Easy and Intelligent Pivot Tables with DuckDB

Josef Machytka

3 min read · Just now

After exploring the various capabilities of DuckDB in my earlier articles, I want to focus more on its powerful data analytical functionality. In this article, I will delve into the PIVOT command, an amazing powerful tool for creating pivot tables, even from complex datasets.

Pivot tables are a staple for data analysis, but they can be tedious to set up in some systems. DuckDB makes this process both simple and efficient. Let me demonstrate it with a classic "sales" example, inspired by a testing scenario I explored using ChatGPT.

The Example Data

I will use a `sales` table that tracks sales figures for various products, salespeople, and years. Here's the table structure and some sample data:

```
CREATE TABLE sales (
    id SERIAL PRIMARY KEY,
    salesperson VARCHAR(50),
    product VARCHAR(50),
    year INT,
    sales_amount NUMERIC
);

INSERT INTO sales (salesperson, product, year, sales_amount) VALUES
('Alice', 'Laptop', 2022, 1200),
('Alice', 'Phone', 2022, 800),
('Alice', 'Tablet', 2022, 300),
('Alice', 'Laptop', 2023, 1400),
('Alice', 'Phone', 2023, 900),
('Alice', 'Tablet', 2023, 400),
('Bob', 'Laptop', 2022, 1000),
('Bob', 'Phone', 2022, 600),
('Bob', 'Laptop', 2023, 1100),
('Charlie', 'Tablet', 2022, 500),
('Charlie', 'Laptop', 2022, 1100),
('Charlie', 'Phone', 2022, 700),
('Charlie', 'Tablet', 2023, 600),
('Charlie', 'Laptop', 2023, 1200),
('Charlie', 'Phone', 2023, 800);
```

Pivoting in PostgreSQL: The Hard Way

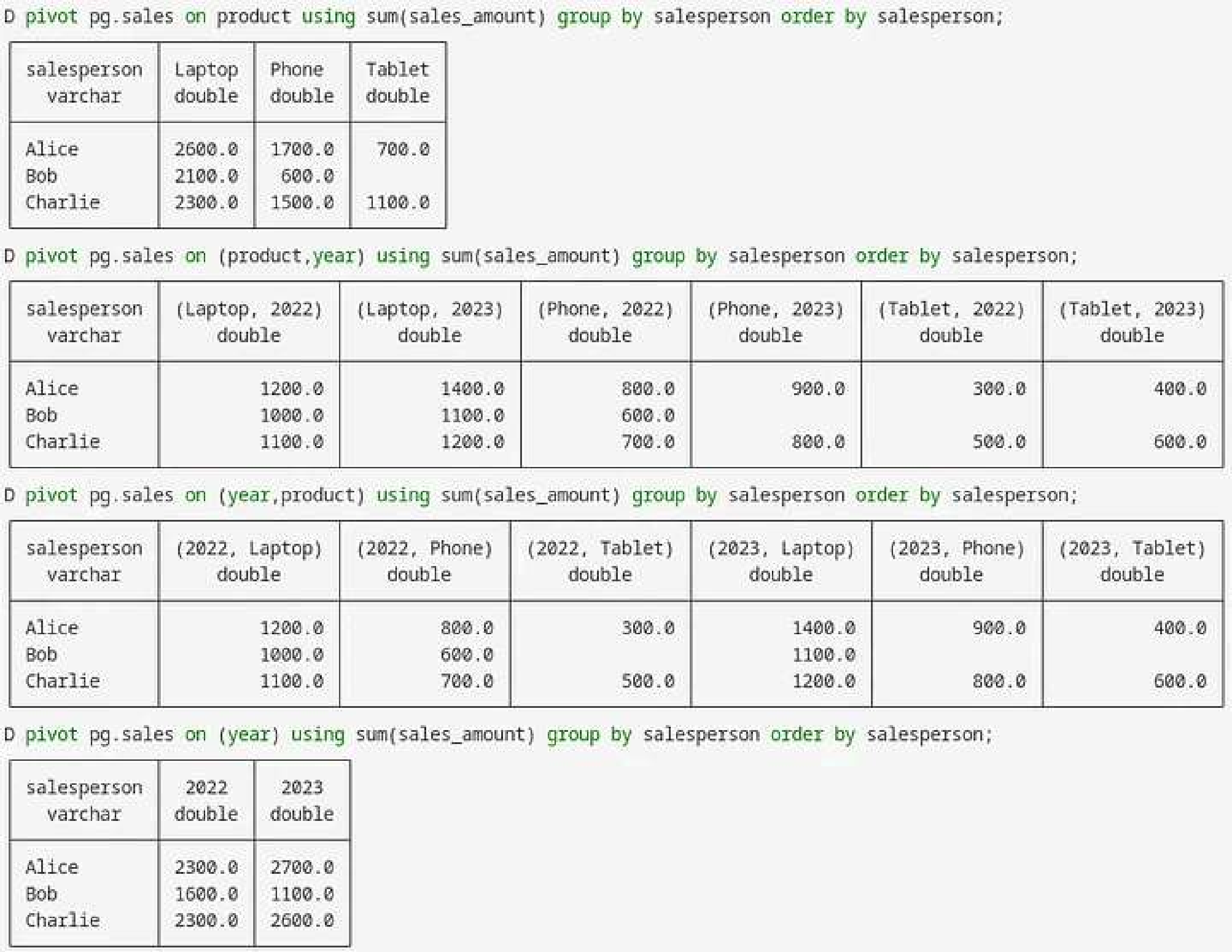
Creating a pivot table in standard PostgreSQL is of course perfectly possible but involves a fair amount of manual work. For example, let’s pivot the `sales` table to display sales amounts for each salesperson by product and year. Here’s the query:

```
SELECT
    salesperson,
    SUM(sales_amount) FILTER (WHERE product = 'Laptop' AND year = 2022) AS Lapto
    SUM(sales_amount) FILTER (WHERE product = 'Phone' AND year = 2022) AS Phone_
    SUM(sales_amount) FILTER (WHERE product = 'Tablet' AND year = 2022) AS Table
    SUM(sales_amount) FILTER (WHERE product = 'Laptop' AND year = 2023) AS Lapto
    SUM(sales_amount) FILTER (WHERE product = 'Phone' AND year = 2023) AS Phone_
    SUM(sales_amount) FILTER (WHERE product = 'Tablet' AND year = 2023) AS Table
    FROM
        sales
    GROUP BY
        salesperson
    ORDER BY
        salesperson;
```

This works but requires explicitly specifying every product and year combination in the query. If people later add new products or years, we must rewrite the query to account for those changes. Clearly, this approach doesn't scale well and involves too much manual effort.

Pivoting in DuckDB: Let's Make Pivot Fun Again

DuckDB offers a simpler and more flexible solution with its internal **PIVOT** command. By attaching a remote PostgreSQL database, we can process the same data in a far more elegant way. Adjusting pivot criteria is effortless, making it an incredibly user-friendly tool for dynamic data exploration.



```
D pivot pg.sales on product using sum(sales_amount) group by salesperson order by salesperson;
┌─────────────┬────────┬────────┬────────┐
│ salesperson │ Laptop │ Phone  │ Tablet │
│   varchar   │ double │ double │ double │
├─────────────┼────────┼────────┼────────┤
│ Alice       │ 2600.0 │ 1700.0 │  700.0 │
│ Bob         │ 2100.0 │  600.0 │        │
│ Charlie     │ 2300.0 │ 1500.0 │ 1100.0 │
└─────────────┴────────┴────────┴────────┘
D pivot pg.sales on (product,year) using sum(sales_amount) group by salesperson order by salesperson;
┌─────────────┬────────────────┬────────────────┬───────────────┬───────────────┬────────────────┬────────────────┐
│ salesperson │ (Laptop, 2022) │ (Laptop, 2023) │ (Phone, 2022) │ (Phone, 2023) │ (Tablet, 2022) │ (Tablet, 2023) │
│   varchar   │     double     │     double     │    double     │    double     │     double     │     double     │
├─────────────┼────────────────┼────────────────┼───────────────┼───────────────┼────────────────┼────────────────┤
│ Alice       │         1200.0 │         1400.0 │         800.0 │         900.0 │          300.0 │          400.0 │
│ Bob         │         1000.0 │         1100.0 │         600.0 │               │                │                │
│ Charlie     │         1100.0 │         1200.0 │         700.0 │         800.0 │          500.0 │          600.0 │
└─────────────┴────────────────┴────────────────┴───────────────┴───────────────┴────────────────┴────────────────┘
D pivot pg.sales on (year,product) using sum(sales_amount) group by salesperson order by salesperson;
┌─────────────┬────────────────┬───────────────┬────────────────┬────────────────┬───────────────┬────────────────┐
│ salesperson │ (2022, Laptop) │ (2022, Phone) │ (2022, Tablet) │ (2023, Laptop) │ (2023, Phone) │ (2023, Tablet) │
│   varchar   │     double     │    double     │     double     │     double     │    double     │     double     │
├─────────────┼────────────────┼───────────────┼────────────────┼────────────────┼───────────────┼────────────────┤
│ Alice       │         1200.0 │         800.0 │          300.0 │         1400.0 │         900.0 │          400.0 │
│ Bob         │         1000.0 │         600.0 │                │         1100.0 │               │                │
│ Charlie     │         1100.0 │         700.0 │          500.0 │         1200.0 │         800.0 │          600.0 │
└─────────────┴────────────────┴───────────────┴────────────────┴────────────────┴───────────────┴────────────────┘
D pivot pg.sales on (year) using sum(sales_amount) group by salesperson order by salesperson;
┌─────────────┬────────┬────────┐
│ salesperson │  2022  │  2023  │
│   varchar   │ double │ double │
├─────────────┼────────┼────────┤
│ Alice       │ 2300.0 │ 2700.0 │
│ Bob         │ 1600.0 │ 1100.0 │
│ Charlie     │ 2300.0 │ 2600.0 │
└─────────────┴────────┴────────┘
```

Conclusion

DuckDB transforms the process of creating pivot tables into a seamless experience, saving us both time and effort. As shown in these examples, its built-in capabilities outperform traditional SQL methods for pivoting. Whether we are dealing with a simple dataset or a complex one, DuckDB makes data analysis not just easier but also more enjoyable.

Duckdb Pivot Tables Pivoting Postgresql

Written by Josef Machytka

Edit profile

16 Followers · 3 Following

I work as Professional Service Consultant - PostgreSQL specialist in NetApp Deutschland GmbH, Open Source Services division.

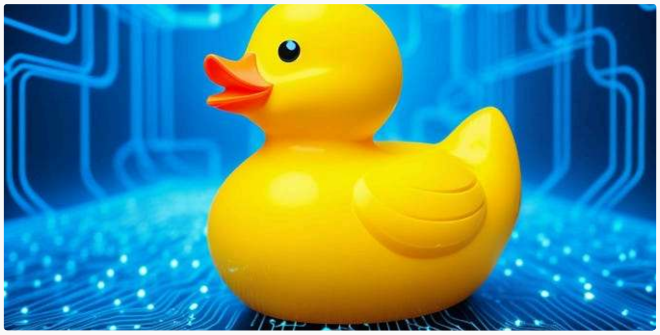
More from Josef Machytka

Josef Machytka

How DuckDB handles data not fitting into memory?

In my previous article about DuckDB I described how to use this database as an…

Nov 13 👏 7

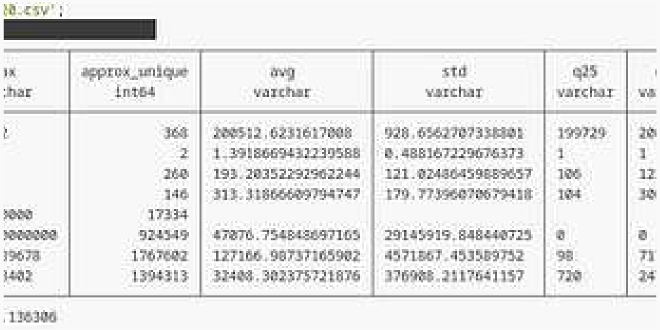


Josef Machytka

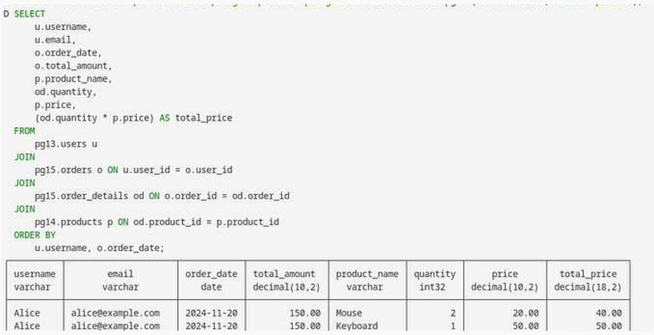
Extending DuckDB ETL Capabilities with Python

DuckDB has recently become my go-to solution for small ETL tasks. It is an…

Nov 24 👏 12



```
0.csv';
[redacted]
 x    │ approx_unique │         avg          │         std          │  q25   │ 
char  │     int64     │       varchar        │       varchar        │ varchar│ va
──────┼───────────────┼──────────────────────┼──────────────────────┼────────┼───
2     │           360 │ 200512.6231617008    │ 928.6562707338001    │ 199729 │ 20
      │             2 │ 1.3918669432239588   │ 0.488167229676373    │ 1      │ 1
      │           260 │ 193.20352292962244   │ 121.02480459889657   │ 106    │ 12
      │           146 │ 313.31866609794747   │ 179.77396070679418   │ 104    │ 30
0000  │         17334 │                      │                      │        │
0000000│       924349 │ 47876.754848697165   │ 29145919.848440725   │ 0      │ 0
79678 │       1767602 │ 127166.98737165902   │ 4571867.453589752    │ 98     │ 71
3402  │       1394313 │ 32408.302375721876   │ 376988.2117641157    │ 720    │ 24
.136306
```



```
D SELECT
      u.username,
      u.email,
      o.order_date,
      o.total_amount,
      p.product_name,
      od.quantity,
      p.price,
      (od.quantity * p.price) AS total_price
  FROM
      pg13.users u
  JOIN
      pg15.orders o ON u.user_id = o.user_id
  JOIN
      pg15.order_details od ON o.order_id = od.order_id
  JOIN
      pg14.products p ON od.product_id = p.product_id
  ORDER BY
      u.username, o.order_date;
┌──────────┬───────────────────┬────────────┬───────────────┬──────────────┬──────────┬───────────────┬───────────────┐
│ username │       email       │ order_date │ total_amount  │ product_name │ quantity │     price     │  total_price  │
│ varchar  │      varchar      │    date    │ decimal(10,2) │   varchar    │  int32   │ decimal(10,2) │ decimal(18,2) │
├──────────┼───────────────────┼────────────┼───────────────┼──────────────┼──────────┼───────────────┼───────────────┤
│ Alice    │ alice@example.com │ 2024-11-20 │        150.00 │ Mouse        │        2 │         20.00 │         40.00 │
│ Alice    │ alice@example.com │ 2024-11-20 │        150.00 │ Keyboard     │        1 │         50.00 │         50.00 │
```

Josef Machytka

Using DuckDB as an Intelligent ETL tool for PostgreSQL

There is a lot of hype around DuckDB these days. At one PostgreSQL conference, I even…

Nov 2 👏 6

Josef Machytka

Easy Cross-Database Selects with DuckDB

DuckDB was created with simplicity and ease of use in mind. In my previous article, I…

Nov 26 👏 2

See all from Josef Machytka

Recommended from Medium

In T3CH by Jesus L. Monroy

Supercharge Your SQL Analysis with Python and DuckDB

Rethinking Data Analysis: A New Paradigm

5d ago 90

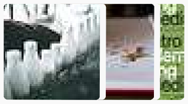
In Code Like A Girl by Chris Ng

I Removed VS Code, and My Productivity Improved…

A Journey from Clunky to Efficient: How The New IDE Transformed My Workflow

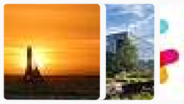
Nov 26 553 7

Lists



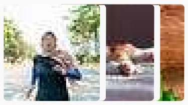
Staff picks

780 stories · 1488 saves



Stories to Help You Level-Up at Work

19 stories · 888 saves

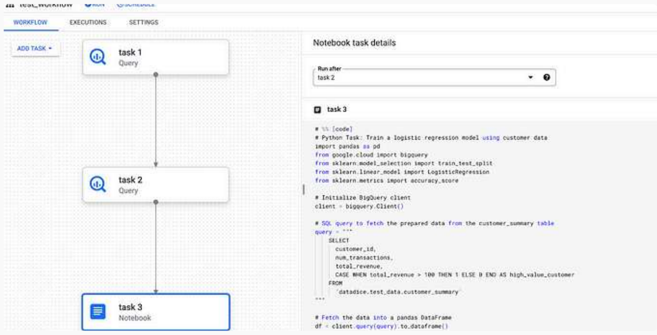


Self-Improvement 101

20 stories · 3114 saves

Productivity 101

20 stories · 2623 saves



```
# %% [code]
# Python Task: Train a logistic regression model using customer data
import pandas as pd
from google.cloud import bigquery
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Initialize BigQuery client
client = bigquery.Client()

# SQL query to fetch the prepared data from the customer_summary table
query = """
    SELECT
      customer_id,
      num_transactions,
      total_revenue,
      CASE WHEN total_revenue > 100 THEN 1 ELSE 0 END AS high_value_customer
    FROM
      `datadice.test_data.customer_summary`
"""

# Fetch the data into a pandas DataFrame
df = client.query(query).to_dataframe()
```

datadice

In Towards Data Science by Eirik Berge, PhD

Introducing BigQuery Workflows

A New Era of Data Orchestration. By Fassil S. Yehuala

Nov 27 108 •••

Use Tablib to Handle Simple Tabular Data in Python

Sometimes a Shallow Abstraction is more Valuable than Performance

✦ Nov 27 211 1 •••

In Level Up Coding by Md Arman Hossen

The Lazy Developer’s Guide to Python Automation (and Why…

It was another late night at the office. The soft glow of my monitor illuminated the scattere…

✦ Nov 7 251 1 •••

Thibaut Gourdel

Data Cleaning Tools in 2024

Data cleaning tools: Code-based (pandas, dbt), Open-source (Amphi, OpenRefine), and…

Nov 26 25 •••

See more recommendations