Database and SQL Course

Eric Hao

2024-01-09

Table of contents

Database and SQL course

Welcome to introduction to Database and SQL

Structure Query Language(SQL, usually pronounced S-Q-L or 'sequel') is a programming language designed for the data manipulation with database system, mostly with relational DBMS, like MySQL, PostgreSQL and Oracle

In this course, we will act as a data analyst for a fictional superstore. Our job is to transform raw data into information and clean it to the tabluar format, and do the data exploration with different business purpose, prepare the story-telling for the stakeholders.

In this folder, we have the two csv files, we will use python to create a basic sales database and use the SQL to query the data. And gradually we will advance to more complex queries with advance data anlysis skills like data wrangling, data filtering, sorting, grouping and statstical analysis.

By the end of this course, you will be able to write queries like the one shown, which can return a list of the sales data, can compare different store, different category goods, their sales volume and profits.

Don't worry if it seems daunting now; we'll master it step by step;

Notes: this course focus on the SQL, so some background knowledge on the jupyter notebook and pandas etc, please see the resource on the web.

Prepare the data

First we need import the necessary python package

```
import numpy as np
import pandas as pd
import duckdb
sql = lambda s: duckdb.sql(s)
```

Load the sales data

```
df1 = pd.read_csv('./data/orders.csv')
df2 = pd.read_csv('./data/details.csv')
sales = pd.merge(df1,df2,on='Order ID')
sales.head(3)
```

	Order ID	Order Date	CustomerName	State	City	Amount	Profit	Quantity	Categ
0	B-26055	10-03-2018	Harivansh	Uttar Pradesh	Mathura	5729	64	14	Furni
1	B-26055	10-03-2018	Harivansh	Uttar Pradesh	Mathura	671	114	9	Elect
2	B-26055	10-03-2018	Harivansh	Uttar Pradesh	Mathura	443	11	1	Cloth

Do some data type tranformation

Transform the column into the table compatible format

```
sales.columns = sales.columns.map(lambda c: '_'.join(c.lower().split(' ')))
sales.columns
```

```
Index(['order_id', 'order_date', 'customername', 'state', 'city', 'amount',
       'profit', 'quantity', 'category', 'sub-category', 'paymentmode'],
      dtype='object')
```

Convert the order date the pandas datetime data type

```
sales['order_date'] = pd.to_datetime(sales['order_date'],format='%d-%m-%Y')
```

```
sales.head(3)
```

	order_id	order_date	customername	state	city	amount	profit	quantity	category
0	B-26055	2018-03-10	Harivansh	Uttar Pradesh	Mathura	5729	64	14	Furnitur
1	B-26055	2018-03-10	Harivansh	Uttar Pradesh	Mathura	671	114	9	Electron
2	B-26055	2018-03-10	Harivansh	Uttar Pradesh	Mathura	443	11	1	Clothing

Query the data with the duckdb sql capability

```
sql('select * from sales limit 3')
```

```
┌──────────┬─────────────────────┬──────────────┬───┬──────────┬─────────────┬──────────────┬────┐
│ order_id │     order_date      │ customername │ … │ quantity │  category   │ sub-category │ pa │
│ varchar  │    timestamp_ns     │   varchar    │   │  int64   │   varchar   │   varchar    │    │
├──────────┼─────────────────────┼──────────────┼───┼──────────┼─────────────┼──────────────┼────┤
│ B-26055  │ 2018-03-10 00:00:00 │ Harivansh    │ … │       14 │ Furniture   │ Chairs       │ EM │
│ B-26055  │ 2018-03-10 00:00:00 │ Harivansh    │ … │        9 │ Electronics │ Phones       │ Cr │
│ B-26055  │ 2018-03-10 00:00:00 │ Harivansh    │ … │        1 │ Clothing    │ Saree        │ CO │
└──────────┴─────────────────────┴──────────────┴───┴──────────┴─────────────┴──────────────┴────┘
3 rows                                                                          11 colum
```

First SQL we try to say hello to database

Now the duckdb database has only one table sales, which contains information about the orders from the superstore stores, our first line of code is to retrieve the first 3 rows of this tables:

```
SELECT *
    FROM sales
    LIMIT 3;
```

Let's explain this code a little further:

- * is a special character meaning 'all the column of the table'. We will learn more about the SELECTclause later, but for now we can read this line as 'Select all columns'
- FROM clause tell database which table that we want to retrieve the data. We will learn more about how to identify the database tables and how to manage them.
- LIMIT 3 tell the database the result return from limit to 3 records. A table might have millions rows of data, attempting to return all the data could get messy.

Different SQL Dialects

There are many different versions of SQL(also called SQL flavors or diablects(. We have benn coding against DuckDB, which is complied to the ANSI SQL standard, other database like MySQL and Oracle Database also support the ANSI SQL standard, but they all have their own dialects to empower their capability, we need take an eyes on the difference when we work on the specific database engine.

Figure 1: sql engine

Database tables and fields

Most database we will work with are like superstore database has multiple tables of data. Like a sheet in Excel, a database table consists of rows and columns which are offen called records and fields.

Figure 2: database tables

First Example: Select all columns from sales table

```
SELECT *
    FROM sales
    LIMIT 3;
```

As we introduce the say hello to the database tables, we can select all the columns from the database tables;

But what about reducing the number of fields shown? As we learned earlier, we can retrieve the data from all data field in a table with help of the * wildcard. If we want to select the specific fields to be return by the query, we can list them in the SELECT clause, seperated by commas:

```
SELECT order_date, customername, category
    FROM sales
    LIMIT 5;
```

```
sql('select order_date, customername, category from sales limit 5')
```

order_date timestamp_ns	customername varchar	category varchar
2018-03-10 00:00:00	Harivansh	Furniture
2018-03-10 00:00:00	Harivansh	Electronics
2018-03-10 00:00:00	Harivansh	Clothing
2018-03-10 00:00:00	Harivansh	Clothing
2018-03-10 00:00:00	Harivansh	Clothing

Exploring the tables and columns

For consistency and effieciency, SQL Database only allow one data type per column. That means integer columns can only store interger data and text columns can only store string data and so on.

Formally, we call these types of data as data type or storage class, when you design the database, you must be carfully select the data type, different data type has different storage requirements and support different computation purpose, for example, numberic data type can be compared and can be add and subtract and string data type can repsent the person name and address, and you need some information of the data type and their specific requirements on the database designation, this course focus on the sql query, please google some more resource on the database design.

Generally, Database all support four type of data, numberic, text, boolean and blob.

- for the text, there are char, varchar, text data type.
- for the numberic, there are int, float, double data type.
- for the boolean, only the bool
- for the blob, there are binary blob and text blob data type.

  For the specific database, we can use the database specific sql command to retrieve the table definition info.

Operation on the numberic columns

A typical task for data analysts is to derive new columns from existing columns by using arithmetic operations.

For example, the sales db, there are Amount and quanity columns, but no item price columns, if we want to know the item price of the each order, how to write the sql?

```
select order_id, amount/quanity as item_price from sales
limit 3
```

That is, we can use the arithmetic operations in the SQL statements, and also we can SQL functions or user defined functions in the sQL statements.

For example, we can only return the two decimal point of the item price by using the round functions:

```
select order_id, round(amount/quantity, 2) as item_price from sales
```

```
sql('select order_id, round(amount/quantity, 2) as item_price from sales limit 3')
```

```
  order_id   item_price
  varchar      double

  B-26055        409.21
  B-26055         74.56
  B-26055         443.0
```

And if your manager want to know the tax of the sales, we can easily calculate the tax with the arithmetic operations

```
SELECT
    order_id,
    round(amount * 0.07,2) as PST ,
    round(amount*0.05,2) as GST
    FROM sales
    LIMIT 5
'''
```

```
::: {.cell execution_count=9}
``` {.python .cell-code}
sql('select order_id, amount, round(amount*0.05,2) as GST, round(amount*0.07,2) as PST from s
```

order_id	amount	GST	PST
varchar	int64	decimal(22,2)	decimal(22,2)
B-26055	5729	286.45	401.03
B-26055	671	33.55	46.97
B-26055	443	22.15	31.01
B-26055	57	2.85	3.99
B-26055	227	11.35	15.89

:::
```

Basic Comparison operations

From the business perspective, we want to knwo which store has the positive profit and how about this month sales increase, and the profit margin. Let's start with the simple $>$ and $<$ to compare whether one value is greater or less than another.

It's time to introduce the WHERE clause to filter the results we want from the SQL statements.

1. SELECT: Specify what fields we want from the table
2. FROM: Specify what table(or tables) we want from
3. WHERE: Specify the criteria that records in those fields should meet, also can be defined as a filter
4. LIMIT: Specify how many records to return.
5. and in statement we can have ORDER BY, GROUP BY clause, we may introduced later.

Let's see the WHERE clause in action. For Example, we want to see how about the order amount greater than $1000 by writing the following statement with WHERE:

```
SELECT order_id, amount
FROM sales
WHERE amount > 1000
LIMIT 5;
```

```
sql('select order_id, amount from sales where amount > 1000 limit 5')
```

```
  order_id   amount
  varchar    int64

  B-26055      5729
  B-26055      1250
  B-26055      1218
  B-25993      4363
  B-25973      4141
```

The Where clause is essentially used as a filter and return the records where the operators evaluate to TRUE

Compare operator not only the > and <, Here is the list of compare operators:

Operator	Description
=	Equal to
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Most compare operator to check the numberic values, like is 10 less than 8, actually is False. = and <> can also be used to check the value of text values. For Example, we can filter the result with category = 'Clothing', any record that category equal to 'Clothing' will be returned.

```
sql('select month(order_date),count(*) from sales group by month(order_date)')
```

```
  month(order_date)   count_star()
        int64             int64

                  1            192
                  2            140
                  3            205
                  4            101
                  5            118
```

```
                 6            101
                 7             64
                 8            120
                 9             83
                10            106
                11            159
                12            111

  12 rows                2 columns
```

```
sql('select sum(profit)/count(order_id) from sales')
```

```
  (sum(profit) / count(order_id))
              double

                          24.642
```

```
sql('select avg(profit) from sales')
```

```
  avg(profit)
    double

       24.642
```

```
sql('select category, sum(amount) as total_amount  from sales group by category order by tota
```

```
   category     total_amount
    varchar        int128

  Furniture           127181
  Clothing            144323
  Electronics         166267
```