Northwestern
INFORMATION TECHNOLOGY

Databases, SQL, and Django

Scott Coughlin, Senior Computational Specialist

Some slides adapted from Professor Michael Coughlin and Matthew Graham

June 4th 2024

Databases, SQL, and Django

Topics Covered

1. Tabular Data
2. Servers, Databases, Schema, Relations, OH MY!
3. Doing Things the Structured Query Language (SQL) Way
4. Django
5. How to use `docker-compose`

Tabular Data

Tabular Data

```
==> IMDB-directors.csv <==
director_id,first_name,last_
name
1,Todd,1
2,Les,12 Poissons
3,Lejaren,a'Hiller
4,Nian,A
```

```
==> IMDB-movies.csv <==
movie_id,name,year,rank
0,#28,2002,0.0
1,"#7 Train: An Immigrant
Journey, The",2000,0.0
2,$,1971,6.4
3,"$1,000 Reward",1913,0.0
```

```
==> IMDB-movies_directors.csv <==
director_id,movie_id
1,378879
2,281325
3,30621
3,304743
```

```
==> IMDB-movies_genres.csv <==
movie_id,genre
1,Documentary
1,Short
2,Comedy
2,Crime
```

Tabular Data

What if you had something that could not only manage all of those tables, i.e. adding new rows, but also could manage the way that those tables relate to one another in a hyper efficient way?

That is where a Database Management System comes in handy.

Some Popular Database Management System

- Different DBMS:
  - Microsoft/Sybase
  - MySQL
  - Oracle
  - PostgreSQL
  - Redis, Hadoop

Servers, Databases, Schema, Relations, OH MY!

Jargon For Days and Days and Days

- The Server
- The Database
- A Database Management System
- Relations (tables)
- Attributes (columns)
- Domain (constraints on values for a column)
- Schema

What is the server?

It is the physical (or virtual) machine where the database (and the application managing the database lives).

- “The database runs at/lives at xxxx.ciera.northwestern.edu”

What is a database?

- A structured collection of data residing on a computer system (server) that can be easily accessed, managed and updated.
- Data is organized according to a database model.
- A Database Management System (DBMS) is a software package designed to store and manage databases.

Some Popular Database Management System

- Different DBMS:
  - Microsoft/Sybase
  - MySQL
  - Oracle
  - PostgreSQL
  - Redis, Hadoop

Why use a Database Management System?

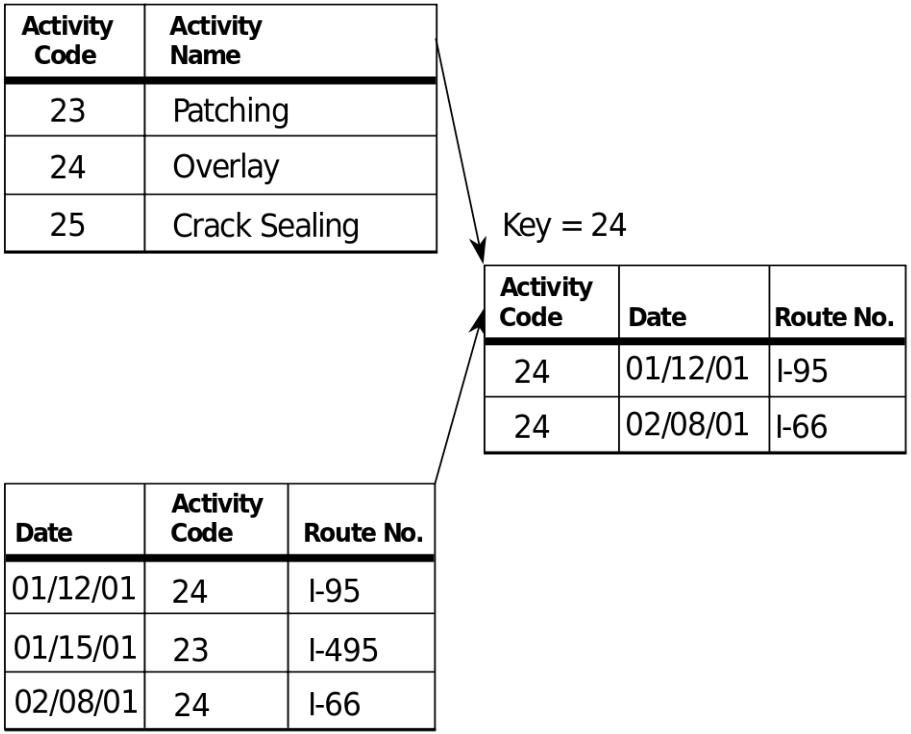
- Provides concurrent access
- data scalability, expandability and flexibility
- Security through managing access to what databases, tables, and even types of queries a individual user can make.
- efficient memory management and indexing of the data
- Integrity constraints

How do we model our data? With tables (or relations)

- Data is organized as **relations (tables)**, **attributes (columns)** and **domains (type)**
- A **relation** is a table with columns (attributes) and rows (tuples)
- The **domain** is the set of values that the attributes are allowed to take
- Within the relation, each row is unique, the column order does not matter, and each row contains a single value for each of its columns

Relational Example

Relational Model

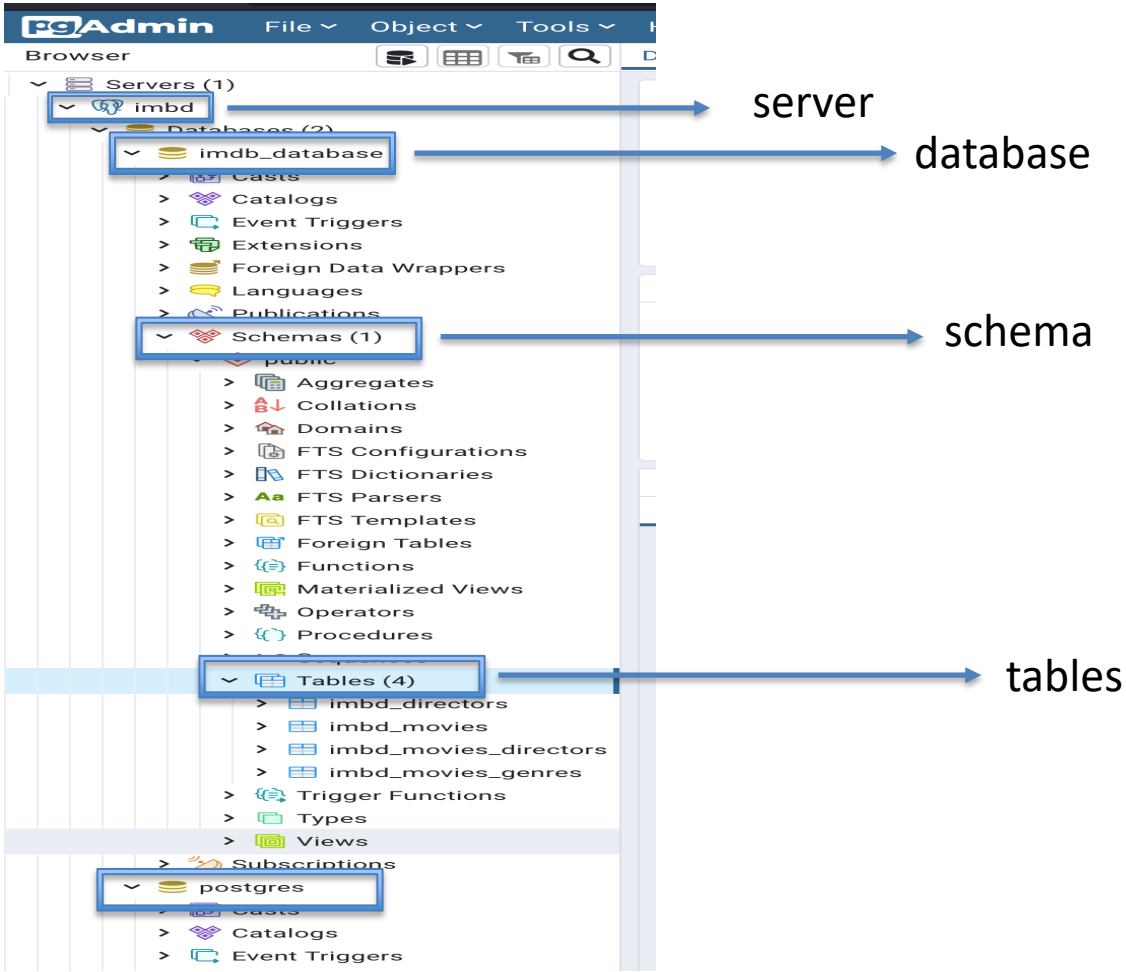


What is the “Schema”

- The **schema** is a named collection of tables.
- A schema can contain many additional pieces of information relevant to a collection of tables including views, indexes, sequences, data types, operators, and functions.

Say What Now?

Say What Now?


pgAdmin
File
Object
Tools
Browser
Servers (1)
imbd
imdb_database
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Schemas (1)
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Tables (4)
imbd_directors
imbd_movies
imbd_movies_directors
imbd_movies_genres
Trigger Functions
Types
Views
Subscriptions
postgres
Catalogs
Event Triggers
server
database
schema
tables

Doing things the Structured Query Language (SQL) Way

Structured Query Language

- Different flavors:
  - DBMS – Flavor of SQL
  - Microsoft/Sybase - Transact-SQL
  - MySQL - MySQL
  - Oracle - PL/SQL
  - PostgreSQL - PL/pgSQL

SELECT

SELECT *column1, column2* FROM *table* WHERE *condition* (LIMIT *#ofrows*) ORDER BY *sort_expression [ASC | DESC];*

- `SELECT name, constellation FROM star WHERE dec > 0 ORDER BY vmag;`
- `SELECT * FROM star WHERE ra BETWEEN 0 AND 90;`
- `SELECT DISTINCT constellation FROM star;`
- `SELECT name FROM star LIMIT 5 ORDER BY vmag;`

JOIN

- Inner join: combining related rows

```
1.  SELECT * FROM imbd_movies as s INNER JOIN imbd_movies_genres as t ON
    s.movie_id = t.movie_id;

2.  SELECT * FROM star s, stellarTypes t WHERE s.stellarType = t.id;
```

- Outer join: each row does not need a matching row

```
1. SELECT * from star s LEFT OUTER JOIN stellarTypes t ON s.stellarType = t.id;

2. SELECT * from star s RIGHT OUTER JOIN stellarTypes t ON s.stellarType = t.id;

3. SELECT * from star s FULL OUTER JOIN stellarTypes t ON s.stellarType = t.id;
```

Aggregate Functions

COUNT, AVG, MIN, MAX, SUM

```
SELECT COUNT(*) FROM star;

SELECT AVG(vmag) FROM star;

SELECT stellarType, MIN(vmag), MAX(vmag) FROM star GROUP BY
stellarType;

SELECT stellarType, AVG(vmag), COUNT(id) FROM star GROUP BY
stellarType HAVING vmag > 14;
```

Create

- CREATE DATABASE *databaseName;*
- CREATE TABLE *tableName* (name1 type1, name2 type2, . . .);

```
CREATE TABLE star (name varchar(20), ra float, dec float, vmag float);
```

- Data types:
  - Boolean, bit, tinyint, smallint, int, bigint;
  - real/float, double, decimal;
  - char, varchar, text, binary, blob, longblob;
  - date, time, datetime, timestamp

```
CREATE TABLE star (name varchar(20) not null, ra float default 0, ...);
```

KEYS

A **primary key** is a unique identifier for a row and is automatically not null

```
CREATE TABLE star (name varchar(20), ra float, dec float,
vmag float, CONSTRAINT PRIMARY KEY (name));
```

A **foreign key** is a referential constraint between two tables identifying a column in one table that refers to a column in another table.

```
CREATE TABLE star (name varchar(20), ...,
stellarType varchar(8), CONSTRAINT stellarType_fk
FOREIGN KEY (stellarType) REFERENCES
stellarTypes(id));
```

Show and Describe

SHOW ...

```
SHOW INDEXES IN star;

SHOW WARNINGS;
```

DESCRIBE…

```
DESCRIBE star;
```

INSERT INTO *table* VALUES(val1, val2, …);

```
INSERT INTO star VALUES('Sirius', 101.287, -16.716,
-1.47);

INSERT INTO star(name, vmag) VALUES('Canopus', -
0.72);

INSERT INTO star SELECT ...;
```

DELETE

DELETE FROM *table* WHERE *condition;*

DROP TABLE *table;*

```
DELETE FROM star WHERE name = 'Canopus';
DELETE FROM star WHERE name LIKE 'C_n%';
```

UPDATE

UPDATE *table* SET *column* = val1 WHERE condition;

```
UPDATE star SET vmag = vmag + 0.5;

UPDATE star SET vmag = -1.47 WHERE name LIKE 'Sirius';

UPDATE star INNER JOIN temp on star.id = temp.id SET star.vmag =
temp.mag;
```

ALTER

ALTER TABLE *table …;*

```
ALTER TABLE star ADD COLUMN bmag double AFTER vmag;

ALTER TABLE star DROP COLUMN bmag;
```

Django

Django

It seems like setting up and managing these databases would require learning a whole other language…can I not do that?

YES! Using a Python program like Django (or flask)

Django

Do either of these print outs make you think of anything (from a Python programming perspective).

```
imdb_database=> SELECT * FROM imdb_movies LIMIT 2;
 movie_id |                  name                   | year | rank
----------+-----------------------------------------+------+------
        0 | #28                                     | 2002 |    0
        1 | #7 Train: An Immigrant Journey, The     | 2000 |    0
```

```
(base) scottycoughlin@EDMRCSLC02G23LAMD6R SQL % head -n 3 IMDB-movies.csv
movie_id,name,year,rank
0,#28,2002,0.0
1,"#7 Train: An Immigrant Journey, The",2000,0.0
```

Some Hints

- Think about what a table is a bit more abstractly…
- It relates to something you are learning about this week…

Django

The best way to think about a table, is from an *Object-Oriented Programming Perspective*.

The table is a **class** and the columns are its **attributes**.

```
class ImdbMovies(models.Model):
    movie_id = models.BigIntegerField(primary_key=True)
    name = models.TextField(blank=True, null=True)
    year = models.BigIntegerField(blank=True, null=True)
    rank = models.FloatField(blank=True, null=True)

    class Meta:
        db_table = 'imdb_movies'
```

Django

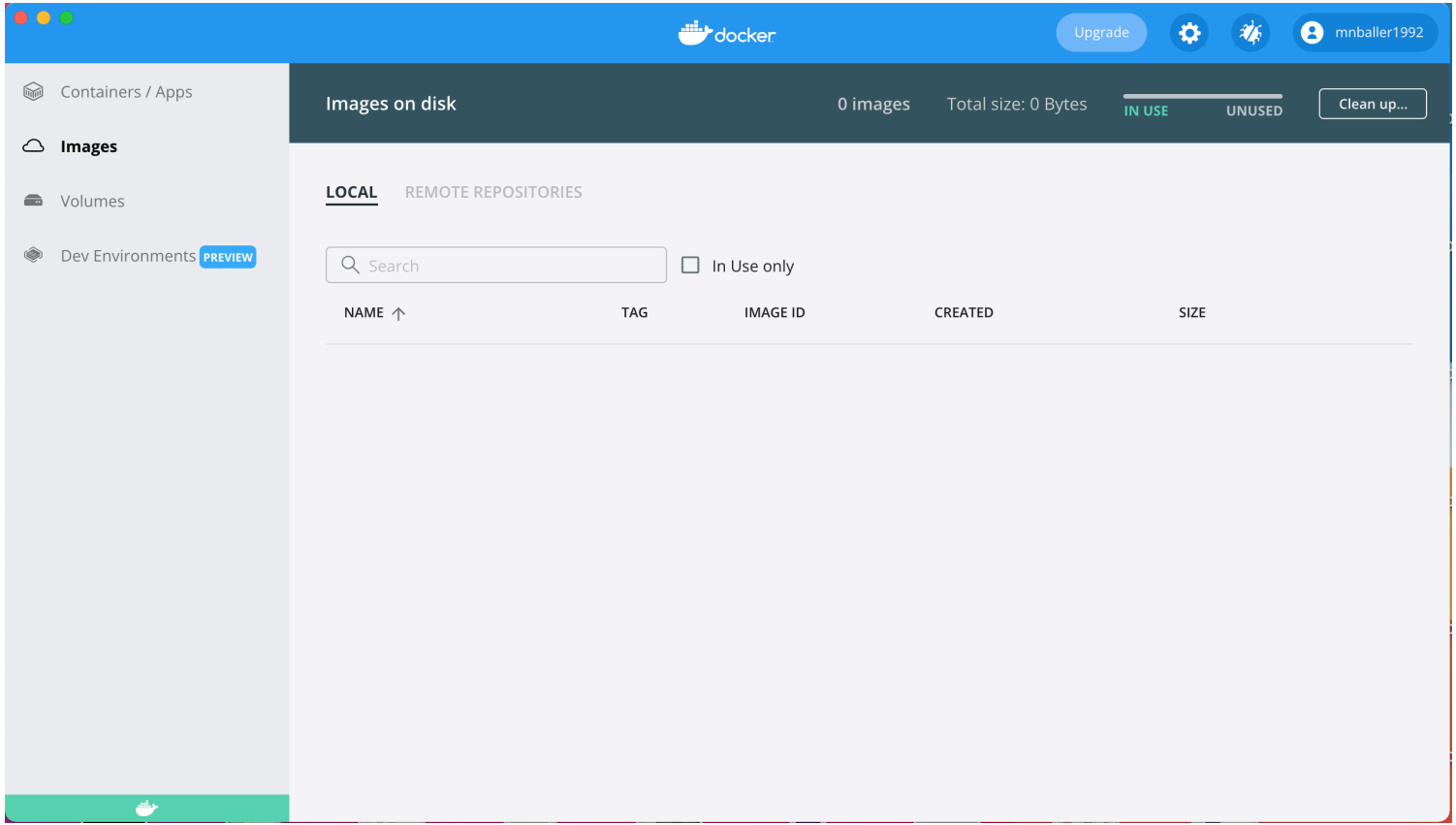
This makes creating the relationships between the tables clearer as you can assign the values for one class as the column for a different class.

Below the column `movie_id` in the table `ImdbMoviesGenre` is the `ImdbMovies` class/table.

```
class ImdbMoviesGenre(models.Model):
    movie_id = models.ForeignKey(ImdbMovies, on_delete=models.CASCADE)
    genre = models.TextField(blank=True, null=True)

    class Meta:
        db_table = 'imdb_movies_genre'
```

How to use `docker-compose`

Launch DockerHub



Open Terminal or PowerShell

```
(base) scottycoughlin@EDMRCSLC02G23LAMD6R SQL % cd ~/Documents/GitHub/Session-21/
Day2/SQL/docker
(base) scottycoughlin@EDMRCSLC02G23LAMD6R docker % cat README.md
# Launch Docker Desktop
In your applications folder, you can double click Docker Desktop and wait for it
to launch

# Launch Docker Containers
From the current folder, i.e. `Session-21/Day2/SQL/docker`, run
```
docker-compose up --detach
```
```

docker-compose

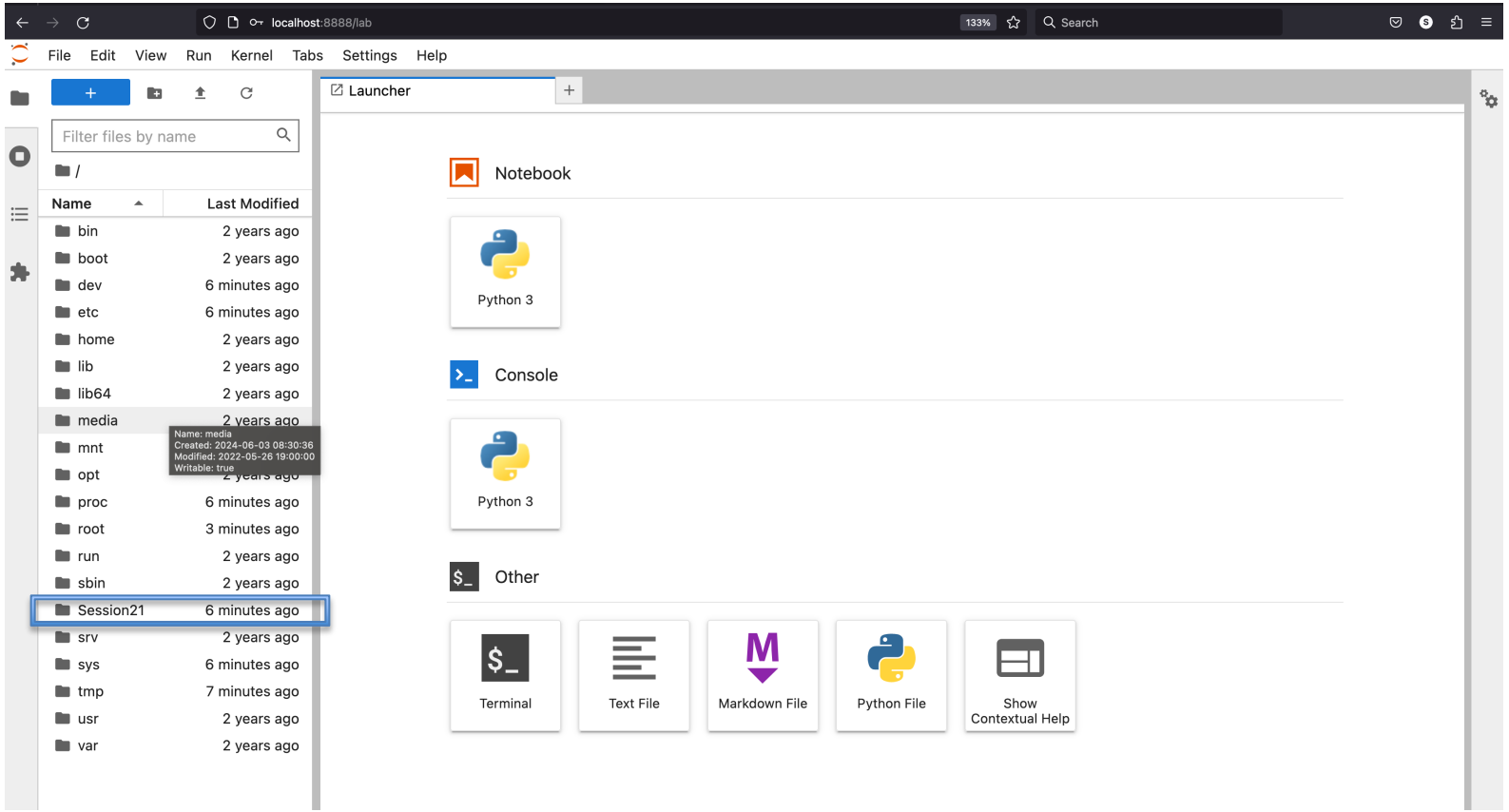
```
# Mac or Linux
$ cd Session-21/Day2/SQL/docker
$ docker-compose up --detach

# Power Shell
$ cd Session-21\Day2\SQL\docker
$ docker-compose up --detach
```

- Three things are happening once you run this command!
  - A container with PostgresSQL installed is being downloaded, a database is being created and a table is being made for each “sheet” of IMBD data.
  - A useful web application, pgAdmin, is also being downloaded so that you can interact with Postgres via your browser.
  - A container with JupyterLab and Django is being made.

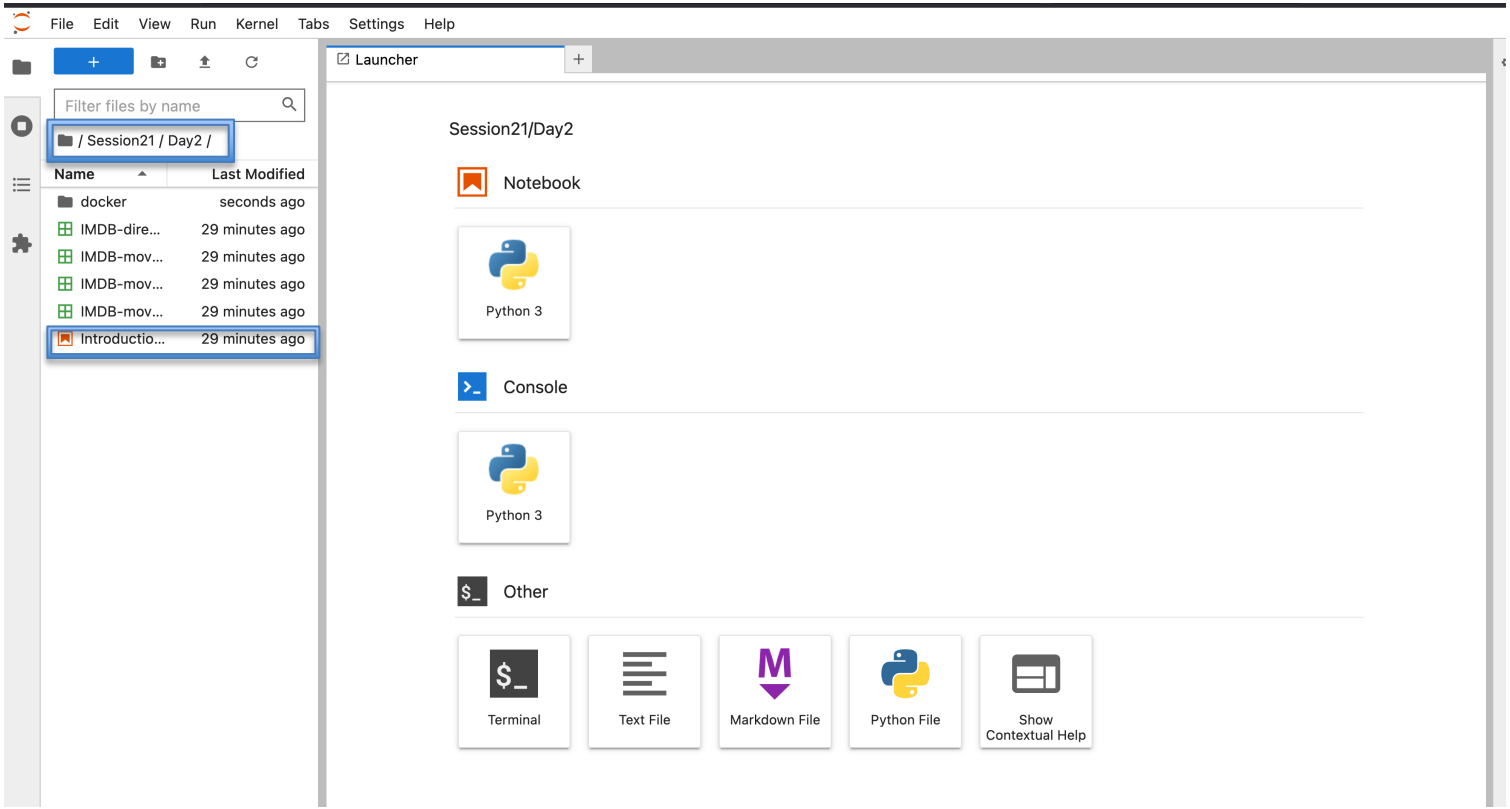
JupyterLab

- Wait about 3 minutes and then…
- In your browser go to localhost:8888
- Password: Session21

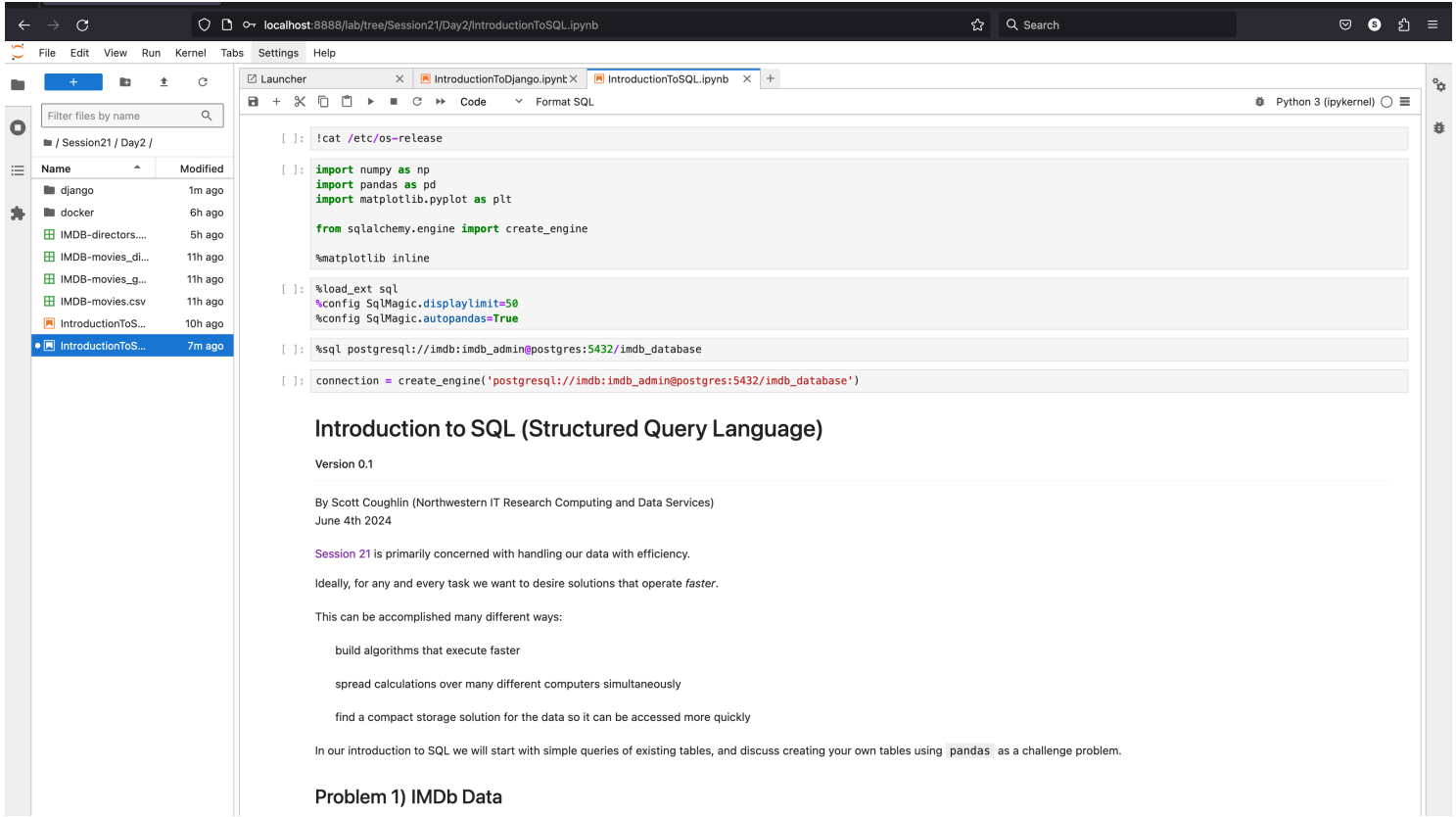
JupyterLab


localhost:8888/lab
133%
Search
File Edit View Run Kernel Tabs Settings Help
Launcher
Filter files by name
/
Name
Last Modified
bin 2 years ago
boot 2 years ago
dev 6 minutes ago
etc 6 minutes ago
home 2 years ago
lib 2 years ago
lib64 2 years ago
media 2 years ago
Name: media
Created: 2024-06-03 08:30:36
Modified: 2022-05-26 19:00:00
Writable: true
mnt
opt
proc 6 minutes ago
root 3 minutes ago
run 2 years ago
sbin 2 years ago
Session21 6 minutes ago
srv 2 years ago
sys 6 minutes ago
tmp 7 minutes ago
usr 2 years ago
var 2 years ago
Notebook
Python 3
Console
Python 3
Other
Terminal
Text File
Markdown File
Python File
Show Contextual Help

JupyterLab


File Edit View Run Kernel Tabs Settings Help
Launcher
Filter files by name
/ Session21 / Day2 /
Name
Last Modified
docker seconds ago
IMDB-dire... 29 minutes ago
IMDB-mov... 29 minutes ago
IMDB-mov... 29 minutes ago
IMDB-mov... 29 minutes ago
Introductio... 29 minutes ago
Session21/Day2
Notebook
Python 3
Console
Python 3
Other
Terminal
Text File
Markdown File
Python File
Show Contextual Help

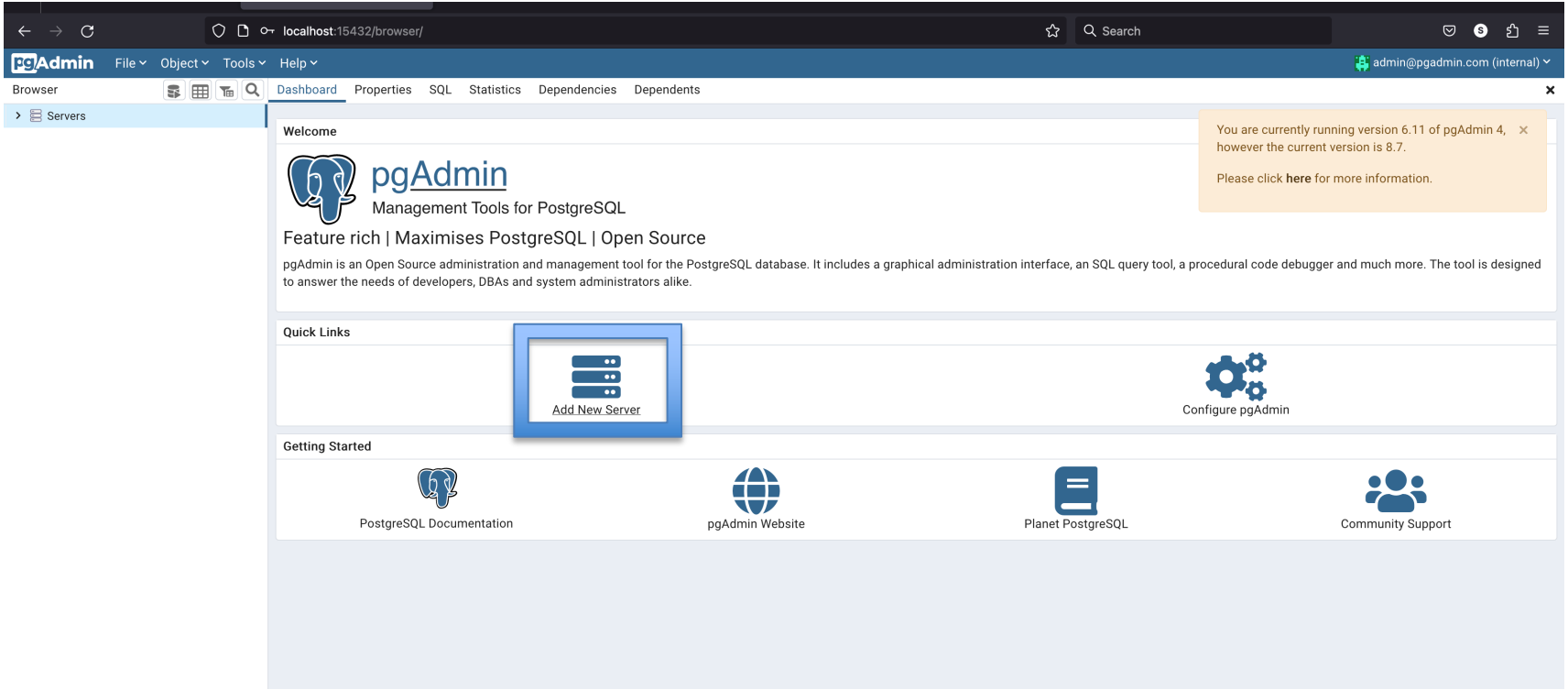
JupyterLab


localhost:8888/lab/tree/Session21/Day2/IntroductionToSQL.ipynb
Search
File Edit View Run Kernel Tabs Settings Help
Launcher
IntroductionToDjango.ipynb
IntroductionToSQL.ipynb
Code
Format SQL
Python 3 (ipykernel)
Filter files by name
/ Session21 / Day2 /
Name
Modified
django 1m ago
docker 6h ago
IMDB-directors.... 5h ago
IMDB-movies_di... 11h ago
IMDB-movies_g... 11h ago
IMDB-movies.csv 11h ago
IntroductionToS... 10h ago
IntroductionToS... 7m ago
[ ]: !cat /etc/os-release
[ ]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sqlalchemy.engine import create_engine
%matplotlib inline
[ ]: %load_ext sql
%config SqlMagic.displaylimit=50
%config SqlMagic.autopandas=True
[ ]: %sql postgresql://imdb:imdb_admin@postgres:5432/imdb_database
[ ]: connection = create_engine('postgresql://imdb:imdb_admin@postgres:5432/imdb_database')
Introduction to SQL (Structured Query Language)
Version 0.1
By Scott Coughlin (Northwestern IT Research Computing and Data Services)
June 4th 2024
Session 21 is primarily concerned with handling our data with efficiency.
Ideally, for any and every task we want to desire solutions that operate faster.
This can be accomplished many different ways:
build algorithms that execute faster
spread calculations over many different computers simultaneously
find a compact storage solution for the data so it can be accessed more quickly
In our introduction to SQL we will start with simple queries of existing tables, and discuss creating your own tables using pandas as a challenge problem.
Problem 1) IMDb Data

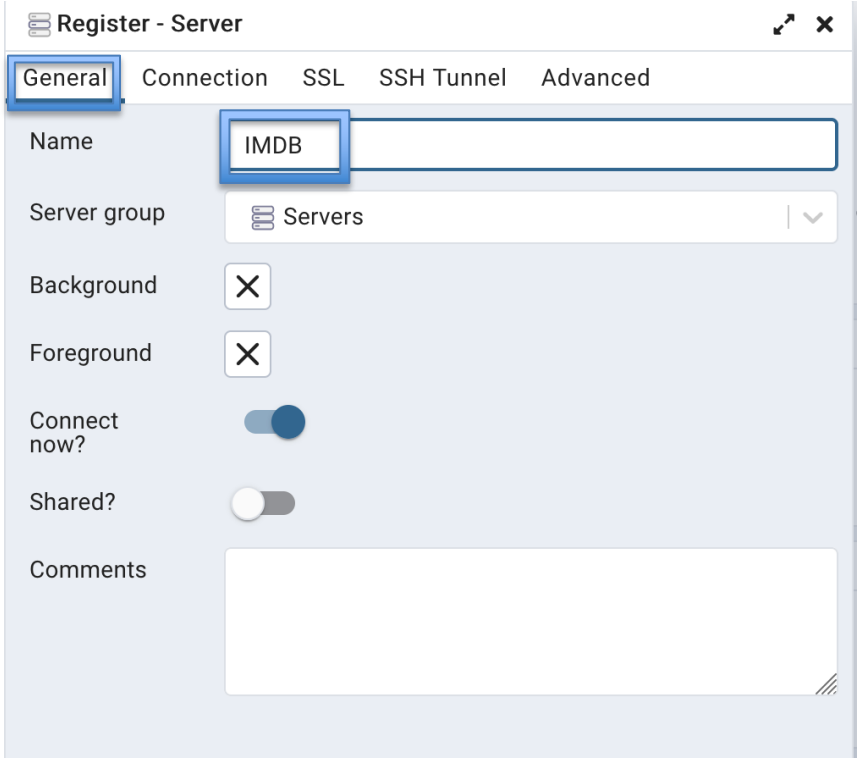
pgAdmin Portal

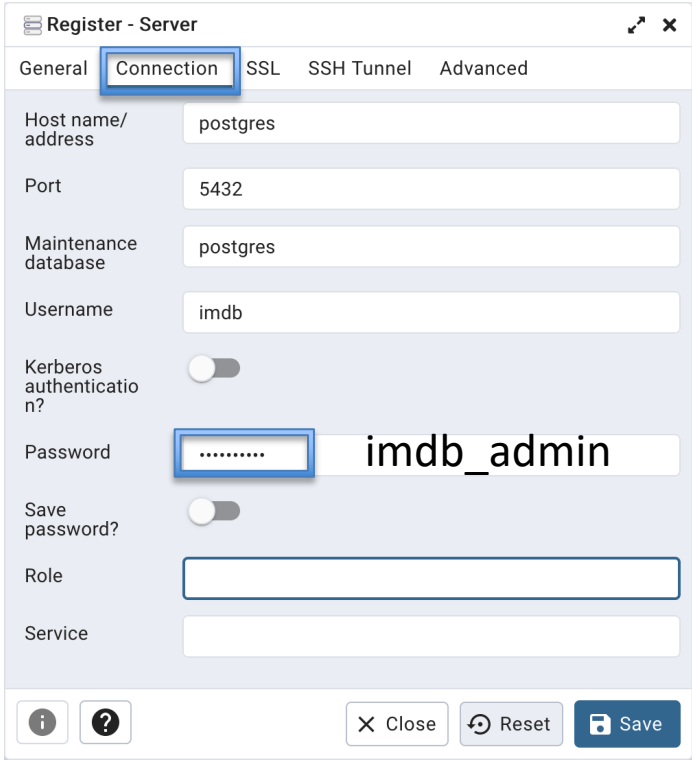
- In your browser go to localhost:15432
- E-mail: admin@pgadmin.com
- Password: password

pgAdmin Portal



pgAdmin Portal





pgAdmin Portal