

## Python commands :

### Chapter 1

#### To print data frame

```
import pandas as pd

data = pd.Series([10, 20, 30, 40, 50])

data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Age': [25, 30, 22, 35, 28],
    'City': ['New York', 'Los Angeles', 'Chicago', 'San Francisco', 'Miami']
}

df = pd.DataFrame(data)

df (to print the data frame with its columns and rows)
```

#### to analyze data

```
import pandas as pd

data = {
    "id": [1, 2, 3, 4, 5],
    "product customer purchased": ["Widget A", "Widget B", "Widget A", "Widget C", "Widget B"],
    "Name of customer": ["John Doe", "Jane Smith", "Bob Johnson", "Susan Brown", "Mary Wilson"],
    "address of customer": ["123 Main St", "456 Elm St", "789 Oak St", "234 Pine St", "567 Birch St"],
    "email of customer": ["john.doe@email.com", "", "bob.johnson@email.com", "susan.brown@email.com", "mary.wilson@email.com"],
    "phone number of customer": ["123-456-7890", "987-654-3210", "555-123-7890", "777-888-9999", "111-222-3333"],
```

```
"time of purchase": ["2023-08-10 09:15:00", "2023-08-10 10:30:00", "2023-08-11  
14:45:00", "2023-08-11 15:30:00", "2023-08-12 11:20:00"]  
}
```

```
sample_data = pd.DataFrame(data)
```

```
sample_data (to print the data frame with its columns and rows)
```

```
sample_data.head() (we can view the top 5 rows of our data)
```

```
sample_data.tail() (we can view the last 5 rows of our data)
```

```
sample_data.shape (we can view numbers of rows and columns )
```

```
sample_data.info() (gives a summary of the data frame structured and content and  
includes the number of null values )
```

```
sample_data.duplicated() (it is used to identify duplicate rows in a data fame )
```

suppose we find various duplicate rows in that case and we want to see them, in that case we can print out duplicates using duplicated method to confirm if any of the rows have duplicates, we can do that by assigning duplicates to a variable and filtering data frame based on that variable so lets call the variable : duplicate\_rows

```
duplicate_rows = sample_data[sample_data.duplicated()]
```

if we print : duplicate\_rows

it should give us all our rows that are duplicated, and we can see that indeed duplicate rows exist in our file if this is the case.

## **Chapter 2 Loading Data into Transforming Data with Python**

Loading .CSV files can be load doing the pandas method, pandas.read.csv

```
import pandas as pd
```

```
sample_csv = pd.read_csv("sample_data_csv.csv")
```

if we sample querying using the keyboard head, we see that this CSV has no headers because we didn't specify it has taken the very first column but we can specify the header is none modifying the following:

```
sample_csv = pd.read_csv("sample_data_csv.csv", header = None)
```

loading excel files are no different the command is the following:

```
sample_excel = pd.read_excel("sample_data_excel.xlsx")
```

now we will extract our data from the excel using Pandas

first import pandas

```
import pandas as pd
```

next define the location of where the excel file exists so we ingested our excel file

```
orders = pd.read_excel("H+ Sport Orders.xlsx")
```

to see if the data was imported correctly

```
orders.head()
```

transforming the data (cleaning, standardizing, removing duplicates, and missing values )

Data standardization

First extract data in this case from the excel file it currently resides in

```
customers = pd.read_excel("H+ Sport Customers.xlsx", sheet_name = "data") (specify sheet name)
```

then to see how "customers" looks like run:

```
customers.head()
```

now how we can find duplicates in this kind of tables

first we should list all the columns in our data, we can do this by using the columns method:

```
customers.columns
```

now we can select the columns we need to check for duplicates and make that into a list.

Let's say the list is : columns\_to\_check (we will use email and phone columns):

```
columns_to_check = ['Email', 'Phone', 'FisrtName', 'LastName']
```

we can check for duplicates using the dataframe.duplicated method, but for this data we'll check for duplicates using a different method:

```
duplicates = customers[customers.duplicated(cloumns_to_check)]
```

then print duplicates to check if we have duplicates

lets use

```
duplicates.shapes (to see how many duplicates we have )
```

now lets deduplicates this data leaving only the first instance to do this we'll use the keyboard keep

```
customers = customers[customers.duplicated(columns_to_check, keep='first')]
```

and we'll be checking for nones and missing values, it'll return a Boolean values which are true for null or non-values

```
customers.isnull().sum()
```

to remove columns

import the excel file and the specific sheet we need to work with

```
employees = pd.read_excel("H+ Sport Employees.xlsx", sheet_name="Employees-Table")
```

we can also Check the columns with the following command:

```
employees.columns
```

we indicate the columns we want to remove :

```
columns_to_remove = ['Job Rating', 'New Salary','Tax Rate', '2.91%']
```

we use the keyboard “drop ” and we use it with the columns we indicate under columns\_to\_remove we want to remove

```
employees = employees.drop(columns=columns_to_remove)
```

we can double check if the columns were removed

```
employees.columns
```

## Chapter 3 Loading Data into Target Systems

What is databases: organized collections of data that are controlled using a database management system(DBMS)

DBMS is a software that allows to manipulate, interact, and access data in a database

Databases focus on the operational or transactional data and managing day to day CRUD (create, read, update, delete) operations

Data can be:

Unstructured doesn't have a predefined schema and are not organized in a tubular form, images, videos, etc.

Semi-structured: does not fit into a table but does have some organization like JSON and XML stored in a mongo DB

Structured: uses table to store data, where each table has a predefined schema, example: MySQL, PostgreSQL, and MS SQL

Data warehouses: are structured and centralized systems or repositories that store data from various sources such as transactional database systems, APIs, ERP Systems, and CRM systems, moreover data warehouses are made to support business intelligence in a consistent format

Data warehouses are used to track historical information and are used by data analyst, data scientist, and other business users to aid in decision making

A data lake is a flexible and scalable data storage system storing both structured and unstructured data

Datalake are used by more technical users like data engineers, data scientists, machine learning engineers. Since the data lake can store data from various sources, such as log files, machine-generated data, IoT devices, social media feeds, and other unstructured data

Summary

Databases provide transactional efficiency

Data warehouses are optimized for complex analytical queries from various sources

Data lakes offer scalability and flexibility in data storage

Data lakehouses provide a balance between raw storage and structured querying

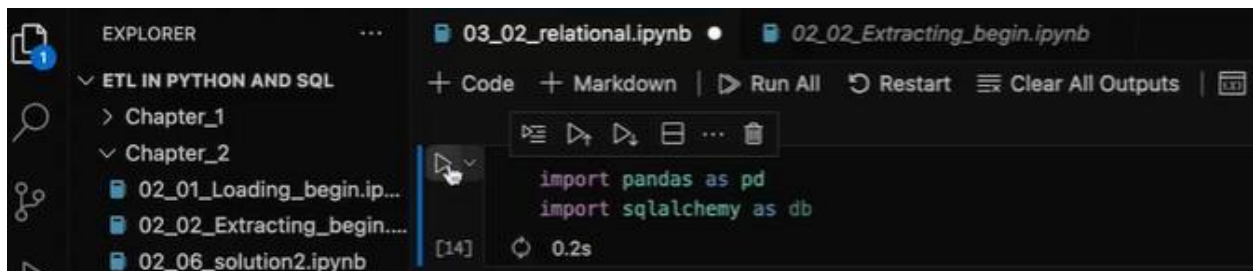
## Loading data into relational databases

For this time we are going to create a PostgreSQL database on Elephant SQL to create, read, update, and delete data directly from a web browser. This will serve as our hplus sport data warehouse.

To create a Postgres database instance: [www.elephantsql.com/plans.html](http://www.elephantsql.com/plans.html) (is not working anymore) I am using MySQL workbench

We also be using SQLAlchemy : [docs.sqlalchemy.org/en/20/](http://docs.sqlalchemy.org/en/20/)

This is an SQL two kit library for python, which allows you to connect and interact with relational databases like Postgres, since this is an external library, we would need to import it just like pandas



```
import pandas as pd
import sqlalchemy as db
```

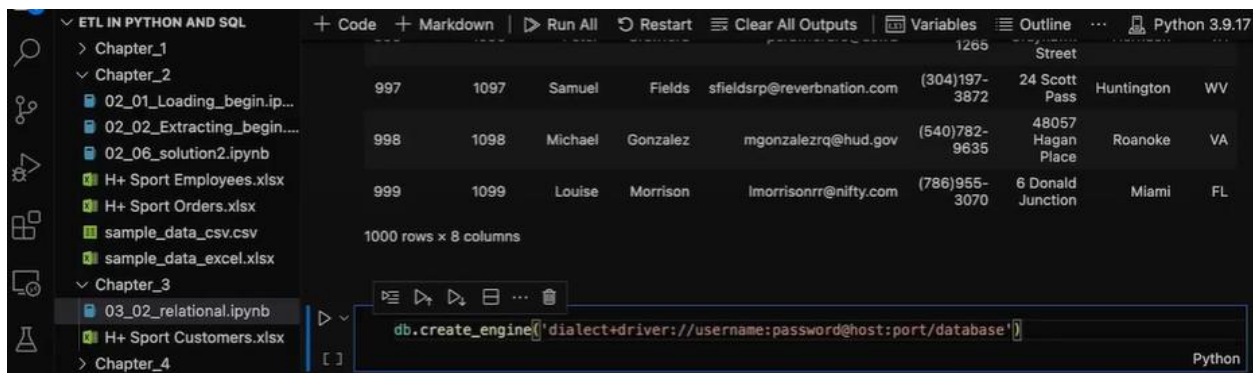
using SQLAlchemy we can connect to relational databases like MySQL, Postgres, SQLite, amongst other databases, using the create engine functions

we have db.CreateEngine, using the Sessionmaker function SQLAlchemy can also be used for data manipulation and querying.

These are just some of the uses cases for SQLAlchemy as use cases are really broad

Let's create a SQL engine, this will be used for tasks like executing SQL queries, managing transactions it'll also serves as a central point for these DBAP interactions. The syntax for creating an engine is this:

`db.create_engine('dialect+driver://username:password@host:port/database')`



| 997 | 1097 | Samuel  | Fields   | sfieldsrp@reverbnation.com | (304)197-3872 | 24 Scott Pass     | Huntington | WV |  |
|-----|------|---------|----------|----------------------------|---------------|-------------------|------------|----|--|
| 998 | 1098 | Michael | Gonzalez | mgonzalezrq@hud.gov        | (540)782-9635 | 48057 Hagan Place | Roanoke    | VA |  |
| 999 | 1099 | Louise  | Morrison | lmorrisonrr@nifty.com      | (786)955-3070 | 6 Donald Junction | Miami      | FL |  |

```
db.create_engine('dialect+driver://username:password@host:port/database')
```

We place the dialects and driver, which is with the database you're working on, which could be Postgres, SQLite, or MySQL, our username and password, our hosts, ports, and database name with the correct details.

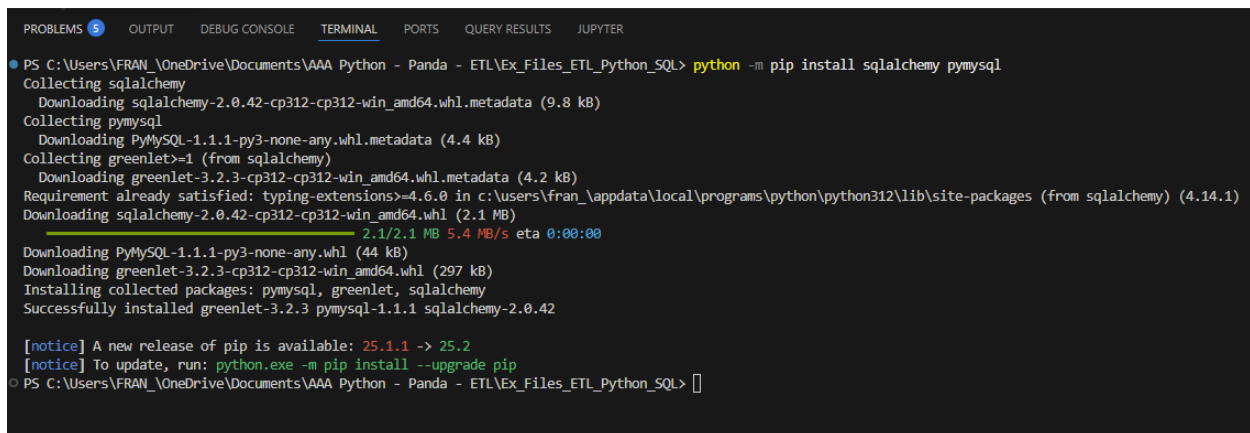
**step-by-step tutorial** to create a **SQLAlchemy engine** in a **VS Code SQL script** using **MySQL**

**Objective: Load the customers DataFrame (from the Excel file) into a MySQL database table using SQLAlchemy.**

Install Required Python Packages

First, run this command in your VS Code terminal:

```
python -m pip install sqlalchemy pymysql
```



```
PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS JUPYTER
● PS C:\Users\FRAN_OneDrive\Documents\AAA Python - Panda - ETL\Ex_Files_ETL_Python_SQL> python -m pip install sqlalchemy pymysql
Collecting sqlalchemy
  Downloading sqlalchemy-2.0.42-cp312-cp312-win_amd64.whl.metadata (9.8 kB)
Collecting pymysql
  Downloading PyMySQL-1.1.1-py3-none-any.whl.metadata (4.4 kB)
Collecting greenlet>=1 (from sqlalchemy)
  Downloading greenlet-3.2.3-cp312-cp312-win_amd64.whl.metadata (4.2 kB)
Requirement already satisfied: typing-extensions>=4.6.0 in c:\users\fran_appdata\local\programs\python\python312\lib\site-packages (from sqlalchemy) (4.14.1)
Downloading sqlalchemy-2.0.42-cp312-cp312-win_amd64.whl (2.1 MB)
  2.1/2.1 MB 5.4 MB/s eta 0:00:00
Downloading PyMySQL-1.1.1-py3-none-any.whl (44 kB)
Downloading greenlet-3.2.3-cp312-cp312-win_amd64.whl (297 kB)
Installing collected packages: pymysql, greenlet, sqlalchemy
Successfully installed greenlet-3.2.3 pymysql-1.1.1 sqlalchemy-2.0.42

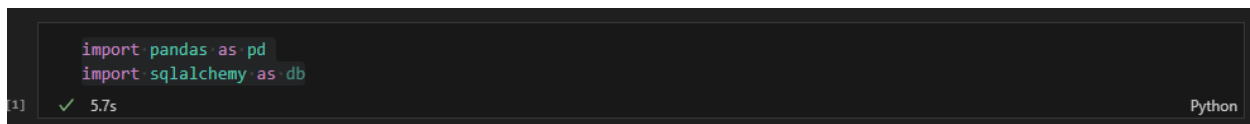
[notice] A new release of pip is available: 25.1.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
○ PS C:\Users\FRAN_OneDrive\Documents\AAA Python - Panda - ETL\Ex_Files_ETL_Python_SQL> []
```

pymysql is the driver used to connect to MySQL with SQLAlchemy.

In VS Code, using Python script lets import the libraries Panda and SQLAlchemy:

```
import pandas as pd
```

```
import sqlalchemy as db
```



```
import pandas as pd
import sqlalchemy as db
```

[1] ✓ 5.7s Python

Then we run the following command to read the excel file that contains the table we are working with:

```
customers = pd.read_excel("H+ Sport Customers.xlsx", sheet_name="data")
```

```
customers = pd.read_excel("H+ Sport Customers.xlsx", sheet_name="data")
```

✓ 0.7s Python

The statement `customers = pd.read_excel("H+ Sport Customers.xlsx", sheet_name="data")` does the following:

- **pd.read\_excel**: This function from the pandas library reads an Excel file.
- **"H+ Sport Customers.xlsx"**: This is the name of the Excel file being read.
- **sheet\_name="data"**: This specifies the sheet named "data" within the Excel file to be read.

The result is stored in the variable `customers` as a `DataFrame`, which allows you to work with the data in Python.

Then we run the command:

```
customers.drop_duplicates()
```

```
customers.drop_duplicates()
```

✓ 0.0s Python

Which is used to remove duplicate rows from the `customers` `DataFrame`. This ensures that each row in the `DataFrame` is unique, which is important for maintaining data integrity before loading it into a relational database.

And we also ran this command:

```
customers = customers.drop(columns='Zipcode')
```

```
customers = customers.drop(columns='Zipcode')
```

✓ 0.0s Python

This command does the following:

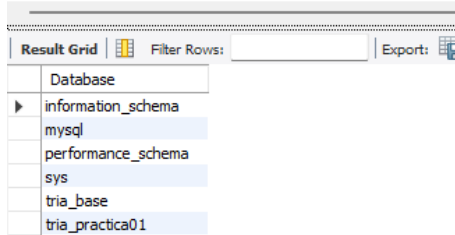
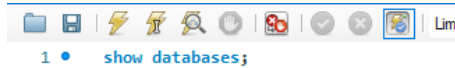
- **customers.drop(columns='Zipcode')**: This removes the 'Zipcode' column from the `customers` `DataFrame`.
- **customers =**: This assigns the resulting `DataFrame` (without the 'Zipcode' column) back to the `customers` variable.

This operation effectively deletes the 'Zipcode' column from the `DataFrame`, ensuring it is no longer part of the data you are working with.



Then we open MySQL workbench and check if the database 'hplussport' is already created, to do this run the following command in the SQL script:

Show databases;



There is another form to check if the MySQL Database Already Exists: Check Programmatically in Python running this code in Python:

```
# We don't need to import SQLAlchemy again if we already did
```

```
import sqlalchemy as db
```

```
# Replace with your actual password
```

```
engine = db.create_engine("mysql+pymysql://root:Password*@localhost:3306")
```

```
with engine.connect() as conn:
```

```
    result = conn.execute(db.text("SHOW DATABASES"))
```

```
    dbs = [row[0] for row in result]
```

```
    print("Databases:", dbs)
```

```
# If 'hplussport' not in the list, you need to create it
```

```
with engine.connect() as conn:
    result = conn.execute(db.text("SHOW DATABASES"))
    dbs = [row[0] for row in result]
    print("Databases:", dbs)
```

✓ 0.3s Python

Databases: ['information\_schema', 'mysql', 'performance\_schema', 'sys', 'tria\_base', 'tria\_practica01']

Since the 'hplussport' database is not listed, we need to create it.

We can create the database hplussport, using MySQL workbench running the following command :

CREATE DATABASE hplussport;

Or we can run the following block in Python and create it from there without using MySQL Workbench, similar to when we check the databases we have in our MySQL

with engine.connect() as conn:

conn.execute(db.text("CREATE DATABASE hplussport"))

print("Database 'hplussport' created!")

```
with engine.connect() as conn:
    conn.execute(db.text("CREATE DATABASE hplussport"))
    print("Database 'hplussport' created!")
```

✓ 0.0s Python

Database 'hplussport' created!

Once the database is created, update your engine to **target the new database**:

engine = db.create\_engine("mysql+pymysql://root:Password@localhost:3306/hplussport")

And now we run the command customers.to\_sql to load the customers DataFrame into a SQL table.

customers.to\_sql("customers", engine, if\_exists='replace', index=False)

print(" Data uploaded to 'customers' table in hplussport database!")

```
customers.to_sql("customers", engine, if_exists='replace', index=False)
print(" Data uploaded to 'customers' table in hplussport database!")
```

✓ 0.1s Python

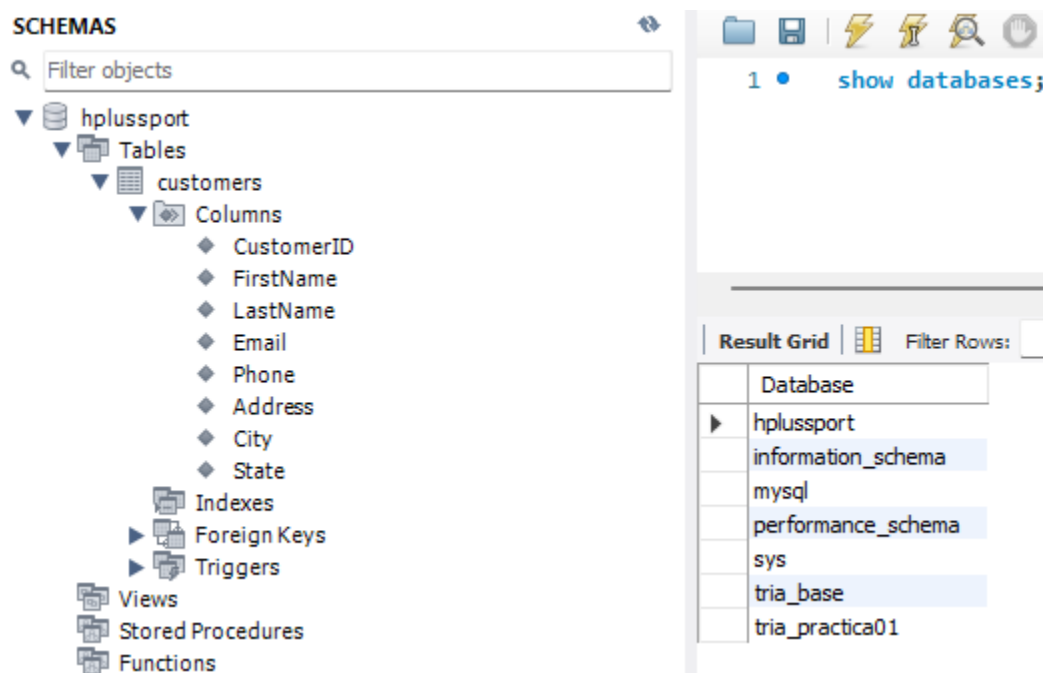
Data uploaded to 'customers' table in hplussport database!

Here's a breakdown of each part:

- **customers.to\_sql**: This is the method used to write records stored in a DataFrame to a SQL database.
- **"customers"**: This specifies the name of the SQL table where the data will be written. In this case, the table is named 'customers'.
- **engine**: This is the SQLAlchemy engine that connects to your database. It manages the connection and interactions with the database.
- **if\_exists='replace'**: This parameter specifies what to do if the table already exists. 'replace' means that if the table exists, it will be dropped and a new one will be created.
- **index=False**: This indicates that the DataFrame's index should not be written as a separate column in the SQL table.

This command effectively transfers the data from the customers DataFrame into the 'customers' table in my MySQL database, replacing any existing table with the same name.

MySQL work bench shows that the data was effectively transferred :



In summary to load the customers DataFrame (from the Excel file) into a MySQL database table using SQLAlchemy, we should do the following :

```
import pandas as pd
```

```

import sqlalchemy as db

# Load and clean Excel file

customers = pd.read_excel("H+ Sport Customers.xlsx", sheet_name="data")

customers = customers.drop_duplicates()

customers = customers.drop(columns='Zipcode')

# Create engine — update your password!

engine =
db.create_engine("mysql+pymysql://root:your_password@localhost:3306/hplussport")

# Upload to SQL

customers.to_sql("customers", con=engine, if_exists='replace', index=False)

```

this is a **ETL-style work** using:

- Python (pandas, sqlalchemy)
- MySQL (database engine)
- MySQL Workbench (GUI client)

## Data Quality checks and validation with SQL

As data moves from one place to the other. It's important to ensure data quality, completeness, and correctness. As your data pipeline becomes more complex, there will be a need to create tests that ensure that the data has not been truncated, lost, or corrupted during the ETL process.

One of these checks and tests include counting the number of rows and columns in the source versus the destination accounting for all the transformations that have taken place, another good data accuracy check can also be checking for nulls, empty rows, duplicates, and default values in fields that should have valid data

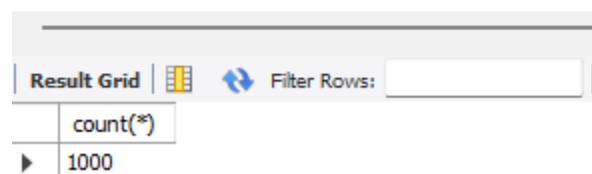
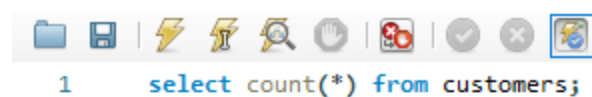
You can use that accuracy checks to make sure the data is right, it stays the same and it's current. These checks often involve comparing data values with a reference source or a predefined rule or limit

You can use data integrity check when data is not good when it is still, this can lead to incorrect conclusions, therefore poor decision making and costing company money a good way to implement data freshness checks is to agree on data warehouse SLAs what is the most amount of time allowed in latency between the source and the data warehouse, then you can create a lot based on these SQL rules. For example, you could agree that data is considered fresh if the maximum timestamp in the table is C minus one day, where C is the current timestamp.

Now let's implement data quality checks in our code. Here we will get the count of rows loaded into our data warehouse and confirm that it's the same as the count of rows in our customer's data frame.

We can confirm that by running a select count from customers. Open MySQL SQL script and run the following command :

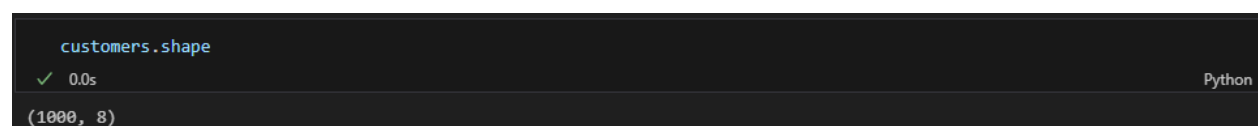
Select count(\*) from customers;



This gives us 1000 rows

Now let's go back to our data in our Jupyter notebook with Python and let's run the command :

Customers.shape



It gives us 1000 rows and 8 columns.

We have successfully performed a data quality check for our ETL

Challenge transform the data and remove duplicates and nulls

Load transformed employees data frame from challenge 2 into the H+ Sport Data Warehouse (the “Employees ” table )

Hint: create an engine and use the to\_sql pandas function.

First we need to export the 2 libraries : pandas and SQLAlchemy

import pandas as pd

import sqlalchemy as db

```
import pandas as pd
import sqlalchemy as db
```

✓ 5.7s

Python

Then we read the excel file, specifically the sheet named employees table:

```
employees = pd.read_excel("../Chapter_2/H+ Sport Employees.xlsx",
sheet_name="Employees-Table")
```

```
employees = pd.read_excel("../Chapter_2/H+ Sport Employees.xlsx", sheet_name="Employees-Table")
```

Python

Then we run the command :

employees.columns

```
employees.columns
Index(['Employee Name', 'Building', 'Department', 'Status', 'Hire Date',
      'Month', 'Years', 'Benefits', 'Salary', 'Job Rating', 'New Salary',
      'Tax Rate', '2.91%'],
      dtype='object')
```

Python

The statement employees.columns retrieves the column labels of the employees DataFrame, allowing you to see the names of all the columns it contains.

The next step is to removed all the duplicates and null values we found. In this case the following columns 'Job Rating', 'New Salary', 'Tax Rate', and '2.91%' must be removed using the following command:

```
columns_to_remove = ['Job Rating', 'New Salary','Tax Rate', '2.91%']
```

```
columns_to_remove = ['Job Rating', 'New Salary', 'Tax Rate', '2.91%']
```

✓ 0.0s

Python

Then we can run the command:

```
employees = employees.drop(columns=columns_to_remove)
```

```
employees = employees.drop(columns=columns_to_remove)
```

✓ 0.0s

Python

The statement `employees = employees.drop(columns=columns_to_remove)` removes the columns listed in `columns_to_remove` from the `employees` DataFrame and updates the DataFrame accordingly.

Now create the engine to target to our MySQL database using the following command:

```
engine = db.create_engine("mysql+pymysql://root:Password@localhost:3306/hplussport")
```

The statement `engine =`

`db.create_engine("mysql+pymysql://root:your_password@localhost:3306/hplussport")` creates a connection engine to a MySQL database using SQLAlchemy.

Finally run the command `employees.to_sql` to load the `employees` DataFrame into a SQL table.

```
employees.to_sql("employees", engine, if_exists='replace', index=False)
```

```
employees.to_sql("employees", engine, if_exists='replace', index=False)
```

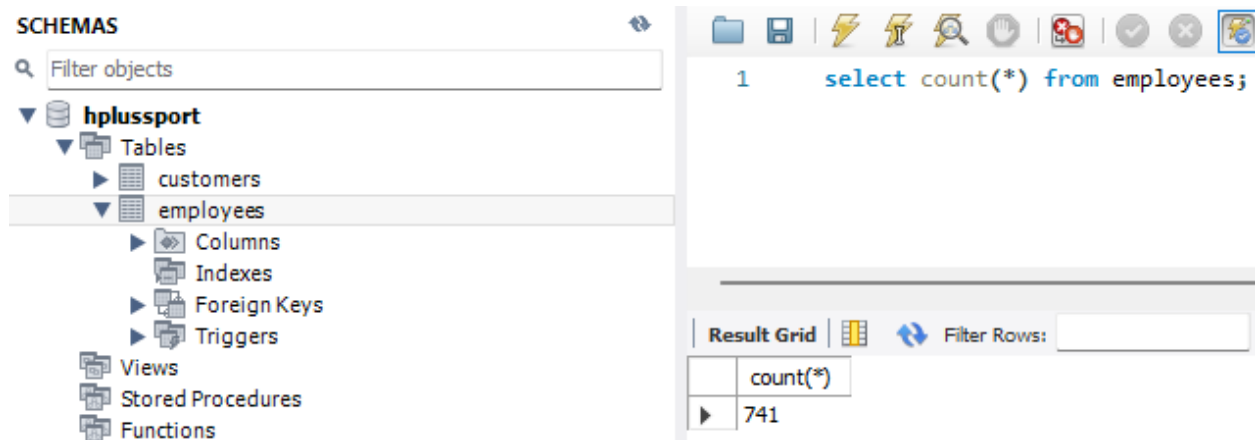
✓ 0.2s

Python

741

The command `employees.to_sql("employees", engine, if_exists='replace', index=False)` writes the `employees` DataFrame to a SQL table named 'employees' using the specified database connection (`engine`). If the table already exists, it will be replaced, and the DataFrame's index will not be included as a separate column.

MySQL work bench shows that the data was effectively transferred :



## Chapter 4 automating ETL Scheduling ETL Jobs with Python

### Query Data with SQL

ETL are great because power that data we need to make important business decisions

For example

Where to open a new H+ Sport Store branch?

To query our data we need to use MySQL workbench and lets create a new tab for executing queries. Always make sure we are using the correct database in schemas

Lets run the following query:

Select \* from customers LIMIT 100;



SCHEMAS

Filter objects

- hplussport
  - Tables
    - customers
      - Columns
      - Indexes
      - Foreign Keys
      - Triggers
    - employees
      - Columns
      - Indexes
      - Foreign Keys
      - Triggers
    - Views
    - Stored Procedures
    - Functions
  - sys
    - trial\_base
    - trial\_practica01

Limit to 1000 rows

1 `select * from customers limit 100;`

Result Grid

| CustomerID | FirstName | LastName   | Email                   | Phone         | Address                 | City           | State |
|------------|-----------|------------|-------------------------|---------------|-------------------------|----------------|-------|
| 100        | Carol     | Shaw       | cshaw0@mb.com           | (206)804-8771 | 8157 Longview Court     | Seattle        | WA    |
| 101        | Elizabeth | Carr       | ecarr1@orade.com        | (512)187-2507 | 3934 Petterle Trail     | Austin         | TX    |
| 102        | Ernest    | Ramos      | eramos2@plala.or.jp     | (816)540-4257 | 8699 Clarendon Terrace  | Kansas City    | MO    |
| 103        | Jane      | Carter     | jcarter3@harvard.edu    | (214)839-0542 | 2830 Novick Lane        | Irving         | TX    |
| 104        | Martha    | Cooper     | mcooper4@go.com         | (727)235-5696 | 4537 Hoard Lane         | Tampa          | FL    |
| 105        | Patricia  | Fox        | pfox5@hubpages.com      | (240)937-3491 | 40748 Stoughton Parkway | Frederick      | MD    |
| 106        | Jack      | Peters     | jpeters6@java.com       | (832)171-7210 | 1 Gerald Hill           | Spring         | TX    |
| 107        | Brian     | Mcdonald   | bmcDonald7@ft.com       | (405)255-8820 | 15 High Crossing Street | Oklahoma City  | OK    |
| 108        | Nicholas  | Clark      | nclark8@mibeian.gov.cn  | (203)941-4125 | 87 Michigan Way         | Danbury        | CT    |
| 109        | Cheryl    | Robinson   | crobison9@tuttocitta.it | (205)646-1280 | 58149 Gale Lane         | Birmingham     | AL    |
| 110        | Doris     | Ramirez    | dramireza@cbslocal.com  | (574)199-9006 | 657 Muir Crossing       | South Bend     | IN    |
| 111        | Sandra    | Ortiz      | sortzb@cloudflare.com   | (757)407-9411 | 2 Moose Terrace         | Richmond       | VA    |
| 112        | Timothy   | Garrett    | tgarrett@cisco.com      | (513)298-2899 | 887 Gina Trail          | Cincinnati     | OH    |
| 113        | Chris     | Williamson | cwillamsond@fotki.com   | (505)896-8356 | 4 Burning Wood Way      | Albuquerque    | NM    |
| 114        | Norma     | Scott      | nscotte@del.com         | (612)432-2596 | 09878 Redwing Court     | Minneapolis    | MN    |
| 115        | Brandon   | Webb       | bwebbf@fastcompany.com  | (915)364-3590 | 98255 Colorado Parkway  | El Paso        | TX    |
| 116        | Stephen   | Mason      | smason@shinystat.com    | (415)861-1390 | 7 Manley Trail          | San Francisco  | CA    |
| 117        | Gloria    | Armstrong  | garmsstrong@unicef.org  | (863)646-4449 | 972 Waywood Alley       | Lakeland       | FL    |
| 118        | Beverly   | Meyer      | bmeyer@cyberchimps.com  | (208)900-5215 | 59460 Rutledge Crossing | Boise          | ID    |
| 119        | Janice    | Mitchell   | jmitchell@lycos.com     | (360)759-7569 | 640 Calypso Drive       | Olympia        | WA    |
| 120        | Ronald    | Murphy     | rmurphyk@yandex.ru      | (303)447-5945 | 452 Parkside Lane       | Denver         | CO    |
| 121        | Ruby      | Lawson     | rlawson@ted.com         | (361)927-9275 | 26332 Granby Junction   | Corpus Christi | TX    |
| 122        | Jean      | Palmer     | jpalmer@delicious.com   | (402)166-3811 | 5 Dwight Point          | Lincoln        | NE    |

We can go further to get account of all the customers where state is in Texas.

Select count(\*)

From customers

Where state = 'TX';

SCHEMAS

Filter objects

- hplussport
  - Tables
    - customers
      - Columns
      - Indexes
      - Foreign Keys
      - Triggers
    - employees
      - Columns
      - Indexes
      - Foreign Keys
      - Triggers
    - Views
    - Stored Procedures
    - Functions
  - sys
    - trial\_base
    - trial\_practica01

Limit to 1000 rows

1 `select count(*) as 'Number of customers whose state is in Texas'`  
 2 `from customers`  
 3 `where state = 'TX';`

Result Grid

| Number of customers whose state is in Texas |
|---|
| 110   |

We have 110 customers whose state is in Texas

We can be more specific and also check not only the state but the city, for example state: Texas and the city: Dallas

select count(\*) as 'Number of customers whose state is in Dallas, Texas'

from customers

where state = 'TX' and city = 'Dallas';

The screenshot shows a database management tool interface. On the left, a 'SCHEMAS' pane displays a tree view of the database structure, including tables like 'customers', 'employees', 'views', 'stored procedures', and 'functions'. The main query editor on the right contains the following SQL code:

```
1 select count(*) as 'Number of customers whose state is in Dallas, Texas'
2 from customers
3 where state = 'TX' and city = 'Dallas';
```

Below the query editor, the 'Result Grid' shows the output of the query:

| Number of customers whose state is in Texas |
|---|
| 11  |

The possibilities are endless, we can connect our table to an analysis tool and create dashboards and reports based on our data.

Now we want to confirm what states we should open a new H+ Sports Store branch. Here is the scenario, the company is making money and wants to invest wisely by opening a new branch in the states where it has the most customers. What states and cities should that be?

To answer the question we should select the states, the city, the count of customers in each state and city, grouping them by states and city, and ordering them with the highest customers descending.

Select state, city, count('customerID') as CustomerCount

From customers

Group by state, city

Order by CustomerCount desc;

The screenshot shows the same database management tool interface. The SQL query editor now contains the following code:

```
1 select state, city, count(CustomerID) as CustomerCount
2 from customers
3 group by state, city
4 order by CustomerCount desc;
```

The 'Result Grid' displays the results of this query, showing the top states and cities by customer count:

| state | city          | CustomerCount |
|-------|---------------|---------------|
| DC    | Washington    | 53            |
| NY    | New York City | 21            |
| TX    | Houston       | 19            |
| MO    | Kansas City   | 16            |
| TX    | El Paso       | 16            |
| IL    | Chicago       | 14            |
| GA    | Atlanta       | 14            |
| FL    | Miami         | 13            |
| OH    | Dayton        | 12            |
| CA    | Los Angeles   | 12            |

Also we can now present our findings and suggestions management with data to back it up. Opening a new branch in Washington DC will be a great investment as we already have a large customer base in the city.

Hence the job of an ETL developer paused the business to make the right decisions.

### **Schedule ETL jobs with Airflow (Python)**

Now we will automate our ETL to ensure it runs unaided. Imagine the sales team has a meeting every day at 8 am, this means that the sales reports and dashboards have to be ready at least 30 to 15 minutes before the meeting, which in turn means the ETL job has to have run and updated the table with the latest datasets. If this was to be done manually by a data engineer the engineer would have to be awake for at least an hour or two before the data is needed to run the pipeline and deal with any issues that may arise. This is why ETL automation is necessary

ETL automation

Uses technology to run ETL processes and workflows between tools and systems without manual intervention based on schedules or events like new data or schemas changes at the source. This ensures that organizations are always working with up to date and reliably updated datasets.

Tools for ETL automation

Apache Airflow

Oozie

NiFi

Luigi

Microsoft Azure Data Factory

SSIS

To automate our ETL, we are going to use Apache Airflow

Apache Airflow is an open source platform designed to schedule and monitor workflows defined as directed acyclic graphs (DAGs) where each node represents a task to be updated independently or in parallel to other tasks which makes it very useful for ETL processes, data pipeline orchestration and other automation scenarios

## Setting up airflow

# Make sure you have Python installed on your Windows machine.

# You can download the latest version of Python from the official Python website. During installation,

# make sure to check the box that says "Add Python to PATH."

In my case I have python environment installed in my VS code, so to Install Apache Airflow Dependencies I used the following:

python -m pip install pywin32

```
PS C:\Users\FRAN_OneDrive\Documents\AAA Python - Panda - ETL\Ex_Files_ETL_Python_SQL> python -m pip install pywin32
Requirement already satisfied: pywin32 in c:\users\fran_\appdata\local\programs\python\python312\lib\site-packages (311)
```

python -m pip install apache-airflow[win]

```
PS C:\Users\FRAN_OneDrive\Documents\AAA Python - Panda - ETL\Ex_Files_ETL_Python_SQL> python -m pip install apache-airflow[win]
Collecting apache-airflow[win]
  Downloading apache_airflow-3.0.4-py3-none-any.whl.metadata (32 kB)
WARNING: apache-airflow 3.0.4 does not provide the extra 'win'
Collecting apache-airflow-core==3.0.4 (from apache-airflow[win])
  Downloading apache_airflow_core-3.0.4-py3-none-any.whl.metadata (7.4 kB)
Collecting apache-airflow-task-sdk<1.1.0,>=1.0.4 (from apache-airflow[win])
  Downloading apache_airflow_task_sdk-1.0.4-py3-none-any.whl.metadata (3.8 kB)
Collecting a2wsgi>=1.10.8 (from apache-airflow-core==3.0.4->apache-airflow[win])
  Downloading a2wsgi-1.10.10-py3-none-any.whl.metadata (4.0 kB)
Collecting aiosqlite>=0.20.0 (from apache-airflow-core==3.0.4->apache-airflow[win])
  Downloading aiosqlite-0.21.0-py3-none-any.whl.metadata (4.3 kB)
Collecting alembic<2.0,>=1.13.1 (from apache-airflow-core==3.0.4->apache-airflow[win])
  Downloading alembic-1.16.4-py3-none-any.whl.metadata (7.3 kB)
Collecting apache-airflow-providers-common-compat>=1.6.0 (from apache-airflow-core==3.0.4->apache-airflow[win])
  Downloading apache_airflow_providers_common_compat-1.7.3-py3-none-any.whl.metadata (5.3 kB)
```

Then, Initialize Airflow Database (**I am using windows as operative system**):

Run the following commands to initialize the Airflow database

airflow db init

I received an error. The issue is that tutorial is for running Airflow natively on Windows, but Airflow isn't supported on Windows; the reliable path is WSL2 (Ubuntu)

Hence, we need to Install WSL2 + Ubuntu

**Step 1:** open Windows Power shell as an administrator and run the following command to see if I already have them installed.

wsl --status

If it says "No installed distributions" or WSL isn't set up, run:

```
wsl --install -d Ubuntu
```

This installs WSL2 and Ubuntu. Reboot when it tells you to.

Now launch Ubuntu (Start menu “Ubuntu” or):

```
wsl -d Ubuntu
```

This should open the Ubuntu terminal where you set up your Linux username and password.

now you got Ubuntu running inside WSL, go to the Ubuntu home directory using the command :

```
cd ~
```

**Step 2:** Update Ubuntu packages. to do this run the following command:

```
sudo apt update && sudo apt upgrade -y
```

Enter your Linux password if asked — the one you just set.

```
fjfigueroa@DESKTOP-I02K5RJ:~$ sudo apt update && sudo apt upgrade -y
[sudo] password for fjfigueroa:
Get:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Hit:2 http://archive.ubuntu.com/ubuntu noble InRelease
Get:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:5 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [21.5 kB]
Get:6 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [878 kB]
Get:7 http://archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:8 http://security.ubuntu.com/ubuntu noble-security/universe Translation-en [194 kB]
Get:9 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [52.3 kB]
Get:10 http://security.ubuntu.com/ubuntu noble-security/universe amd64 c-n-f Metadata [17.0 kB]
Get:11 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 B]
Get:12 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages [18.5 kB]
Get:13 http://security.ubuntu.com/ubuntu noble-security/multiverse Translation-en [4288 B]
Get:14 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [212 B]
Get:15 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 c-n-f Metadata [380 B]
Get:16 http://archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
Get:17 http://archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]
Get:18 http://archive.ubuntu.com/ubuntu noble/universe amd64 c-n-f Metadata [301 kB]
Get:19 http://archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [269 kB]
Get:20 http://archive.ubuntu.com/ubuntu noble/multiverse Translation-en [118 kB]
Get:21 http://archive.ubuntu.com/ubuntu noble/multiverse amd64 Components [35.0 kB]
Get:22 http://archive.ubuntu.com/ubuntu noble/multiverse amd64 c-n-f Metadata [8328 B]
Get:23 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [1313 kB]
Get:24 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [164 kB]
Get:25 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1120 kB]
Get:26 http://archive.ubuntu.com/ubuntu noble-updates/universe Translation-en [287 kB]
Get:27 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [377 kB]
Get:28 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 c-n-f Metadata [26.0 kB]
```

**Step 3:** Install Python and tools

Ubuntu already has Python 3, but we'll add venv and pip:

sudo apt install -y python3-venv python3-pip

```
fjfigueroa@DESKTOP-I02K5RJ:~$ sudo apt install -y python3-venv python3-pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  build-essential bzip2 cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu dpkg-dev fakeroot g++ g++-13 g++-13-x86-64-linux-gnu g++-x86-64-linux-gnu gcc
  gcc-13 gcc-13-base gcc-13-x86-64-linux-gnu gcc-x86-64-linux-gnu javascript-common libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl
  libao3 libasan8 libatomic1 libc-dev-bin libc-devtools libc6-dev libcc1-0 libcrypt-dev libde265-0 libdpkg-perl libexpat1-dev libfakeroot libfile-fcntllock-perl
  libgcc-13-dev libgd3 libgomp1 libheif-plugin-aomdec libheif-plugin-aomenc libheif-plugin-libde265 libheif1 libhwasan0 libisl23 libitm1 libjs-jquery
  libjs-sphinxdoc libjs-underscore liblsan0 libmpc3 libpython3-dev libpython3.12-dev libquadmath0 libstdc++-13-dev libtsan2 libubsan1 libxpm4 linux-libc-dev
  lto-disabled-list make manpages-dev python3-dev python3-pip-whl python3-setuptools-whl python3-wheel python3.12-dev python3.12-venv rpcsvc-proto zlib1g-dev
Suggested packages:
  bzip2-doc cpp-doc gcc-13-locales cpp-13-doc debian-keyring g++-multilib g++-13-multilib gcc-13-doc gcc-multilib autoconf automake libtool flex bison gdb gcc-doc
  gcc-13-multilib gdb-x86-64-linux-gnu apache2 | lighttpd | httpd glibc-doc bsr libgd-tools libheif-plugin-x265 libheif-plugin-ffmpegdec libheif-plugin-jpegdec
  libheif-plugin-jpegenc libheif-plugin-j2kdec libheif-plugin-j2kenc libheif-plugin-rav1e libheif-plugin-svtenc libstdc++-13-doc make-doc
The following NEW packages will be installed:
  build-essential bzip2 cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu dpkg-dev fakeroot g++ g++-13 g++-13-x86-64-linux-gnu g++-x86-64-linux-gnu gcc
  gcc-13 gcc-13-base gcc-13-x86-64-linux-gnu gcc-x86-64-linux-gnu javascript-common libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl
  libao3 libasan8 libatomic1 libc-dev-bin libc-devtools libc6-dev libcc1-0 libcrypt-dev libde265-0 libdpkg-perl libexpat1-dev libfakeroot libfile-fcntllock-perl
  libgcc-13-dev libgd3 libgomp1 libheif-plugin-aomdec libheif-plugin-aomenc libheif-plugin-libde265 libheif1 libhwasan0 libisl23 libitm1 libjs-jquery
  libjs-sphinxdoc libjs-underscore liblsan0 libmpc3 libpython3-dev libpython3.12-dev libquadmath0 libstdc++-13-dev libtsan2 libubsan1 libxpm4 linux-libc-dev
  lto-disabled-list make manpages-dev python3-dev python3-pip python3-pip-whl python3-setuptools-whl python3-venv python3-wheel python3.12-dev python3.12-venv
  rpcsvc-proto zlib1g-dev
0 upgraded, 70 newly installed, 0 to remove and 0 not upgraded.
Need to get 83.0 MB of archives.
After this operation, 286 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libc-dev-bin amd64 2.39-0ubuntu8.5 [20.4 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 linux-libc-dev amd64 6.8.0-71.71 [1897 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble/main amd64 libcrypt-dev amd64 1:4.4.36-4build1 [112 kB]
Setting up python3.12-dev (3.12.3-1ubuntu0.7) ...
Setting up g++-x86-64-linux-gnu (4:13.2.0-7ubuntu1) ...
Setting up g++-13 (13.3.0-6ubuntu2~24.04) ...
Setting up libpython3-dev:amd64 (3.12.3-0ubuntu2) ...
Setting up python3-dev (3.12.3-0ubuntu2) ...
Setting up g++ (4:13.2.0-7ubuntu1) ...
update-alternatives: using /usr/bin/g++ to provide /usr/bin/c++ (c++) in auto mode
Setting up build-essential (12.10ubuntu1) ...
Setting up libheif1:amd64 (1.17.6-1ubuntu4.1) ...
Setting up libgd3:amd64 (2.3.3-9ubuntu5) ...
Setting up libc-devtools (2.39-0ubuntu8.5) ...
Setting up libheif-plugin-aomdec:amd64 (1.17.6-1ubuntu4.1) ...
Setting up libheif-plugin-libde265:amd64 (1.17.6-1ubuntu4.1) ...
Setting up libheif-plugin-aomenc:amd64 (1.17.6-1ubuntu4.1) ...
Processing triggers for libc-bin (2.39-0ubuntu8.5) ...
Processing triggers for man-db (2.12.0-4build2) ...
fjfigueroa@DESKTOP-I02K5RJ:~$
```

#### Step 4 : Create Airflow project folder

mkdir airflow\_project

cd airflow\_project

```
fjfigueroa@DESKTOP-I02K5RJ:~$ mkdir airflow_project
cd airflow_project
fjfigueroa@DESKTOP-I02K5RJ:~/airflow_project$
```

#### Step 5 : Create & activate a virtual environment

python3 -m venv .venv

source .venv/bin/activate



```
fjfigueroa@DESKTOP-I02K5RJ:~/airflow_project$ python3 -m venv .venv
source .venv/bin/activate
(.venv) fjfigueroa@DESKTOP-I02K5RJ:~/airflow_project$
```

You should now see (.venv) at the start of your prompt.

## Step 6 : Upgrade pip & setuptools

python -m pip install --upgrade pip setuptools wheel

```
(.venv) fjfigueroa@DESKTOP-I02K5RJ:~/airflow_project$ python -m pip install --upgrade pip setuptools wheel
Requirement already satisfied: pip in ./venv/lib/python3.12/site-packages (24.0)
Collecting pip
  Downloading pip-25.2-py3-none-any.whl.metadata (4.7 kB)
Collecting setuptools
  Downloading setuptools-80.9.0-py3-none-any.whl.metadata (6.6 kB)
Collecting wheel
  Downloading wheel-0.45.1-py3-none-any.whl.metadata (2.3 kB)
Downloading pip-25.2-py3-none-any.whl (1.8 MB)
  1.8/1.8 MB 5.4 MB/s eta 0:00:00
Downloading setuptools-80.9.0-py3-none-any.whl (1.2 MB)
  1.2/1.2 MB 5.5 MB/s eta 0:00:00
Downloading wheel-0.45.1-py3-none-any.whl (72 kB)
  72.5/72.5 kB 2.8 MB/s eta 0:00:00
Installing collected packages: wheel, setuptools, pip
  Attempting uninstall: pip
    Found existing installation: pip 24.0
    Uninstalling pip-24.0:
      Successfully uninstalled pip-24.0
Successfully installed pip-25.2 setuptools-80.9.0 wheel-0.45.1
(.venv) fjfigueroa@DESKTOP-I02K5RJ:~/airflow_project$
```

## Step 7: Install Airflow with the correct constraints

Choose an Airflow version known to work well:

AIRFLOW\_VERSION=3.0.4

Detect your Python version (e.g., 3.10 or 3.12):

```
PYTHON_VERSION=$(python -c 'import sys;
print(f"{sys.version_info.major}.{sys.version_info.minor}")')
```

Constraint file must match the Airflow + Python minor version:

```
CONSTRAINT_URL="https://raw.githubusercontent.com/apache/airflow/constraints-
${AIRFLOW_VERSION}/constraints-${PYTHON_VERSION}.txt"
```

And finally:

```
pip install "apache-airflow==${AIRFLOW_VERSION}" --constraint "${CONSTRAINT_URL}"
```

```
(.venv) ffigueroa@DESKTOP-102K5RJ:~/airflow_project$ AIRFLOW_VERSION=3.0.4
(.venv) ffigueroa@DESKTOP-102K5RJ:~/airflow_project$ PYTHON_VERSION=$(python -c 'import sys; print(f"{sys.version_info.major}.{sys.version_info.minor}")')
(.venv) ffigueroa@DESKTOP-102K5RJ:~/airflow_project$ CONSTRAINT_URL="https://raw.githubusercontent.com/apache/airflow/constraints-${AIRFLOW_VERSION}/constraints-${PYTHON_VERSION}.txt"
(.venv) ffigueroa@DESKTOP-102K5RJ:~/airflow_project$ pip install "apache-airflow==${AIRFLOW_VERSION}" --constraint "${CONSTRAINT_URL}"
Collecting apache-airflow==3.0.4
  Downloading apache-airflow-3.0.4-py3-none-any.whl.metadata (32 kB)
Collecting apache-airflow-core==3.0.4 (from apache-airflow==3.0.4)
  Downloading apache-airflow-core-3.0.4-py3-none-any.whl.metadata (7.4 kB)
Collecting apache-airflow-task-sdk<1.1.0,>=1.0.4 (from apache-airflow==3.0.4)
  Downloading apache-airflow-task-sdk-1.0.4-py3-none-any.whl.metadata (3.8 kB)
Collecting a2wsgi>=1.10.8 (from apache-airflow-core==3.0.4->apache-airflow==3.0.4)
  Downloading a2wsgi-1.10.10-py3-none-any.whl.metadata (4.0 kB)
Collecting aiosqlite>=0.20.0 (from apache-airflow-core==3.0.4->apache-airflow==3.0.4)
  Downloading aiosqlite-0.21.0-py3-none-any.whl.metadata (4.3 kB)
Collecting alembic<2.0,>=1.13.1 (from apache-airflow-core==3.0.4->apache-airflow==3.0.4)
  Downloading alembic-1.16.4-py3-none-any.whl.metadata (7.3 kB)
Collecting apache-airflow-providers-common-compat>=1.6.0 (from apache-airflow-core==3.0.4->apache-airflow==3.0.4)
  Downloading apache-airflow-providers-common-compat-1.7.3-py3-none-any.whl.metadata (5.3 kB)
Collecting apache-airflow-providers-common-io>=1.5.3 (from apache-airflow-core==3.0.4->apache-airflow==3.0.4)
  Downloading apache-airflow-providers-common-io-1.6.2-py3-none-any.whl.metadata (5.3 kB)
Collecting apache-airflow-providers-common-sql>=1.26.0 (from apache-airflow-core==3.0.4->apache-airflow==3.0.4)
  Downloading apache-airflow-providers-common-sql-1.27.4-py3-none-any.whl.metadata (5.5 kB)
Collecting apache-airflow-providers-smtp>=2.0.2 (from apache-airflow-core==3.0.4->apache-airflow==3.0.4)
  Downloading apache-airflow-providers-smtp-2.1.2-py3-none-any.whl.metadata (5.1 kB)
Collecting apache-airflow-providers-standard>=0.4.0 (from apache-airflow-core==3.0.4->apache-airflow==3.0.4)
  Downloading apache-airflow-providers-standard-1.5.0-py3-none-any.whl.metadata (3.8 kB)
Collecting argcomplete>=1.10 (from apache-airflow-core==3.0.4->apache-airflow==3.0.4)
  Downloading argcomplete-3.6.2-py3-none-any.whl.metadata (16 kB)
Collecting asgiref>=2.3.0 (from apache-airflow-core==3.0.4->apache-airflow==3.0.4)
  Downloading asgiref-2.3.0-py3-none-any.whl.metadata (9.3 kB)
```

## Step 8: Set AIRFLOW\_HOME (analogous to your Windows step)

```
echo 'export AIRFLOW_HOME=$HOME/airflow_home' >> ~/.bashrc
```

```
source ~/.bashrc
```

```
mkdir -p "$AIRFLOW_HOME"
```

You don't need to hand-write airflow.cfg—Airflow will generate one. The default executor is already SequentialExecutor, which is perfect for local dev.

```
(.venv) ffigueroa@DESKTOP-102K5RJ:~/airflow_project$ echo 'export AIRFLOW_HOME=$HOME/airflow_home' >> ~/.bashrc
source ~/.bashrc
mkdir -p "$AIRFLOW_HOME"
```

## Step 9: Initialize the Airflow metadata DB

```
airflow db migrate
```

```
fjfigueroa@DESKTOP-102K5RJ:~/airflow_project$ airflow db migrate
[2025-08-08T17:13:28.538-0700] {providers_manager.py:953} INFO - The hook_class 'airflow.providers.standard.hooks.filesystem.FSHook' is not fully initialized (UI widgets will be missing), because the 'flask_appbuilder' package is not installed, however it is not required for Airflow components to work
[2025-08-08T17:13:28.542-0700] {providers_manager.py:953} INFO - The hook_class 'airflow.providers.standard.hooks.package_index.PackageIndexHook' is not fully initialized (UI widgets will be missing), because the 'flask_appbuilder' package is not installed, however it is not required for Airflow components to work
DB: sqlite:///home/fjfigueroa/airflow_home/airflow.db
Performing upgrade to the metadata database sqlite:///home/fjfigueroa/airflow_home/airflow.db
[2025-08-08T17:13:28.874-0700] {migration.py:211} INFO - Context impl SQLiteImpl.
[2025-08-08T17:13:28.874-0700] {migration.py:214} INFO - Will assume non-transactional DDL.
[2025-08-08T17:13:28.875-0700] {migration.py:211} INFO - Context impl SQLiteImpl.
[2025-08-08T17:13:28.876-0700] {migration.py:214} INFO - Will assume non-transactional DDL.
[2025-08-08T17:13:28.876-0700] {db.py:730} INFO - Creating Airflow database tables from the ORM
[2025-08-08T17:13:29.411-0700] {migration.py:211} INFO - Context impl SQLiteImpl.
[2025-08-08T17:13:29.412-0700] {migration.py:214} INFO - Will assume non-transactional DDL.
[2025-08-08T17:13:29.498-0700] {migration.py:622} INFO - Running stamp_revision -> fe199e1abd77
[2025-08-08T17:13:29.498-0700] {db.py:741} INFO - Airflow database tables created
Database migrating done!
fjfigueroa@DESKTOP-102K5RJ:~/airflow_project$
```

## Step 10: use airflow standalone so ubuntu environment creates an admin user for you automatically

```
airflow standalone
```

you'll see a bunch of logs:

Webserver listening on <http://localhost:8080>



[illegible]

Then Log in. Open your Windows browser to <http://localhost:8080> and use the printed username/password.

Useful notes:

- Keep this terminal open; standalone runs in the foreground.
- To stop it: press Ctrl+C.
- If port 8080 is busy:

AIRFLOW\_\_WEBSERVER\_\_WEB\_SERVER\_PORT=8081 airflow standalone

- Files are in:

`$AIRFLOW_HOME` (usually `~/airflow_home`) for configs/logs

~/airflow\_home/dags/ for your DAG files (create this if it isn't there)

- If you ever reopen a new terminal, re-activate your venv:

```
cd ~/airflow_project
```

```
source .venv/bin/activate
```

in case you want to read the password file created by the command airflow standalone do the following:

When standalone runs, it writes creds to a file.

1. Switch to Ubuntu (WSL)  
Open the Ubuntu app from Start, or in PowerShell run:  

```
wsl -d Ubuntu
```
2. Open a new Ubuntu terminal (leave the one with logs running).
3. Activate your venv:  

```
cd ~/airflow_project  
source .venv/bin/activate
```
4. Find your Airflow home:  

```
echo $AIRFLOW_HOME
```
5. Show the generated creds:  

```
cat $AIRFLOW_HOME/standalone_admin_password.txt
```
6. You should see something like:  
username: admin  
password: <random-string>
7. if you still don't see the creds:  

```
grep -i 'password' "$AIRFLOW_HOME/standalone.out"  
ls -l "$AIRFLOW_HOME"  
cat "$AIRFLOW_HOME/standalone_admin_password.txt"
```

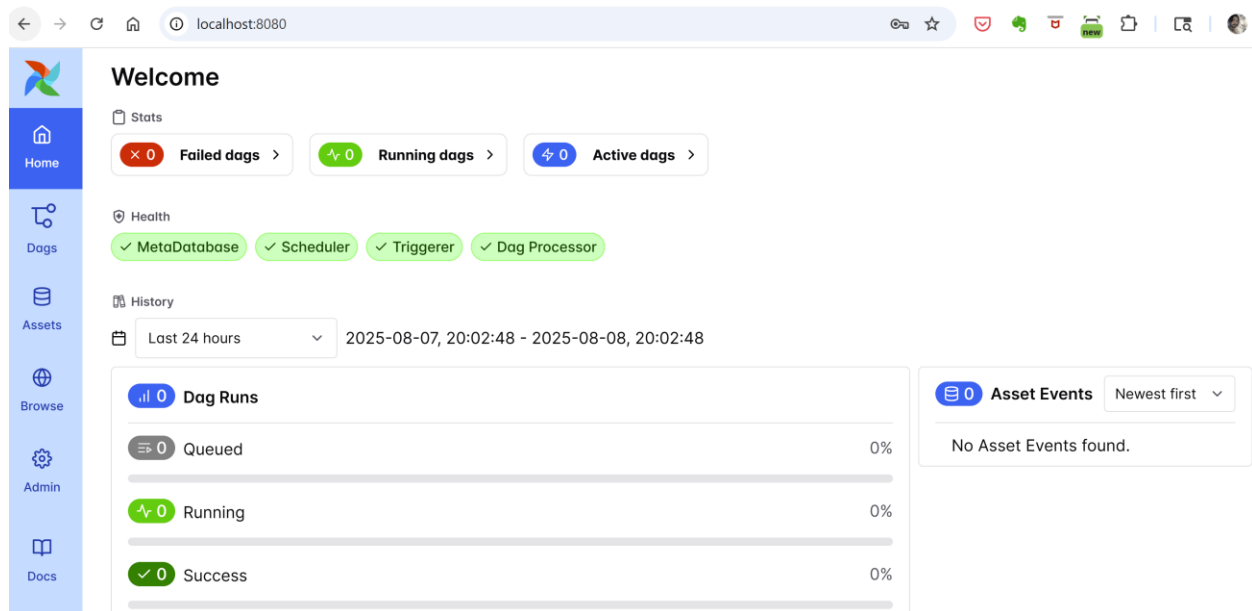
log literally tells us where the password went: "Password for the admin user has been previously generated in..."
8. So your creds are in that file. Do this in your Ubuntu terminal:  

```
cat "$AIRFLOW_HOME/simple_auth_manager_passwords.json.generated"
```

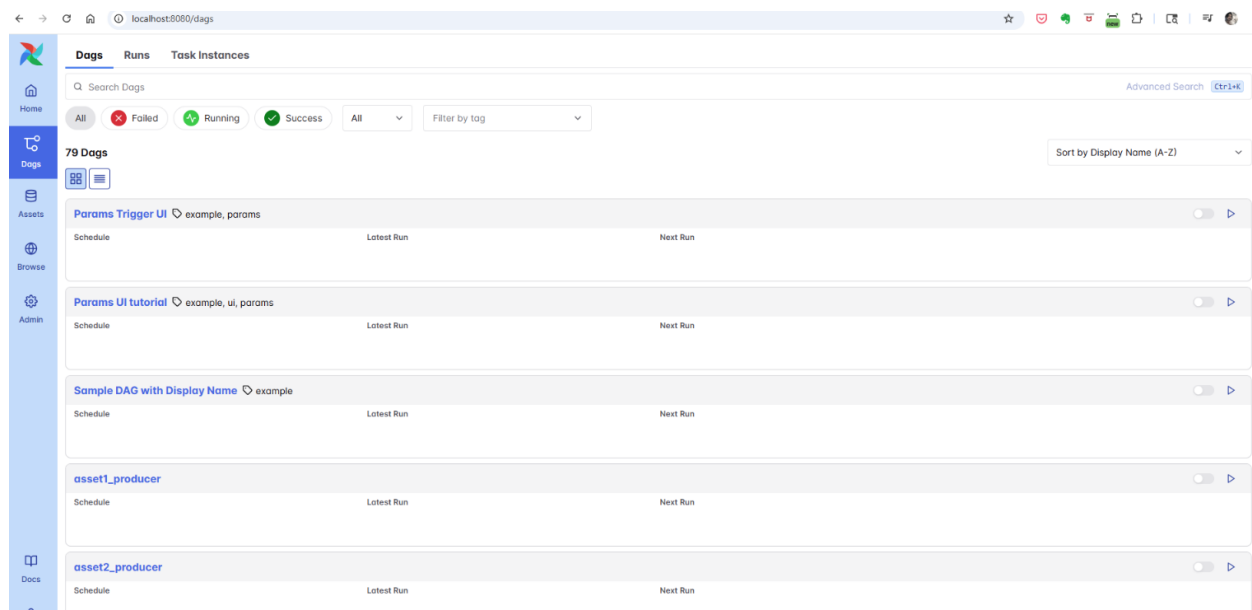
You should see JSON with the admin user and a generated password.
9. Then open <http://localhost:8080>, log in with:  
Username: admin  
Password: the one from that file

**Step 11: Finally Open a web browser and go to <http://localhost:8080>.**

You should see the Airflow web UI



## Create Your First DAG:



You can create your first DAG (Directed Acyclic Graph) by placing Python scripts with DAG definitions in the `AIRFLOW_HOME/dags` directory.

First we have to create a folder called DAGs in our Airflow Home Directory.- Airflow already created a home directory automatically inside Ubuntu:

```
/home/fjfigueroa/airflow_home
```

To create the folder DAGs we have to run the following command into our ubuntu terminal:

```
mkdir -p "$AIRFLOW_HOME/dags"
```

```
(.venv) fjfigueroa@DESKTOP-I02K5RJ:~/airflow_project$ mkdir -p "$AIRFLOW_HOME/dags"  
(.venv) fjfigueroa@DESKTOP-I02K5RJ:~/airflow_project$
```

From now on all of our dags will live here, DAGs are Python files, so we can no longer use Jupyter notebooks here. We have to use .py files, python files. So a neat trick is to create your ETL pipeline logic on jupyter notebook and then, after testing, move it to Python.

For the ETL that will be used in this step we would ingest and transform the Orders table by removing two columns, the customer's comment and salesperson's comment using VS code. We will then load it into our H Plus Sports Data warehouse, naming the table orders

```
import pandas as pd
```

```
import sqlalchemy as sa
```

```
from sqlalchemy.engine import URL
```

```
from airflow.hooks.base import BaseHook
```

```
ORDERS_XLSX = (
```

```
    "/mnt/c/Users/... "H+ Sport orders.xlsx"
```

```
)
```

```
def _sqlalchemy_engine_from_conn_id(conn_id: str = "mysql_conn"):
```

```
    """Build and return a SQLAlchemy Engine from an Airflow connection."""
```

```
    af_conn = BaseHook.get_connection(conn_id)    # Airflow Connection (credentials  
only)
```

```
    url = URL.create(
```

```
        "mysql+pymysql",
```

```
        username=af_conn.login,
```

```
        password=af_conn.password,
```

```
        host=af_conn.host,
```

```
        port=af_conn.port or 3306,
```

```

        database=af_conn.schema,
    )

    return sa.create_engine(url, pool_pre_ping=True, future=True) # <-- SQLAlchemy Engine

def main():

    # 1) Read Excel

    orders = pd.read_excel(ORDERS_XLSX, sheet_name="data")

    # 2) Keep required columns

    orders = orders[["OrderID", "Date", "TotalDue", "Status", "CustomerID", "SalespersonID"]]

    # 3) Build a SQLAlchemy ENGINE from the Airflow connection

    engine = _sqlalchemy_engine_from_conn_id("mysql_conn")

    # --- sanity checks so we fail early if anything is wrong ---

    import sqlalchemy as sa

    print("Engine type:", type(engine))

    assert isinstance(engine, sa.engine.Engine), f"EXPECTED Engine, got {type(engine)}"

    with engine.begin() as sa_conn:

        print("SA connection type:", type(sa_conn))

        assert isinstance(sa_conn, sa.engine.Connection), f"EXPECTED SA Connection, got {type(sa_conn)}"

    # Now write using the SQLAlchemy Connection

    orders.to_sql(

```

```

name="orders",

con=sa_conn,          # <-- MUST be sa.engine.Connection

if_exists="replace",

index=False,

)

```

```

print("ETL script executed successfully.")

```

```

3  import pandas as pd
4  import sqlalchemy as sa
5  from sqlalchemy.engine import URL
6  from airflow.hooks.base import BaseHook
7
8  ✓ ORDERS_XLSX = (
9      "/mnt/c/Users/Fran_/OneDrive/Documents/AAA Python - Panda - ETL/"
10     "Ex_Files_ETL_Python_SQL/Ex_Files_ETL_Python_SQL/Exercise Files/Chapter_4/"
11     "H+ Sport orders.xlsx"
12 )
13
14 ✓ def _sqlalchemy_engine_from_conn_id(conn_id: str = "mysql_conn"):
15     """Build and return a SQLAlchemy Engine from an Airflow connection."""
16     af_conn = BaseHook.get_connection(conn_id)          # Airflow Connection (credentials only)
17     ✓ url = URL.create(
18         "mysql+pymysql",
19         username=af_conn.login,
20         password=af_conn.password,
21         host=af_conn.host,
22         port=af_conn.port or 3306,
23         database=af_conn.schema,
24     )
25     return sa.create_engine(url, pool_pre_ping=True, future=True) # <-- SQLAlchemy Engine
26

```

```

27 def main():
28     # 1) Read Excel
29     orders = pd.read_excel(ORDERS_XLSX, sheet_name="data")
30
31     # 2) Keep required columns
32     orders = orders[["OrderID", "Date", "TotalDue", "Status", "CustomerID", "SalespersonID"]]
33
34     # 3) Build a SQLAlchemy ENGINE from the Airflow connection
35     engine = _sqlalchemy_engine_from_conn_id("mysql_conn")
36
37     # --- sanity checks so we fail early if anything is wrong ---
38     import sqlalchemy as sa
39     print("Engine type:", type(engine))
40     assert isinstance(engine, sa.engine.Engine), f"EXPECTED Engine, got {type(engine)}"
41
42     with engine.begin() as sa_conn:
43         print("SA connection type:", type(sa_conn))
44         assert isinstance(sa_conn, sa.engine.Connection), f"EXPECTED SA Connection, got {type(sa_conn)}"
45
46         # Now write using the SQLAlchemy Connection
47         orders.to_sql(
48             name="orders",
49             con=sa_conn,          # <-- MUST be sa.engine.Connection
50             if_exists="replace",
51             index=False,
52         )
53
54     print("ETL script executed successfully.")

```

Now let's go to create our DAG file

First it will be necessary to import all the classes we need: date time, pendulum, Airflow as DAGs, Airflow operators, which is a python operator and from our orders\_etl\_logic I imported the function main

For default arguments, we have defined owners, depends on past, start date, email on failure, email on retry, number of retries, and retry delays

For dag, we name our dag: orders\_full\_load, we are saying that default\_args are equal to the args above, and our description, it is my Orders table ETL DAG, and the schedule we want it to run daily

and use the python operator to call our function from the orders\_etl\_logic file. We define a function called run\_etl, all it does is run our python file which has a Python callable main.

we have set dependencies and we only wanted to run one task, which is a run\_etl

```

Ex_Files_ETL_Python_SQL > DAGS > orders_dag.py > ...
1  from datetime import timedelta
2  from pendulum import datetime
3  from airflow import DAG
4  from airflow.operators.python import PythonOperator
5  from orders_etl_logic import main
6
7  default_args = {
8      'owner': 'airflow',
9      'depends_on_past': False,
10     'email_on_failure': False,
11     'email_on_retry': False,
12     'retries': 1,
13     'retry_delay': timedelta(minutes=5),
14 }
15
16 dag = DAG(
17     dag_id="orders_full_load",
18     default_args=default_args,
19     description='My Orders Table ETL DAG',
20     schedule='@daily',
21     start_date=datetime(2024, 1, 1),
22     catchup=False,
23 )
24
25 run_etl= PythonOperator(
26     task_id='run_etl',
27     python_callable=main,
28     dag=dag,
29 )
30
31 run_etl

```

Now lets move the files where Airflow can see them, since Airflow is running inside Ubuntu (WSL) we need to move the two files into Ubuntu's DAGs folder:

# in Ubuntu terminal (venv on)

echo \$AIRFLOW\_HOME

mkdir -p "\$AIRFLOW\_HOME/dags"

# copy from Windows to WSL



```
cp "/mnt/c/Users/.../DAGS/orders_etl_logic.py" "$AIRFLOW_HOME/dags/"
```

```
cp "/mnt/c/Users/.../DAGS/orders_dag.py" "$AIRFLOW_HOME/dags/"
```

```
(.venv) ffiguerola@DESKTOP-162K5R2:~/airflow_project$ cp "/mnt/c/Users/FRAN_OneDrive/Documents/AAA Python - Panda - ETL/Ex_Files_ETL_Python_SQL/Ex_Files_ETL_Python_SQL/DAGS/orders_etl_logic.py" "$AIRFLOW_HOME/dags/"
(.venv) ffiguerola@DESKTOP-162K5R2:~/airflow_project$ cp "/mnt/c/Users/FRAN_OneDrive/Documents/AAA Python - Panda - ETL/Ex_Files_ETL_Python_SQL/Ex_Files_ETL_Python_SQL/DAGS/orders_dag.py" "$AIRFLOW_HOME/dags/"
(.venv) ffiguerola@DESKTOP-162K5R2:~/airflow_project$
```

(we can also download the extension WSL of Microsoft from VS code so we can connect our WSL to VS code and run the python code in the home library of ubuntu)

Then we go to check our Airflow Web UI and we will see that is a new Dag created (we had 79 plus the one we just created there is a total of 80)

**Dags** Runs Task Instances

Search Dags

All Failed Running Success All Filter by tag

80 Dags

orders\_full\_load

| Schedule  | Latest Run | Next Run             |
|-----------|------------|----------------------|
| 0 0 * * * |            | 2025-08-10, 17:00:00 |

| Schedule  | Latest Run           | Next Run             |
|-----------|----------------------|----------------------|
| 0 0 * * * | 2025-08-11, 17:37:25 | 2025-08-12, 17:00:00 |

orders\_full\_load

Schedule: 0 0 \* \* \* Latest Run: 2025-08-09, 17:34:04 - 2025-08-10, 17:34:04 Next Run: 2025-08-10, 17:00:00 Owner: airflow Tags: Reparse Dag Latest Dag Version: v1

Overview Runs Tasks Backfills Events Code Details

Last 24 hours

Failed Tasks

Failed Runs

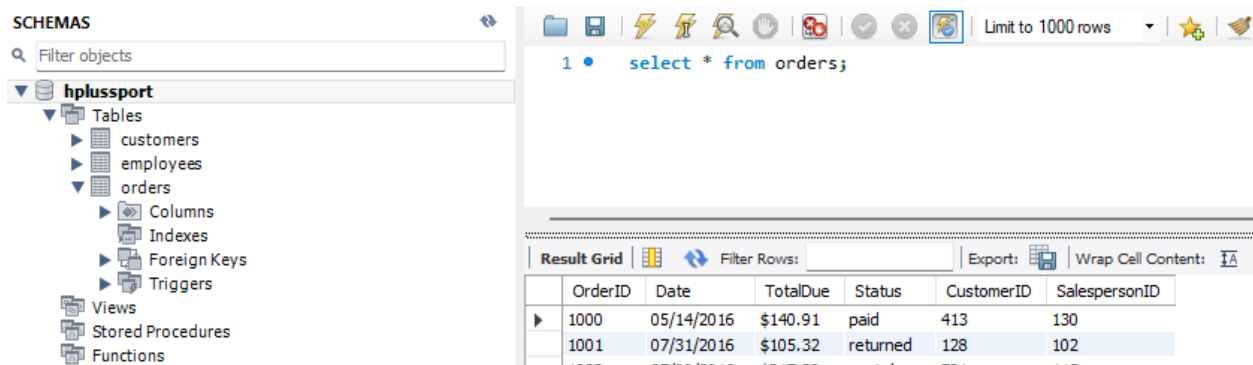
Last 0 Dag Runs

Duration (seconds)

Run After

After click in our new dag: oreders\_full\_load we have to click on trigeer button to airflow do the following:

- Airflow triggers → task starts.
- Your code builds SQLAlchemy Engine from Airflow connection.
- Pandas sees SQLAlchemy → uses SA integration path.
- Data writes successfully to MySQL.



We have successfully brought our orders table to our MySQL database, unaided

In this course we have explored:

The fundamentals of pandas, for data ingestion, manipulation, and transformation.

We have also discussed best practices for data extraction, transformation and loading

We learned how to design and implement ETL workflows

And gained hands-on experience with Apache Airflow using WSL/ubuntu for workflow orchestration

