PART 3 – Replication + Sharding

TASK

1. **Jelaskan perbedaan antara replication dan sharding**
   Jawabannya : Berikut penjelasan perbedaan antara replication dan sharding
   - **Replication (Replikasi)**
     Replikasi adalah proses menyalin dan memelihara salinan data yang sama pada beberapa server atau node. Tujuan utamanya adalah meningkatkan keandalan dan ketersediaan data. Ada dua jenis utama replikasi:
     - **Master-Slave Replication:**
       Master: Satu server (master) menerima semua operasi tulis (write operations).
       Slave: Server lainnya (slave) menerima salinan data dari master dan hanya digunakan untuk operasi baca (read operations).
       Keuntungan: Meningkatkan ketersediaan baca, memungkinkan backup lebih mudah, dan meningkatkan kinerja baca.
       Kekurangan: Penulisan tetap bergantung pada satu master, yang dapat menjadi bottleneck.
     - **Multi-Master Replication:**
       Semua node bisa menerima operasi tulis dan saling mereplikasi data.
       Keuntungan: Meningkatkan ketersediaan penulisan dan mengurangi bottleneck pada satu server.
       Kekurangan: Lebih kompleks dalam mengatasi konflik data.

   - **Sharding (Partisi Data)**
     Sharding adalah teknik memecah data besar menjadi beberapa bagian lebih kecil, yang disebut shards, yang disimpan di server atau node yang berbeda. Tujuan utama sharding adalah meningkatkan skalabilitas dan kinerja database.
     - **Horizontal Sharding (Pembagian Horizontal):** Membagi baris data dalam tabel yang sama ke dalam beberapa shard. Misalnya, jika Anda memiliki tabel pengguna, sebagian pengguna akan disimpan di shard pertama, sebagian lainnya di shard kedua, dan seterusnya.
       Keuntungan: Mengurangi beban pada satu server dan memungkinkan database menangani jumlah data yang sangat besar.
       Kekurangan: Memerlukan mekanisme untuk menentukan shard mana yang harus dituju untuk setiap operasi, yang bisa menambah kompleksitas.

     - **Vertical Sharding (Pembagian Vertikal):** Membagi kolom data dalam tabel yang sama ke dalam beberapa shard. Misalnya, data profil pengguna disimpan di satu shard dan data transaksi pengguna di shard lain.
       Keuntungan: Membuat setiap shard lebih ringan dan fokus pada jenis data tertentu.
       Kekurangan: Kurang umum digunakan karena seringkali tidak membagi beban secara merata.

   Perbandingan:
   - **Replication:**
     - Menyalin data yang sama ke beberapa node.

- Meningkatkan ketersediaan dan keandalan.
- Berguna untuk beban baca yang tinggi.

- **Sharding:**
  - Membagi data menjadi beberapa bagian.
  - Meningkatkan skalabilitas dan kinerja.
  - Berguna untuk mengelola set data yang sangat besar.

Keduanya sering digunakan bersama untuk mengoptimalkan kinerja dan keandalan database dalam skenario yang berbeda.

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

- ***Reference table***
  - **Membuat table users dan products untuk reference table**

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

```
CREATE TABLE products (
  product_id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  price NUMERIC(10, 2) NOT NULL
);
SELECT create_reference_table('products');
```

  - **Menambahkan data kedalam tabel users dan products**

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

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

- ***Distributed table***
  - **Membuat sequence pada tabel orders**

```
CREATE SEQUENCE orders_order_id_seq;
```

  - **Membuat tabel orders dengan menggunakan tabel distributed**

```
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');
```

- **Membuat sequence pada tabel order_details**

```
CREATE SEQUENCE order_details_order_detail_id_seq;
```

- **Membuat tabel order_details dengan menggunakan tabel distributed**

```
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');
```

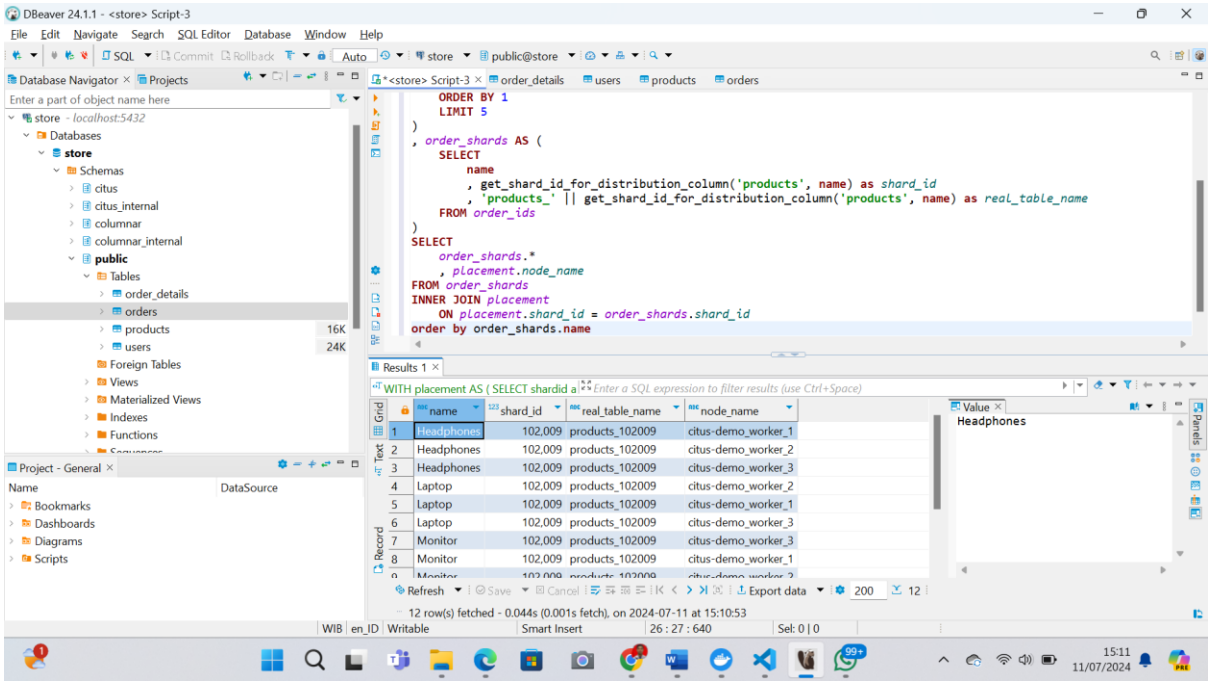
- **Menambahkan data ke dalam tabel orders dan tabel order_details**

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

3. **Di node/worker mana saja product "Headphone" tersimpan? Tunjukkan shard id nya**
   Jawabannya : Di node/worker 1, 2 dan 3 dan shard_id nya 102,009

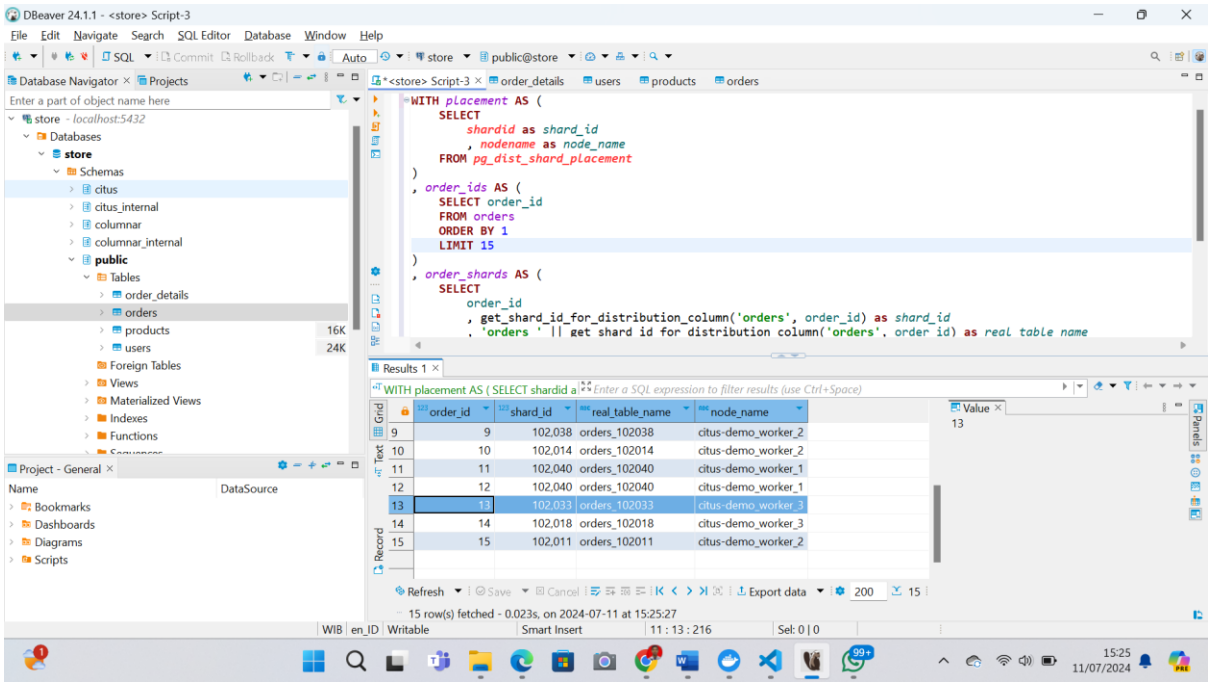
```
WITH placement AS (
    SELECT
        shardid as shard_id
        , nodename as node_name
    FROM pg_dist_shard_placement
)
, order_ids AS (
    SELECT name
    FROM products
    ORDER BY 1
    LIMIT 5
)
, order_shards AS (
    SELECT
        name
        , get_shard_id_for_distribution_column('products', name) as shard_id
        , 'products_' || get_shard_id_for_distribution_column('products', name) 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
order by order_shards.name
;
```



4. **Di node/worker mana saja order dengan id 13 tersimpan? Tunjukkan shard id nya**
   Jawabannya : Di node/worker 3 dan shard_id nya 102,033

```
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 1
    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
order by order_shards.order_id
;
```



5. **Kapan sebaiknya kita menggunakan replication?**

   Jawabannya : Replikasi sebaiknya digunakan dalam situasi-situasi tertentu untuk meningkatkan kinerja, keandalan, dan ketersediaan data dalam sistem basis data. Berikut adalah beberapa kasus di mana replikasi sangat bermanfaat:

   1. Meningkatkan Kinerja Baca

   Jika aplikasi Anda memiliki beban baca yang sangat tinggi dan beban tulis yang relatif rendah, replikasi dapat digunakan untuk mendistribusikan permintaan baca ke beberapa salinan data (replica). Dengan demikian, beban pada server utama (master) berkurang, dan kinerja baca meningkat.

   2. Toleransi Kesalahan dan Ketersediaan Tinggi

   Replikasi dapat meningkatkan ketersediaan sistem dengan menyediakan salinan data di beberapa node. Jika satu node gagal atau mengalami gangguan, node lain yang menyimpan salinan data tetap dapat melayani permintaan, sehingga sistem tetap berfungsi tanpa downtime yang signifikan.

   3. Backup dan Recovery

   Replikasi memungkinkan pembuatan salinan data secara otomatis, yang dapat digunakan sebagai backup. Jika terjadi kerusakan data atau kesalahan pengguna, data dapat dipulihkan dari salinan yang direplikasi, meminimalkan risiko kehilangan data.

   4. Skalabilitas Horizontal

   Dengan replikasi, Anda dapat menambahkan lebih banyak node untuk menangani beban kerja yang meningkat. Ini membantu meningkatkan skalabilitas sistem secara horizontal, memungkinkan penambahan kapasitas tanpa harus meningkatkan kapasitas perangkat keras pada satu server.

5. Isolasi Workload
Replikasi dapat digunakan untuk mengisolasi workload yang berbeda. Misalnya, Anda dapat memiliki satu node yang khusus untuk melayani analisis data berat (heavy analytics) sementara node lain menangani operasi transaksi harian (daily transactions). Ini mencegah satu jenis workload mengganggu kinerja jenis workload lainnya.

6. Lokasi Geografis yang Terpencar
Jika pengguna Anda tersebar di berbagai lokasi geografis, replikasi memungkinkan Anda menyimpan salinan data di beberapa lokasi. Ini mengurangi latensi karena permintaan baca dapat dilayani oleh server yang paling dekat dengan pengguna.

7. Maintenance Tanpa Downtime
Dengan replikasi, Anda dapat melakukan pemeliharaan pada satu node tanpa mengganggu ketersediaan sistem. Misalnya, Anda dapat memperbarui atau memelihara server master sementara permintaan baca tetap dilayani oleh node slave.

Jenis-Jenis Replikasi dan Kapan Digunakan:

- **Master-Slave Replication:**
  Kapan digunakan: Ketika beban baca tinggi dan Anda ingin mendistribusikan beban baca ke beberapa server tanpa mempengaruhi server utama.

- **Multi-Master Replication:**
  Kapan digunakan: Ketika Anda memerlukan ketersediaan tinggi untuk operasi tulis dan beban tulis tersebar di beberapa lokasi geografis atau ketika beberapa server perlu melakukan operasi tulis secara bersamaan.

- **Synchronous Replication:**
  Kapan digunakan: Ketika Anda memerlukan konsistensi data yang tinggi dan tidak boleh ada kehilangan data meskipun ada kegagalan.

- **Asynchronous Replication:**
  Kapan digunakan: Ketika Anda dapat mentoleransi sedikit ketidakkonsistenan data dan lebih mengutamakan kinerja serta latensi rendah.

Kesimpulan
Replikasi sebaiknya digunakan untuk meningkatkan kinerja baca, ketersediaan data, toleransi kesalahan, dan skalabilitas sistem. Jenis replikasi yang dipilih harus disesuaikan dengan kebutuhan spesifik aplikasi Anda, termasuk pola beban kerja, kebutuhan konsistensi data, dan lokasi pengguna.

6. **Kapan sebaiknya kita menggunakan sharding?**
   Jawabannya : Sharding, atau pemisahan data secara horizontal, sebaiknya digunakan dalam situasi-situasi tertentu untuk mengelola volume data yang sangat besar dan meningkatkan kinerja basis data. Berikut adalah beberapa skenario di mana sharding sangat bermanfaat:

   1. Volume Data yang Sangat Besar

Jika volume data Anda sangat besar sehingga tidak dapat dikelola dengan baik oleh satu server, sharding adalah solusi yang tepat. Dengan memecah data menjadi beberapa shard, Anda dapat menyimpan dan mengelola lebih banyak data secara efisien.

2. Pertumbuhan Data yang Pesat
Jika aplikasi Anda mengalami pertumbuhan data yang sangat cepat, sharding memungkinkan Anda untuk mengatasi peningkatan ini tanpa harus terus-menerus meningkatkan kapasitas server tunggal. Anda dapat menambahkan shard baru sesuai kebutuhan untuk mengakomodasi pertumbuhan data.

3. Beban Kerja yang Tidak Merata
Jika beban kerja Anda tidak merata (misalnya, beberapa tabel atau jenis data lebih sering diakses daripada yang lain), sharding memungkinkan Anda untuk mendistribusikan beban ini secara lebih merata di seluruh server, menghindari bottleneck pada satu server.

4. Kinerja Query yang Buruk
Jika query terhadap database menjadi lambat karena ukuran tabel yang sangat besar, sharding dapat membantu meningkatkan kinerja dengan memecah tabel besar menjadi beberapa bagian yang lebih kecil. Query terhadap shard yang lebih kecil biasanya lebih cepat karena data yang dicari lebih sedikit.

5. Isolasi Data Berdasarkan Pengguna atau Lokasi
Jika aplikasi Anda melayani pengguna dari berbagai lokasi geografis atau berbagai segmen pengguna, sharding dapat digunakan untuk mengisolasi data berdasarkan lokasi atau segmen pengguna. Misalnya, data pengguna dari wilayah tertentu dapat disimpan di shard khusus untuk wilayah tersebut, mengurangi latensi akses data.

6. Skalabilitas Horizontal
Jika Anda ingin meningkatkan kapasitas basis data secara horizontal (dengan menambahkan lebih banyak server), sharding adalah metode yang tepat. Dengan memecah data ke beberapa shard, Anda dapat menambahkan lebih banyak server untuk menangani peningkatan beban kerja tanpa harus meningkatkan kapasitas perangkat keras dari satu server.

7. Manajemen Sumber Daya yang Lebih Baik
Jika Anda ingin mengelola sumber daya server secara lebih efektif, sharding memungkinkan Anda untuk mendistribusikan beban pemrosesan dan penyimpanan di beberapa server. Ini membantu dalam mengoptimalkan penggunaan CPU, memori, dan ruang penyimpanan di seluruh kluster.

Kapan Sebaiknya Tidak Menggunakan Sharding:

- Volume Data Kecil: Jika volume data Anda masih kecil dan dapat dikelola dengan satu server, sharding mungkin tidak diperlukan karena akan menambah kompleksitas.
- Konsistensi Transaksi: Jika aplikasi Anda memerlukan konsistensi transaksi yang sangat tinggi dan sulit untuk dipecah menjadi shard yang independen, sharding bisa menjadi tantangan.
- Beban Kerja Tulis Tinggi pada Shard Tunggal: Jika satu shard menerima beban kerja tulis yang sangat tinggi, masalah performa mungkin tidak sepenuhnya terselesaikan

dengan sharding dan dapat memerlukan pendekatan lain seperti optimasi indeks atau replikasi.

Kesimpulan

Sharding sebaiknya digunakan ketika volume data sangat besar, pertumbuhan data cepat, beban kerja tidak merata, atau kinerja query menurun karena ukuran tabel yang besar. Sharding memungkinkan peningkatan kapasitas dan kinerja database secara horizontal, mengisolasi data berdasarkan pengguna atau lokasi, dan mengelola sumber daya secara lebih efektif. Namun, sharding juga menambah kompleksitas, jadi pastikan bahwa manfaatnya sepadan dengan upaya yang dibutuhkan untuk implementasi dan pemeliharaan.