

TASK I - Introduction to Data Warehouse

1. Perbedaan Antara Data Warehouse dan Data Lake

a. Struktur Data:

- **Data Warehouse:** Data yang disimpan dalam data warehouse umumnya terstruktur dan dioptimalkan untuk query analitis. Data biasanya diolah dan diubah menjadi format yang konsisten sebelum dimasukkan ke dalam data warehouse.
- **Data Lake:** Data lake dapat menyimpan data dalam format mentah, baik terstruktur, semi-terstruktur, maupun tidak terstruktur. Data ini dapat berupa file log, data streaming, atau file multimedia yang belum diolah.

b. Tujuan:

- **Data Warehouse:** Digunakan untuk analisis data bisnis dan pelaporan, yang memerlukan data terstruktur dengan kinerja query yang cepat.
- **Data Lake:** Dirancang untuk penyimpanan data dalam jumlah besar dan heterogen, yang dapat digunakan untuk analitik lanjutan seperti machine learning dan big data processing.

c. Desain dan Skema:

- **Data Warehouse:** Memiliki skema yang ditentukan sebelumnya (schema-on-write), yang berarti skema ditentukan sebelum data dimasukkan.
- **Data Lake:** Menggunakan pendekatan schema-on-read, yang berarti skema diterapkan saat data diakses atau dibaca.

d. Biaya dan Skala:

- **Data Warehouse:** Lebih mahal dalam hal penyimpanan karena data harus diolah dan disimpan dalam format terstruktur.
- **Data Lake:** Lebih murah untuk penyimpanan karena dapat menyimpan data dalam format mentah dan tidak terstruktur.

2. Perbedaan Teknologi Database untuk Data Warehouse (OLAP) dan Database Konvensional (OLTP)

a. **Tujuan Penggunaan:**

- **OLAP (Online Analytical Processing):** Digunakan untuk analisis data dan pelaporan. Fokus pada query yang kompleks dan agregasi data.
- **OLTP (Online Transaction Processing):** Digunakan untuk transaksi harian yang memerlukan operasi read/write cepat dan efisiensi tinggi dalam memasukkan data.

b. **Desain Skema:**

- **OLAP:** Biasanya menggunakan skema star atau snowflake yang dioptimalkan untuk query analitis.
- **OLTP:** Menggunakan skema normalisasi untuk menghindari duplikasi data dan memastikan integritas data.

c. **Ukuran Transaksi:**

- **OLAP:** Transaksi lebih besar dan lebih kompleks, berfokus pada analisis data historis.
- **OLTP:** Transaksi lebih kecil dan sederhana, berfokus pada operasi insert, update, dan delete secara cepat.

d. **Kinerja:**

- **OLAP:** Dioptimalkan untuk query baca yang kompleks.
- **OLTP:** Dioptimalkan untuk operasi read/write yang cepat dan efisien.

3. **Teknologi yang Digunakan untuk Data Warehouse**

a. **Database Management Systems (DBMS):**

- **Amazon Redshift**
- **Google BigQuery**
- **Snowflake**
- **Microsoft Azure Synapse Analytics**

b. ETL Tools:

- **Apache NiFi**
- **Talend**
- **Informatica**
- **Apache Airflow**

c. Data Integration:

- **Kafka**
- **Apache Flink**

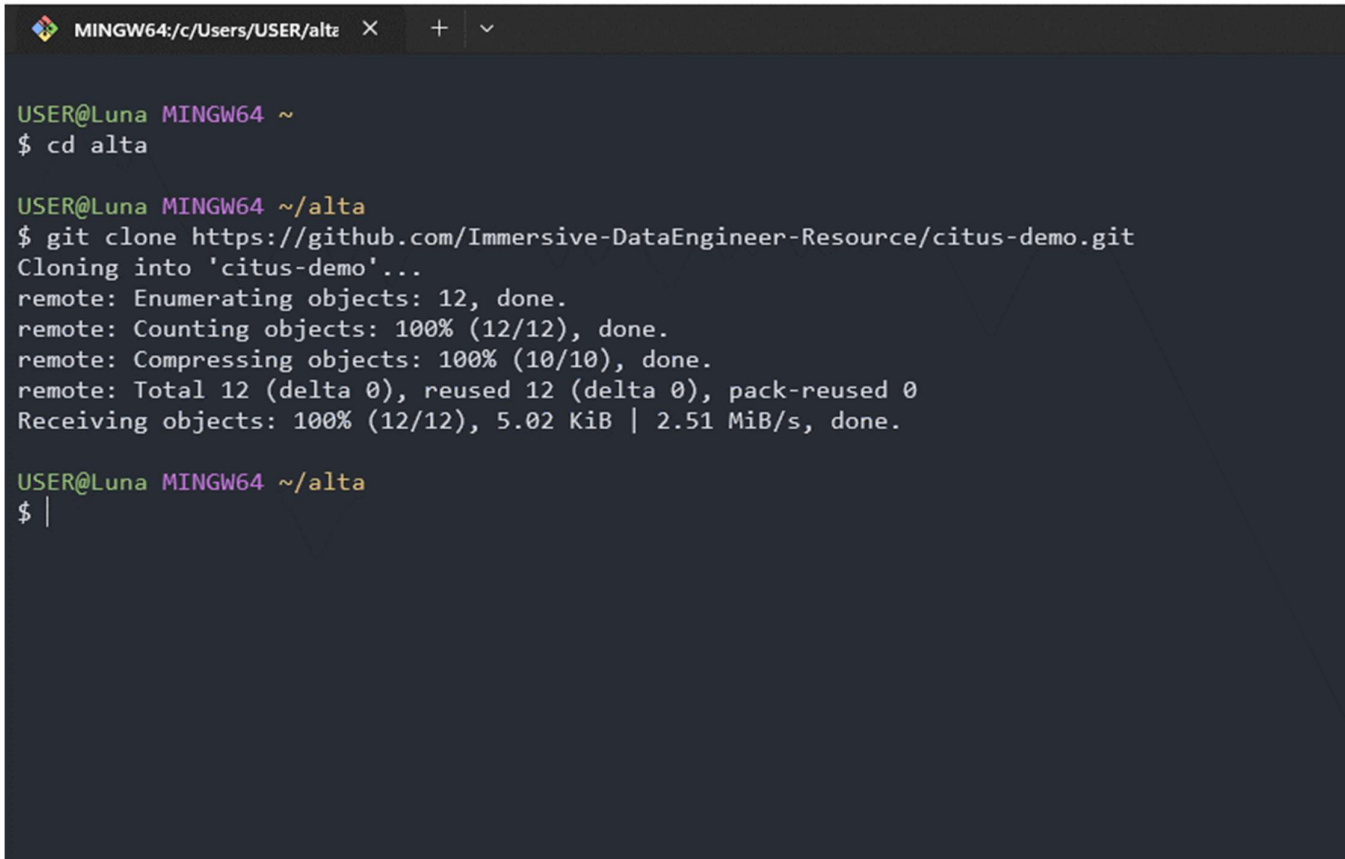
4. Tuliskan setiap perintah dari proses instalasi citus menggunakan docker compose sampai tabel terbentuk, berikan juga tangkapan layar untuk setiap langkah dan hasilnya!

DATA ENGINEER BATCH 4

Mentee: Yovina Silvia

Mentor: Bilal Benefit

- a. Pindah ke directory yang ingin kita clone citus-demonya dari repository github alta.
- b. Setelah itu clone repo dari github alta dengan command git clone.

A screenshot of a Windows terminal window with a dark background. The title bar shows "MINGW64:/c/Users/USER/alta" with window control buttons. The terminal text shows a user navigating to the "alta" directory and cloning a repository from GitHub. The output of the git clone command shows the progress of cloning 12 objects, with all steps (enumerating, counting, compressing, and receiving) completed at 100%.

```
USER@Luna MINGW64 ~  
$ cd alta  
  
USER@Luna MINGW64 ~/alta  
$ git clone https://github.com/Immersive-DataEngineer-Resource/citus-demo.git  
Cloning into 'citus-demo'...  
remote: Enumerating objects: 12, done.  
remote: Counting objects: 100% (12/12), done.  
remote: Compressing objects: 100% (10/10), done.  
remote: Total 12 (delta 0), reused 12 (delta 0), pack-reused 0  
Receiving objects: 100% (12/12), 5.02 KiB | 2.51 MiB/s, done.  
  
USER@Luna MINGW64 ~/alta  
$ |
```

- c. Lalu pindah ke directory yang telah diclone sebelumnya, dan buka directory di vs code.

```
MINGW64:/c/Users/USER/alta X + v

USER@Luna MINGW64 ~
$ cd alta

USER@Luna MINGW64 ~/alta
$ git clone https://github.com/Immersive-DataEngineer-Resource/citus-demo.git
Cloning into 'citus-demo'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 12 (delta 0), reused 12 (delta 0), pack-reused 0
Receiving objects: 100% (12/12), 5.02 KiB | 2.51 MiB/s, done.

USER@Luna MINGW64 ~/alta
$ cd citus-demo

USER@Luna MINGW64 ~/alta/citus-demo (main)
$ code citus-demo

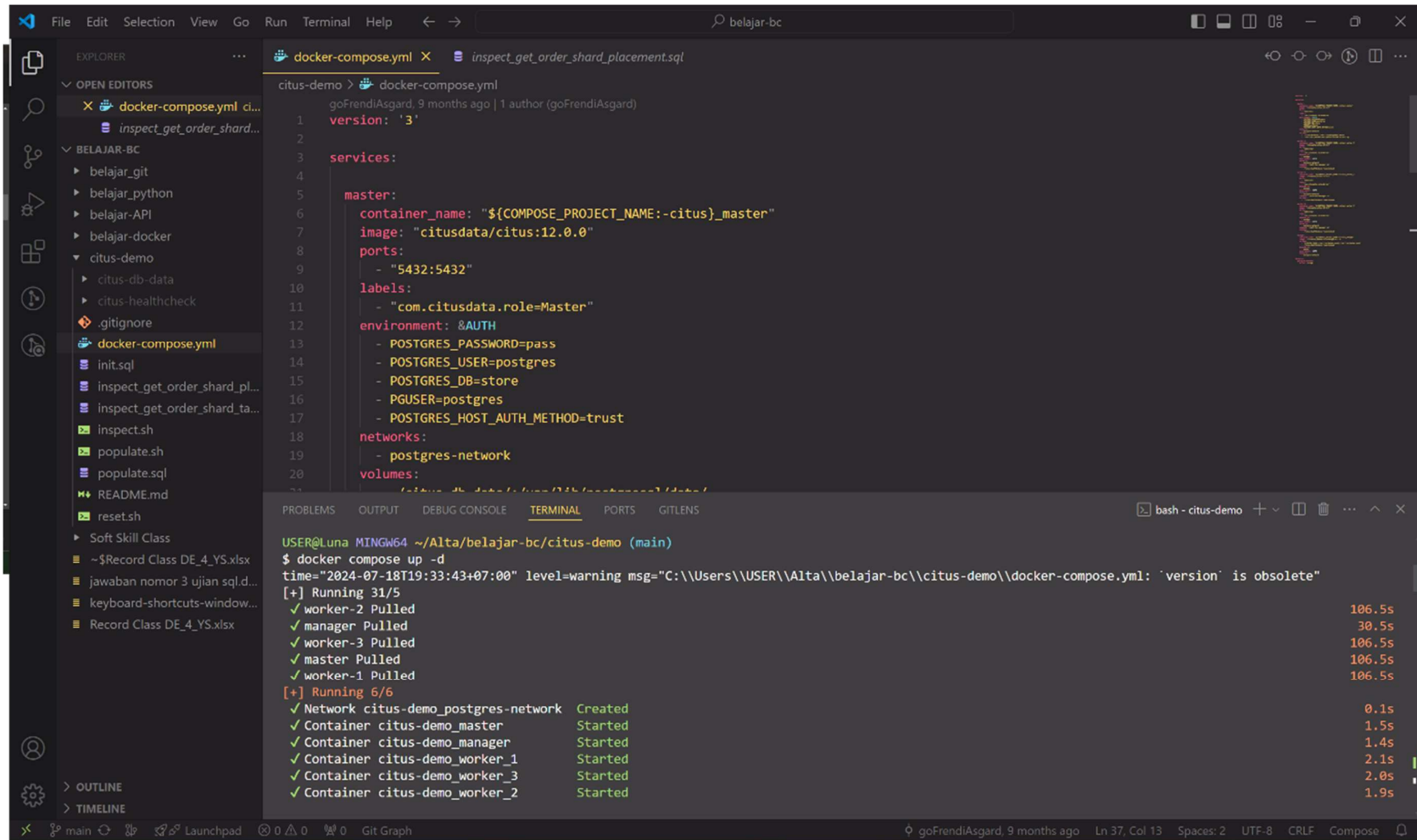
USER@Luna MINGW64 ~/alta/citus-demo (main)
$ |
```

DATA ENGINEER BATCH 4

Mentee: Yovina Silvia

Mentor: Bilal Benefit

d. Jalankan docker compose untuk menjalankan container-container citus-demo yang ada di docker-compose.yml



```
citrus-demo > docker-compose.yml
goFrendiAsgard, 9 months ago | 1 author (goFrendiAsgard)
1 version: '3'
2
3 services:
4
5     master:
6         container_name: "${COMPOSE_PROJECT_NAME:-citus}_master"
7         image: "citusdata/citus:12.0.0"
8         ports:
9             - "5432:5432"
10        labels:
11            - "com.citusdata.role=Master"
12        environment: &AUTH
13            - POSTGRES_PASSWORD=pass
14            - POSTGRES_USER=postgres
15            - POSTGRES_DB=store
16            - PGUSER=postgres
17            - POSTGRES_HOST_AUTH_METHOD=trust
18        networks:
19            - postgres-network
20        volumes:
21            - citus-db-data:/var/lib/postgresql/data

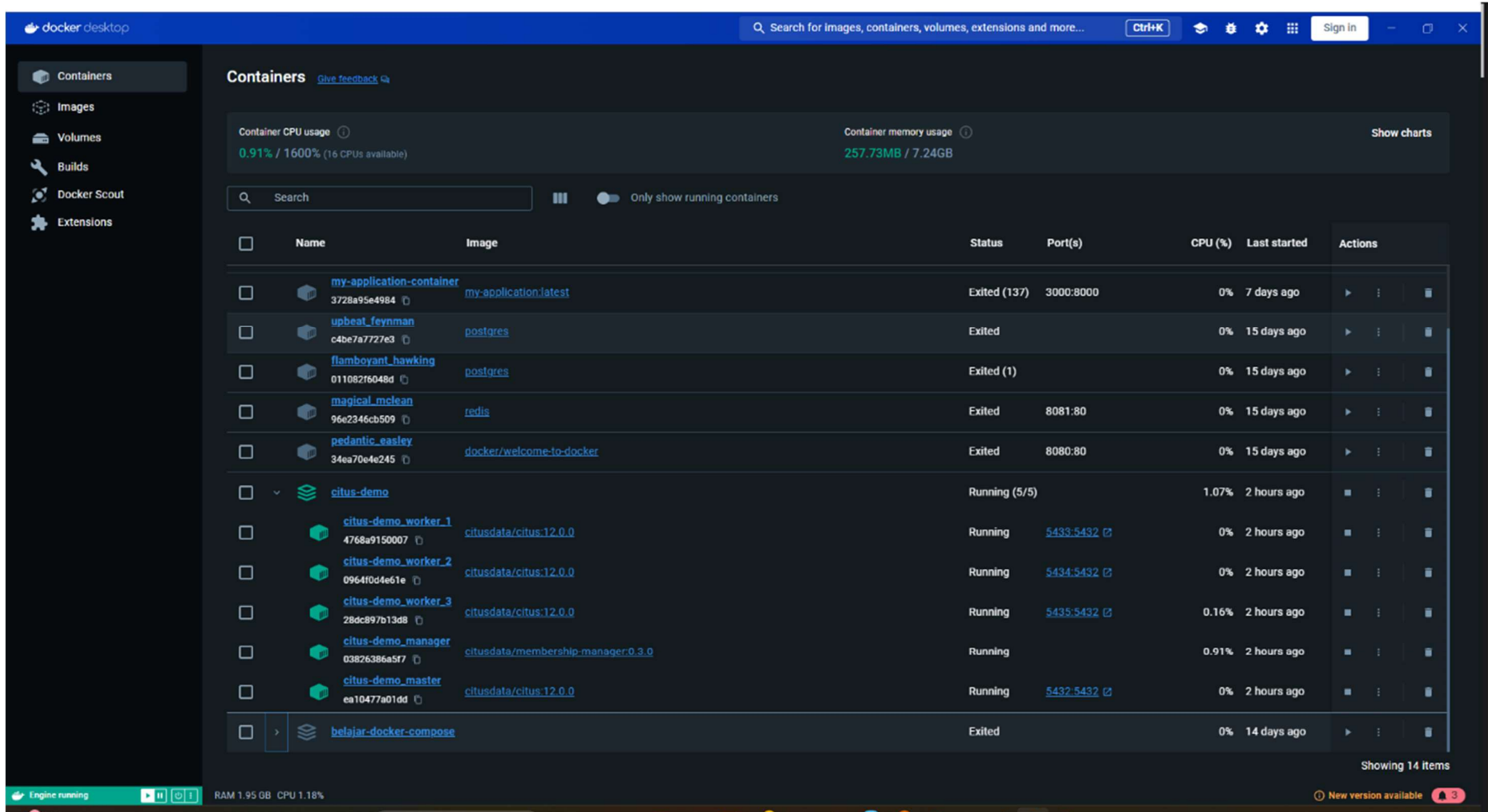
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
bash - citrus-demo
USER@Luna MINGW64 ~/Alta/belajar-bc/citus-demo (main)
$ docker compose up -d
time="2024-07-18T19:33:43+07:00" level=warning msg="C:\\Users\\USER\\Alta\\belajar-bc\\citus-demo\\docker-compose.yml: 'version' is obsolete"
[+] Running 31/5
  ✓ worker-2 Pulled                                106.5s
  ✓ manager Pulled                                30.5s
  ✓ worker-3 Pulled                                106.5s
  ✓ master Pulled                                  106.5s
  ✓ worker-1 Pulled                                106.5s
[+] Running 6/6
  ✓ Network citrus-demo_postgres-network Created      0.1s
  ✓ Container citrus-demo_master Started              1.5s
  ✓ Container citrus-demo_manager Started             1.4s
  ✓ Container citrus-demo_worker_1 Started            2.1s
  ✓ Container citrus-demo_worker_3 Started            2.0s
  ✓ Container citrus-demo_worker_2 Started            1.9s
```

DATA ENGINEER BATCH 4

Mentee: Yovina Silvia

Mentor: Bilal Benefit

e. Lalu cek di docker desktop untuk memastikan container citus-demo benar sudah bejalan.



The screenshot shows the Docker Desktop interface. The left sidebar contains navigation options: Containers, Images, Volumes, Builds, Docker Scout, and Extensions. The main panel is titled 'Containers' and displays a summary of container CPU usage (0.91% / 1600%) and memory usage (257.73MB / 7.24GB). Below this, there is a search bar and a toggle for 'Only show running containers'. A table lists the containers, including their names, images, status, ports, CPU usage, and last started time. The 'citus-demo' container is highlighted, showing it is running with 5/5 instances. The status bar at the bottom indicates the engine is running, with RAM usage at 1.95 GB and CPU at 1.16%.

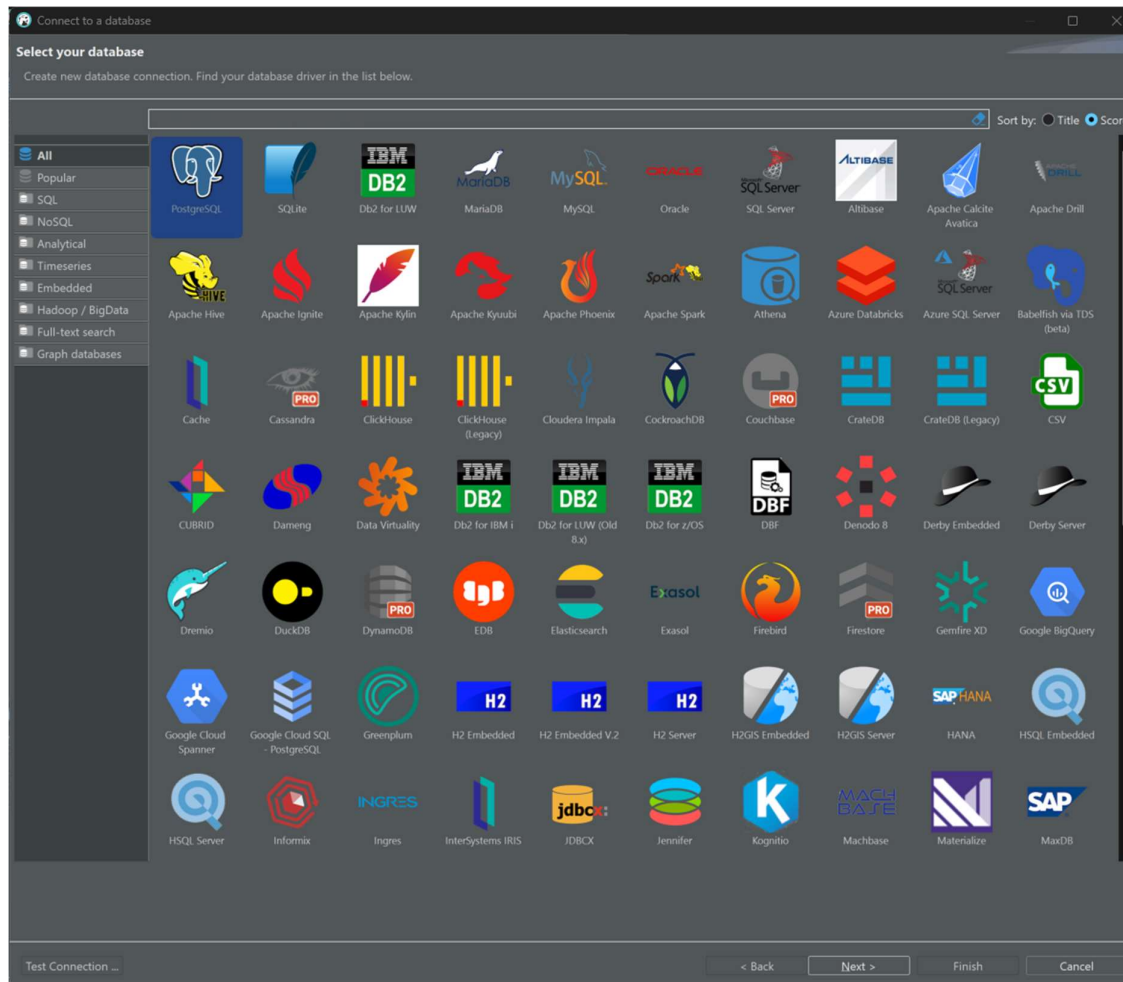
Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
my-application-container	my-application:latest	Exited (137)	3000:8000	0%	7 days ago	
upbeat_feynman	postgres	Exited		0%	15 days ago	
flamboyant_hawking	postgres	Exited (1)		0%	15 days ago	
magical_mclean	redis	Exited	8081:80	0%	15 days ago	
pedantic_easley	docker/welcome-to-docker	Exited	8080:80	0%	15 days ago	
citus-demo		Running (5/5)		1.07%	2 hours ago	
citus-demo_worker_1	citusdata/citus:12.0.0	Running	5433:5432	0%	2 hours ago	
citus-demo_worker_2	citusdata/citus:12.0.0	Running	5434:5432	0%	2 hours ago	
citus-demo_worker_3	citusdata/citus:12.0.0	Running	5435:5432	0.16%	2 hours ago	
citus-demo_manager	citusdata/membership-manager:0.3.0	Running		0.91%	2 hours ago	
citus-demo_master	citusdata/citus:12.0.0	Running	5432:5432	0%	2 hours ago	
belajar-docker-compose		Exited		0%	14 days ago	

DATA ENGINEER BATCH 4

Mentee: Yovina Silvia

Mentor: Bilal Benefit

- f. Buka postgresql pada dbeaver, lalu buat connection baru ke database postgresql menggunakan port master yang sudah kita jalankan di docker-compose.yml tadi.



DATA ENGINEER BATCH 4

Mentee: Yovina Silva

Mentor: Bilal Benefit



Connect to a database

Connection Settings

PostgreSQL connection settings

Main PostgreSQL Driver properties SSH SSL + Network configurations...

Server

Connect by: ☒ Host ☐ URL

URL: jdbc:postgresql://localhost:5432/postgres

Host: localhost Port: 5432

Database: postgres ☐ Show all databases

Authentication

Authentication: Database Native

Username: store

Password: Save password

Advanced

Session role: Local Client: PostgreSQL 16

[Connection variables information](#) [Database documentation](#) [Connection details \(name, type, ...\)](#)

Driver name: PostgreSQL Driver Settings Driver license

Test Connection ... < Back Next > Finish Cancel

- g. Create table events_columnar dengan using columnar dan buat table events_row sebagai table biasa dan dari sana kita dapat membandingkan bahwa data pada table yang menggunakan metode columnar tercompress, sehingga hanya memakan storage yang kecil dibandingkan data yang menggunakan table biasa.

```
WITH placement AS (  
    SELECT  
        shardid as shard_id  
        , nodename as node_name  
    FROM pg_dist_shard_placement  
)  
, order_ids AS (  
    SELECT order_id  
    FROM orders  
    ORDER BY order_id  
    LIMIT 15  
)  
, order_shards AS (  
    SELECT  
        order_id  
        , get_shard_id_for_distribution_column('orders', order_id) as shard_id  
        , 'orders_' || get_shard_id_for_distribution_column('orders', order_id) as real_table_name  
    FROM order_ids  
)  
SELECT  
    order_shards.*  
    , placement.node_name  
FROM order_shards  
INNER JOIN placement  
    ON placement.shard_id = order_shards.shard_id  
;
```

DATA ENGINEER BATCH 4

Mentee: Yovina Silva

Mentor: Bilal Benefit

The screenshot displays the DBeaver 24.1.1 interface. The left sidebar shows the 'Database Navigator' with a tree view of the 'store' database, including schemas like 'citius', 'citius_internal', 'columnar', 'columnar_internal', and 'public'. The 'public' schema is expanded, showing tables like 'events_columnar' and 'events_row'. The main editor window shows a SQL script with the following content:

```
CREATE TABLE events_columnar (
  device_id bigint,
  event_id bigserial,
  event_time timestampz default now(),
  data jsonb not null
)
USING columnar;
--insert some data
INSERT INTO events_columnar (device_id, data)
SELECT d, '{"hello": "columnar"}' FROM generate_series (1, 100000) d;

--create a row-based table to compare
CREATE TABLE events_row AS SELECT * FROM events_columnar;

select * from events_columnar
select * from events_row
```

The bottom panel shows the results of the SQL execution in a table view. The table has four columns: 'device_id', 'event_id', 'event_time', and 'data'. The data is displayed in a grid format, showing the first 24 rows of the results.

Grid	device_id	event_id	event_time	data
1	1	1	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
2	2	2	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
3	3	3	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
4	4	4	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
5	5	5	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
6	6	6	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
7	7	7	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
8	8	8	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
9	9	9	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
10	10	10	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
11	11	11	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
12	12	12	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
13	13	13	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
14	14	14	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
15	15	15	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
16	16	16	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
17	17	17	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
18	18	18	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
19	19	19	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
20	20	20	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
21	21	21	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
22	22	22	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
23	23	23	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}
24	24	24	2024-07-19 16:38:48.863 +0700	{"hello": "columnar"}

The bottom status bar indicates that 200 rows were fetched, with a fetch time of 0.007s (0.001s fetch) on 2024-07-19 at 22:44:12.

5. Perbedaan Antara Access Method Heap dan Columnar pada Citus

1. Heap:

- Heap adalah metode akses default di PostgreSQL dan Citus. Data disimpan dalam format baris demi baris. Heap cocok untuk beban kerja OLTP di mana operasi read/write sering terjadi dan data dimasukkan dan diambil dengan cepat.

2. Columnar:

- Columnar adalah metode akses di mana data disimpan dalam format kolom demi kolom. Ini cocok untuk beban kerja OLAP di mana query analitis yang kompleks memerlukan pemrosesan data dalam jumlah besar. Format columnar mengurangi I/O dengan hanya membaca kolom yang diperlukan oleh query.