DATA WAREHOUSE

PART 1

1. Perbedaan Data Warehouse dan Data Lake

- **Data Warehouse:**
  - **Struktur Data:** Terstruktur dan dikelola dengan baik.
  - **Tujuan:** Digunakan untuk analisis bisnis, pelaporan, dan query.
  - **Jenis Data:** Data historis yang sudah diproses dan terstruktur.
  - **Penyimpanan:** Menggunakan tabel dengan skema yang telah ditentukan (schema-on-write).
  - **Kecepatan Query:** Cepat dalam pengambilan data karena data sudah terstruktur.
  - **Contoh Teknologi:** Amazon Redshift, Google BigQuery, Snowflake.
- **Data Lake:**
  - **Struktur Data:** Tidak terstruktur, semi-terstruktur, dan terstruktur.
  - **Tujuan:** Menyimpan data mentah untuk analisis lebih lanjut.
  - **Jenis Data:** Data mentah dari berbagai sumber, termasuk file log, media, teks, dll.
  - **Penyimpanan:** Menggunakan pendekatan schema-on-read, data disimpan dalam format asli.
  - **Kecepatan Query:** Bisa lebih lambat karena data belum terstruktur.
  - **Contoh Teknologi:** Apache Hadoop, Amazon S3, Azure Data Lake.

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

- **OLAP (Online Analytical Processing):**
  - **Fokus:** Analisis data dan query yang kompleks.
  - **Jenis Query:** Agregasi, laporan, dan analisis data besar.
  - **Struktur Data:** Dimensi dan fakta, sering menggunakan skema bintang atau salju.
  - **Optimisasi:** Dioptimalkan untuk query read-heavy dan analisis.
  - **Transaksi:** Mendukung transaksi batch besar dengan frekuensi update yang rendah.
  - **Contoh Teknologi:** Microsoft SQL Server Analysis Services (SSAS), Oracle OLAP, Teradata.
- **OLTP (Online Transaction Processing):**
  - **Fokus:** Pengolahan transaksi harian.
  - **Jenis Query:** Insert, update, delete, dan select sederhana.
  - **Struktur Data:** Normalisasi tinggi untuk mengurangi redundansi data.
  - **Optimisasi:** Dioptimalkan untuk transaksi write-heavy dengan latensi rendah.
  - **Transaksi:** Mendukung transaksi kecil dengan frekuensi update yang tinggi.
  - **Contoh Teknologi:** MySQL, PostgreSQL, Microsoft SQL Server.

3. Teknologi yang Biasanya Dipakai untuk Data Warehouse

- **Amazon Redshift**
- **Google BigQuery**
- **Snowflake**
- **Teradata**
- **Microsoft Azure Synapse Analytics**
- **IBM Db2 Warehouse**
- **Oracle Exadata**

4. Perintah Instalasi Citus Menggunakan Docker Compose

Jalankan Docker Compose:

Di terminal, jalankan perintah berikut untuk memulai layanan:

docker-compose up -d

Verifikasi Instalasi:

Gunakan perintah berikut untuk memeriksa apakah kontainer sudah berjalan:

docker ps

Konfigurasi Citus:

Masuk ke kontainer master dan tambahkan worker:

docker exec -it <container_id_of_master> psql -U postgres -c "SELECT master_add_node('citus_worker', 5432);"

5. Perbedaan Antara Access Method Heap dan Columnar Citus

- **Heap:**
  - **Struktur:** Menyimpan data dalam format baris (row-based).
  - **Kinerja:** Optimal untuk OLTP, di mana transaksi sering melakukan operasi insert, update, dan delete.
  - **Contoh:** PostgreSQL heap table.
  - **Keunggulan:** Bagus untuk query yang mengakses banyak kolom dari satu baris.
- **Columnar:**
  - **Struktur:** Menyimpan data dalam format kolom (column-based).
  - **Kinerja:** Optimal untuk OLAP, di mana query lebih sering membaca data dalam jumlah besar untuk analisis.
  - **Contoh:** Citus columnar table.
  - **Keunggulan:** Bagus untuk query yang membaca beberapa kolom tetapi dari banyak baris, karena pengambilan data lebih cepat dan efisien.

PART 2

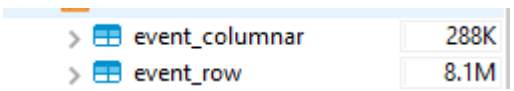
Menjalankan citus di docker compose :

```
$ docker compose start
time="2024-07-20T12:40:21+07:00" level=warning msg="D:\\Altera\\AnggiSDC\\demo
 citus\\citus-demo\\docker-compose.yml: `version` is obsolete"
[+] Running 5/5
 ✓ Container citus-demo_master    Start...                    0.6s
 ✓ Container citus-demo_manager   Star...                     0.7s
```

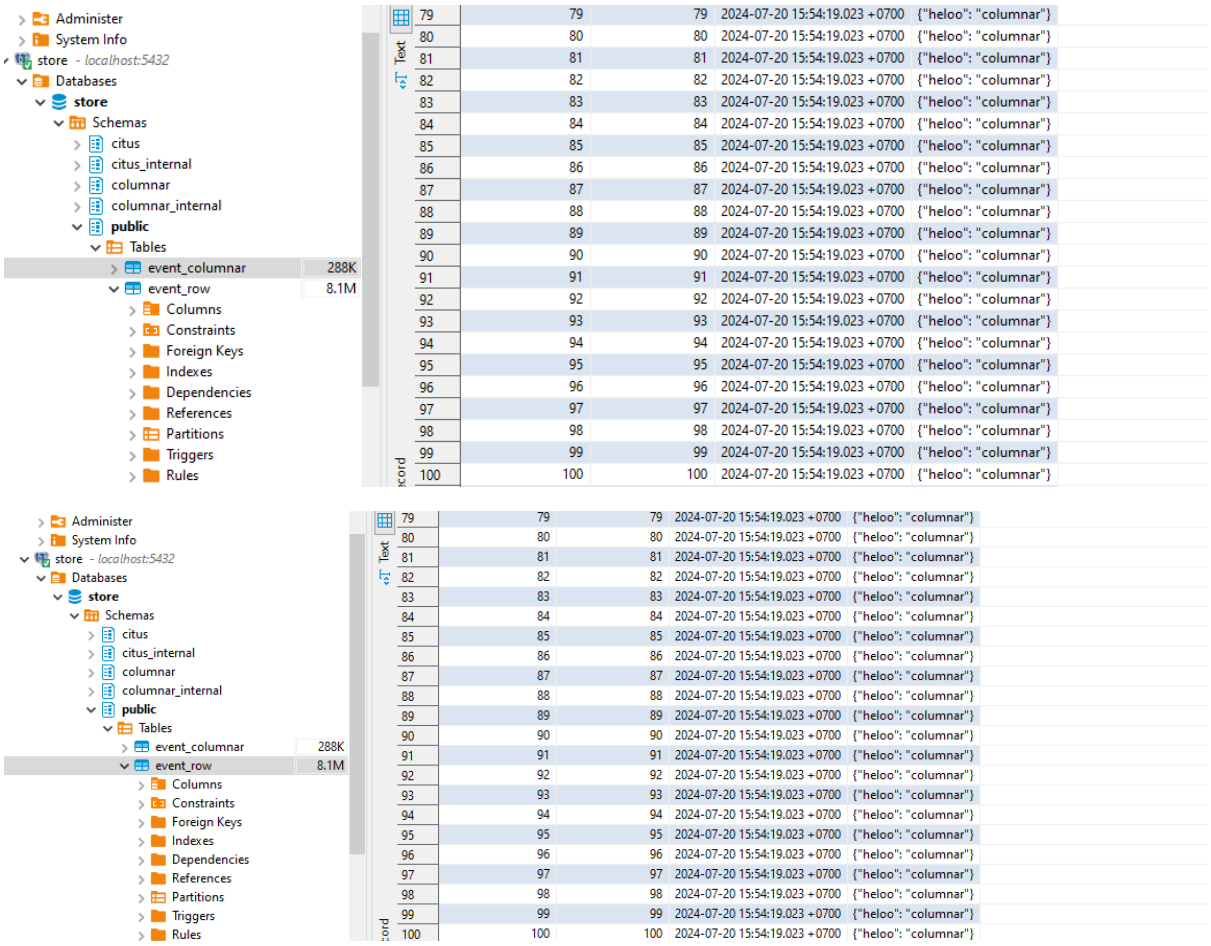
Perintah untuk membuat table biasa dan columnar:

```
CREATE TABLE event_columnar (
       device_id bigint,
       event_id bigserial,
       event_time timestamptz default now(),
       data jsonb not null
)
USING columnar;

-- insert some data
INSERT INTO event_columnar (device_id, data)
SELECT d, '{"heloo":"columnar"}' from generate_series(1,100000) d;

--create a row-based table to compare

CREATE TABLE event_row AS SELECT * FROM event_columnar;
```

Perbedaan ukuran table :



100 data biasa dan columnar :



Kesimpulan :

Ukuran yang di hasilkan dari table columnar menjadi lebih kecil di banding ukuran table biasa, tetapi tidak mengurangi isi table itu sendiri

PART 3

Replication dan sharding adalah dua teknik yang sering digunakan dalam manajemen basis data untuk meningkatkan kinerja, skalabilitas, dan keandalan. Berikut adalah penjelasan mengenai perbedaan antara keduanya:

Replication

Definisi: Replication adalah proses menduplikasi data dari satu server database ke server database lain. Tujuannya adalah untuk memastikan bahwa salinan data yang sama tersedia di beberapa lokasi.

Manfaat:

1. **Ketersediaan Tinggi:** Jika satu server gagal, salinan lain tetap tersedia untuk melayani permintaan.

2. **Performa Bacaan:** Membaca data dari beberapa salinan bisa meningkatkan performa bacaan karena permintaan bisa didistribusikan ke beberapa server.
3. **Pemulihan Bencana:** Memiliki salinan data di lokasi yang berbeda membantu dalam pemulihan data jika terjadi kerusakan fisik pada salah satu server.

Cara Kerja:

- **Master-Slave Replication:** Satu server bertindak sebagai master yang menerima semua penulisan, dan perubahan tersebut direplikasi ke beberapa server slave yang hanya digunakan untuk membaca.
- **Multi-Master Replication:** Beberapa server bertindak sebagai master, dan semua server dapat menerima penulisan dan pembacaan. Perubahan direplikasi ke semua master lainnya.

Sharding

Definisi: Sharding adalah teknik membagi data menjadi beberapa bagian yang lebih kecil, disebut shard, dan mendistribusikan shard tersebut ke beberapa server.

Manfaat:

1. **Skalabilitas:** Membagi data ke beberapa server memungkinkan sistem untuk menangani lebih banyak permintaan dengan membagi beban kerja.
2. **Performa:** Mengurangi ukuran dataset pada setiap server bisa meningkatkan kinerja karena setiap server menangani lebih sedikit data.
3. **Isolasi Kesalahan:** Jika satu shard mengalami masalah, shard lain tetap berfungsi normal.

Cara Kerja:

- **Horizontal Partitioning:** Data dibagi berdasarkan baris. Misalnya, baris dengan ID 1-1000 disimpan di server A, 1001-2000 di server B, dan seterusnya.
- **Range-Based Sharding:** Data dibagi berdasarkan rentang nilai tertentu.
- **Hash-Based Sharding:** Data di-hash dan kemudian didistribusikan ke shard berdasarkan hasil hash tersebut.

Perbedaan Utama

1. **Tujuan:**
    - **Replication:** Meningkatkan ketersediaan dan keandalan dengan menyediakan salinan data yang sama di beberapa lokasi.
    - **Sharding:** Meningkatkan skalabilitas dan performa dengan membagi data ke beberapa server.
2. **Implementasi:**
    - **Replication:** Semua salinan (replica) menyimpan data yang sama.
    - **Sharding:** Setiap shard menyimpan subset data yang berbeda.
3. **Pemakaian:**
    - **Replication:** Cocok untuk aplikasi yang memerlukan ketersediaan tinggi dan performa bacaan yang cepat.

- **Sharding:** Cocok untuk aplikasi dengan dataset yang sangat besar yang memerlukan pembagian beban kerja.

Dengan memahami perbedaan dan manfaat dari kedua teknik ini, organisasi dapat memilih strategi yang paling sesuai dengan kebutuhan mereka untuk mengelola basis data.

Node/worker dengan ID 13 seperti pada gambar di bawah ini, dan untuk melakukannya lakukan query seperti di bawah ini :

```
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
```

	order_id	shard_id	real_table_name	node_name
6	7	102,018	orders_102018	citus-demo_worker_3
7	4	102,018	orders_102018	citus-demo_worker_3
8	6	102,030	orders_102030	citus-demo_worker_3
9	13	102,033	orders_102033	citus-demo_worker_3
10	8	102,010	orders_102010	citus-demo_worker_1
11	5	102,016	orders_102016	citus-demo_worker_1
12	3	102,025	orders_102025	citus-demo_worker_1
13	2	102,034	orders_102034	citus-demo_worker_1

Penggunaan Replication

Replication sebaiknya digunakan dalam situasi berikut:

1. **Ketersediaan Tinggi (High Availability)**:
    - Untuk memastikan bahwa sistem tetap dapat beroperasi meskipun salah satu node atau server mengalami kegagalan. Dengan replication, data disalin ke beberapa server sehingga jika satu server gagal, data masih bisa diakses dari server lain.
2. **Peningkatan Kapasitas Pembacaan (Read Scalability)**:
    - Untuk menangani beban baca yang tinggi. Dengan adanya replika, permintaan pembacaan dapat didistribusikan ke beberapa server, sehingga mengurangi beban pada server utama.
3. **Pemulihan Bencana (Disaster Recovery)**:
    - Untuk menjaga cadangan data yang dapat diakses jika terjadi bencana atau kegagalan sistem yang luas. Data yang direplikasi ke lokasi geografis yang berbeda dapat membantu dalam pemulihan cepat.
4. **Mengurangi Latensi Akses Data**:
    - Untuk memberikan akses data yang lebih cepat kepada pengguna yang berada di lokasi geografis yang berbeda. Data dapat direplikasi ke server yang lebih dekat dengan pengguna, sehingga mengurangi latensi.

Penggunaan Sharding

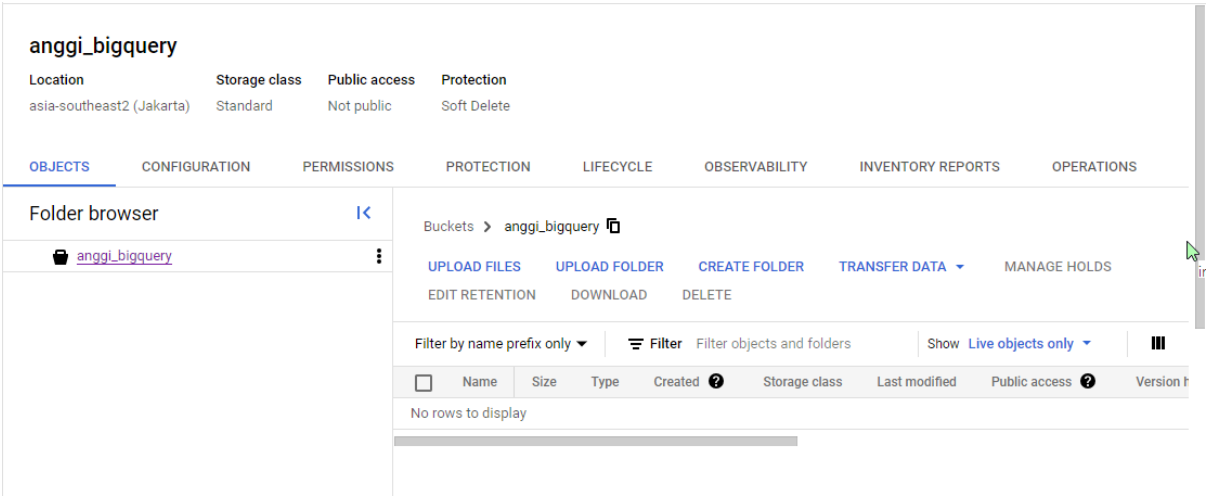
Sharding sebaiknya digunakan dalam situasi berikut:

1. **Peningkatan Kapasitas Penulisan (Write Scalability)**:
    - Untuk menangani beban tulis yang sangat tinggi. Dengan sharding, data dibagi ke beberapa shard, sehingga beban penulisan dapat didistribusikan ke beberapa server, memungkinkan peningkatan kapasitas penulisan.
2. **Mengelola Volume Data yang Sangat Besar**:
    - Untuk sistem dengan volume data yang sangat besar sehingga tidak mungkin disimpan dalam satu server. Sharding memungkinkan data dibagi ke beberapa server, sehingga setiap server hanya menyimpan sebagian dari total data.
3. **Menghindari Batasan Kapasitas Penyimpanan Server**:
    - Untuk mengatasi batasan kapasitas penyimpanan dari satu server. Dengan membagi data ke beberapa server, batasan kapasitas penyimpanan dapat diatasi.
4. **Peningkatan Performa Query Khusus**:
    - Untuk meningkatkan performa query tertentu yang dapat dioptimalkan melalui distribusi data yang lebih spesifik. Misalnya, data pengguna dapat di-shard berdasarkan wilayah geografis untuk meningkatkan efisiensi query terkait lokasi.

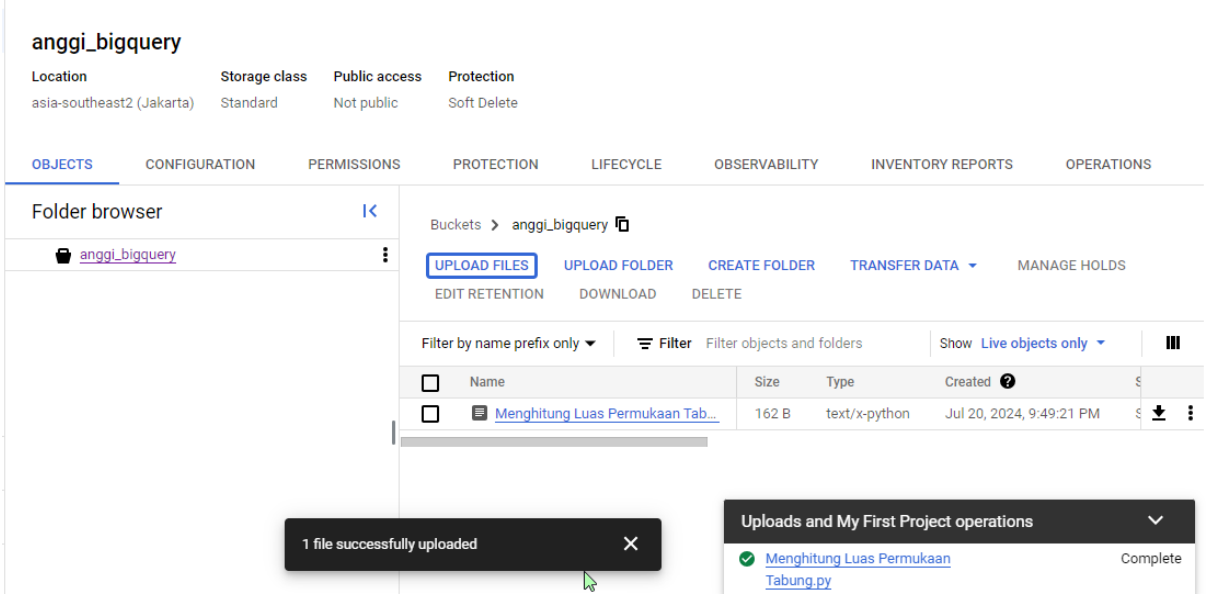
Dengan memahami kebutuhan spesifik dari sistem yang Anda kelola, Anda dapat memilih antara replication dan sharding, atau bahkan menggabungkan keduanya untuk mendapatkan kinerja dan ketersediaan yang optimal.

PART 4

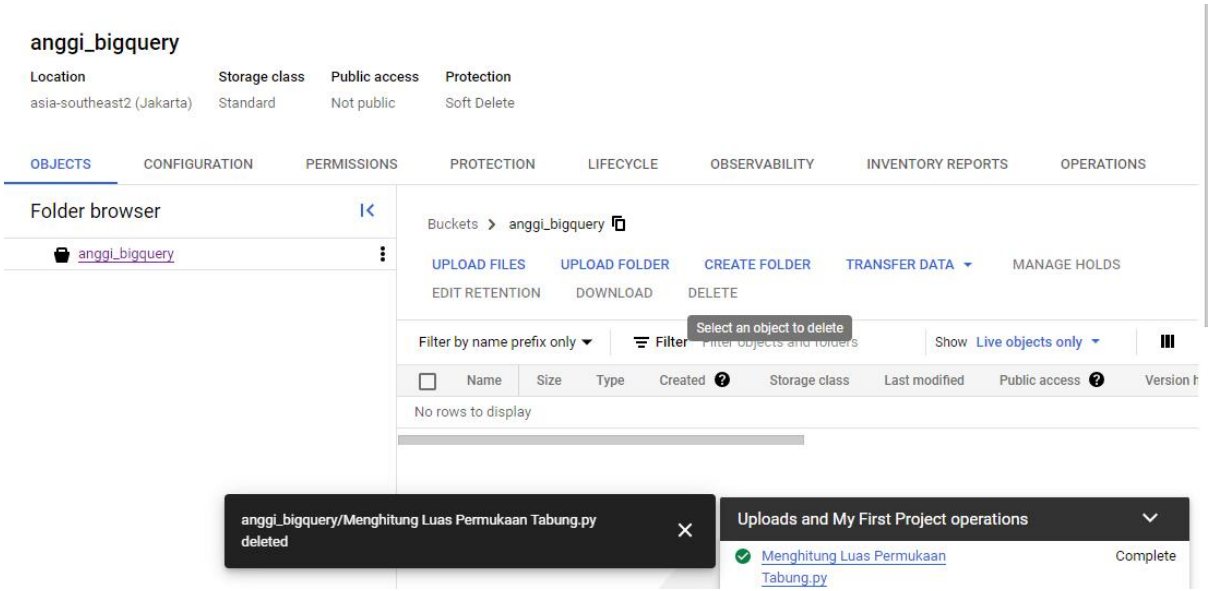
Buat bucket :



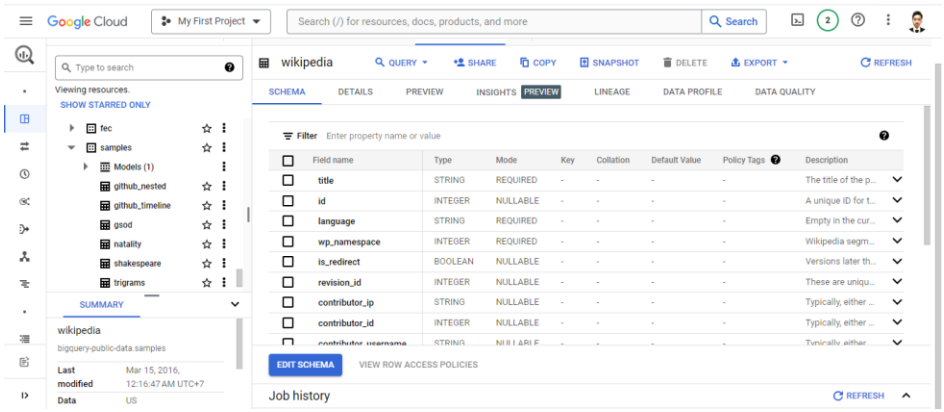
Upload file :



Hapus File :



EKSPLORASI WIKIPEDIA :



MEMUNCULKAN JUMLAH KONTRIBUSI DARI BESAR KE KECIL:

```
SELECT *FROM(SELECT contributor_ip, COUNT(id) AS TotalContrib FROM `bigquery-public-data.samples.wikipedia` GROUP BY contributor_ip LIMIT 10) as a ORDER BY a.TotalContrib DESC
```