

### TASK 3 - Replication + Sharding

#### 1. Jelaskan perbedaan antara replication dan sharding

##### a. Replication:

- **Definisi:** Replication adalah proses menduplikasi data dari satu database ke database lain untuk meningkatkan ketersediaan dan keandalan data.
- **Tujuan:** Tujuannya adalah untuk memastikan bahwa data tetap tersedia dan dapat diakses meskipun ada kegagalan pada salah satu node. Replication juga dapat digunakan untuk load balancing pada aplikasi yang memerlukan akses baca tinggi.
- **Karakteristik:**
  - Meningkatkan ketersediaan data.
  - Menggunakan salinan data yang sama di beberapa node.
  - Memungkinkan failover dan pemulihan bencana.
- **Contoh Penggunaan:** Digunakan pada sistem yang memerlukan toleransi kesalahan tinggi dan ketersediaan data yang sangat baik, seperti dalam aplikasi perbankan atau e-commerce.

##### b. Sharding:

- **Definisi:** Sharding adalah teknik membagi data besar menjadi beberapa fragmen atau "shards" yang didistribusikan di beberapa database atau server.
- **Tujuan:** Tujuannya adalah untuk meningkatkan kinerja dengan mendistribusikan beban kerja dan kapasitas penyimpanan di beberapa node.
- **Karakteristik:**
  - Meningkatkan kinerja dan skalabilitas.
  - Data dipecah menjadi beberapa bagian berdasarkan kriteria tertentu (misalnya, range atau hash).
  - Setiap shard hanya berisi subset dari keseluruhan data.
- **Contoh Penggunaan:** Digunakan pada sistem dengan volume data sangat besar yang memerlukan skalabilitas tinggi, seperti media sosial atau platform streaming video.

2. Lakukan percobaan untuk membuat reference table + distributed table seperti pada repo <https://github.com/Immersive-DataEngineer-Resource/citus-demo>

```
-- Create users table (Reference Table)
CREATE TABLE users (
  user_id SERIAL PRIMARY KEY,
  username TEXT NOT NULL,
  email TEXT NOT NULL UNIQUE
);
SELECT create_reference_table('users');

select * from users u

-- Create products table (Reference Table)
CREATE TABLE products (
  product_id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  price NUMERIC(10, 2) NOT NULL
);
SELECT create_reference_table('products');

select * from products

-- Populate data
-- Users
INSERT INTO users (username, email) VALUES ('JohnDoe', 'john.doe@example.com'), ('JaneSmith', 'jane.smith@example.com');

-- Products
INSERT INTO products (name, price) VALUES ('Laptop', 1000.00), ('Phone', 500.00), ('Headphones', 200.00), ('Monitor', 300.00);

select * from users u
select * from products
```

- Membuat Tabel Referensi 'users'
- **CREATE TABLE users:** Membuat tabel `users` dengan kolom-kolom berikut:

- `user_id SERIAL PRIMARY KEY`: Pengidentifikasi unik untuk setiap pengguna, secara otomatis bertambah.
- `username TEXT NOT NULL`: Nama pengguna yang tidak boleh kosong.
- `email TEXT NOT NULL UNIQUE`: Email pengguna yang tidak boleh kosong dan harus unik.

- **SELECT create\_reference\_table('users')**: Ini adalah fungsi dari Citus untuk mengatur tabel `users` sebagai tabel referensi. Tabel referensi digunakan untuk menyimpan data yang sering digunakan bersama oleh beberapa shard (pecahan) dalam database yang terdistribusi.

➤ Membuat Tabel Referensi 'products'

- **CREATE TABLE products**: Membuat tabel `products` dengan kolom-kolom berikut:

- `product_id SERIAL PRIMARY KEY`: Pengidentifikasi unik untuk setiap produk, secara otomatis bertambah.
- `name TEXT NOT NULL`: Nama produk yang tidak boleh kosong.
- `price NUMERIC(10, 2) NOT NULL`: Harga produk yang tidak boleh kosong, dengan format numerik dua desimal.

- **SELECT create\_reference\_table('products')**: Fungsi ini mengatur tabel `products` sebagai tabel referensi.

➤ Memasukkan Data ke Tabel 'users' dan 'products'

- **INSERT INTO users**: Menyisipkan data pengguna ke dalam tabel `users`.

- `('JohnDoe', 'john.doe@example.com')`: Menambahkan pengguna dengan nama `JohnDoe` dan email `john.doe@example.com`.
- `('JaneSmith', 'jane.smith@example.com')`: Menambahkan pengguna dengan nama `JaneSmith` dan email `jane.smith@example.com`.

➤ Memasukkan Data ke Tabel 'products'

- **INSERT INTO products**: Menyisipkan data produk ke dalam tabel `products`.

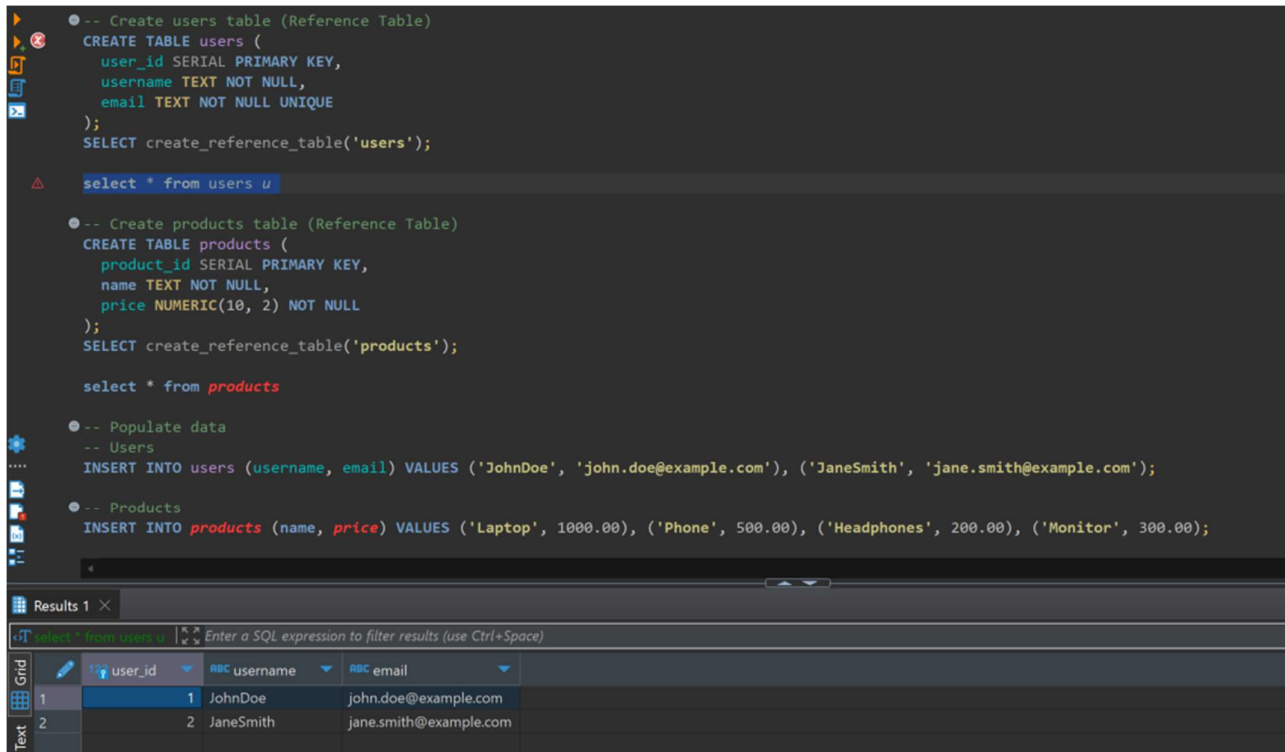
- `('Laptop', 1000.00)`: Menambahkan produk `Laptop` dengan harga `1000.00`.
- `('Phone', 500.00)`: Menambahkan produk `Phone` dengan harga `500.00`.
- `('Headphones', 200.00)`: Menambahkan produk `Headphones` dengan harga `200.00`.

- o ('Monitor', 300.00): Menambahkan produk Monitor dengan harga 300.00.

Skrip SQL ini:

1. Membuat dua tabel referensi, `users` dan `products`, menggunakan PostgreSQL dan ekstensi Citus.
2. Mengisi tabel `users` dengan dua pengguna.
3. Mengisi tabel `products` dengan empat produk.
4. Menampilkan semua data yang telah dimasukkan ke dalam kedua tabel tersebut.

a. Tampilan dari table `users` (table references)



```
-- Create users table (Reference Table)
CREATE TABLE users (
  user_id SERIAL PRIMARY KEY,
  username TEXT NOT NULL,
  email TEXT NOT NULL UNIQUE
);
SELECT create_reference_table('users');

select * from users u

-- Create products table (Reference Table)
CREATE TABLE products (
  product_id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  price NUMERIC(10, 2) NOT NULL
);
SELECT create_reference_table('products');

select * from products

-- Populate data
-- Users
INSERT INTO users (username, email) VALUES ('JohnDoe', 'john.doe@example.com'), ('JaneSmith', 'jane.smith@example.com');

-- Products
INSERT INTO products (name, price) VALUES ('Laptop', 1000.00), ('Phone', 500.00), ('Headphones', 200.00), ('Monitor', 300.00);
```

Results 1 X

select \* from users u Enter a SQL expression to filter results (use Ctrl+Space)

	user_id	username	email
1	1	JohnDoe	john.doe@example.com
2	2	JaneSmith	jane.smith@example.com

**b. Tampilan dari table products (table references)**

```
-- Create users table (Reference Table)
CREATE TABLE users (
  user_id SERIAL PRIMARY KEY,
  username TEXT NOT NULL,
  email TEXT NOT NULL UNIQUE
);
SELECT create_reference_table('users');

select * from users u

-- Create products table (Reference Table)
CREATE TABLE products (
  product_id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  price NUMERIC(10, 2) NOT NULL
);
SELECT create_reference_table('products');

select * from products

-- Populate data
-- Users
INSERT INTO users (username, email) VALUES ('JohnDoe', 'john.doe@example.com'), ('JaneSmith', 'jane.smith@example.com');

-- Products
INSERT INTO products (name, price) VALUES ('Laptop', 1000.00), ('Phone', 500.00), ('Headphones', 200.00), ('Monitor', 300.00);
```

Results 1 X

select \* from products Enter a SQL expression to filter results (use Ctrl+Space)

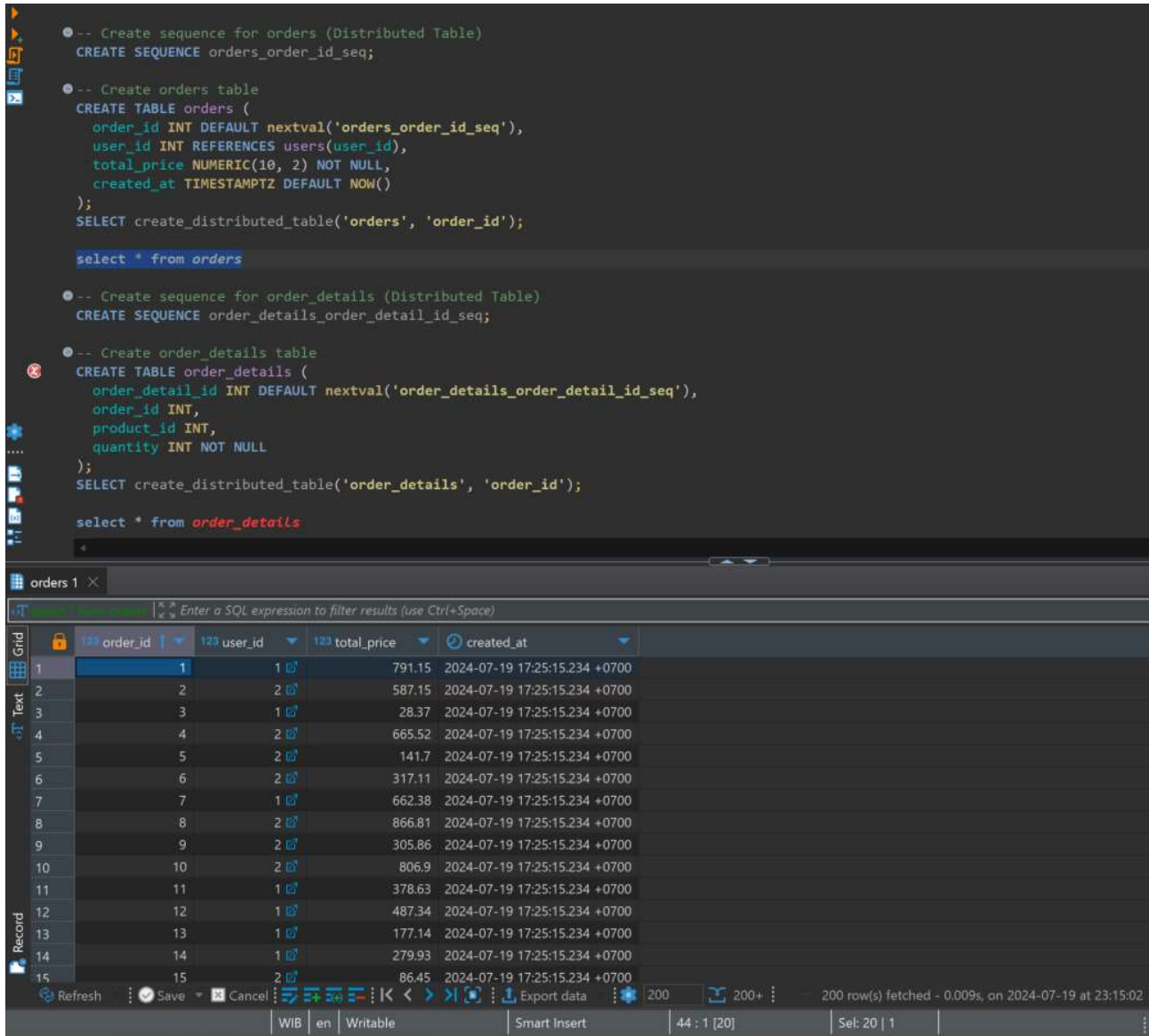
Grid	123 product_id	ABC name	123 price
1	1	Laptop	1,000
2	2	Phone	500
3	3	Headphones	200
4	4	Monitor	300

#### DATA ENGINEER BATCH 4

Mentee: Yovina Silvia

Mentor: Bilal Benefit

#### c. Tampilan dari table orders (table distributed)



```
-- Create sequence for orders (Distributed Table)
CREATE SEQUENCE orders_order_id_seq;

-- Create orders table
CREATE TABLE orders (
  order_id INT DEFAULT nextval('orders_order_id_seq'),
  user_id INT REFERENCES users(user_id),
  total_price NUMERIC(10, 2) NOT NULL,
  created_at TIMESTAMPTZ DEFAULT NOW()
);
SELECT create_distributed_table('orders', 'order_id');

select * from orders

-- Create sequence for order_details (Distributed Table)
CREATE SEQUENCE order_details_order_detail_id_seq;

-- Create order_details table
CREATE TABLE order_details (
  order_detail_id INT DEFAULT nextval('order_details_order_detail_id_seq'),
  order_id INT,
  product_id INT,
  quantity INT NOT NULL
);
SELECT create_distributed_table('order_details', 'order_id');

select * from order_details
```

orders 1 X

Enter a SQL expression to filter results (use Ctrl+Space)

	order_id	user_id	total_price	created_at
1	1	1	791.15	2024-07-19 17:25:15.234 +0700
2	2	2	587.15	2024-07-19 17:25:15.234 +0700
3	3	1	28.37	2024-07-19 17:25:15.234 +0700
4	4	2	665.52	2024-07-19 17:25:15.234 +0700
5	5	2	141.7	2024-07-19 17:25:15.234 +0700
6	6	2	317.11	2024-07-19 17:25:15.234 +0700
7	7	1	662.38	2024-07-19 17:25:15.234 +0700
8	8	2	866.81	2024-07-19 17:25:15.234 +0700
9	9	2	305.86	2024-07-19 17:25:15.234 +0700
10	10	2	806.9	2024-07-19 17:25:15.234 +0700
11	11	1	378.63	2024-07-19 17:25:15.234 +0700
12	12	1	487.34	2024-07-19 17:25:15.234 +0700
13	13	1	177.14	2024-07-19 17:25:15.234 +0700
14	14	1	279.93	2024-07-19 17:25:15.234 +0700
15	15	2	86.45	2024-07-19 17:25:15.234 +0700

Refresh Save Cancel Export data 200 200+ 200 row(s) fetched - 0.009s, on 2024-07-19 at 23:15:02

WIB en Writable Smart Insert 44 : 1 [20] Set: 20 | 1

```
-- Create sequence for orders (Distributed Table)
CREATE SEQUENCE orders_order_id_seq;

-- Create orders table
CREATE TABLE orders (
  order_id INT DEFAULT nextval('orders_order_id_seq'),
  user_id INT REFERENCES users(user_id),
  total_price NUMERIC(10, 2) NOT NULL,
  created_at TIMESTAMPTZ DEFAULT NOW()
);
SELECT create_distributed_table('orders', 'order_id');

select * from orders
```

➤ Membuat Sekuens untuk Tabel 'orders'

**CREATE SEQUENCE orders\_order\_id\_seq:** Membuat sekuens bernama `orders_order_id_seq`. Sekuens ini akan menghasilkan nilai unik yang secara otomatis bertambah untuk digunakan sebagai `order_id` dalam tabel `orders`.

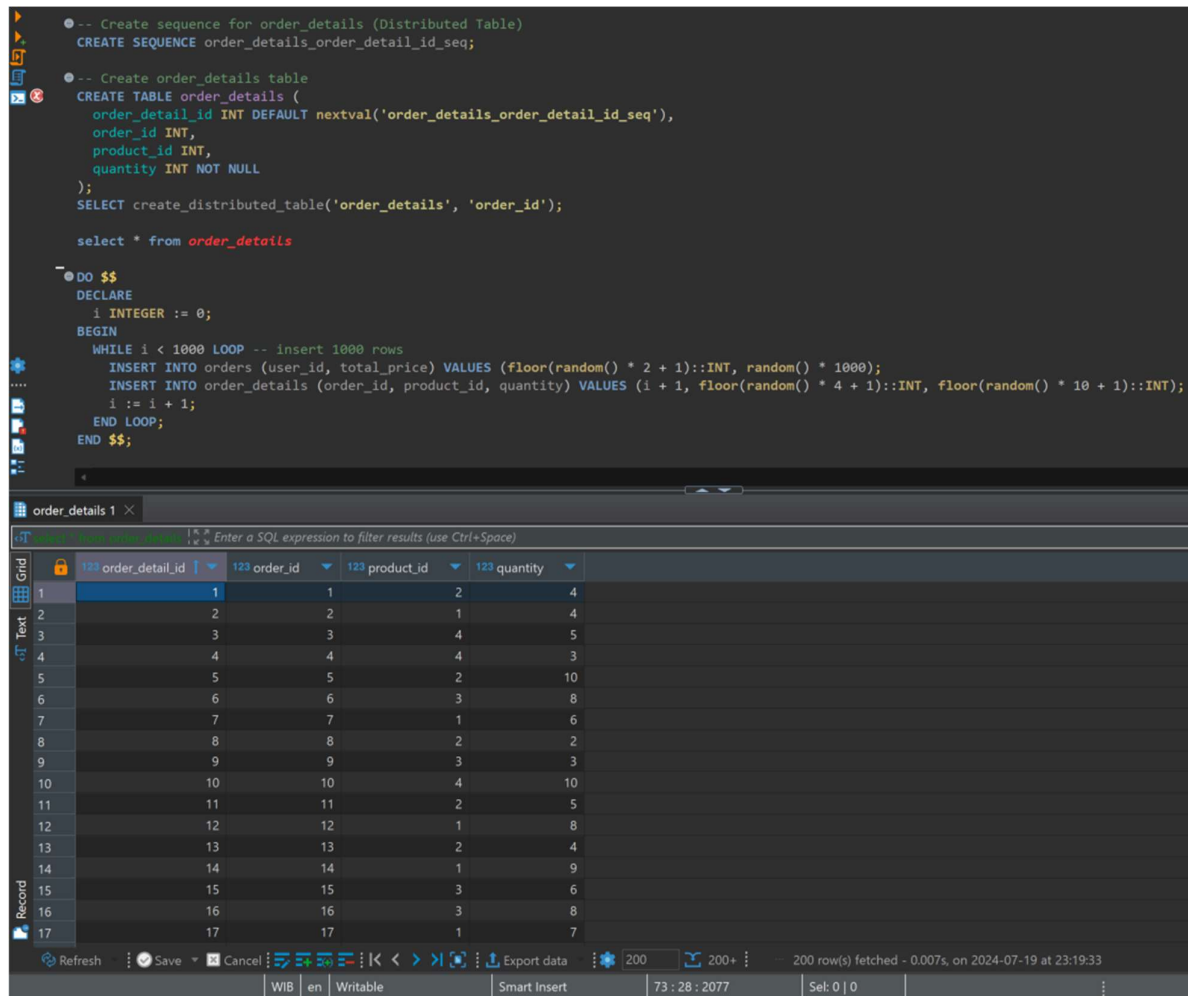
➤ Membuat Tabel 'orders'

- **CREATE TABLE orders:** Membuat tabel `orders` dengan kolom-kolom berikut:
  - `order_id INT DEFAULT nextval('orders_order_id_seq')`: Pengidentifikasi unik untuk setiap pesanan, menggunakan nilai berikutnya dari sekuens `orders_order_id_seq`.
  - `user_id INT REFERENCES users(user_id)`: Pengidentifikasi unik pengguna yang melakukan pesanan, dengan referensi ke kolom `user_id` di tabel `users`.
  - `total_price NUMERIC(10, 2) NOT NULL`: Total harga pesanan yang tidak boleh kosong.
  - `created_at TIMESTAMPTZ DEFAULT NOW()`: Waktu pembuatan pesanan, dengan nilai default waktu saat ini.
- **SELECT create\_distributed\_table('orders', 'order\_id');** Mengatur tabel `orders` sebagai tabel terdistribusi menggunakan Citus, dengan kolom distribusi `order_id`. Tabel terdistribusi berarti data akan dipecah dan disebar ke beberapa node pekerja dalam kluster Citus untuk meningkatkan kinerja.

➤ Menampilkan Data dari Tabel 'orders'

**SELECT \* from orders:** Menampilkan semua data dari tabel orders.

d. Tampilan dari table order\_details (table distributed)



```

-- Create sequence for order_details (Distributed Table)
CREATE SEQUENCE order_details_order_detail_id_seq;

-- Create order_details table
CREATE TABLE order_details (
  order_detail_id INT DEFAULT nextval('order_details_order_detail_id_seq'),
  order_id INT,
  product_id INT,
  quantity INT NOT NULL
);
SELECT create_distributed_table('order_details', 'order_id');

select * from order_details

-- DO $$
DECLARE
  i INTEGER := 0;
BEGIN
  WHILE i < 1000 LOOP -- Insert 1000 rows
    INSERT INTO orders (user_id, total_price) VALUES (floor(random() * 2 + 1)::INT, random() * 1000);
    INSERT INTO order_details (order_id, product_id, quantity) VALUES (i + 1, floor(random() * 4 + 1)::INT, floor(random() * 10 + 1)::INT);
    i := i + 1;
  END LOOP;
END $$;

```

order_detail_id	order_id	product_id	quantity
1	1	2	4
2	2	1	4
3	3	4	5
4	4	4	3
5	5	2	10
6	6	3	8
7	7	1	6
8	8	2	2
9	9	3	3
10	10	4	10
11	11	2	5
12	12	1	8
13	13	2	4
14	14	1	9
15	15	3	6
16	16	3	8
17	17	1	7

200 row(s) fetched - 0.007s, on 2024-07-19 at 23:19:33



```
-- Create sequence for order_details (Distributed Table)
CREATE SEQUENCE order_details_order_detail_id_seq;

-- Create order_details table
CREATE TABLE order_details (
  order_detail_id INT DEFAULT nextval('order_details_order_detail_id_seq'),
  order_id INT,
  product_id INT,
  quantity INT NOT NULL
);
SELECT create_distributed_table('order_details', 'order_id');

select * from order_details
```

➤ Membuat Sekuens untuk Tabel 'order\_details'

**CREATE SEQUENCE order\_details\_order\_detail\_id\_seq:** Membuat sekuens bernama `order_details_order_detail_id_seq`. Sekuens ini akan menghasilkan nilai unik yang secara otomatis bertambah untuk digunakan sebagai `order_detail_id` dalam tabel `order_details`.

➤ Membuat Tabel 'order\_details'

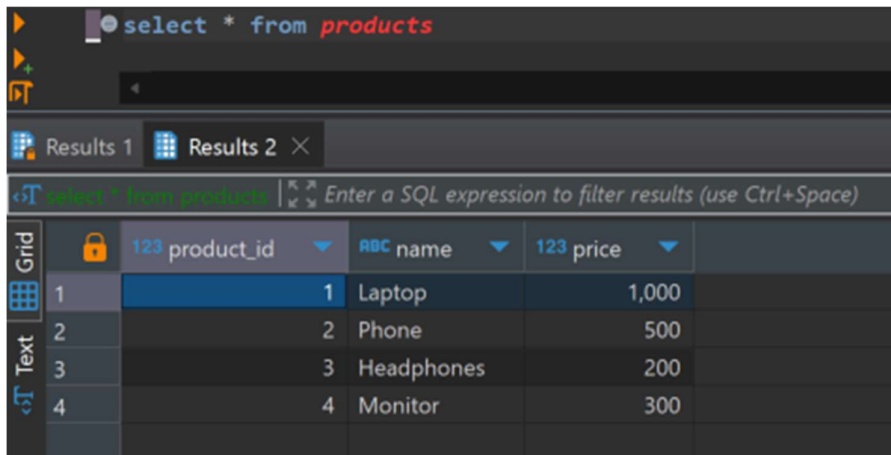
- **CREATE TABLE order\_details:** Membuat tabel `order_details` dengan kolom-kolom berikut:
  - `order_detail_id INT DEFAULT nextval('order_details_order_detail_id_seq')`: Pengidentifikasi unik untuk setiap detail pesanan, menggunakan nilai berikutnya dari sekuens `order_details_order_detail_id_seq`.
  - `order_id INT`: Pengidentifikasi unik untuk pesanan, digunakan untuk menghubungkan dengan tabel `orders`.
  - `product_id INT`: Pengidentifikasi unik untuk produk, digunakan untuk menghubungkan dengan tabel `products`.
  - `quantity INT NOT NULL`: Jumlah produk dalam pesanan yang tidak boleh kosong.
- **SELECT create\_distributed\_table('order\_details', 'order\_id');** Mengatur tabel `order_details` sebagai tabel terdistribusi menggunakan Citus, dengan kolom distribusi `order_id`.

- Menampilkan Data dari Tabel 'order\_details'

**SELECT \* from order\_details:** Menampilkan semua data dari tabel order\_details.

3. Di node/worker mana saja product "Headphone" tersimpan? Tunjukkan shard id nya

- Headphones memiliki product\_id = 3



	product_id	name	price
1	1	Laptop	1,000
2	2	Phone	500
3	3	Headphones	200
4	4	Monitor	300

```
WITH product_placements AS (
  SELECT
    shardid as shard_id,
    nodename as node_name
  FROM pg_dist_shard_placement
),
product_shards AS (
  SELECT
    product_id,
    get_shard_id_for_distribution_column('products', product_id) as shard_id,
    'products_' || get_shard_id_for_distribution_column('products', product_id) as real_table_name
```

```
FROM products
WHERE name = 'Headphones'
)
SELECT
    product_shards.product_id,
    product_shards.shard_id,
    product_placements.node_name
FROM product_shards
INNER JOIN product_placements
    ON product_placements.shard_id = product_shards.shard_id;
```

Skrip SQL ini bertujuan untuk menemukan lokasi fisik (node) dari shard yang menyimpan informasi tentang produk tertentu (dalam hal ini, 'Headphones') di dalam kluster Citus yang terdistribusi.

- CTE `product_placements`
  - **pg\_dist\_shard\_placement:** Ini adalah tabel sistem Citus yang menyimpan informasi tentang penempatan shard, termasuk ID shard dan nama node di mana shard tersebut ditempatkan.
  - **CTE `product_placements`:** Common Table Expression (CTE) ini memilih `shardid` dan `nodename`, mengganti nama kolom menjadi `shard_id` dan `node_name` untuk digunakan nanti.
- CTE `product_shards`
  - **products:** Tabel `products` berisi informasi produk seperti `product_id`, `name`, dan `price`.
  - **get\_shard\_id\_for\_distribution\_column:** Fungsi Citus yang mengambil ID shard untuk kolom distribusi tertentu. Dalam hal ini, kolom distribusi adalah `product_id` dari tabel `products`.
  - **CTE `product_shards`:** CTE ini memilih `product_id`, menghitung `shard_id` menggunakan `get_shard_id_for_distribution_column`, dan membuat nama tabel nyata (`real_table_name`) untuk produk dengan nama 'Headphones'.
- Query Utama
  - **product\_shards:** CTE dari langkah sebelumnya yang berisi `product_id`, `shard_id`, dan `real_table_name` untuk produk 'Headphones'.
  - **product\_placements:** CTE dari langkah pertama yang berisi `shard_id` dan `node_name`.

- **INNER JOIN:** Menggabungkan `product_shards` dengan `product_placements` berdasarkan `shard_id`, sehingga menghubungkan informasi produk dengan lokasi fisiknya di kluster.

Query ini menghasilkan:

- **product\_id:** ID dari produk 'Headphones'.
  - **shard\_id:** ID dari shard yang menyimpan data tentang produk 'Headphones'.
  - **node\_name:** Nama node di kluster Citus tempat shard tersebut disimpan.
- Product Headphones tersimpan di node\_name:

`citus-demo_worker_1`, `citus-demo_worker_2`, dan `citus-demo_worker_3`

```
WITH product_placements AS (
  SELECT
    shardid as shard_id,
    nodename as node_name
  FROM pg_dist_shard_placement
),
product_shards AS (
  SELECT
    product_id,
    get_shard_id_for_distribution_column('products', product_id) as shard_id,
    'products_' || get_shard_id_for_distribution_column('products', product_id) as real_table_name
  FROM products
  WHERE name = 'Headphones'
)
SELECT
  product_shards.product_id,
  product_shards.shard_id,
  product_placements.node_name
FROM product_shards
INNER JOIN product_placements
  ON product_placements.shard_id = product_shards.shard_id;
```

Results 1 Results 2 X

WITH product\_placements AS ( SELECT shardid as shard\_id ) Enter a SQL expression to filter results (use Ctrl+Space)

Grid	123 product_id	123 shard_id	node_name
1	3	102,009	citus-demo_worker_1
2	3	102,009	citus-demo_worker_2
3	3	102,009	citus-demo_worker_3

4. Di node/worker mana saja order dengan id 13 tersimpan? Tunjukkan shard id nya

```
WITH order_placements AS (  
    SELECT  
        shardid as shard_id,  
        nodename as node_name  
    FROM pg_dist_shard_placement  
),  
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 orders  
    WHERE order_id = 13  
)  
SELECT  
    order_shards.order_id,  
    order_shards.shard_id,  
    order_placements.node_name  
FROM order_shards  
INNER JOIN order_placements  
    ON order_placements.shard_id = order_shards.shard_id;
```

Skrip SQL ini bertujuan untuk menemukan lokasi fisik (node) dari shard yang menyimpan informasi tentang pesanan tertentu (dalam hal ini, order\_id 13) di dalam kluster Citus yang terdistribusi.

- CTE order\_placements
  - **pg\_dist\_shard\_placement:** Ini adalah tabel sistem Citus yang menyimpan informasi tentang penempatan shard, termasuk ID shard dan nama node di mana shard tersebut ditempatkan.
  - **CTE order\_placements:** Common Table Expression (CTE) ini memilih shardid dan nodename, mengganti nama kolom menjadi shard\_id dan node\_name untuk digunakan nanti.

- CTE `order_shards`
  - **orders:** Tabel `orders` berisi informasi pesanan seperti `order_id`, `user_id`, `total_price`, dan `created_at`.
  - **get\_shard\_id\_for\_distribution\_column:** Fungsi Citus yang mengambil ID shard untuk kolom distribusi tertentu. Dalam hal ini, kolom distribusi adalah `order_id` dari tabel `orders`.
  - **CTE `order_shards`:** CTE ini memilih `order_id`, menghitung `shard_id` menggunakan `get_shard_id_for_distribution_column`, dan membuat nama tabel nyata (`real_table_name`) untuk pesanan dengan `order_id` 13.
- Query Utama
  - **order\_shards:** CTE dari langkah sebelumnya yang berisi `order_id`, `shard_id`, dan `real_table_name` untuk pesanan `order_id` 13.
  - **order\_placements:** CTE dari langkah pertama yang berisi `shard_id` dan `node_name`.
  - **INNER JOIN:** Menggabungkan `order_shards` dengan `order_placements` berdasarkan `shard_id`, sehingga menghubungkan informasi pesanan dengan lokasi fisiknya di kluster.

Query ini menghasilkan:

- **order\_id:** ID dari pesanan 13.
- **shard\_id:** ID dari shard yang menyimpan data tentang pesanan 13.
- **node\_name:** Nama node di kluster Citus tempat shard tersebut disimpan.

#### DATA ENGINEER BATCH 4

Mentee: Yovina Silvia

Mentor: Bilal Benefit

order dengan id = 13 tersimpan di node\_name = citus-demo\_worker\_3

```
WITH order_placements AS (
  SELECT
    shardid as shard_id,
    nodename as node_name
  FROM pg_dist_shard_placement
),
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 orders
  WHERE order_id = 13
)
SELECT
  order_shards.order_id,
  order_shards.shard_id,
  order_placements.node_name
FROM order_shards
INNER JOIN order_placements
  ON order_placements.shard_id = order_shards.shard_id;
```

Results 1

Grid	order_id	shard_id	node_name
1	13	102,033	citus-demo_worker_3

## 5. Kapan sebaiknya kita menggunakan replication?

### Replication:

- Digunakan ketika memerlukan ketersediaan data yang tinggi dan pemulihan bencana.
- Cocok untuk aplikasi yang memerlukan load balancing untuk query baca.
- Digunakan pada sistem yang memerlukan toleransi kesalahan dan uptime tinggi.

## 6. Kapan sebaiknya kita menggunakan sharding?

### Sharding:

- Digunakan ketika volume data sangat besar dan memerlukan skalabilitas tinggi.
- Cocok untuk aplikasi dengan beban kerja baca dan tulis yang tinggi.
- Digunakan pada sistem yang memerlukan pemrosesan data cepat dan kapasitas penyimpanan yang luas.