SQL - Implementation of a Relational Database

This hands-on consists of three main parts:

The first one is about creating a database and manipulating information in it. The second one involves setting up a more complex database and retrieving information from it. The last part is to automate database manipulation by putting the commands into a Python script.

[oracle site: https://www.oracle.com/database/what-is-database/]: A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database.

Database Creation for the hands-on:

to complete our hands-on, we are going to install PostgreSQL, which is an open-source, advanced object-relational DBMS, using its Docker image. To do this, we create a docker-compose.yaml file in a dedicated folder and define the following setup:

```
version: '3.8'
services:
  db:
    container_name: pg_container
    image: postgres:16-alpine
    restart: always
    environment:
      POSTGRES_USER: afa_user
      POSTGRES_PASSWORD: afa_password
      POSTGRES_DB: dst_db
    ports:
      - "5432:5432"
  pgadmin:
    container_name: pgadmin4_container
    image: dpage/pgadmin4
    restart: always
    environment:
      PGADMIN_DEFAULT_EMAIL: afa@gmail.com
      PGADMIN_DEFAULT_PASSWORD: afa_data_engineer
    ports:
      - "5050:80"
```

To improve the docker-compose file, let's move sensitive information into an environment file *.env*.

```
POSTGRES_USER=afa_user
POSTGRES_PASSWORD=afa_password
```

```
POSTGRES_DB=dst_db
PGADMIN_DEFAULT_EMAIL=afa@gmail.com
PGADMIN_DEFAULT_PASSWORD=afa_data_engineer
```

The best practice is to hide the *.env* file by adding it to the *.gitignore* file, while creating a *.env.example* file that will serves as documentation. This file should have the same structure as the *.env* file but without real values as follows :

```
POSTGRES_USER=your_user
POSTGRES_PASSWORD=your_password
POSTGRES_DB=your_db_name
PGADMIN_DEFAULT_EMAIL=your_email
PGADMIN_DEFAULT_PASSWORD=your_pgadmin_password
```

Then we modify the docker-compose.yml to use these environment variables:

```
version: '3.8'
services:
  db:
    container_name: pg_container
    image: postgres:16-alpine
    restart: always
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    ports:
      - "5432:5432"
  pgadmin:
    container_name: pgadmin4_container
    image: dpage/pgadmin4
    restart: always
    environment:
      PGADMIN_DEFAULT_EMAIL: ${PGADMIN_DEFAULT_EMAIL}
      PGADMIN_DEFAULT_PASSWORD: ${PGADMIN_DEFAULT_PASSWORD}
    ports:
      - "5050:80"
```

To run the docker-compose file :

```
docker-compose up -d
```

Figure 1: Launch our containers

to check the running containers :

```
docker ps
```

Here is the output showing the docker containers that are currently running :

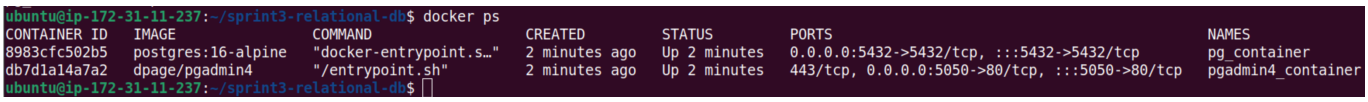


Figure 1: output running containers

We see that we have two running containers, named *pg_container* and *pgadmin4_container*, respectively.

To access the container and run bash commands :

```
docker exec -it pg_container bash
```

To create the database :

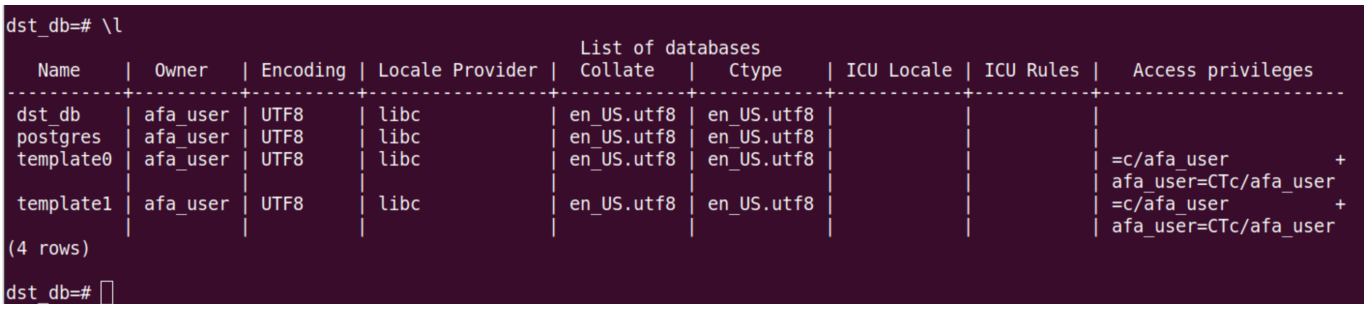
```
createdb -h localhost -U afa_user <db_name>
```

To connect to the DB :

```
psql -h localhost -U afa_user <db_name>
```

to list the existing DB :

```
\l
```



```
dst_db=# \l
                                                List of databases
   Name    |  Owner   | Encoding | Locale Provider |  Collate   |   Ctype    | ICU Locale | ICU Rules |   Access privileges
-----------+----------+----------+-----------------+------------+------------+------------+-----------+-----------------------
 dst_db    | afa_user | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           |
 postgres  | afa_user | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           |
 template0 | afa_user | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           | =c/afa_user          +
           |          |          |                 |            |            |            |           | afa_user=CTc/afa_user
 template1 | afa_user | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           | =c/afa_user          +
           |          |          |                 |            |            |            |           | afa_user=CTc/afa_user
(4 rows)

dst_db=#
```

Figure 2: Display the existing databases

To create a database :

```
CREATE DATABASE <db_name>;
```

Part 1: Creating the tables:

Considering the dataset for Anime that we present in the table bellows :

Anime_ID	English_name	Score	Genres	Type	Episodes	Aired	Premiered	Producers	Studios	Source	Duration	Rating	Ranked	Popularity
1	Cowboy Bebop	8.78	Action, Adventure, Drama, Sci-Fi, Space	TV	26	Apr 3, 1998 to Apr 24, 1999	Spring 1998	Bandai Visual	Sunrise	Original	24 min. per ep	R - 17+ (violence & profanity)	28.0	39
2	Cowboy Bebop:The Movie	8.39	Action, Drama, Sci-Fi, Space	Movie	1	Sep 1, 2001		Sunrise, Bandai Visual	Bones	Original	1 hr. 55 min.	R - 17+ (violence & profanity)	159.0	518
3	Naruto	7.91	Action, Adventure, Shounen	TV	220	Oct 3, 2002 to Feb 8, 2007	Fall 2002	TV Tokyo, Shueisha	Studio Pierrot	Manga	23 min. per ep.	PG-13 - Teens 13 or older	660.0	8
4	One Piece	8.52	Action, Adventure, Shounen	TV		Oct 20, 1999 to ?	Fall 1999	Fuji TV, Shueisha	Toei Animation	Manga	24 min.	PG-13 - Teens 13 or older	95.0	31
5	Mobile Suit Gundam SEED	7.79	Action, Drama,Military, Sci-Fi, Space	TV	50	Oct 5, 2002 to Sep 27, 2003	Fall 2002	Sotsu, Sony Music Entertainment	Sunrise	Original	24 min. per ep.	R - 17+ (violence & profanity)	850.0	1057
6	Mobile Suit Gundam SEED Destiny	7.22	Action, Drama, Military, Sci-Fi, Space	TV	50	Oct 9, 2004 to Oct 1, 2005	Fall 2004	Sotsu, Sony Music Entertainment	Sunrise	Original	24 min. per ep.	R - 17+ (violence & profanity)	2687.0	1530

Figure 4: Anime dataset

Using the MERISE method, we are building an Logical Data Model (LDM) for this dataset while following the rules.

MERISE is a method for designing, developing, and implementing IT projects. It is based on the separation of data and processes into multiple conceptual and physical models.

The figure below illustrates the different steps through which we pass from the dataset to achieve the Physical Data Model (Modèle Physique de Données), which is the final step of database design before implementation.

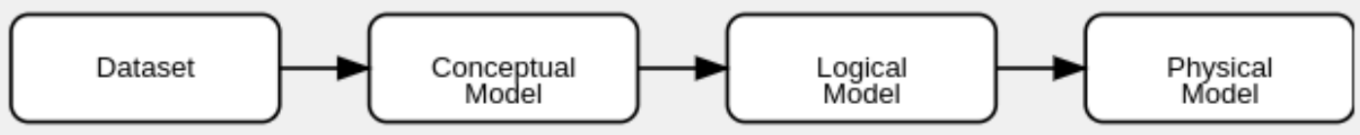


Figure 5: steps of Merise method

Construction of the Logical Data Model (MLD - Modèle Logique de Données)

The proposed structure is justified by normalization principles, separation of concerns, relationship management between entities, and model scalability. It ensures the creation of a robust database that is easy to maintain and evolve.

1. Normalization Normalization is a process aimed at eliminating data redundancy and organizing data in a consistent manner. Here's how it applies in this case:

Eliminating Redundant Data In the initial table, columns such as Genres, Producers, and Studios contain lists of values separated by commas (e.g., [Action, Adventure, Drama]). This violates the First Normal Form (1NF), which requires that each column contain atomic (indivisible) values.

By creating separate tables for Genre, Producer, and Studio, we comply with 1NF and avoid redundancy.

Preventing Update Anomalies If genres, producers, or studios were stored directly in the Anime table, any modification (e.g., changing a studio's name) would require updating multiple rows, potentially leading to update anomalies.

With separate tables, modifications only need to be made in a single location.

2. Separation of Concerns

Each table has a clear and distinct responsibility, improving maintainability and readability of the model.

- Anime: Stores basic information about an anime (name, score, type, etc.).
- Genre: Stores the different available genres.
- Producer: Stores producer details.
- Studio: Stores studio information.
- Aired: Stores broadcasting details (start and end dates).
- Premiered: Stores premiere information (season and year).

3. Relationships Between Tables

The structure allows for clear and well-defined relationships, which are crucial for a relational database.

- One-to-Many (1-n) Relationships: An anime can have a single broadcast entry (Aired) and a single premiere entry (Premiered), but each broadcast or premiere corresponds to only one anime. This justifies the 1-n relationships between Anime and Aired, and between Anime and Premiered.

- Many-to-Many (n-n) Relationships An anime can belong to multiple genres, and a genre can be associated with multiple animes. Similarly, an anime can have multiple producers/studios, and a producer/studio can be linked to multiple animes. This justifies the use of junction tables (Anime_Genre, Anime_Producer, Anime_Studio) to properly manage n-n relationships.

4. Extensibility

The structure makes the model more flexible and scalable.

Adding New Genres, Producers, or Studios

If new genres, producers, or studios need to be added, a new row can simply be inserted into the corresponding table without modifying the database schema. Adding New Information If additional details (e.g., extra information about studios) need to be stored, they can be added to the relevant table without affecting other tables.

The following figure shows the proposed Physical Data Model, taking our arguments into account.

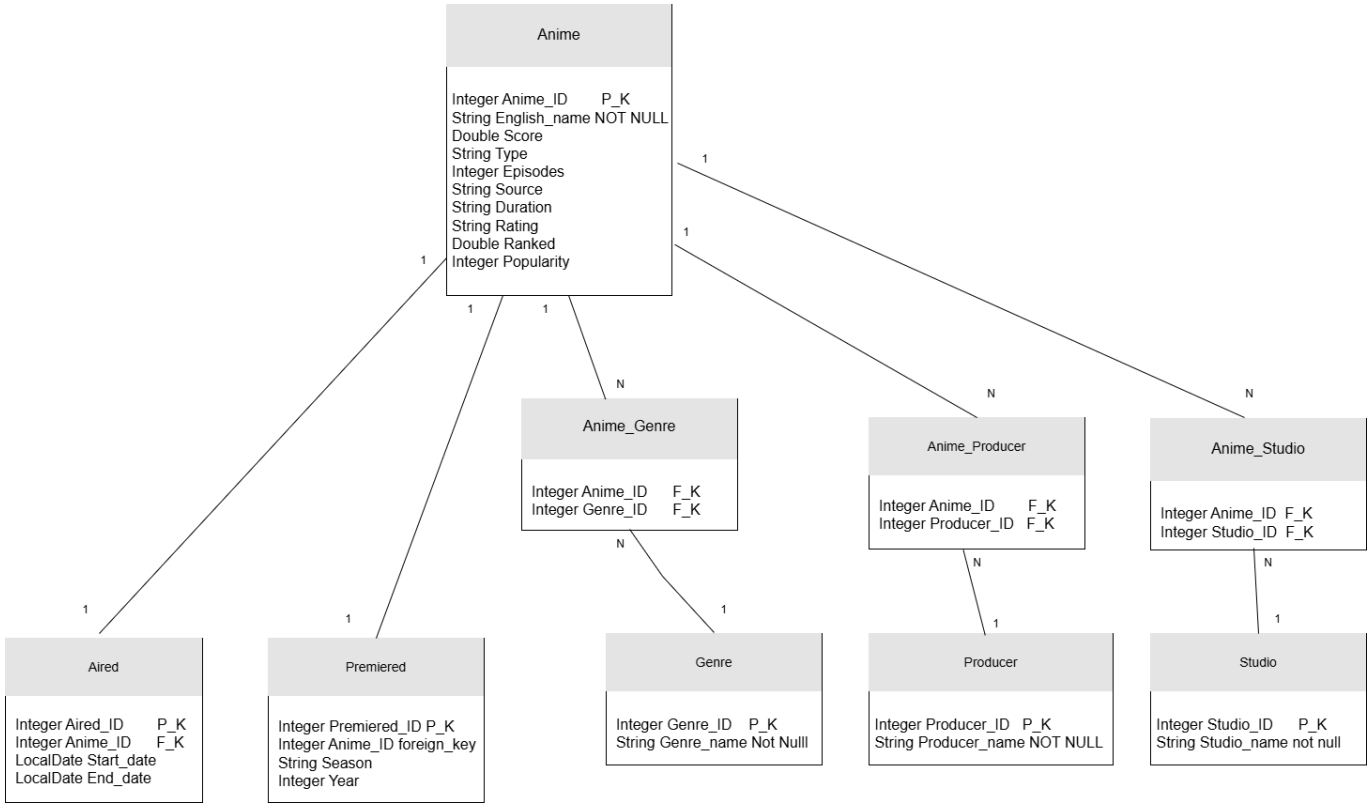


Figure 6: Physical model

```

## Creation of tables:

here is how to create tables and their relationships:

```
-- Table Anime
CREATE TABLE Anime (
    Anime_ID INT PRIMARY KEY,
    English_name VARCHAR(255) NOT NULL,
    Score FLOAT,
    Type VARCHAR(50),
    Episodes INT,
    Source VARCHAR(50),
    Duration VARCHAR(50),
```
```

```
    Rating VARCHAR(50),
    Ranked FLOAT,
    Popularity INT
);

-- Table Genre
CREATE TABLE Genre (
    Genre_ID INT PRIMARY KEY,
    Genre_name VARCHAR(50) NOT NULL
);

-- Table Producer
CREATE TABLE Producer (
    Producer_ID INT PRIMARY KEY,
    Producer_name VARCHAR(255) NOT NULL
);

-- Table Studio
CREATE TABLE Studio (
    Studio_ID INT PRIMARY KEY,
    Studio_name VARCHAR(255) NOT NULL
);

-- Table Aired
CREATE TABLE Aired (
    Aired_ID INT PRIMARY KEY,
    Anime_ID INT,
    Start_date DATE,
    End_date DATE,
    FOREIGN KEY (Anime_ID) REFERENCES Anime(Anime_ID)
);

-- Table Premiered
CREATE TABLE Premiered (
    Premiered_ID INT PRIMARY KEY,
    Anime_ID INT,
    Season VARCHAR(50),
    Year INT,
    FOREIGN KEY (Anime_ID) REFERENCES Anime(Anime_ID)
);

-- Table de liaison Anime_Genre
CREATE TABLE Anime_Genre (
    Anime_ID INT,
    Genre_ID INT,
    PRIMARY KEY (Anime_ID, Genre_ID),
    FOREIGN KEY (Anime_ID) REFERENCES Anime(Anime_ID),
    FOREIGN KEY (Genre_ID) REFERENCES Genre(Genre_ID)
);

-- Table de liaison Anime_Producer
CREATE TABLE Anime_Producer (
    Anime_ID INT,
    Producer_ID INT,
```

```
    PRIMARY KEY (Anime_ID, Producer_ID),
    FOREIGN KEY (Anime_ID) REFERENCES Anime(Anime_ID),
    FOREIGN KEY (Producer_ID) REFERENCES Producer(Producer_ID)
);

-- Table de liaison Anime_Studio
CREATE TABLE Anime_Studio (
    Anime_ID INT,
    Studio_ID INT,
    PRIMARY KEY (Anime_ID, Studio_ID),
    FOREIGN KEY (Anime_ID) REFERENCES Anime(Anime_ID),
    FOREIGN KEY (Studio_ID) REFERENCES Studio(Studio_ID)
);
```

Data insertion queries:

```
INSERT INTO Anime (Anime_ID, English_name, Score, Type, Episodes, Source,
Duration, Rating, Ranked, Popularity)
VALUES
(1, 'Cowboy Bebop', 8.78, 'TV', 26, 'Original', '24 min. per ep.', 'R - 17+
(violence & profanity)', 28.0, 39),
(2, 'Cowboy Bebop: The Movie', 8.39, 'Movie', 1, 'Original', '1 hr. 55 min.', 'R -
17+ (violence & profanity)', 159.0, 518),
(3, 'Naruto', 7.91, 'TV', 220, 'Manga', '23 min. per ep.', 'PG-13 - Teens 13 or
older', 660.0, 8),
(4, 'One Piece', 8.52, 'TV', NULL, 'Manga', '24 min.', 'PG-13 - Teens 13 or
older', 95.0, 31),
(5, 'Mobile Suit Gundam SEED', 7.79, 'TV', 50, 'Original', '24 min. per ep.', 'R -
17+ (violence & profanity)', 850.0, 1057),
(6, 'Mobile Suit Gundam SEED Destiny', 7.22, 'TV', 50, 'Original', '24 min. per
ep.', 'R - 17+ (violence & profanity)', 2687.0, 1530);
```

```
INSERT INTO Genre (Genre_ID, Genre_name)
VALUES
(1, 'Action'),
(2, 'Adventure'),
(3, 'Drama'),
(4, 'Sci-Fi'),
(5, 'Space'),
(6, 'Shounen'),
(7, 'Military');
```

```
INSERT INTO Producer (Producer_ID, Producer_name)
VALUES
(1, 'Bandai Visual'),
(2, 'Sunrise'),
(3, 'TV Tokyo'),
```

```
(4, 'Shueisha'),
(5, 'Fuji TV'),
(6, 'Sotsu'),
(7, 'Sony Music Entertainment'),
(8, 'Bones');
```

```
INSERT INTO Studio (Studio_ID, Studio_name)
VALUES
(1, 'Sunrise'),
(2, 'Bones'),
(3, 'Studio Pierrot'),
(4, 'Toei Animation');
```

```
INSERT INTO Aired (Aired_ID, Anime_ID, Start_date, End_date)
VALUES
(1, 1, '1998-04-03', '1999-04-24'),
(2, 2, '2001-09-01', NULL),
(3, 3, '2002-10-03', '2007-02-08'),
(4, 4, '1999-10-20', NULL),
(5, 5, '2002-10-05', '2003-09-27'),
(6, 6, '2004-10-09', '2005-10-01');
```

```
INSERT INTO Premiered (Premiered_ID, Anime_ID, Season, Year)
VALUES
(1, 1, 'Spring', 1998),
(2, 2, NULL, NULL),
(3, 3, 'Fall', 2002),
(4, 4, 'Fall', 1999),
(5, 5, 'Fall', 2002),
(6, 6, 'Fall', 2004);
```

```
INSERT INTO Anime_Genre (Anime_ID, Genre_ID)
VALUES
-- Cowboy Bebop
(1, 1), (1, 2), (1, 3), (1, 4), (1, 5),
-- Cowboy Bebop: The Movie
(2, 1), (2, 3), (2, 4), (2, 5),
-- Naruto
(3, 1), (3, 2), (3, 6),
-- One Piece
(4, 1), (4, 2), (4, 6),
-- Mobile Suit Gundam SEED
(5, 1), (5, 3), (5, 7), (5, 4), (5, 5),
-- Mobile Suit Gundam SEED Destiny
(6, 1), (6, 3), (6, 7), (6, 4), (6, 5);
```

```
INSERT INTO Anime_Producer (Anime_ID, Producer_ID)
VALUES
-- Cowboy Bebop
(1, 1),
-- Cowboy Bebop: The Movie
(2, 1), (2, 2),
-- Naruto
(3, 3), (3, 4),
-- One Piece
(4, 5), (4, 4),
-- Mobile Suit Gundam SEED
(5, 6), (5, 7),
-- Mobile Suit Gundam SEED Destiny
(6, 6), (6, 7);
```

```
INSERT INTO Anime_Studio (Anime_ID, Studio_ID)
VALUES
-- Cowboy Bebop
(1, 1),
-- Cowboy Bebop: The Movie
(2, 2),
-- Naruto
(3, 3),
-- One Piece
(4, 4),
-- Mobile Suit Gundam SEED
(5, 1),
-- Mobile Suit Gundam SEED Destiny
(6, 1);
```

To show list of tables

```
\dt
```

```
examen_sql=# \dt
            List of relations
 Schema |      Name      | Type  |  Owner
--------+----------------+-------+----------
 public | aired          | table | afa_user
 public | anime          | table | afa_user
 public | anime_genre    | table | afa_user
 public | anime_producer | table | afa_user
 public | anime_studio   | table | afa_user
 public | genre          | table | afa_user
 public | premiered      | table | afa_user
 public | producer       | table | afa_user
 public | studio         | table | afa_user
(9 rows)

examen_sql=#
```

Figure 6: List of tables

To show details of a table

```
\d+ <table_name>
```

```
examen_sql=# \d+ anime
                                                Table "public.anime"
    Column    |          Type          | Collation | Nullable | Default | Storage  | Compression | Stats target | Descriptio
n
--------------+------------------------+-----------+----------+---------+----------+-------------+--------------+-----------
--
 anime_id     | integer                |           | not null |         | plain    |             |              |
 english_name | character varying(255) |           | not null |         | extended |             |              |
 score        | double precision       |           |          |         | plain    |             |              |
 type         | character varying(50)  |           |          |         | extended |             |              |
 episodes     | integer                |           |          |         | plain    |             |              |
 source       | character varying(50)  |           |          |         | extended |             |              |
 duration     | character varying(50)  |           |          |         | extended |             |              |
 rating       | character varying(50)  |           |          |         | extended |             |              |
 ranked       | double precision       |           |          |         | plain    |             |              |
 popularity   | integer                |           |          |         | plain    |             |              |
Indexes:
    "anime_pkey" PRIMARY KEY, btree (anime_id)
Referenced by:
    TABLE "aired" CONSTRAINT "aired_anime_id_fkey" FOREIGN KEY (anime_id) REFERENCES anime(anime_id)
    TABLE "anime_genre" CONSTRAINT "anime_genre_anime_id_fkey" FOREIGN KEY (anime_id) REFERENCES anime(anime_id)
    TABLE "anime_producer" CONSTRAINT "anime_producer_anime_id_fkey" FOREIGN KEY (anime_id) REFERENCES anime(anime_id)
    TABLE "anime_studio" CONSTRAINT "anime_studio_anime_id_fkey" FOREIGN KEY (anime_id) REFERENCES anime(anime_id)
    TABLE "premiered" CONSTRAINT "premiered_anime_id_fkey" FOREIGN KEY (anime_id) REFERENCES anime(anime_id)
Access method: heap

examen_sql=#
```

Figure 7: Details of table Anime

To insert data from a file, create first a file named *insert-data.sql*, whithin our container, containing all the insertion code above, and then run the following command

```
\i /path/to/insert_anime.sql
```

```
examen_sql-# \i /home/insert_data.sql
psql:/home/insert_data.sql:8: ERROR:  duplicate key value violates unique constraint "anime_pkey"
DETAIL:  Key (anime_id)=(1) already exists.
INSERT 0 7
INSERT 0 8
INSERT 0 4
INSERT 0 6
INSERT 0 6
INSERT 0 25
INSERT 0 11
INSERT 0 6
examen_sql-#
```

Figure 8: output when running the script that inserts data in tables

Part II Perform SQL Query Execution

Download data using this command :

```
cd && wget https://dst-de.s3.eu-west-3.amazonaws.com/bdd_postgres_fr/database/examen.sql
```

We create a database called *examen_afa*

```
docker exec -i pg_container psql -U afa_user -d postgres -c "CREATE DATABASE examen_afa;"
```

To check that the database was created by listing all databases :

```
docker exec -i pg_container psql -U afa_user -d postgres -c "\l"
```

```
ubuntu@ip-172-31-11-237:~$ docker exec -i pg_container psql -U afa_user -d postgres -c "CREATE DATABASE examen_afa;"
CREATE DATABASE
ubuntu@ip-172-31-11-237:~$ docker exec -i pg_container psql -U afa_user -d postgres -c "\l"
                                                List of databases
    Name    |  Owner   | Encoding | Locale Provider |  Collate   |   Ctype    | ICU Locale | ICU Rules |   Access privileges
------------+----------+----------+-----------------+------------+------------+------------+-----------+-----------------------
 dst_db     | afa_user | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           |
 examen_afa | afa_user | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           |
 examen_sql | afa_user | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           |
 postgres   | afa_user | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           |
 template0  | afa_user | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           | =c/afa_user          +
            |          |          |                 |            |            |            |           | afa_user=CTc/afa_user
 template1  | afa_user | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           | =c/afa_user          +
            |          |          |                 |            |            |            |           | afa_user=CTc/afa_user
(6 rows)

ubuntu@ip-172-31-11-237:~$
```

Figure 9: listing the existing databases

To execute the SQL script on a specified database inside a running Docker container, run this command:

```
docker exec -i <container-name> psql -U <user-name> -d <db_name> < <path-to/script-sql>
```

```
docker exec -i pg_container psql -U afa_user -d exam_afa < ./data/examen.sql
```

To display the tables created whithin the database *examen_afa*, we run the command :

```
docker exec -i pg_container psql -U afa_user -d postgres -c "\dt"
```

```
ubuntu@ip-172-31-11-237:~/sprint3-relational-db$ docker exec -i pg_container psql -U afa_user -d examen_afa -c "\dt"
            List of relations
 Schema |      Name      | Type  |  Owner
--------+----------------+-------+----------
 public | abilities      | table | afa_user
 public | classification | table | afa_user
 public | pokemon        | table | afa_user
 public | pokemonability | table | afa_user
 public | pokemontype    | table | afa_user
 public | stats          | table | afa_user
 public | types          | table | afa_user
(7 rows)

ubuntu@ip-172-31-11-237:~/sprint3-relational-db$
```

Figure 10: Display tables whithin the database examan_afa

SQL queries:

For the sake of clarity, we do not run the SQL statement from outside the container using this command:

```
docker exec -it pg_container psql -U afa_user -d examen_afa -c "SQL-statement"
```

However, we prefer connect to the database first, then run sql statements.

This is the command to connect to the database *examen_afa*

SQL query 1

```
SELECT
    t.name_type AS type, COUNT(pt.pokedex_number) AS count
FROM
    pokemontype pt JOIN pokemon p
ON
    pt.pokedex_number = p.pokedex_number
JOIN
    types t
ON
    pt.type_id = t.type_id
GROUP
    BY type
ORDER
    BY count DESC;
```

This query joins the three tables (pokemon, types and pokemontype) and groups the result by type.

```
examen_afa=# SELECT t.name_type AS type, COUNT(pt.pokedex_number) AS count FROM pokemontype pt JOIN pokemon p ON pt.pokedex_number =
p.pokedex_number JOIN types t ON pt.type_id = t.type_id GROUP BY type ORDER BY count DESC;
   type   | count
----------+-------
 water    |   131
 normal   |   109
 flying   |    98
 grass    |    98
 psychic  |    82
 bug      |    77
 ground   |    66
 poison   |    66
 fire     |    65
 rock     |    59
 fighting |    53
 dark     |    50
 electric |    48
 fairy    |    47
 steel    |    46
 dragon   |    44
 ghost    |    41
 ice      |    38
(18 rows)
```

Figure 11: group pokemon by type and order them according to their number, decreasingly.

SQL query 2

List pokemon names and base stat total grater than 600 in descending order.

```
SELECT
    p.name, p.base_total
FROM
    pokemon p
WHERE
    p.base_total > 600
ORDER BY
    base_total DESC;
```

```
examen_afa=# SELECT p.name, p.base_total FROM pokemon p WHERE p.base_total > 600 ORDER BY base_total DESC;
    name    | base_total
------------+------------
 Mewtwo     |        780
 Rayquaza   |        780
 Groudon    |        770
 Kyogre     |        770
 Arceus     |        720
 Zygarde    |        708
 Diancie    |        700
 Latias     |        700
 Latios     |        700
 Metagross  |        700
 Salamence  |        700
 Kyurem     |        700
 Garchomp   |        700
 Tyranitar  |        700
 Lunala     |        680
 Lugia      |        680
 Ho-Oh      |        680
 Dialga     |        680
 Palkia     |        680
 Giratina   |        680
 Reshiram   |        680
 Zekrom     |        680
 Xerneas    |        680
 Yveltal    |        680
 Hoopa      |        680
 Solgaleo   |        680
 Regigigas  |        670
```

Figure 12: Display pokemon name and base total with two conditions.

SQL query 3

Display the average base stats of Pokemon, grouped by their type, and sorts the result in ascending order according to the average:

```
SELECT
    t.name_type AS type,
    AVG(p.base_total)::NUMERIC AS average
FROM
    pokemon p
JOIN
    pokemontype pt ON p.pokedex_number = pt.pokedex_number
JOIN
    types t ON t.type_id = pt.type_id
GROUP BY
    t.name_type
ORDER BY
    average;
```

```
examen_afa=# select t.name_type AS type, AVG(p.base_total)::NUMERIC AS average from pokemon p join pokemontype pt on p.pokedex_numbe
r=pt.pokedex_number join types t on t.type_id = pt.type_id GROUP BY type Order by average;
   type   |       average
----------+----------------------
 bug      | 380.4415584415584416
 poison   | 397.9090909090909091
 normal   | 401.6697247706422018
 grass    | 413.4081632653061224
 water    | 424.0381679389312977
 fairy    | 425.8085106382978723
 ground   | 428.1212121212121212
 electric | 440.1458333333333333
 flying   | 441.7346938775510204
 ghost    | 444.8048780487804878
 rock     | 446.9830508474576271
 ice      | 447.1315789473684211
 dark     | 447.7400000000000000
 fire     | 453.5846153846153846
 fighting | 456.5849056603773585
 psychic  | 463.7926829268292683
 steel    | 493.6739130434782609
 dragon   | 513.0000000000000000
(18 rows)

examen_afa=#
```

Figure 13: Display pokemon name and average base total with two conditions.

SQL query 4

List Pokemons with the special ability 'Overgrow' and sort them by base stats in descending order

```
SELECT
    p.name,
    p.base_total
FROM
    pokemon p
JOIN
    pokemonability pa ON p.pokedex_number = pa.pokedex_number
JOIN
    abilities a ON a.ability_id = pa.ability_id
WHERE
    a.name_ability = 'Overgrow'
ORDER BY
    p.base_total DESC;
```

```
examen_afa=# SELECT p.name, p.base_total FROM pokemon p JOIN pokemonability pa ON p.pokedex_number=pa.pokedex_number JOIN abilities
a ON a.ability_id=pa.ability_id where a.name_ability='Overgrow' ORDER BY p.base_total DESC;
    name    | base_total
------------+------------
 Sceptile   |        630
 Venusaur   |        625
 Decidueye  |        530
 Chesnaught |        530
 Serperior  |        528
 Torterra   |        525
 Meganium   |        525
 Simisage   |        498
 Dartrix    |        420
 Servine    |        413
 Grovyle    |        405
 Grotle     |        405
 Bayleef    |        405
 Quilladin  |        405
 Ivysaur    |        405
 Rowlet     |        320
 Turtwig    |        318
 Chikorita  |        318
 Bulbasaur  |        318
 Pansage    |        316
 Chespin    |        313
 Treecko    |        310
 Snivy      |        308
(23 rows)
```

Figure 14: Display pokemon with special ability in descending order according to base stats.

SQL query 5

To list pokemon by their names and their primary and secondary types, we can group the pokemon by their names, and for each name we print the types by using conditional operations on *type_id* as shown by the following query:

```
SELECT
    p.name,
    MAX(CASE WHEN pt.type_id = LEAST(pt.type_id, pt2.type_id) THEN t.name_type
END) AS primary_type,
    MAX(CASE WHEN pt2.type_id IS NOT NULL AND pt.type_id = GREATEST(pt.type_id,
pt2.type_id) THEN t.name_type END) AS secondary_type
FROM
    pokemon p
JOIN
    pokemontype pt ON p.pokedex_number = pt.pokedex_number
JOIN
    types t ON t.type_id = pt.type_id
LEFT JOIN
    pokemontype pt2 ON p.pokedex_number = pt2.pokedex_number AND pt.type_id !=
pt2.type_id
GROUP BY
    p.name
ORDER BY
    p.name;
```

```
examen_afa=# SELECT
    p.name,
    MAX(CASE WHEN pt.type_id = LEAST(pt.type_id, pt2.type_id) THEN t.name_type END) AS primary_type,
    MAX(CASE WHEN pt2.type_id IS NOT NULL AND pt.type_id = GREATEST(pt.type_id, pt2.type_id) THEN t.name_type END) AS secondary_type

FROM
    pokemon p
JOIN
    pokemontype pt ON p.pokedex_number = pt.pokedex_number
JOIN
    types t ON t.type_id = pt.type_id
LEFT JOIN
    pokemontype pt2 ON p.pokedex_number = pt2.pokedex_number AND pt.type_id != pt2.type_id
GROUP BY
    p.name
ORDER BY
    p.name;
     name     | primary_type | secondary_type
--------------+--------------+----------------
 Abomasnow    | grass        | ice
 Abra         | psychic      |
 Absol        | dark         |
 Accelgor     | bug          |
 Aegislash    | steel        | ghost
 Aerodactyl   | flying       | rock
 Aggron       | rock         | steel
 Aipom        | normal       |
 Alakazam     | psychic      |
 Alomomola    | water        |
 Altaria      | flying       | dragon
 Amaura       | ice          | rock
 Ambipom      | normal       |
```

Figure 15: Display pokemon names and their primary and secondary types.

SQL query 6

Display Pokemon with a *base_total* greater than the average *base_total* per generation:

```
SELECT
    p.name,
    p.generation,
    p.base_total AS total_stats
FROM
    pokemon p
WHERE
    p.base_total > (
        SELECT
            AVG(p2.base_total)
        FROM
            pokemon p2
        WHERE
            p2.generation = p.generation
    );
```

```
examen_afa=# SELECT
    p.name,
    p.generation,
    p.base_total AS total_stats
FROM
    pokemon p
WHERE
    base_total > (
        SELECT
            AVG(p2.base_total)
        FROM
            pokemon p2
        WHERE
            p2.generation = p.generation
    );
     name     | generation | total_stats
--------------+------------+-------------
 Venusaur     |          1 |         625
 Charizard    |          1 |         634
 Blastoise    |          1 |         630
 Beedrill     |          1 |         495
 Pidgeot      |          1 |         579
 Fearow       |          1 |         442
 Arbok        |          1 |         448
 Raichu       |          1 |         485
 Sandslash    |          1 |         450
 Nidoqueen    |          1 |         505
 Nidoking     |          1 |         505
 Clefable     |          1 |         483
 Ninetales    |          1 |         505
 Wigglytuff   |          1 |         435
 Golbat       |          1 |         455
 Vileplume    |          1 |         490
 Venomoth     |          1 |         450
```

Figure 16: Display pokemon when base_total is greater the the average base_total per generation.

SQL query 7

To find pokemon of type "fire" with attack greater than 100 :

```
SELECT
    p.name,
    s.attack
FROM
    pokemon p
JOIN
    stats s
    ON p.pokedex_number = s.pokedex_number
JOIN
    pokemontype pt
    ON p.pokedex_number = pt.pokedex_number
JOIN
    types t
    ON t.type_id = pt.type_id
WHERE
    t.name_type = 'fire'
    AND s.attack > 100;
```

```
examen_afa=# select p.name, s.attack from pokemon p join pokemontype pt on p.pokedex_number=pt.pokedex_number join types t on t.type
_id=pt.type_id join stats s on s.pokedex_number = p.pokedex_number where t.name_type='fire' and s.attack > 100;
    name    | attack
------------+--------
 Charizard  |    104
 Arcanine   |    110
 Flareon    |    130
 Entei      |    115
 Ho-Oh      |    130
 Blaziken   |    160
 Camerupt   |    120
 Infernape  |    104
 Emboar     |    123
 Reshiram   |    120
 Volcanion  |    110
 Incineroar |    115
(12 rows)

examen_afa=#
```

Figure 17: Display pokemon the type is 'fire' and the attack is greater than 100.

SQL query 8

Indiquer si le total des stats d'un Pokémon est supérieur ou inférieur à la moyenne par génération. Indicate whether a Pokémon's total stats of is great or less than the average for its generation.

```
SELECT
    p.name,
    p.generation,
    p.base_total,
    CASE
        WHEN p.base_total > (
            SELECT AVG(p2.base_total)
            FROM pokemon p2
            WHERE p2.generation = p.generation
```

```
        ) THEN 'greater than the average'
        ELSE 'less or equal to the average'
    END AS total_stats_comparison
FROM
    pokemon p;
```

```
examen_afa=# SELECT
    p.name,
    p.generation,
    p.base_total,
    CASE
        WHEN p.base_total > (
            SELECT AVG(p2.base_total)
            FROM pokemon p2
            WHERE p2.generation = p.generation
        ) THEN 'greater than the average'
        ELSE 'less or equal to the average'
    END AS total_stats_comparison
FROM
    pokemon p;
     name     | generation | base_total |    total_stats_comparison
--------------+------------+------------+------------------------------
 Bulbasaur    |          1 |        318 | less or equal to the average
 Ivysaur      |          1 |        405 | less or equal to the average
 Venusaur     |          1 |        625 | greater than the average
 Charmander   |          1 |        309 | less or equal to the average
 Charmeleon   |          1 |        405 | less or equal to the average
 Charizard    |          1 |        634 | greater than the average
 Squirtle     |          1 |        314 | less or equal to the average
 Wartortle    |          1 |        405 | less or equal to the average
 Blastoise    |          1 |        630 | greater than the average
 Caterpie     |          1 |        195 | less or equal to the average
 Metapod      |          1 |        205 | less or equal to the average
 Butterfree   |          1 |        395 | less or equal to the average
 Weedle       |          1 |        195 | less or equal to the average
 Kakuna       |          1 |        205 | less or equal to the average
 Beedrill     |          1 |        495 | greater than the average
 Pidgey       |          1 |        251 | less or equal to the average
 Pidgeotto    |          1 |        349 | less or equal to the average
```

Figure 18: Display pokemon with new column according to a given condition.

Part III : Automate all queries using a Python script :

In our docker there is no Python installed, so we start by installing Python3 on the alpine distribution.

We copy the Python script in the container using these commands:

```
docker cp ./run-queries-python.py pg_container:/home

docker cp ./queries.txt pg_container:/home
```

To connect to the container :

```
docker exec -it pg_container sh
```

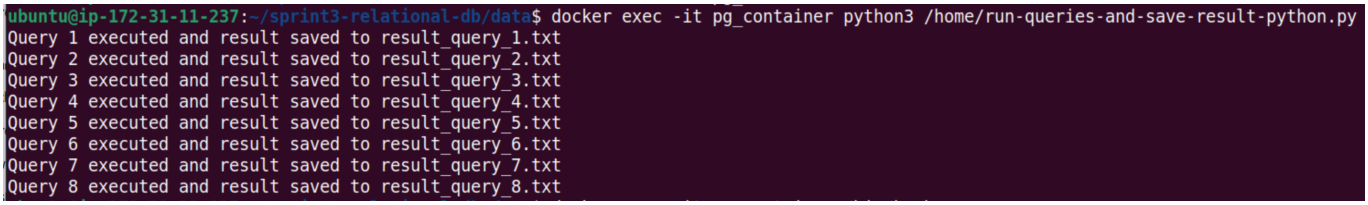
Install python3 and the required dependencies :

```
apk add --no-cache python3

apk add --no-cache py3-psycopg2
```

To run the python script, we use the following command :

```
docker exec -it pg_container python3 /home/run-queries-and-save-result-python.py
```



/19.the-output-executed-python.png

Figure 19: The output of the executed Python command.

Restaoring the Database:

To make a backup of the database :

- first create the backup inside the container, in a directory that the user *PostgreSQL* has write access to

```
docker exec -it pg_container pg_dump -U afa_user examen_afa -f /tmp/examen_afa.sql
```

- Then we copy the backup from the container to the host machine :

```
docker cp pg_container:/tmp/examen_afa.sql .
```

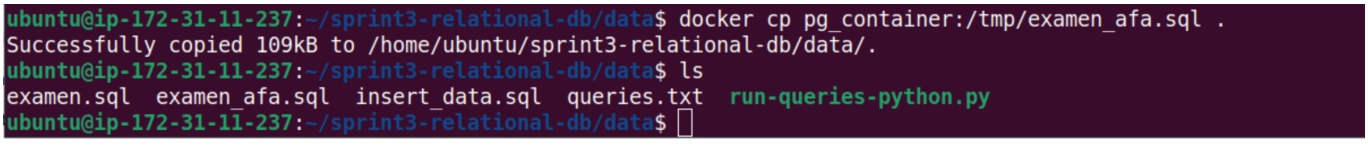


Figure 20: The output when copying the backup on local.