

12/01/2023

API & SQL avec la BDD

PostgreSQL et PGAdmin

Rapport TP1&2



TCHONGOUANG DJOMO GATIEN JUNIOR

Installation de la base de données depuis docker

Pour mener à bien ce TP, préalablement Docker doit être installé sur notre machine.

1. Téléchargez l'image Docker PostgreSQL et pgAdmin :

```
$ docker pull totofunku/sql-cours
$ docker run --name postgresql -e POSTGRES_USER=admin \
-e POSTGRES_PASSWORD=adminadmin -p 5432:5432 \
-v /data:/var/lib/postgresql/data -d totofunku/sql-cours
```

```
C:\Users\33749>docker pull totofunku/sql-cours
Using default tag: latest
latest: Pulling from totofunku/sql-cours
Digest: sha256:a46add136a2aebfe3483a1392710f0e1d2d2df22cd13f11b8a5cdf00774ce7e0
Status: Image is up to date for totofunku/sql-cours:latest
docker.io/totofunku/sql-cours:latest
```

```
$ docker pull totofunku/api-pg-cours
$ docker run --name postgresql_db -d totofunku/api-pg-cours
```

```
C:\Users\33749>docker pull totofunku/api-pg-cours
Using default tag: latest
latest: Pulling from totofunku/api-pg-cours
bb263680fed1: Already exists
75a54e59e691: Already exists
3ce7f8df2b36: Already exists
f30287ef02b9: Already exists
dc1f0e9024d8: Already exists
7f0a68628bce: Already exists
32b11818cae3: Already exists
48111fe612c1: Already exists
fcedb9c04393: Already exists
8943748d4e1f: Already exists
204b98eddef7: Already exists
9e0624990483: Already exists
01ebe7b28449: Already exists
551401fbd028: Pull complete
Digest: sha256:8fece6a2d835e76fc67d1932bae5d9e5138fde22bbc2bd44adaa971d2a359fd6
Status: Downloaded newer image for totofunku/api-pg-cours:latest
docker.io/totofunku/api-pg-cours:latest
```

```
C:\Users\33749>docker run --name postgresql_db -d totofunku/api-pg-cours
eb50707c26f35c7146f478cc61dcbbf8860a97999d725e4e3798ac0980617611
```

```
$ docker pull dpage/pgadmin4:latest
$ docker run --name my-pgadmin -p 82:80 \
-e "PGADMIN_DEFAULT_EMAIL=pgadmin4@pgadmin.org" \
-e "PGADMIN_DEFAULT_PASSWORD=test1234" \
-d dpage/pgadmin4
```

```
C:\Users\33749>docker pull dpage/pgadmin4:latest
```

```
latest: Pulling from dpage/pgadmin4
```

```
661ff4d9561e: Pull complete
```

```
5afa59b8f408: Pull complete
```

```
2d4a6138fafc: Pull complete
```

```
8fb424a8a983: Pull complete
```

```
0011ff13e560: Pull complete
```

```
13f0840603e2: Pull complete
```

```
42a4ae4e73a6: Pull complete
```

```
91c1c38904e7: Pull complete
```

```
fb06a9fc0ca4: Pull complete
```

```
779b87a8a026: Pull complete
```

```
dc7968d095ac: Pull complete
```

```
2716e11486a8: Pull complete
```

```
54175cb5dd12: Pull complete
```

```
d5ccf38e9033: Pull complete
```

```
7008c2bb0275: Pull complete
```

```
618162bf675e: Pull complete
```

```
Digest: sha256:3b480788bfd84c9c46f09ce4984867f36fa8b8d8736b2044b376506cb0702ae2
```

```
Status: Downloaded newer image for dpage/pgadmin4:latest
```

```
docker.io/dpage/pgadmin4:latest
```

```
C:\Users\33749>docker run --name my-pgadmin -p 82:80 -e "PGADMIN_DEFAULT_EMAIL=pgadmin4@pgadmin.org" -e "PGADMIN_DEFAULT_PASSWORD=test1234" -d dpage/pgadmin4
```

```
dbf4d56346938a62db6f59a99e0b9072821880f93371b9bef6d63eb4001fd6ec
```

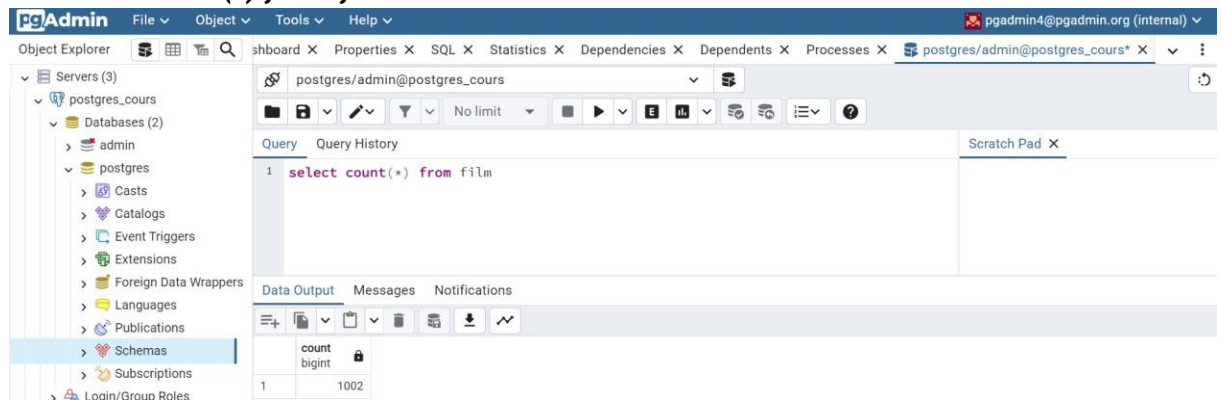
TP1

Cas d'utilisation : Location de DVD

Exécution des 4 requêtes :

a. Combien de films possède le magasin ?

⇒ ***select count(*) from film***



The screenshot shows the pgAdmin 4 web interface. On the left, the 'Object Explorer' pane shows the database structure for 'postgres_cours'. The 'Query' tab is active, displaying the SQL query: `select count(*) from film`. Below the query, the 'Data Output' pane shows the result of the query as a table with one row and one column.

count
1002

A= 1002 movies

b. Combien de films sont disponibles ?

⇒ ***SELECT COUNT(DISTINCT inventory.film_id) AS
nombre_de_films_disponibles FROM inventory***



The screenshot shows the 'Data Output' pane for the second query. It displays a table with one row and one column.

nombre_de_films_disponibles
958

A= 958 movies

c. Quel est le chiffre d'affaires mensuel du magasin ?

⇒ ***SELECT SUM(amount) AS
chiffre_affaires_mensuel FROM payment
WHERE payment_date BETWEEN '2007-01-01' AND '2007-01-31';***

postgres/admin@postgres_cours

Query Query History

```

1 SELECT SUM(amount) AS chiffre_affaires_mensuel
2 FROM payment
3 WHERE payment_date BETWEEN '2007-01-01' AND '2007-01-31';
4
5
6 select * from payment

```

Data Output Messages Notifications

	payment_id [PK] integer	customer_id smallint	staff_id smallint	rental_id integer	amount numeric (5,2)	payment_date timestamp without time zone
1	17503	341	2	1520	7.99	2007-02-15 22:25:46.996577
2	17504	341	1	1778	1.99	2007-02-16 17:23:14.996577
3	17505	341	1	1849	7.99	2007-02-16 22:41:45.996577
4	17506	341	2	2829	2.99	2007-02-19 19:39:56.996577

d. Pour chaque client, donnez les 3 catégories de films les plus vues.

⇒ **SELECT customer_id, first_name, last_name, category_rank,
category_name, category_id**

FROM (

SELEC

c.customer_id, c.first_name,

c.last_name, cat.category_id,

cat.name AS category_name,

**RANK() OVER (PARTITION BY c.customer_id ORDER BY COUNT(*) DESC) AS
category_rank**

FROM customer c

JOIN rental r ON c.customer_id = r.customer_id

JOIN inventory i ON r.inventory_id =

i.inventory_id JOIN film f ON i.film_id = f.film_id

JOIN film_category fc ON f.film_id = fc.film_id

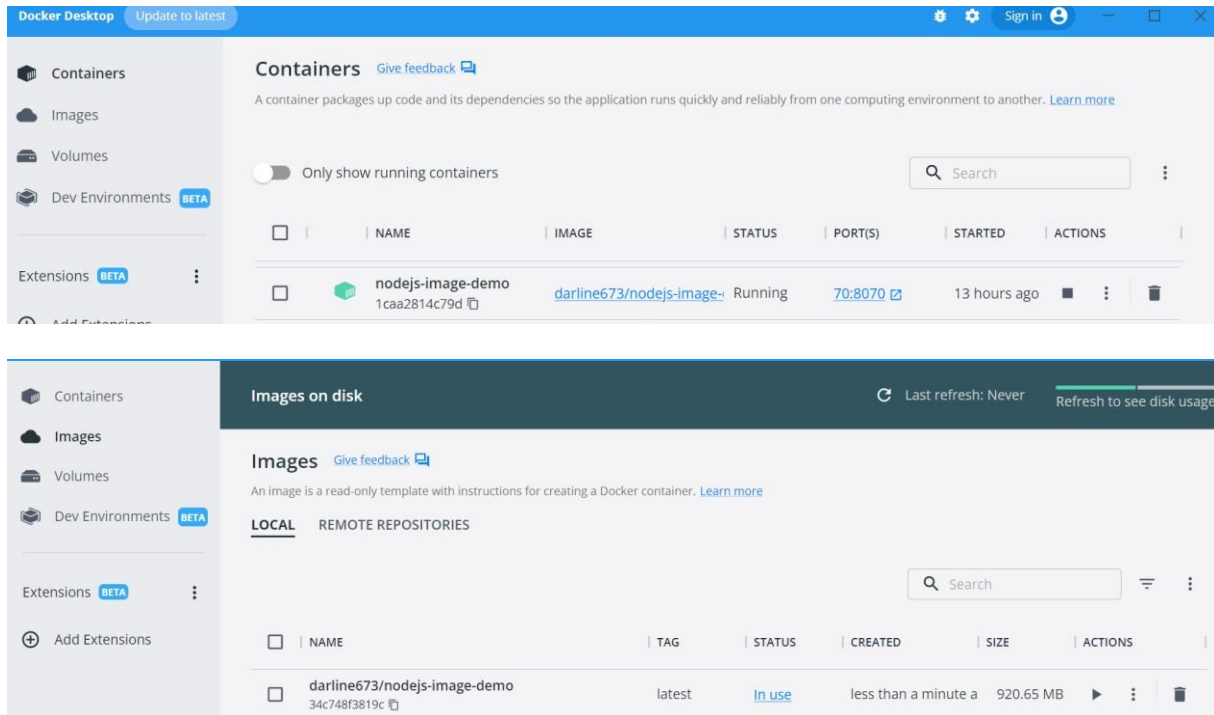
JOIN category cat ON fc.category_id = cat.category_id

TP2

Cas d'utilisation : Location de DVD

Durant ce TP nous avons créé API REST en GO avec Docker pour interagir avec une base de données PostgreSQL à l'aide. Etant familiarisé avec JavaScript, j'ai décidé de faire un REST API Node js et de la tester avec Thunder Client (Extension VS Code pour tester des api)

Création du conteneur :



Ajout d'un nouveau film :

Ici, je récupère un film sous format Json puis je le convertis en objet JavaScript avant de l'insérer dans la base de données et je retourne un code 200 en cas de succès

Liste des films:

Je récupère tous les films puis je le convertis en Json avant de les renvoyer au client avec un code

Afficher les informations d'un Films :

Tout d'abord, je récupère l'ID du film recherché via la method GET, j'exécute une requête vers ma data base afin d'obtenir les informations sur le film souhaité et le retourner sous format Json.

Conclusion

L'intérêt de réaliser une API est de faciliter la communication avec notre data base et ainsi on il sera plus facile de manipuler nos données tout en conservant de façon confidentielle l'intégrité de nos données afin de respecter au minimum la RGPD.

