Member-only story

MySQL: How to Write a Query That Returns the Top Records in a Group

Casey McMullen · Follow
Published in Towards Data Science
6 min read · Feb 10, 2019

Listen Share More

This article will show you a simple query example in MySQL 5.7 (along with an example using the ***rank()*** *function in MySQL 8.0) that will return the top 3 orders per month out of an orders table.*

If you've ever wanted to write a query that returns the top ***n*** number of records out of a group or category, you've come to the right place. Over time I've needed this type of query every once in a while, and I always wind up with either an overly complex multi-query union effort, or just iterate through a result set in code. Both methods are highly inefficient and (now in retrospect) silly.

In my quest to learn new database query techniques I've come across the rank() function in MySQL 8.0 which makes this effort incredibly simple. But there's an equally simple technique you can use in MySQL 5.7 and earlier that provides a solution to this common conundrum.

Let's jump right into the example.

The Sample Table

First we're going to build a sample database table for our example. It's a simple orders table that spans three different customers over four months. It includes an ordered GUID as the primary key and contains an order number, customer number, customer name, order date and order amount.

We'll use this same table in both of our MySQL version examples:

```
CREATE TABLE orders(
  id BINARY(16),
  order_number INT,
  customer_number INT,
  customer_name VARCHAR(90),
  order_date DATE,
  order_amount DECIMAL(13,2),
  PRIMARY KEY (`id`)
);

INSERT INTO orders VALUES
  (UNHEX('11E92BDEA738CEB7B78E0242AC110002'), 100, 5001, 'Wayne
Enterprises', '2018-11-14', 100.00),
  (UNHEX('11E92BDEA73910BBB78E0242AC110002'), 101, 6002, 'Star
Labs', '2018-11-15', 200.00),
  (UNHEX('11E92BDEA7395C95B78E0242AC110002'), 102, 7003, 'Daily
Planet', '2018-11-15', 150.00),
  (UNHEX('11E92BDEA739A057B78E0242AC110002'), 103, 5001, 'Wayne
Enterprises', '2018-11-21', 110.00),
  (UNHEX('11E92BDEA739F892B78E0242AC110002'), 104, 6002, 'Star
Labs', '2018-11-22', 175.00),
  (UNHEX('11E92BE00BADD97CB78E0242AC110002'), 105, 6002, 'Star
Labs', '2018-11-23', 117.00),
  (UNHEX('11E92BE00BAE15ACB78E0242AC110002'), 106, 7003, 'Daily
Planet', '2018-11-24', 255.00),
  (UNHEX('11E92BE00BAE59FEB78E0242AC110002'), 107, 5001, 'Wayne
Enterprises', '2018-12-07', 321.00),
  (UNHEX('11E92BE00BAE9D7EB78E0242AC110002'), 108, 6002, 'Star
Labs', '2018-12-14', 55.00),
  (UNHEX('11E92BE00BAED1A4B78E0242AC110002'), 109, 7003, 'Daily
Planet', '2018-12-15', 127.00),
  (UNHEX('11E92BE021E2DF22B78E0242AC110002'), 110, 6002, 'Star
Labs', '2018-12-15', 133.00),
  (UNHEX('11E92BE021E31638B78E0242AC110002'), 111, 5001, 'Wayne
Enterprises', '2018-12-17', 145.00),
  (UNHEX('11E92BE021E35474B78E0242AC110002'), 112, 7003, 'Daily
Planet', '2018-12-21', 111.00),
  (UNHEX('11E92BE021E39950B78E0242AC110002'), 113, 6002, 'Star
Labs', '2018-12-31', 321.00),
  (UNHEX('11E92BE021E3CEC5B78E0242AC110002'), 114, 6002, 'Star
Labs', '2019-01-03', 223.00),
  (UNHEX('11E92BE035EF4BE5B78E0242AC110002'), 115, 6002, 'Star
Labs', '2019-01-05', 179.00),
  (UNHEX('11E92BE035EF970DB78E0242AC110002'), 116, 5001, 'Wayne
Enterprises', '2019-01-14', 180.00),
  (UNHEX('11E92BE035EFD540B78E0242AC110002'), 117, 7003, 'Daily
Planet', '2019-01-21', 162.00),
  (UNHEX('11E92BE035F01B8AB78E0242AC110002'), 118, 5001, 'Wayne
Enterprises', '2019-02-02', 133.00),
  (UNHEX('11E92BE035F05EF0B78E0242AC110002'), 119, 7003, 'Daily
Planet', '2019-02-05', 55.00),
  (UNHEX('11E92BE0480B3CBAB78E0242AC110002'), 120, 5001, 'Wayne
Enterprises', '2019-02-08', 25.00),
  (UNHEX('11E92BE25A9A3D6DB78E0242AC110002'), 121, 6002, 'Star
Labs', '2019-02-08', 222.00);
```

The MySQL 5.7 Example

The rank() function is pretty cool, but it's not available prior to MySQL 8.0. Therefore we'll need to write a creative nested query to rank our records and provide the results.

We're going to start by writing a query that ranks all of the records in our table in order of year, month and order amount in descending sequence (so that the largest orders get the lowest scores).

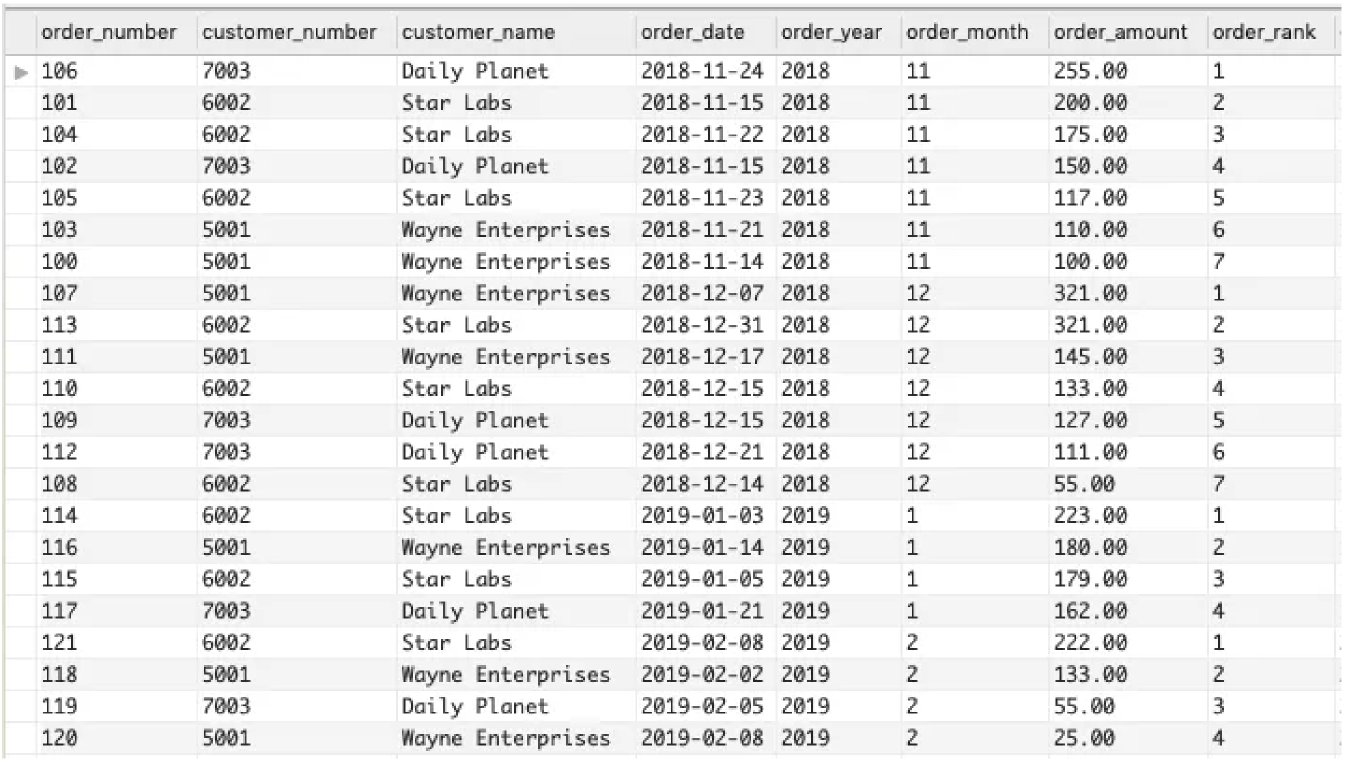
```
SELECT order_number, customer_number, customer_name, order_date,
  YEAR(order_date) AS order_year,
  MONTH(order_date) AS order_month,
  order_amount,
  @order_rank := IF(@current_month = MONTH(order_date),
  @order_rank + 1, 1) AS order_rank,
  @current_month := MONTH(order_date)
FROM orders
ORDER BY order_year, order_month, order_amount DESC;
```

In our example SELECT statement we're getting all the fields from the table along with getting the YEAR from the order date as well as the MONTH. Since our goal is to rank the orders by month I'm creating a temporary MySQL variable called @current_month that will keep track of each month. On every change of month we reset the @order_rank variable to one, otherwise we increment by one.

** Note ** using the := operand allows us to create a variable on the fly without requiring the SET command.

** 2nd Note ** keep in mind this SELECT statement will rank all of the records in our table. Normally you'd want to have a WHERE clause that limits the size of the result set. Perhaps by customer or date range.

The query above produces a result set that looks like this:



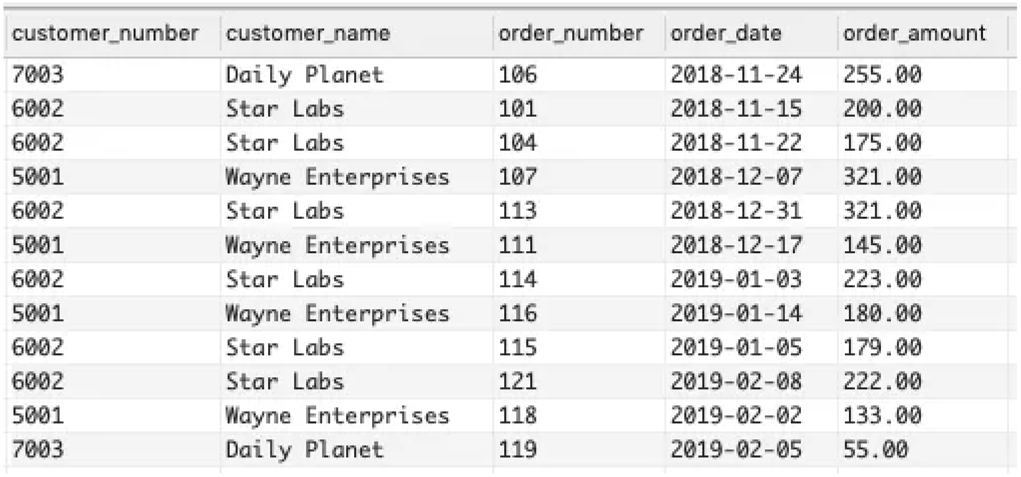
order_number	customer_number	customer_name	order_date	order_year	order_month	order_amount	order_rank
106	7003	Daily Planet	2018-11-24	2018	11	255.00	1
101	6002	Star Labs	2018-11-15	2018	11	200.00	2
104	6002	Star Labs	2018-11-22	2018	11	175.00	3
102	7003	Daily Planet	2018-11-15	2018	11	150.00	4
105	6002	Star Labs	2018-11-23	2018	11	117.00	5
103	5001	Wayne Enterprises	2018-11-21	2018	11	110.00	6
100	5001	Wayne Enterprises	2018-11-14	2018	11	100.00	7
107	5001	Wayne Enterprises	2018-12-07	2018	12	321.00	1
113	6002	Star Labs	2018-12-31	2018	12	321.00	2
111	5001	Wayne Enterprises	2018-12-17	2018	12	145.00	3
110	6002	Star Labs	2018-12-15	2018	12	133.00	4
109	7003	Daily Planet	2018-12-15	2018	12	127.00	5
112	7003	Daily Planet	2018-12-21	2018	12	111.00	6
108	6002	Star Labs	2018-12-14	2018	12	55.00	7
114	6002	Star Labs	2019-01-03	2019	1	223.00	1
116	5001	Wayne Enterprises	2019-01-14	2019	1	180.00	2
115	6002	Star Labs	2019-01-05	2019	1	179.00	3
117	7003	Daily Planet	2019-01-21	2019	1	162.00	4
121	6002	Star Labs	2019-02-08	2019	2	222.00	1
118	5001	Wayne Enterprises	2019-02-02	2019	2	133.00	2
119	7003	Daily Planet	2019-02-05	2019	2	55.00	3
120	5001	Wayne Enterprises	2019-02-08	2019	2	25.00	4

You can see that the orders are sorted by year and month and then by order amount in descending sequence. The new order_rank column is included that ranks every order in 1–2–3 sequence by month.

Now we can include this query as a subquery to a SELECT that only pulls the top 3 orders out of every group. That final query looks like this:

```
SELECT customer_number, customer_name, order_number, order_date,
order_amount
FROM
  (SELECT order_number, customer_number, customer_name, order_date,
    YEAR(order_date) AS order_year,
    MONTH(order_date) AS order_month,
    order_amount,
    @order_rank := IF(@current_month = MONTH(order_date),
    @order_rank + 1, 1) AS order_rank,
    @current_month := MONTH(order_date)
   FROM orders
   ORDER BY order_year, order_month, order_amount DESC)
ranked_orders
WHERE order_rank <= 3;
```

With our ranking query as the subquery, we only need to pull out the final fields that we need for reporting. A WHERE clause is added that only pulls records with a rank of 3 or less. Our final result set is shown below:



customer_number	customer_name	order_number	order_date	order_amount
7003	Daily Planet	106	2018-11-24	255.00
6002	Star Labs	101	2018-11-15	200.00
6002	Star Labs	104	2018-11-22	175.00
5001	Wayne Enterprises	107	2018-12-07	321.00
6002	Star Labs	113	2018-12-31	321.00
5001	Wayne Enterprises	111	2018-12-17	145.00
6002	Star Labs	114	2019-01-03	223.00
5001	Wayne Enterprises	116	2019-01-14	180.00
6002	Star Labs	115	2019-01-05	179.00
6002	Star Labs	121	2019-02-08	222.00
5001	Wayne Enterprises	118	2019-02-02	133.00
7003	Daily Planet	119	2019-02-05	55.00

You can see in the results that we got the top 3 orders out of every month.

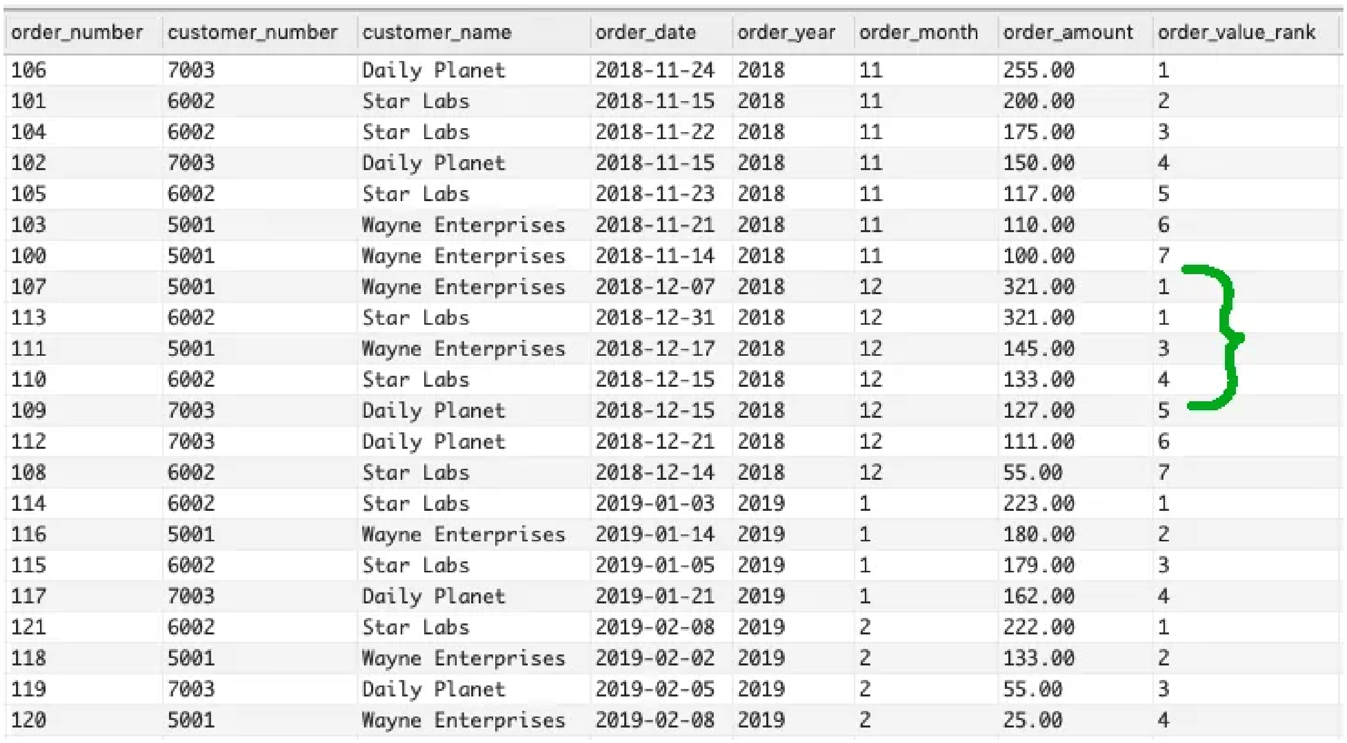
The MySQL 8.0 Example

MySQL 8.0 introduces a rank() function that adds some additional functionality for ranking records in a result set. With the rank() function the result set is partitioned by a value that you specify, then a rank is assigned to each row within each partition. Ties are given the same rank and the subsequent new number is given a rank of one plus the number of ranked records before it.

Our ranking query with this new feature looks like this:

```
SELECT order_number, customer_number, customer_name, order_date,
 YEAR(order_date) AS order_year,
 MONTH(order_date) AS order_month,
 order_amount,
 RANK() OVER (
 PARTITION BY YEAR(order_date), MONTH(order_date)
ORDER BY YEAR(order_date), MONTH(order_date), order_amount DESC)
order_value_rank
FROM orders;
```

This produces a result that looks like the following:



order_number	customer_number	customer_name	order_date	order_year	order_month	order_amount	order_value_rank
106	7003	Daily Planet	2018-11-24	2018	11	255.00	1
101	6002	Star Labs	2018-11-15	2018	11	200.00	2
104	6002	Star Labs	2018-11-22	2018	11	175.00	3
102	7003	Daily Planet	2018-11-15	2018	11	150.00	4
105	6002	Star Labs	2018-11-23	2018	11	117.00	5
103	5001	Wayne Enterprises	2018-11-21	2018	11	110.00	6
100	5001	Wayne Enterprises	2018-11-14	2018	11	100.00	7
107	5001	Wayne Enterprises	2018-12-07	2018	12	321.00	1
113	6002	Star Labs	2018-12-31	2018	12	321.00	1
111	5001	Wayne Enterprises	2018-12-17	2018	12	145.00	3
110	6002	Star Labs	2018-12-15	2018	12	133.00	4
109	7003	Daily Planet	2018-12-15	2018	12	127.00	5
112	7003	Daily Planet	2018-12-21	2018	12	111.00	6
108	6002	Star Labs	2018-12-14	2018	12	55.00	7
114	6002	Star Labs	2019-01-03	2019	1	223.00	1
116	5001	Wayne Enterprises	2019-01-14	2019	1	180.00	2
115	6002	Star Labs	2019-01-05	2019	1	179.00	3
117	7003	Daily Planet	2019-01-21	2019	1	162.00	4
121	6002	Star Labs	2019-02-08	2019	2	222.00	1
118	5001	Wayne Enterprises	2019-02-02	2019	2	133.00	2
119	7003	Daily Planet	2019-02-05	2019	2	55.00	3
120	5001	Wayne Enterprises	2019-02-08	2019	2	25.00	4

In this example you can see that in December the two greatest orders had the same amount of $321.00. The rank() function gives these two records the same rank of 1 and the subsequent record gets a rank of 3 and so on.

Like before, this ranking query is used as a subquery for our final query:

```
WITH ranked_orders AS (
 SELECT order_number, customer_number, customer_name, order_date,
 YEAR(order_date) AS order_year,
 MONTH(order_date) AS order_month,
 order_amount,
 RANK() OVER (
 PARTITION BY YEAR(order_date), MONTH(order_date)
 ORDER BY YEAR(order_date),
 MONTH(order_date), order_amount DESC) order_rank
 FROM orders
)
SELECT customer_number, customer_name, order_number, order_date,
 order_amount
FROM ranked_orders
WHERE order_rank <= 3;
```

The final query is very similar to our MySQL 5.7 example, but uses some MySQL 8.0 goodness (like the availability of the WITH statement) along with more ranking capabilities that you can research in the MySQL 8.0 documentation. The final results to this query are identical to our MySQL 5.7 results in the example above.

I hope these two examples gives you some help the next time you're wanting to get the top results out of a group.

There's more!

Make sure to check out my other articles here to learn more about setting up your PHP development environment on your Mac and other coding tips and examples.

MySQL Mysql 8

Follow

Open in app ↗

More from Casey McMullen and Towards Data Science

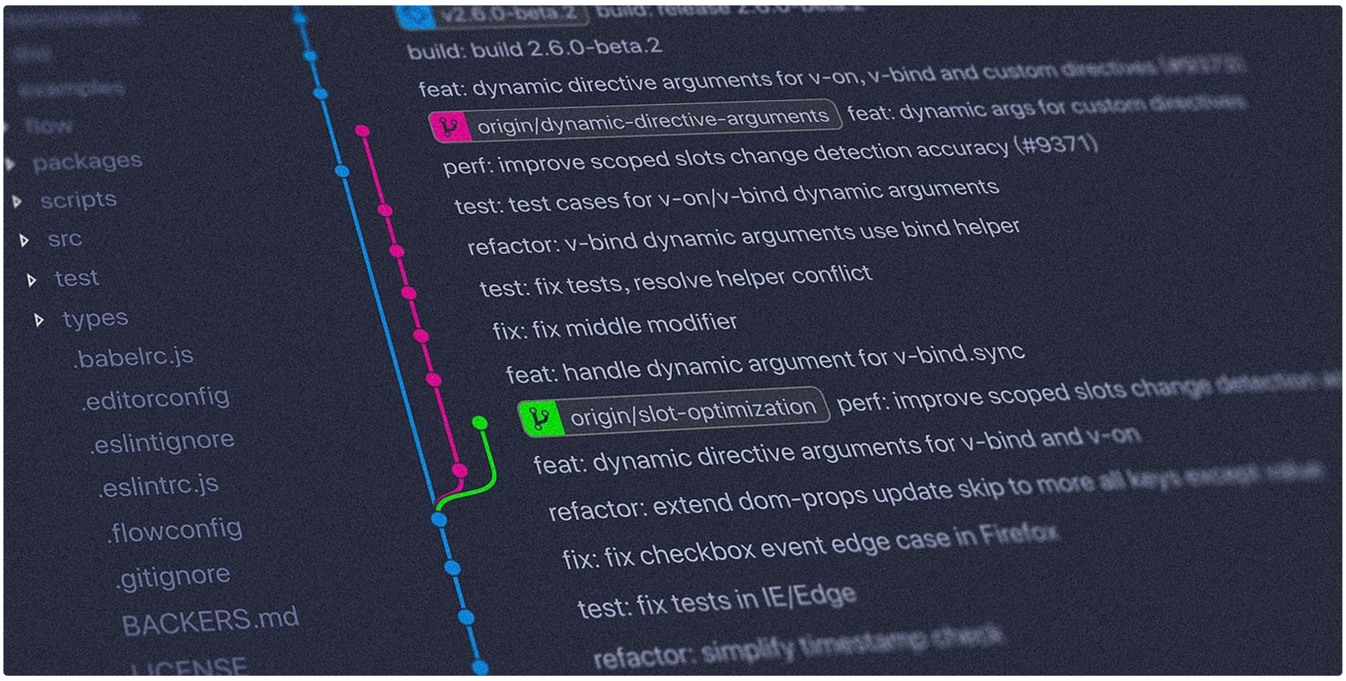
Casey McMullen

How to Run MySQL 8.0 with Native Password Authentication

This article will show you the steps to run MySQL 8.0 in your macOS development environment with mysql_native_password rather than…

✦ · 4 min read · Dec 21, 2018

165 1



```
v2.6.0-beta.2  build: release 2.6.0-beta.2
build: build 2.6.0-beta.2
feat: dynamic directive arguments for v-on, v-bind and custom directives (#9373)
origin/dynamic-directive-arguments  feat: dynamic args for custom directives
perf: improve scoped slots change detection accuracy (#9371)
test: test cases for v-on/v-bind dynamic arguments
refactor: v-bind dynamic arguments use bind helper
test: fix tests, resolve helper conflict
fix: fix middle modifier
feat: handle dynamic argument for v-bind.sync
origin/slot-optimization  perf: improve scoped slots change detection ac
feat: dynamic directive arguments for v-bind and v-on
refactor: extend dom-props update skip to more all keys except value
fix: fix checkbox event edge case in Firefox
test: fix tests in IE/Edge
refactor: simplify timestamp check
```

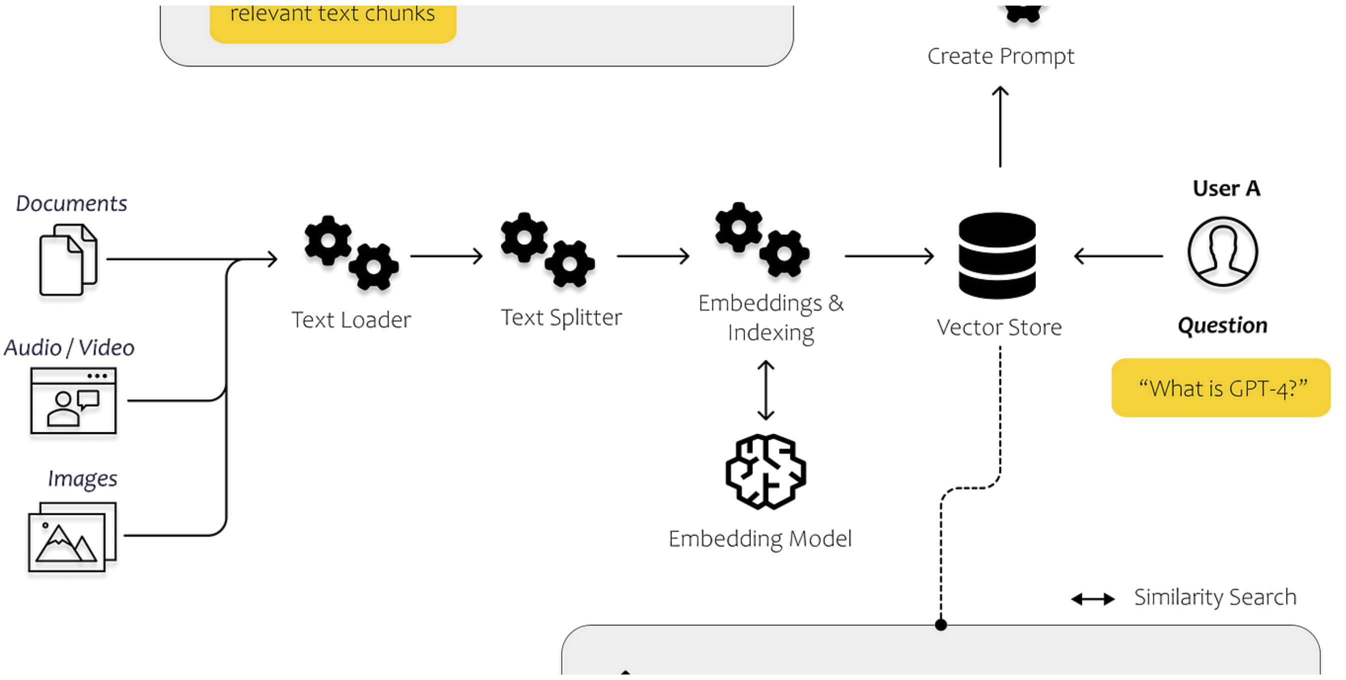
Miriam Santos in Towards Data Science

Pandas 2.0: A Game-Changer for Data Scientists?

The Top 5 Features for Efficient Data Manipulation

7 min read · Jun 27

1.3K 14



Dominik Polzer in Towards Data Science

All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt templates

✦ · 25 min read · Jun 22

1.6K 19

Casey McMullen

How to Run MySQL in a Docker Container on macOS with Persistent Local Data

This article is an easy step-by-step guide to setup MySQL containers for your macOS development environment and have the data remain on...

✦ · 5 min read · Dec 21, 2018

375 11

See all from Casey McMullen

See all from Towards Data Science

Recommended from Medium

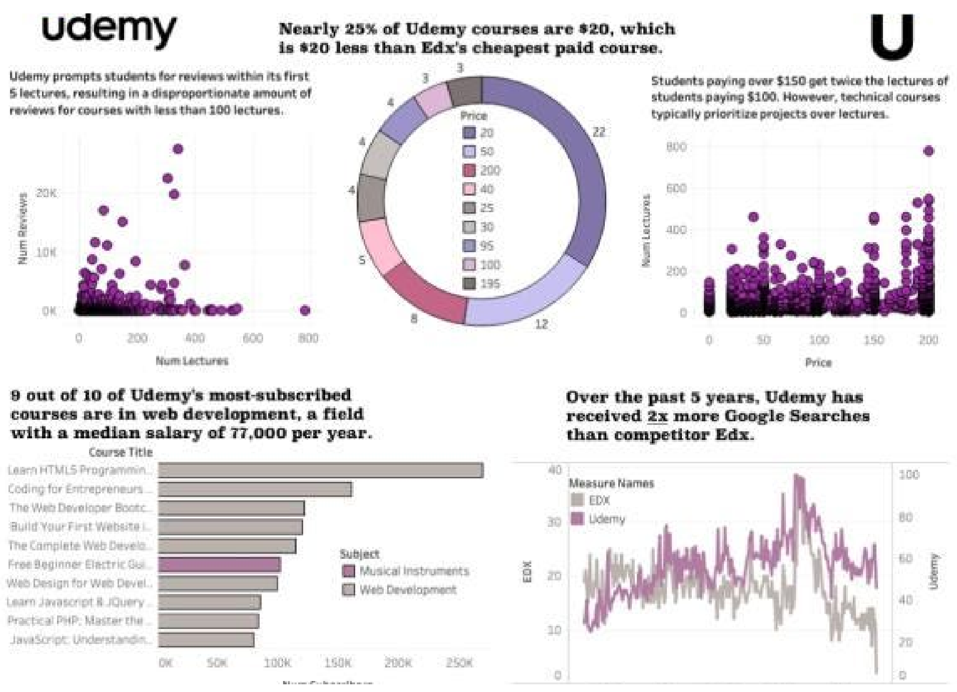
Youssef Hosni in Level Up Coding

13 SQL Statements for 90% of Your Data Science Tasks

Structured Query Language (SQL) is a programming language designed for managing and manipulating relational databases. It is widely used by...

· 15 min read · Feb 27

2.9K 33



Zach Quinn in Pipeline: Your Data Engineering Resource

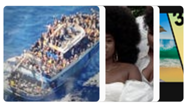
Creating The Dashboard That Got Me A Data Analyst Job Offer

A walkthrough of the Udemy dashboard that got me a job offer from one of the biggest names in academic publishing.

· 9 min read · Dec 5, 2022

1.2K   20

Lists

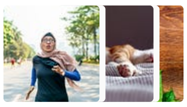


Staff Picks

387 stories · 146 saves

Stories to Help You Level-Up at Work

19 stories · 133 saves



Self-Improvement 101

20 stories · 239 saves

Productivity 101

20 stories · 254 saves

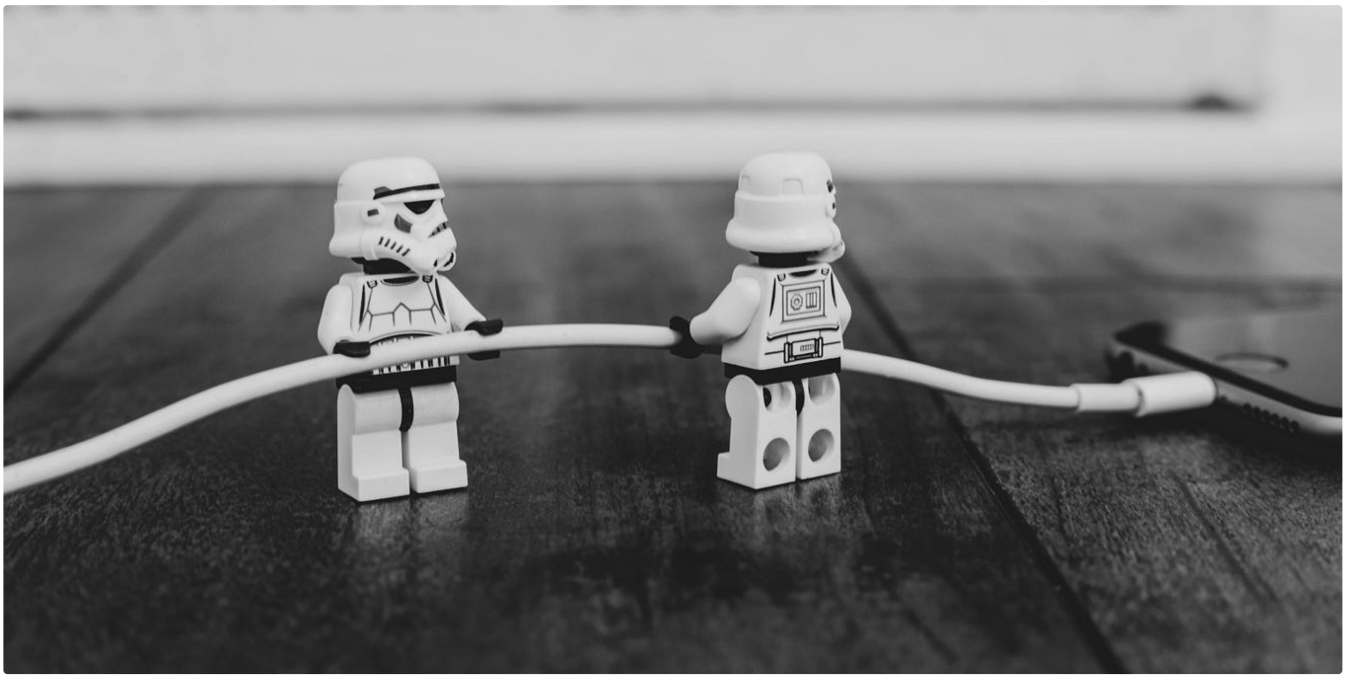
Axel Thevenot in Google Cloud - Community

BigQuery WINDOW Functions | Advanced Techniques for Data Professionals

A complete guide for maximizing the potential of BigQuery WINDOW functions to manipulate and transform data.

· 13 min read · Jan 9

170 2



Giorgos Myrianthous in Towards Data Science

ETL vs ELT: What’s the Difference?

A comparison between ETL and ELT in the context of Data Engineering

· 8 min read · Jan 18

160 2

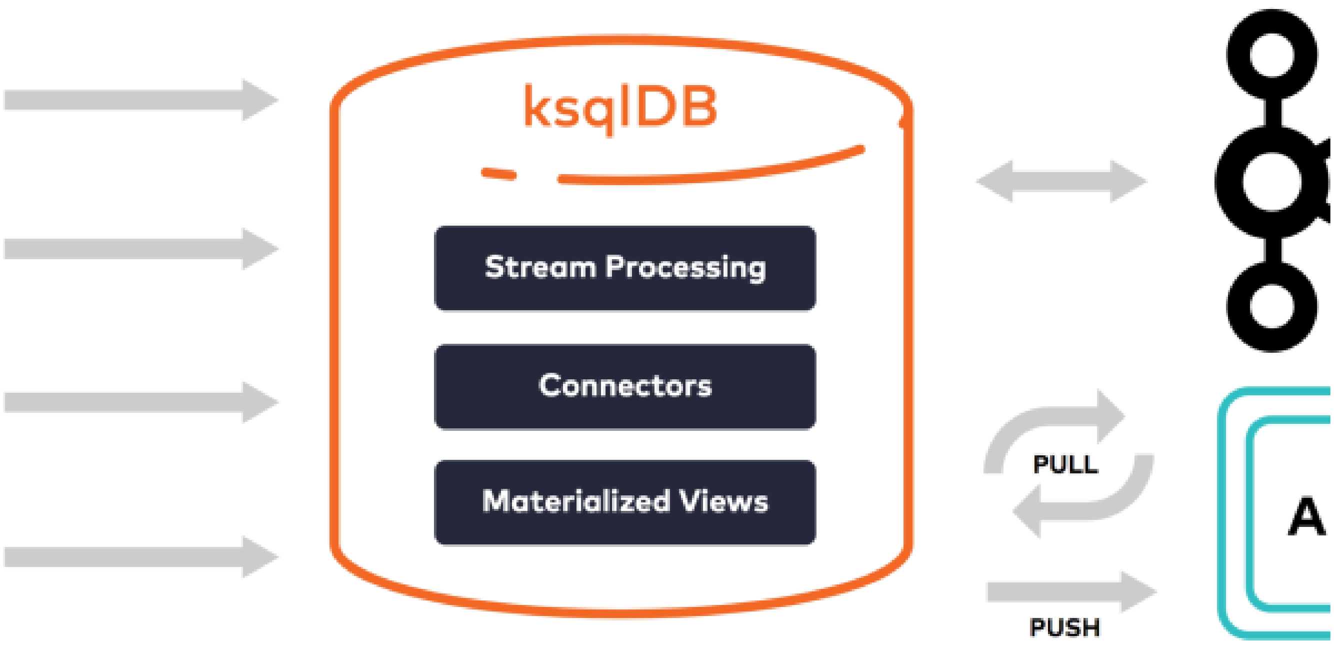
Ansam Yousry

SQL Window Functions With Examples

In SQL, a window function (also called an “over clause”) allows you to perform calculations on a set of rows that are related to the...

· 4 min read · Jan 18

5



Thomas Reid in Level Up Coding

Streaming change data capture (CDC) data between databases using Kafka

This article will briefly introduce Kafka, how to connect database sources to it using the Kafka SQL client ksqlDB and create and...

· 21 min read · Mar 27

61

See more recommendations