.

Solution

```
SELECT title
FROM film
ORDER BY length
LIMIT 1;
```

Question 26. Second shortest film **medium**

Instruction

- Write a query to return the title of the second shortest film based on its duration/length.
- A movie's duration can be found using the **length** column.
- If there are ties, return just one of them.

Solution

```
WITH shortest_2 AS (
    SELECT film_id, length
    FROM film
    ORDER BY length
    LIMIT 2
```

```
)
SELECT title
FROM film
WHERE film_id IN (
    SELECT film_id
    FROM shortest_2
    ORDER BY length DESC
    LIMIT 1
);
```

Question 27. Film with the largest cast **easy**

Instruction

- Write a query to return the title of the film with the largest cast (most actors).
- If there are ties, return just one of them.

Solution

```
WITH film_size AS (
    SELECT film_id,
    COUNT(*) AS actors_cnt
    FROM film_actor
    GROUP BY film_id
    ORDER BY actors_cnt DESC
    LIMIT 1
)
SELECT title
FROM film
WHERE film_id IN (
    SELECT film_id
    FROM film_size
);
```

Question 28. Film with the second largest cast **medium**

Instruction

- Write a query to return the title of the film with the second largest cast.
- If there are ties, e.g., two movies have the same number of actors, return either one of the movie.

Solution

```
WITH film_size AS (
    SELECT film_id,
    COUNT(*) AS actors_cnt
    FROM film_actor
    GROUP BY film_id
    ORDER BY actors_cnt DESC
    LIMIT 2
)
SELECT title
FROM film
WHERE film_id IN (
    SELECT film_id
    FROM film_size
    ORDER BY actors_cnt
    LIMIT 1
);
```

Question 29. Second highest spend customer **medium**

Instruction

- Write a query to return the name of the customer who spent the second highest for movie rentals in May 2020.
- If there are ties, return any one of them.

Solution

```
WITH cust_spend_may AS (
    SELECT customer_id,
    SUM(amount) AS cust_spend
    FROM payment
    WHERE payment_ts >= '2020-05-01'
    AND payment_ts < '2020-06-01'
    GROUP BY customer_id
    ORDER BY cust_spend DESC
    LIMIT 2
)
SELECT
    first_name,
    last_name
FROM customer
WHERE customer_id IN (
    SELECT customer_id
    FROM cust_spend_may
    ORDER BY cust_spend
    LIMIT 1
);
```

Question 30. Inactive customers in May **easy**

Instruction

- Write a query to return the total number of customers who didn't rent any movies in May 2020.

Hint

- You can use `NOT IN` to exclude customers who have rented movies in May 2020.

Solution

```
SELECT COUNT(*)
FROM customer
WHERE customer_id NOT IN(
    SELECT customer_id
    FROM rental
    WHERE  DATE(rental_ts) >= '2020-05-01'
    AND    DATE(rental_ts) <= '2020-05-31'
);
```

Question 31. Movies that have not been returned **easy**

Instruction

- Write a query to return the titles of the films that have not been returned by our customers.

Hint

- If a movie is not returned, the `return_ts` will be `NULL` in the rental table.

Solution

```
WITH out_film AS (
    SELECT DISTINCT film_id
    FROM inventory
    WHERE inventory_id IN (
    SELECT inventory_id
        FROM rental
    WHERE return_ts IS NULL
    )
)

SELECT title
FROM film
WHERE film_id IN (
    SELECT film_id
    FROM out_film
)
;
```

Question 32. Unpopular movies **difficult**

Instruction

- Write a query to return the number of films with no rentals in Feb 2020.
- Count the entire movie catalog from the `film` table.

Solution

```
WITH rented_film AS (
    SELECT DISTINCT film_id
    FROM inventory
    WHERE inventory_id IN(
        SELECT inventory_id
        FROM rental
        WHERE DATE(rental_ts) >= '2020-02-01'
        AND   DATE(rental_ts) <= '2020-02-29'
    )
)

SELECT COUNT(*)
FROM film
WHERE film_id NOT IN(
    SELECT film_id
    FROM rented_film
);
```

Question 33. Returning customers **medium**

Instruction

- Write a query to return the number of customers who rented at least one movie in both May 2020 and June 2020.

Solution

```
WITH may_cust AS (
    SELECT DISTINCT customer_id AS may_cust_id
    FROM rental
    WHERE DATE(rental_ts) >= '2020-05-01'
    AND   DATE(rental_ts) <= '2020-05-31'
)
```

```
SELECT COUNT(DISTINCT customer_id)
FROM rental
WHERE DATE(rental_ts) >= '2020-06-01'
AND   DATE(rental_ts) <= '2020-06-30'
AND  customer_id IN (
    SELECT may_cust_id
    FROM may_cust
);
```

Question 34. Stocked up movies easy

Instruction

- Write a query to return the titles of movies with more than `>7` dvd copies in the inventory.

Solution

```
SELECT title
FROM film
WHERE film_id IN (
    SELECT
        film_id
    FROM inventory
    GROUP BY film_id
    HAVING COUNT(*) >=8
);
```

Question 35. Film length report easy

Instruction

- Write a query to return the number of films in the following categories: short, medium, and long.
- The order of your results doesn't matter.

Definition

- short: less `<60` minutes.
- medium: `>=60` minutes, but `<100` minutes.
- long: `>=100` minutes

Solution

```
SELECT
  CASE WHEN length < 60 THEN 'short'
       WHEN length < 100 THEN 'medium'
       WHEN length >= 100 THEN 'long'
       ELSE NULL
       END AS film_category,
  COUNT(*)
FROM film
GROUP BY film_category;
```

Question 81. How many people searched on new year's day easy

Write a query to return the total number of users who have searched on new year's day: 2021-01-01. **Solution**

```
SELECT COUNT(DISTINCT user_id)
FROM search
WHERE date = '2021-01-01';
```

Solution

```
SELECT COUNT(user_id) FROM (
    SELECT
    user_id
    FROM search
    WHERE date = '2021-01-01'
    GROUP BY 1
) X;
```

Question 82. The top search query on new year's day easy

Write a query to return the top search term on new year's day: 2021-01-01 **Solution**

```
SELECT query FROM (
    SELECT
    query,
    COUNT(*)
    FROM search
    WHERE date = '2021-01-01'
    GROUP BY 1
    ORDER BY 2 DESC
    LIMIT 1
) X
```

Question 84. Click through rate on new year's day easy

- Write a query to compute the click through rate for the search results on new year's day (2021-01-01).
- `Click through rate:` number of searches end up with at least one click.
- Convert your result into a percentage (* 100.0).

Solution

```
SELECT
COUNT(DISTINCT CASE WHEN action = 'click' THEN search_id ELSE NULL END) *
100.0/COUNT(DISTINCT search_id)
FROM search_result
WHERE date = '2021-01-01';
```

Question 85. Top 5 queries based on click through rate on new year's day difficult

Instruction

- Write a query to return the top 5 search terms with the highest click through rate on new year's day (2021-01-01)
- The search term has to be searched by more than 2 `(>2)` distinct users.
- `Click through rate:` number of searches end up with at least one click.

Solution

```
WITH click_through_rate AS (
  SELECT
      S.query,
```

```
      COUNT(DISTINCT CASE WHEN action='click' THEN S.search_id ELSE NULL
END) * 100/COUNT(DISTINCT S.search_id) ctr
  FROM
  search S
  INNER JOIN search_result R
  ON S.search_id = R.search_id
  WHERE S.date = '2021-01-01'
  GROUP BY S.query
  HAVING COUNT(DISTINCT S.user_id) > 2
)
SELECT query
FROM click_through_rate
ORDER BY ctr DESC
LIMIT 5;
```

Question 86. Top song in the US **easy**

Write a query to return the name of the top song in the US yesterday. **Solution**

```
SELECT title FROM SONG
WHERE song_id IN (
    SELECT d.song_id, s.name, d.plays
    FROM daily_plays d
    WHERE d.country = 'US'
    AND d.date = CURRENT_DATE - 1
    ORDER BY plays DESC
    LIMIT 1
);
```

Question 89. Top 5 artists in the US **medium**

Instruction

- Write a query to return the top 5 artists id for US yesterday.
- Return a unique row for each country
- For simplicity, let's assume there is no tie.
- The order of your results doesn't matter.

Solution

```
WITH artist_plays AS (
    SELECT
        S.artist_id,
        P.country,
        SUM(num_plays) num_plays
    FROM song_plays P
    INNER JOIN song S
    ON S.song_id = P.song_id
    WHERE P.date = CURRENT_DATE - 1
    GROUP BY 1,2
)
SELECT artist_id
FROM artist_plays
ORDER BY num_plays DESC
LIMIT 5;
```

MULTI-TABLE OPERATIONS

Question 36. Actors from film 'AFRICAN EGG' easy

Instruction

- Write a query to return the first name and last name of all actors in the film 'AFRICAN EGG'.
- The order of your results doesn't matter.

Solution

```
SELECT A.first_name, A.last_name
FROM film F
INNER JOIN film_actor FA
ON FA.film_id = F.film_id
INNER JOIN actor A
ON A.actor_id = FA.actor_id
WHERE F.title = 'AFRICAN EGG';
```

Question 37. Most popular movie category easy

Instruction

- Return the name of the category that has the most films.
- If there are ties, return just one of them.

Solution

```
SELECT
    C.name
FROM film_category FC
INNER JOIN category C
ON C.category_id = FC.category_id
GROUP BY C.name
ORDER BY COUNT(*) DESC
LIMIT 1;
```

Question 38. Most popular movie category (name and id) medium

Instruction

- Write a query to return the name of the most popular film category and its category id
- If there are ties, return just one of them.

Solution

```
SELECT
    C.category_id,
    MAX(C.name) AS name
FROM film_category FC
INNER JOIN category C
ON C.category_id = FC.category_id
GROUP BY C.category_id
ORDER BY COUNT(*) DESC
LIMIT 1;
```

Question 39. Most productive actor with inner join easy

Instruction

- Write a query to return the name of the actor who appears in the most films.

- You have to use INNER JOIN in your query.

Solution

```
SELECT
    FA.actor_id,
    MAX(A.first_name) first_name,
    MAX(A.last_name) last_name
FROM film_actor FA
INNER JOIN actor A
ON A.actor_id = FA.actor_id
GROUP BY FA.actor_id
ORDER BY COUNT(*) DESC
LIMIT 1;
```

Question 40. Top 5 most rented movie in June 2020 **medium**

Instruction

- Write a query to return the `film_id` and `title` of the top 5 movies that were rented the most times in June 2020
- Use the `rental_ts` column from the `rental` for the transaction time.
- The order of your results doesn't matter.
- If there are ties, return any 5 of them.

Solution

```
SELECT
    F.film_id,
    MAX(F.title) AS title
FROM rental R
INNER JOIN inventory I
ON I.inventory_id = R.inventory_id
INNER JOIN film F
ON F.film_id = I.film_id
WHERE DATE(rental_ts) >= '2020-06-01'
AND   DATE(rental_ts) <= '2020-06-30'
GROUP BY F.film_id
ORDER BY COUNT(*) DESC
LIMIT 5;
```

Question 41. Productive actors vs less-productive actors **medium**

Instruction

- Write a query to return the number of `productive` and `less-productive` actors.
- The order of your results doesn't matter.

Definition

- `productive`: appeared in `>= 30` films.
- `less-productive`: appeared in `<30` films.

Solution

```
SELECT actor_category,
    COUNT(*)
FROM (
    SELECT
        A.actor_id,
        CASE WHEN  COUNT(DISTINCT FA.film_id) >= 30 THEN 'productive' ELSE
'less productive' END AS actor_category
    FROM actor A
```

```
    LEFT JOIN film_actor FA
    ON FA.actor_id = A.actor_id
    GROUP BY A.actor_id
) X
GROUP BY actor_category;
```

Question 42. Films that are in stock vs not in stock **medium**

Instruction

- Write a query to return the number of films that we have inventory vs no inventory.
- A film can have multiple inventory ids
- Each film dvd copy has a unique inventory ids

Solution

```
SELECT in_stock, COUNT(*)
FROM (
    SELECT
        F.film_id,
        MAX(CASE WHEN I.inventory_id IS NULL THEN 'not in stock' ELSE 'in
stock' END) in_stock
    FROM film F
    LEFT JOIN INVENTORY I
    ON F.film_id =I.film_id
    GROUP BY F.film_id
) X
GROUP BY in_stock;
```

Question 43. Customers who rented vs. those who did not **medium**

Instruction

- Write a query to return the number of customers who rented at least one movie vs. those who didn't in **May 2020**.
- The order of your results doesn't matter.
- Use `customer` table as the base table for all customers (assuming all customers have signed up before May 2020)
- `Rented`: if a customer rented at least one movie.
- **Bonus:** Develop a `LEFT JOIN` as well as a `RIGHT JOIN` solution

Solution

```
SELECT have_rented, COUNT(*)
FROM (
    SELECT
        C.customer_id,
        CASE WHEN R.customer_id IS NOT NULL THEN 'rented' ELSE 'never-
rented' END AS have_rented
    FROM customer C
    LEFT JOIN (
        SELECT DISTINCT customer_id
        FROM rental
        WHERE DATE(rental_ts) >= '2020-05-01'
        AND DATE(rental_ts) <= '2020-05-31'
    ) R
    ON R.customer_id = C.customer_id
) X
GROUP BY have_rented;
```

Question 44. In-demand vs not-in-demand movies **medium**

Instruction

- Write a query to return the number of `in demand` and `not in demand` movies in May 2020.
- Assumptions (great to clarify in your interview): all films are available for rent before May.
- But if a film is not in stock, it is not in demand.
- The order of your results doesn't matter.

Definition

- `in-demand`: rented `>1 times` in May 2020.
- `not-in-demand`: rented `<= 1 time` in May 2020.

Solution

```
SELECT demand_category, COUNT(*)
FROM (
    SELECT
        F.film_id,
        CASE WHEN COUNT(R.rental_id) >1 THEN 'in demand' ELSE 'not in
demand' END AS demand_category
    FROM film F
    LEFT JOIN INVENTORY I
    ON F.film_id =I.film_id
    LEFT JOIN (
        SELECT inventory_id, rental_id
        FROM rental
        WHERE DATE(rental_ts) >= '2020-05-01'
        AND DATE(rental_ts) <= '2020-05-31'
    ) R
    ON R.inventory_id = I.inventory_id
    GROUP BY F.film_id
)X
GROUP BY demand_category;
```

Question 45. Movie inventory optimization **difficult**

Instruction

- For movies that are not in demand (rentals = 0 in May 2020), we want to remove them from our inventory.
- Write a query to return the number of unique `inventory_id` from those movies with **0 demand**.
- **Hint**: a movie can have multiple `inventory_id`.

Solution

```
SELECT COUNT(inventory_id )
FROM inventory I
INNER JOIN (
    SELECT F.film_id
    FROM film F
    LEFT JOIN (
        SELECT  DISTINCT I.film_id
        FROM inventory I
        INNER JOIN (
        SELECT inventory_id, rental_id
        FROM rental
```

```
        WHERE DATE(rental_ts) >= '2020-05-01'
        AND DATE(rental_ts) <= '2020-05-31'
        ) R
        ON I.inventory_id = R.inventory_id
    ) X ON X.film_id = F.film_id
    WHERE X.film_id IS NULL
)Y
ON Y.film_id = I.film_id;
```

Question 46. Actors and customers whose last name starts with 'A' easy

Instruction

- Write a query to return unique names (first_name, last_name) of our `customers` and `actors` whose last name starts with letter 'A'.

Solution

```
SELECT first_name, last_name
FROM customer
WHERE last_name LIKE 'A%'
UNION
SELECT first_name, last_name
FROM actor
WHERE last_name LIKE 'A%';
```

Question 47. Actors and customers whose first names end in 'D'. easy

Instruction

- Write a query to return all actors and customers whose first names ends in 'D'.
- Return their ids (for actor: use `actor_id`, customer: `customer_id`), `first_name` and `last_name`.
- The order of your results doesn't matter.

Solution

```
SELECT customer_id, first_name, last_name
FROM customer
WHERE first_name LIKE '%D'
UNION
SELECT actor_id, first_name, last_name
FROM actor
WHERE first_name LIKE '%D';
```

Question 48. Movie and TV actors easy

Instruction

- Write a query to return actors who appeared in both tv and movies
- The order of your results doesn't matter.
- You need to use `INNER JOIN`.

Solution

```
SELECT
    M.actor_id,
    M.first_name,
    M.last_name
FROM actor_movie M
INNER JOIN actor_tv T
ON T.actor_id = M.actor_id;
```

Question 49. Top 3 money making movie categories **medium**

Instruction

- Write a query to return the `name` of the 3 movie categories that generated the most rental revenue
- And rental revenue from each of the category.
- The order of your results doesn't matter.
- If there are ties, return just one of them.

Solution

```
SELECT C.name, SUM(P.amount)
FROM
    payment P
INNER JOIN rental R
ON R.rental_id = P.rental_id
INNER JOIN inventory I
ON I.inventory_id = R.inventory_id
INNER JOIN film F
ON F.film_id = I.film_id
INNER JOIN film_category FC
ON FC.film_id = F.film_id
INNER JOIN category C
ON C.category_id = FC.category_id
GROUP BY C.name
ORDER BY SUM(P.amount) DESC
LIMIT 3
;
```

Question 50. Top 5 cities for movie rentals **easy**

Instruction

- Write a query to return the names of the top 5 cities with the most rental revenues in 2020.
- Include each city's revenue in the second column.
- The order of your results doesn't matter.
- If there are ties, return any one of them.
- Yours results should have exactly 5 rows.

Solution

```
SELECT
    T.city,
    SUM(P.amount)
FROM payment P
INNER JOIN customer C
ON C.customer_id = P.customer_id
INNER JOIN address A
ON A.address_id = C.address_id
INNER JOIN city T
ON T.city_id = A.city_id
WHERE DATE(P.payment_ts) >= '2020-01-01'
AND DATE(P.payment_ts) <= '2020-12-31'
GROUP BY T.city
ORDER BY SUM(P.amount) DESC
LIMIT 5;
```

Question 51. Movie only actor easy

Instruction

- Write a query to return the first name and last name of actors who only appeared in movies.
- Actor appeared in tv should not be included .
- The order of your results doesn't matter.

Solution

```
SELECT M.first_name, M.last_name
FROM actor_movie M
LEFT JOIN actor_tv T
ON M.actor_id = T.actor_id
WHERE T.actor_id IS NULL;
```

Question 52. Movies cast by movie only actors difficult

Instruction

- Write a query to return the `film_id` with movie only casts (actors who never appeared in tv).
- The order of your results doesn't matter.
- You should exclude movies with one or more tv actors

Solution

```
SELECT F.film_id
FROM film F
LEFT JOIN (
    SELECT DISTINCT FA.film_id
    FROM film_actor FA
    INNER JOIN actor_tv T
    ON T.actor_id = FA.actor_id
) X
ON F.film_id = X.film_id
WHERE X.film_id IS NULL;
```

Question 53. Movie groups by rental income difficult

Instruction

- Write a query to return the number of films in 3 separate groups: high, medium, low.
- The order of your results doesn't matter.

Definition

- high: revenue `>= $100`.
- medium: revenue `>= $20, <$100` .
- low: revenue `<$20`.

Hint

- If a movie has no rental revenue, it belongs to the `low` group

Solution

```
SELECT film_group, COUNT(*)
FROM (
    SELECT
        F.film_id,
        CASE WHEN SUM(P.amount) >= 100 THEN 'high'
             WHEN SUM(P.amount) >= 20 THEN 'medium'
```

```
            ELSE 'low' END film_group
    FROM film F
    LEFT JOIN inventory I
    ON I.film_id = F.film_id
    LEFT JOIN rental R
    ON R.inventory_id = I.inventory_id
    LEFT JOIN payment P
    ON P.rental_id = R.rental_id
    GROUP BY F.film_id
) X
GROUP BY film_group
;
```

Question 54. Customer groups by movie rental spend **medium**

Instruction

- Write a query to return the number of customers in 3 separate groups: high, medium, low.
- The order of your results doesn't matter.

Definition

- high: movie rental spend `>= $150`.
- medium: movie rental spend `>= $100, <$150`.
- low: movie rental spend `<$100`.

Hint

- If a customer spend 0 in movie rentals, he/she belongs to the `low` group.

Solution

```
SELECT customer_group, COUNT(*)
FROM (
    SELECT
        C.customer_id,
        CASE WHEN SUM(P.amount) >= 150 THEN 'high'
             WHEN SUM(P.amount) >= 100 THEN 'medium'
             ELSE 'low' END customer_group
    FROM customer C
    LEFT JOIN payment P
    ON P.customer_id = C.customer_id
    GROUP BY C.customer_id
) X
GROUP BY customer_group
;
```

Question 55. Busy days and slow days **medium**

Instruction

- Write a query to return the number of busy days and slow days in **May 2020** based on the number of movie rentals.
- The order of your results doesn't matter.
- If there are ties, return just one of them.

Definition

- busy: rentals `>= 100`.
- slow: rentals `< 100`.

Solution

```
SELECT date_category, COUNT(*)
```

```
FROM (
    SELECT  D.date,
        CASE WHEN COUNT(*) >= 100 THEN 'busy' ELSE 'slow' END date_category
    FROM dates D
    LEFT JOIN (
        SELECT * FROM rental
    ) R
    ON D.date = DATE(R.rental_ts)
    WHERE D.date >= '2020-05-01'
    AND   D.date <= '2020-05-31'
    GROUP BY D.date
) X
GROUP BY date_category
;
```

Question 56. Total number of actors **easy**

Instruction

- Write a query to return the total number of actors from actor_tv, actor_movie with `FULL OUTER JOIN`.
- Use `COALESCE` to return the first non-null value from a list.
- Actors who appear in both tv and movie share the same value of `actor_id` in both actor_tv and actor_movie tables.

Solution

```
SELECT COUNT(DISTINCT actor_id) FROM (
    SELECT
         COALESCE(T.actor_id, M.actor_id) AS actor_id
    FROM actor_tv T
    FULL OUTER JOIN
        actor_movie M
    ON M.actor_id = T.actor_id
) X;
```

Question 57. Total number of actors (with UNION) **easy**

Instruction

- Write a query to return the total number of actors using `UNION`.
- Actor who appeared in both tv and movie has the same value of `actor_id` in both actor_tv and actor_movie tables.

Solution

```
SELECT COUNT(*) FROM (
    SELECT
         T.actor_id
    FROM actor_tv T
    UNION
    SELECT
        M.actor_id
    FROM actor_movie M
) X;
```

Question 87. Top song in the US and UK **medium**

Write a query to return the name of the top song in the US and UK yesterday, respectively. **Solution**

```
WITH top_song AS (
    SELECT
        S.song_id,
        P.country,
        MAX(S.name) AS song_name,
        SUM(plays) num_plays
    FROM daily_plays P
    INNER JOIN song S
    ON P.song_id = S.id
    WHERE P.date = CURRENT_DATE - 1
    AND country = 'US'
    ORDER BY num_plays DESC
    LIMIT 1
    UNION ALL
    SELECT
        S.song_id,
        P.country,
        MAX(S.name) AS song_name,
        SUM(plays) num_plays
    FROM daily_plays P
    INNER JOIN song S
    ON P.song_id = S.id
    WHERE P.date = CURRENT_DATE - 1
    AND country = 'UK'
    ORDER BY num_plays DESC
    LIMIT 1
)
SELECT country, song_name
FROM top_song;
```

WINDOW FUNCTIONS

Question 58. Percentage of revenue per movie **medium**

Instruction

- Write a query to return the percentage of revenue for each of the following films: `film_id <= 10`.
- Formula: `revenue (film_id x) * 100.0/ revenue of all movies`.
- The order of your results doesn't matter.

Solution

```
WITH movie_revenue AS (
    SELECT
        I.film_id, SUM(P.amount) revenue
    FROM payment P
    INNER JOIN rental R
    ON R.rental_id = P.rental_id
    INNER JOIN inventory I
    ON I.inventory_id = R.inventory_id
    GROUP BY I.film_id
)
SELECT film_id, revenue * 100.0 / SUM(revenue) OVER() revenue_percentage
FROM movie_revenue
ORDER BY film_id
LIMIT 10
;
```

Question 59. Percentage of revenue per movie by category **medium**

Instruction

- Write a query to return the percentage of revenue for each of the following films: `film_id <= 10` by its category.
- Formula: `revenue (film_id x) * 100.0/ revenue of all movies in the same category`.
- The order of your results doesn't matter.
- Return 3 columns: film_id, category name, and percentage.

Solution

```
WITH movie_revenue AS (
    SELECT
        I.film_id, SUM(P.amount) revenue
    FROM payment P
    INNER JOIN rental R
    ON R.rental_id = P.rental_id
    INNER JOIN inventory I
    ON I.inventory_id = R.inventory_id
    GROUP BY I.film_id
)
SELECT
    MR.film_id,
    C.name category_name,
    revenue * 100.0 / SUM(revenue) OVER(PARTITION BY C.name)
revenue_percent_category
FROM movie_revenue MR
INNER JOIN film_category FC
  ON FC.film_id = MR.film_id
INNER JOIN category C
  ON C.category_id = FC.category_id
ORDER BY film_id
LIMIT 10
;
```

Question 60. Movie rentals and average rentals in the same category **medium**

Instruction

- Write a query to return the number of rentals per movie, and the average number of rentals in its same category.
- You only need to return results for `film_id <= 10`.
- Return 4 columns: film_id, category name, number of rentals, and the average number of rentals from its category.

Solution

```
WITH movie_rental AS (
    SELECT
        I.film_id,
        COUNT(*) rentals
    FROM rental R
    INNER JOIN inventory I
    ON I.inventory_id = R.inventory_id
    GROUP BY I.film_id
)
SELECT
    film_id,
    category_name,
    rentals,
    avg_rentals_category
FROM (
    SELECT
        MR.film_id,
        C.name category_name,
        rentals,
        AVG(rentals) OVER(PARTITION BY C.name) avg_rentals_category
    FROM movie_rental MR
    INNER JOIN film_category FC
      ON FC.film_id = MR.film_id
    INNER JOIN category C
      ON C.category_id = FC.category_id
) X
WHERE film_id <= 10
;
```

Question 61. Customer spend vs average spend in the same store **difficult**

Instruction

- Write a query to return a customer's life time value for the following: `customer_id IN (1, 100, 101, 200, 201, 300, 301, 400, 401, 500)`.
- Add a column to compute the average LTV of all customers from the same store.
- Return 4 columns: customer_id, store_id, customer total spend, average customer spend from the same store.
- The order of your results doesn't matter.

Hint

- Assumptions: a customer can only be associated with one store.

Solution

```
WITH customer_ltd_spend AS (
    SELECT
        P.customer_id,
        MAX(store_id) store_id,
        SUM(P.amount) ltd_spend
    FROM payment P
    INNER JOIN customer C
```

```
    ON C.customer_id = P.customer_id
    GROUP BY P.customer_id
)

SELECT customer_id, store_id, ltd_spend, store_avg
FROM (
    SELECT
        customer_id,
        store_id,
        ltd_spend,
        AVG(ltd_spend) OVER(PARTITION BY store_id) as store_avg
    FROM customer_ltd_spend CLS
) X
WHERE X.customer_id IN (1,100,101, 200, 201, 300,301, 400, 401, 500)
ORDER BY 1;
```

Question 62. Shortest film by category **medium**

Instruction

- Write a query to return the shortest movie from each category.
- The order of your results doesn't matter.
- If there are ties, return just one of them.
- Return the following columns: `film_id, title, length, category, row_num`

Solution

```
WITH movie_ranking AS (
  SELECT
    F.film_id,
    F.title,
    F.length,
    C.name category,
    ROW_NUMBER() OVER(PARTITION BY C.name ORDER BY F.length) row_num
  FROM film F
  INNER JOIN film_category FC
  ON FC.film_id = F.film_id
  INNER JOIN category C
  ON C.category_id = FC.category_id
)

SELECT
  film_id,
  title,
  length,
  category,
  row_num
FROM movie_ranking
WHERE row_num = 1
;
```

Solution

```
SELECT
  film_id,
  title,
  length,
  category,
  row_num
FROM (
```

```
  SELECT
    F.film_id,
    F.title,
    F.length,
    C.name category,
    ROW_NUMBER() OVER(PARTITION BY C.name ORDER BY F.length) row_num
  FROM film F
  INNER JOIN film_category FC
  ON FC.film_id = F.film_id
  INNER JOIN category C
  ON C.category_id = FC.category_id
) X
WHERE row_num = 1
;
```

Question 63. Top 5 customers by store **medium**

Instruction

- Write a query to return the top 5 customer ids and their rankings based on their spend for each store.
- The order of your results doesn't matter.
- If there are ties, return just one of them.

Solution

```
WITH cust_revenue AS (
  SELECT
    C.customer_id,
    MAX(store_id) store_id,
    SUM(amount) revenue
  FROM customer C
  INNER JOIN payment P
  ON P.customer_id = C.customer_id
  GROUP BY C.customer_id
)
SELECT * FROM (
  SELECT
    store_id,
    customer_id,
    revenue,
    DENSE_RANK() OVER(PARTITION BY store_id ORDER BY revenue DESC) ranking
  FROM cust_revenue
) X
WHERE ranking <= 5;
```

Question 64. Top 2 films by category **difficult**

Instruction

- Write a query to return top 2 films based on their rental revenues in their category.
- A film can only belong to one category.
- The order of your results doesn't matter.
- If there are ties, return just one of them.
- Return the following columns: `category, film_id, revenue, row_num`

Solution

```
WITH film_revenue AS (
  SELECT
```

```
    F.film_id,
    MAX(C.name) AS category,
    SUM(P.amount) revenue
  FROM payment P
  INNER JOIN rental R
  ON R.rental_id = P.rental_id
  INNER JOIN inventory I
  ON I.inventory_id = R.inventory_id
  INNER JOIN film F
  ON F.film_id = I.film_id
  INNER JOIN film_category FC
  ON FC.film_id = F.film_id
  INNER JOIN category C
  ON C.category_id = FC.category_id
  GROUP BY F.film_id
)

SELECT * FROM (
  SELECT
    category,
    FR.film_id,
    revenue,
    ROW_NUMBER() OVER(PARTITION BY category ORDER BY revenue DESC) row_num
  FROM film_revenue FR
  INNER JOIN film_category FC
  ON FC.film_id = FR.film_id
  INNER JOIN category C
  ON C.category_id = FC.category_id
) X
WHERE row_num <= 2;
```

Question 65. Movie revenue percentiles **easy**

Instruction

- Write a query to return percentile distribution for the following movies by their total rental revenues in the entire movie catalog.
- `film_id IN (1,10,11,20,21,30)`.
- A film can only belong to one category.
- The order of your results doesn't matter.
- Return the following columns: `film_id, revenue, percentile`

Solution

```
WITH film_revenue AS (
  SELECT
    F.film_id,
    SUM(P.amount) revenue,
    NTILE(100) OVER(ORDER BY SUM(P.amount)) AS percentile
  FROM payment P
  INNER JOIN rental R
  ON R.rental_id = P.rental_id
  INNER JOIN inventory I
  ON I.inventory_id = R.inventory_id
  INNER JOIN film F
  ON F.film_id = I.film_id
  GROUP BY F.film_id
)
```

```
SELECT
  film_id,
  revenue,
  percentile
FROM film_revenue
WHERE film_id IN (1,10,11,20,21,30);
```

Question 66. Movie percentiles by revenue by category **medium**

Instruction

- Write a query to generate percentile distribution for the following movies by their total rental revenue in their category.
- `film_id <= 20`.
- Use NTILE(100) to create percentile.
- The order of your results doesn't matter.
- Return the following columns: `category, film_id, revenue, percentile`

Solution

```
WITH film_revenue_by_cat AS (
  SELECT
    F.film_id,
    MAX(C.name) AS category,
    SUM(P.amount) revenue
  FROM payment P
  INNER JOIN rental R
  ON R.rental_id = P.rental_id
  INNER JOIN inventory I
  ON I.inventory_id = R.inventory_id
  INNER JOIN film F
  ON F.film_id = I.film_id
  INNER JOIN film_category FC
  ON FC.film_id = F.film_id
  INNER JOIN category C
  ON C.category_id = FC.category_id
  GROUP BY F.film_id
)

SELECT
    category,
    film_id,
    revenue,
    percentile
FROM (
  SELECT
    category,
    FR.film_id,
    revenue,
    NTILE(100) OVER(PARTITION BY category ORDER BY revenue) percentile
  FROM film_revenue_by_cat FR
  INNER JOIN film_category FC
  ON FC.film_id = FR.film_id
  INNER JOIN category C
  ON C.category_id = FC.category_id
) X
WHERE film_id <=20
ORDER BY category, revenue;
```

Question 67. Quartile by number of rentals easy

Instruction

- Write a query to return quartiles for the following movies by number of rentals among all movies.
- `film_id IN (1,10,11,20,21,30)`.
- Use `NTILE(4)` to create quartile buckets.
- The order of your results doesn't matter.
- Return the following columns: `film_id, number of rentals, quartile`.

Solution

```
WITH movie_rentals AS (
  SELECT
    F.film_id,
    COUNT(*) AS num_rentals,
    NTILE(4) OVER(ORDER BY COUNT(*)) AS quartile
  FROM rental R
  INNER JOIN inventory I
  ON I.inventory_id = R.inventory_id
  INNER JOIN film F
  ON F.film_id = I.film_id
  GROUP BY F.film_id
)
SELECT *
FROM movie_rentals
WHERE film_id IN (1,10,11,20,21,30);
```

Question 68. Spend difference between first and second rentals difficult

Instruction

- Write a query to return the difference of the spend amount between the following customers' first movie rental and their second rental.
- `customer_id in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)`.
- Use `first spend - second spend` to compute the difference.
- Skip users who only rented once.

Hint:

- You can use `ROW_NUMBER` to identify the first and second transactions.
- You can use `LAG or LEAD` to find previous or following transaction amount.

Solution

```
SELECT customer_id,
    prev_amount - current_amount AS delta
FROM (
    SELECT
      customer_id,
      payment_ts,
      amount as current_amount,
      LAG(amount, 1) OVER(PARTITION BY customer_id ORDER BY payment_ts ) AS
prev_amount,
      ROW_NUMBER() OVER(PARTITION BY customer_id ORDER BY payment_ts) AS
payment_idx
    FROM payment
    WHERE customer_id IN
    (1,2,3,4,5,6,7,8,9,10)
```

```
) X
WHERE payment_idx = 2;
```

Question 69. Number of happy customers **difficult**

Instructions:

- Write a query to return the number of happy customers from May 24 (inclusive) to May 31 (inclusive).

Definition

- Happy customer: customers who made at least 1 rental in each day of any 2 consecutive days.

Hint

- For customer 1, you can create the following temporary table:
- customer 1, first rental date, second rental date
- customer 1, second rental date, third rental date
- .....
- customer 1, second last rental date, last rental date
- customer 1, last rental date, NULL
- As long as there is at least one row, where the delta of the last 2 columns are not null, and less or equal than 1 day, this customer must be a happy customer.

Solution

```
WITH customer_rental_date AS (
    SELECT
            customer_id,
            DATE(rental_ts) AS rental_date
    FROM rental
    WHERE DATE(rental_ts) >= '2020-05-24'
    AND DATE(rental_ts) <= '2020-05-31'
    GROUP BY
            customer_id,
            DATE(rental_ts)
),

customer_rental_date_diff AS (
    SELECT
        customer_id,
        rental_date AS current_rental_date,
        LAG( rental_date, 1) OVER(PARTITION BY customer_id ORDER BY
rental_date) AS prev_rental_date
    FROM customer_rental_date
)

SELECT COUNT(*) FROM (
    SELECT
        customer_id,
        MIN(current_rental_date - prev_rental_date)
    FROM customer_rental_date_diff
    GROUP BY customer_id
    HAVING MIN(current_rental_date - prev_rental_date) = 1
) X
;
```

Question 70. Cumulative spend **easy**

Instructions

- Write a query to return the cumulative daily spend for the following customers:
- `customer_id in (1, 2, 3)`.
- Each day a user has a rental, return their total spent until that day.
- If there is no rental on that day, you can skip that day.

Solution

```
WITH customer_spend AS (
    SELECT
      DATE(payment_ts) date,
      customer_id,
      SUM(amount) AS daily_spend
    FROM payment
    WHERE customer_id IN (1, 2, 3)
    GROUP BY DATE(payment_ts), customer_id
)

SELECT
  date,
  customer_id,
  daily_spend,
  SUM(daily_spend) OVER(PARTITION BY customer_id ORDER BY date)
cumulative_spend
FROM customer_spend;
```

Question 71. Cumulative rentals easy

Instructions

- Write a query to return the cumulative daily rentals for the following customers:
- `customer_id in (3, 4, 5)`.
- Each day a user had a rental, return their total spent until that day.
- If there is no rental on that day, ignore that day.

Solution

```
WITH customer_rentals AS (
    SELECT
      DATE(rental_ts) date,
      customer_id,
      COUNT(*) AS daily_rental
    FROM rental
    WHERE customer_id IN (3,4,5)
    GROUP BY DATE(rental_ts), customer_id
)

SELECT
  date,
  customer_id,
  daily_rental,
  SUM(daily_rental) OVER(PARTITION BY customer_id ORDER BY date)
cumulative_rentals
FROM customer_rentals;
```

Question 72. Days when they became happy customers easy

Instructions:

- Any customers who made at least 10 movie rentals are happy customers, write a query to return the dates when the following customers became happy customers:
- `customer_id in (1,2,3,4,5,6,7,8,9,10)`.
- You can skip a customer if he/she never became a 'happy customer'.

Solution

```
WITH cust_rental_dates AS (
    SELECT
      customer_id,
      DATE(rental_ts) date,
      ROW_NUMBER() OVER(PARTITION BY customer_id ORDER BY rental_ts)
rental_idx
    FROM rental
    WHERE customer_id IN (1,2,3,4,5,6,7,8,9,10)
)
SELECT
  customer_id,
  date
FROM cust_rental_dates
WHERE rental_idx = 10;
```

Question 73. Number of days to become a happy customer **medium**

Instructions:

- Any customers who made 10 movie rentals are happy customers
- Write a query to return the average number of days for a customer to make his/her 10th rental.
- If a customer has never become a 'happy' customer, you should skip this customer when computing the average.
- You can use `EXTRACT(DAYS FROM tenth_rental_ts - first_rental_ts)` to get the number of days in between the 1st rental and 10th rental
- Use `ROUND(average_days)` to return an integer

Solution

```
WITH cust_rental_ts AS (
    SELECT
        customer_id,
    rental_ts,
    ROW_NUMBER() OVER(PARTITION BY customer_id ORDER BY rental_ts)
rental_idx
    FROM rental
)
SELECT ROUND(AVG(delta)) AS avg_days FROM (
    SELECT
        customer_id,
    first_rental_ts,
    tenth_rental_ts,
    EXTRACT(DAYS FROM tenth_rental_ts - first_rental_ts) AS delta
    FROM (
        SELECT
            customer_id,
        MAX(CASE WHEN rental_idx = 1 THEN rental_ts ELSE NULL END) AS
first_rental_ts,
        MAX(CASE WHEN rental_idx = 10 THEN rental_ts ELSE NULL END) AS
tenth_rental_ts
     FROM cust_rental_ts
```

```
     GROUP BY customer_id
    ) X
    WHERE tenth_rental_ts IS NOT NULL
)Y;
```

Question 74. The most productive actors by category **medium**

Instructions:

- An actor's productivity is defined as the number of movies he/she has played.
- Write a query to return the `category_id, actor_id and number of movies` by the most productive actor in that category.
- For example: John Doe filmed the most **action** movies, your query will return John as the result for action movie category.
- Do this for every movie category.

Solution

```
WITH actor_movies AS (
  SELECT
    FC.category_id,
    FA.actor_id,
    COUNT(DISTINCT F.film_id) num_movies
  FROM film_actor FA
  INNER JOIN film F
  ON F.film_id = FA.film_id
  INNER JOIN film_category FC
  ON FC.film_id = F.film_id
  GROUP BY FC.category_id, FA.actor_id
)
SELECT category_id, actor_id, num_movies
FROM (
    SELECT
        category_id,
        actor_id,
        num_movies,
        ROW_NUMBER()OVER(PARTITION BY category_id ORDER BY num_movies DESC)
AS productivity_idx
    FROM actor_movies
) X
WHERE productivity_idx = 1;
```

Question 75. Top customer by movie category **medium**

Instruction

- For each movie category: return the customer id who spend the most in rentals.
- If there are ties, return any customer id.

Hint

- To save you some time, you can use the following CTE (Common table expression) to get each customer's spend by movie category:

```
WITH cust_revenue_by_cat AS (
    SELECT
        P.customer_id,
    FC.category_id,
    SUM(P.amount) AS revenue
    FROM payment P
    INNER JOIN rental R
```

```
    ON R.rental_id = P.rental_id
    INNER JOIN inventory I
    ON I.inventory_id = R.inventory_id
    INNER JOIN film F
    ON F.film_id = I.film_id
    INNER JOIN film_category FC
    ON FC.film_id = F.film_id
    GROUP BY P.customer_id, FC.category_id
)
```

Solution

```
WITH cust_revenue_by_cat AS (
    SELECT
      P.customer_id,
      FC.category_id,
      SUM(P.amount) AS revenue
    FROM payment P
    INNER JOIN rental R
    ON R.rental_id = P.rental_id
    INNER JOIN inventory I
    ON I.inventory_id = R.inventory_id
    INNER JOIN film F
    ON F.film_id = I.film_id
    INNER JOIN film_category FC
    ON FC.film_id = F.film_id
    GROUP BY P.customer_id, FC.category_id
)
SELECT category_id, customer_id
FROM (
    SELECT
      customer_id,
      category_id,
      ROW_NUMBER() OVER(PARTITION BY category_id ORDER BY revenue DESC) AS
rev_cat_idx
    FROM cust_revenue_by_cat
) X
WHERE rev_cat_idx = 1;
```

Question 76. Districts with the most and least customers easy

Instructions

- Return the `district` where the most and least number of customers are.
- Append a column to indicate whether this district has the most customers or least customers with **'most'** or **'least'** category.
- **HINT**: it is possible an address is not associated with any customer.

Solution

```
WITH district_cust_cnt AS (
    SELECT
        A.district,
        COUNT(DISTINCT C.customer_id) cust_cnt,
        ROW_NUMBER() OVER(ORDER BY COUNT(DISTINCT C.customer_id) ASC) AS
cust_asc_idx,
        ROW_NUMBER() OVER(ORDER BY COUNT(DISTINCT C.customer_id) DESC) AS
cust_desc_idx
    FROM address A
    LEFT JOIN customer C
```

```
    ON A.address_id = C.address_id
    GROUP BY A.district
)
SELECT
    district,
    'least' AS city_cat
FROM district_cust_cnt
WHERE cust_asc_idx = 1
UNION
SELECT
    district,
    'most' AS city_cat
FROM district_cust_cnt
WHERE cust_desc_idx = 1
;
```

Question 77. Movie revenue percentiles by category easy

Instructions:

- Write a query to return revenue percentiles (ordered ascendingly) of the following movies within their category:
- `film_id IN (1,2,3,4,5)`.

Hint

- Use `NTILE(100)` to create percentiles.
- To save you some time, here is the CTE to create a movie's revenue by category.

```
WITH movie_rev_by_cat AS (
    SELECT
       F.film_id,
       MAX(FC.category_id) AS category_id,
       SUM(P.amount) AS revenue
    FROM film F
    INNER JOIN inventory I
    ON I.film_id = F.film_id
    INNER JOIN rental R
    ON R.inventory_id = I.inventory_id
    INNER JOIN payment P
    ON P.rental_id = R.rental_id
    INNER JOIN film_category FC
    ON FC.film_id = F.film_id
    GROUP BY F.film_id
)
```

Solution

```
WITH movie_rev_by_cat AS (
    SELECT
       F.film_id,
       MAX(FC.category_id) AS category_id,
       SUM(P.amount) AS revenue
    FROM film F
    INNER JOIN inventory I
    ON I.film_id = F.film_id
    INNER JOIN rental R
    ON R.inventory_id = I.inventory_id
    INNER JOIN payment P
    ON P.rental_id = R.rental_id
    INNER JOIN film_category FC
```

```
    ON FC.film_id = F.film_id
    GROUP BY F.film_id
)
SELECT film_id, perc_by_cat
FROM (
    SELECT film_id,
        NTILE(100) OVER(PARTITION BY category_id ORDER BY revenue) AS
perc_by_cat
    FROM movie_rev_by_cat
)X
WHERE film_id IN (1,2,3,4,5);
```

Question 78. Quartiles buckets by number of rentals **medium**

Instructions:

- Write a query to return the quartile by the number of rentals for the following customers:
- `customer_id IN (1,2,3,4,5,6,7,8,9,10)`

Hint

- USE `NTILE(4)` to create quartiles.
- To save you some time, here is the CTE to create customer rentals by store:

```
WITH cust_rentals AS (
    SELECT C.customer_id,
    MAX(C.store_id) AS store_id, -- one customer can only belong to one
store
    COUNT(*) AS num_rentals FROM
    rental R
    INNER JOIN customer C
    ON C.customer_id = R.customer_id
    GROUP BY C.customer_id
)
```

Solution

```
WITH cust_rentals AS (
    SELECT C.customer_id,
    MAX(C.store_id) AS store_id, -- one customer can only belong to one
store
    COUNT(*) AS num_rentals FROM
    rental R
    INNER JOIN customer C
    ON C.customer_id = R.customer_id
    GROUP BY C.customer_id
)
SELECT customer_id, store_id, quartile
FROM (
    SELECT
        customer_id,
        store_id,
        NTILE(4) OVER(PARTITION BY store_id ORDER BY num_rentals) AS
quartile
    FROM cust_rentals
) X
WHERE customer_id IN (1,2,3,4,5,6,7,8,9,10);
```

Question 79. Spend difference between the last and the second last rentals **difficult**

Instructions

- Write a query to return the spend amount difference between the last and the second last movie rentals for the following customers:
- `customer_id IN (1,2,3,4,5,6,7,8,9,10)`.
- Skip customers if they made less than 2 rentals.

Hint

- Use `ROW_NUMBER` to determine the sequence of movie rental

Solution

```
WITH cust_spend_seq AS (
    SELECT
      customer_id,
      payment_ts,
      amount AS current_payment,
      LAG(amount, 1) OVER(PARTITION BY customer_id ORDER BY payment_ts) AS
prev_payment,
      ROW_NUMBER() OVER(PARTITION BY customer_id ORDER BY payment_ts DESC)
AS payment_idx
    FROM payment P
)
SELECT
    customer_id,
    current_payment - prev_payment AS delta
FROM cust_spend_seq
WHERE customer_id IN(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
AND payment_idx = 1;
```

Question 80. DoD revenue growth for each store **difficult**

Instruction

- Write a query to return DoD(day over day) growth for each store from May 24 (inclusive) to May 31 (inclusive).
- `DoD:` (current_day/ prev_day -1) * 100.0
- Multiply dod growth to **100.0** to get percentage of growth.
- Use `ROUND` to convert dod growth to the nearest integer.

Hint

- To save you some time, use the following CTE to create a store's daily revenue:

```
WITH store_daily_rev AS (
  SELECT
    I.store_id,
    DATE(P.payment_ts) date,
    SUM(amount) AS daily_rev
  FROM
    payment P
  INNER JOIN rental R
  ON R.rental_id = P.rental_id
  INNER JOIN inventory I
  ON I.inventory_id = R.inventory_id
  WHERE DATE(P.payment_ts) >= '2020-05-01'
  AND DATE(P.payment_ts) <= '2020-05-31'
  GROUP BY I.store_id, DATE(P.payment_ts)
)
```

Solution

```
WITH store_daily_rev AS (
  SELECT
```

```
    I.store_id,
    DATE(P.payment_ts) date,
    SUM(amount) AS daily_rev
  FROM
    payment P
  INNER JOIN rental R
  ON R.rental_id = P.rental_id
  INNER JOIN inventory I
  ON I.inventory_id = R.inventory_id
  WHERE DATE(P.payment_ts) >= '2020-05-01'
  AND DATE(P.payment_ts) <= '2020-05-31'
  GROUP BY I.store_id, DATE(P.payment_ts)
)
SELECT
  store_id,
  date,
  ROUND( (daily_rev / LAG(daily_rev, 1) OVER(PARTITION BY store_id ORDER BY
date) -1) * 100.0 ) AS dod_growth
FROM store_daily_rev;
```

Question 83. Top search_query in US and UK on new year's day **medium**

- Write a query to return the top searched term in the US and UK on new year's day (2021-01-01), separately
- The order of your results doesn't matter.
- Rank them based on search volume.

Solution

```
SELECT
    country,
    query
FROM (
    SELECT
        query,
        country,
        COUNT(DISTINCT user_id),
        ROW_NUMBER() OVER(PARTITION BY country ORDER BY COUNT(DISTINCT
user_id) DESC) AS row_num
    FROM search
    WHERE country IN ('US', 'UK')
    AND date = '2021-01-01'
    GROUP BY 1,2
) X
WHERE row_num = 1;
```

Question 88. Top song report **medium**

Instruction

- Write a query to return the top song id for every country
- Return a unique row for each country
- For simplicity, let's assume there is no tie.
- The order of your results doesn't matter.

Solution

```
WITH song_rankings AS (
    SELECT
        P.song_id,
        P.country,
        ROW_NUMBER() OVER(PARTITION BY country ORDER BY num_plays DESC) AS
ranking
    FROM daily_plays P
    WHERE P.date = CURRENT_DATE - 1
)
SELECT
    song_id,
    country
FROM song_rankings
WHERE ranking = 1;
```

Question 90. Top artist report **medium**

Instruction

- Write a query to return the top artist id for every country
- Return a unique row for each country
- For simplicity, let's assume there is no tie.
- The order of your results doesn't matter.

Solution

```
WITH artist_plays AS (
    SELECT
        S.artist_id,
        P.country,
        SUM(num_plays) num_plays
    FROM daily_plays P
    INNER JOIN song S
    ON S.song_id = P.song_id
    WHERE P.date = CURRENT_DATE - 1
    GROUP BY 1,2
),

artist_ranking AS (
    SELECT
    artist_id,
    country,
    ROW_NUMBER() OVER(PARTITION BY country ORDER BY num_plays DESC) ranking
FROM artist_plays
)

SELECT country, artist_id
FROM artist_ranking
WHERE ranking = 1;
```

RECAP

CONGRATULATIONS!

You've finished **90 SQL questions** from **Single table operations**

1. SELECT, WHERE, ORDER BY, LIMIT;
2. COUNT, GROUP BY HAVING;
3. IN, BETWEEN, LIKE, CASE WHEN

To **multi-table operations** including

1. INNER JOIN;
2. LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN;
3. UNION, UNION ALL

And **advanced Window functions**:

1. AVG, MIN/MAX, COUNT, SUM;
2. ROW_NUMBER, RANK, DENSE_RANK;
3. NTILE;
4. LAG, LEAD

But what's more important, you've finished **this EBook! That is incredible!**

And if you are in a good mood 😁, can I ask you for a favor?

I need to collect testimonials for this book, could you please take a minute and write about your experience on Twitter?

Please make sure to add the **#CrackSQLInterviewBook** hashtag and mention me @**DataLeonWei** so I will be notified. Share on Twitter

I hope you enjoyed this book, and good luck at your work, job hunting and your new skills in SQL! If you have any questions, please feel free to drop me a line hello@sqlpad.io

March 2021

Leon Wei