DATA BUILD TOOL

STUDENT : WARTADI
MENTOR : BILAL BENEFIT

alterra

PART 1

DBT

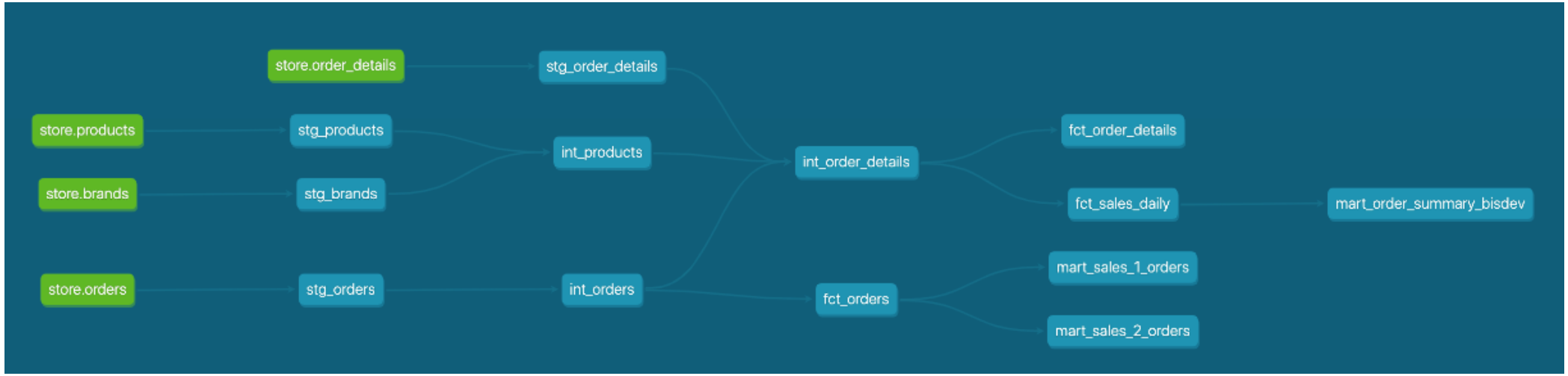
- DBT adalah alat yang menyediakan framework yang kuat dan efisien untuk transformasi data dalam data warehouse, memastikan data yang siap digunakan untuk analisis selalu up-to-date, konsisten, dan terorganisir dengan baik. Hal ini membantu organisasi untuk lebih efektif dalam pengambilan keputusan berbasis data.

Perkembangan teknologi yang semakin maju dan di dunia data atau big data dalam beberapa tahun terakhir ini terutama teknologi Data Warehouse menjadi sangat fleksibel (contohnya UDF) dan powerful.Misalnya adanya fitur yang dapat memisahkan penyimpanan dan pemrosesan, proses scaling yang elastis dan kemampuan Machine Learning (Big Query ML). Hal ini mengakibatkan banyak perusahaan yang menggunakan data warehouse untuk meningkatkan performa proses transformasi data. Oleh sebab itu teknologi DBT mulai populer karena menyediakan cara yang mudah dalam melakukan proses transformasi hanya dengan menggunakan query SQL. Selain itu menyediakan pemeriksaan data quality secara asli.

DEPEDENCY TREE

- **Dependency Tree** dalam DBT menggambarkan bagaimana model-model transformasi data saling bergantung satu sama lain. Ini memungkinkan pengelolaan urutan eksekusi transformasi data secara otomatis, memastikan bahwa setiap model dieksekusi dalam urutan yang tepat berdasarkan ketergantungannya.
- **Cara Kerja Dependency Tree:**
- **Model Hierarki:** Setiap model dalam DBT dapat bergantung pada satu atau lebih model lainnya. Ketergantungan ini didefinisikan melalui penggunaan referensi (ref()) dalam skrip SQL DBT.
- **Urutan Eksekusi:** DBT secara otomatis menentukan urutan eksekusi yang tepat berdasarkan ketergantungan yang ditentukan, sehingga data selalu konsisten dan up-to-date.
- **Visualisasi:** Dependency tree dapat divisualisasikan dalam DBT, memungkinkan pengguna untuk melihat dan memahami bagaimana model-model saling terhubung dan mempengaruhi satu sama lain.

DEPEDENCY TREE



Alur	Fungsi
• Data mentah (store.*) diproses menjadi staging models (stg_*). • Staging models kemudian diolah menjadi intermediate models (int_*), yang menggabungkan dan memproses data lebih lanjut. • Intermediate models kemudian digunakan untuk membuat fact models (fct_*), yang merupakan representasi data yang siap untuk analisis. • Fact models kemudian diolah menjadi mart models (mart_*), yang biasanya digunakan untuk kebutuhan bisnis atau analisis yang lebih spesifik.	**Staging Models (stg_*):** Menyiapkan dan membersihkan data mentah untuk digunakan lebih lanjut.**Intermediate Models (int_*):** Menggabungkan dan mengolah data dari staging models untuk membuat data lebih konsisten dan terstruktur.**Fact Models (fct_*):** Menyediakan data yang siap dianalisis dengan metrik dan dimensi yang telah diproses.**Mart Models (mart_*):** Mengubah data fact menjadi bentuk yang lebih spesifik sesuai kebutuhan analisis bisnis.

Virsioning

Virsioning dalam DBT menggunakan version control system seperti Git untuk melacak perubahan, memungkinkan branching, merging, dan rollback, serta mendukung praktik pengembangan perangkat lunak yang baik seperti code review.

Fungsi	Kelebihan
Pelacakan Perubahan: Setiap perubahan yang dibuat pada model DBT dicatat dalam VCS, memungkinkan pengguna untuk melihat siapa yang membuat perubahan, kapan, dan apa yang diubah. **Branching dan Merging:** Pengguna dapat bekerja pada cabang (branch) yang terpisah, mengembangkan dan menguji perubahan sebelum digabungkan (merge) ke cabang utama. **Revert Changes:** Jika terjadi kesalahan atau masalah, pengguna dapat dengan mudah mengembalikan proyek ke versi sebelumnya. **Code Review:** Versioning memungkinkan praktik code review, di mana perubahan dapat ditinjau oleh anggota tim lain sebelum diterapkan.	**Kolaborasi yang Lebih Baik:** Memungkinkan banyak anggota tim untuk bekerja pada proyek yang sama tanpa mengganggu pekerjaan satu sama lain. **Transparansi dan Akuntabilitas:** Memastikan bahwa semua perubahan dapat dilacak kembali ke individu tertentu, meningkatkan akuntabilitas. **Stabilitas dan Keandalan:** Meminimalkan risiko kesalahan dengan memungkinkan pengujian dan tinjauan perubahan sebelum diterapkan ke produksi.

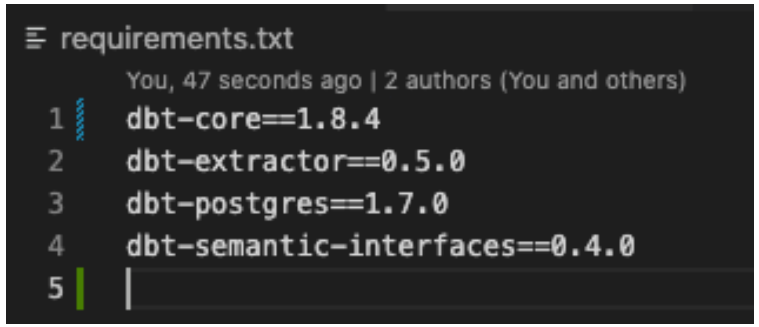
DATA BUILD TOOL

PART 2 – CREATE BDT PROJECT (STAGING LAYER)

Setup venv and install DBT

No	Perintah	Output
1.1	Jika dalam computer belum terinstall tools yang digunakan untuk proses pembuatan di DBT, maka perlu melakukan perintah sebagai berikut **python -m venv**	wartadi@Wartadis-MacBook-Pro my_project % Dalam hal ini output perintah tidak menghasilkan perbedaan
	source .venv/bin/activate Setelah menjalankan perintah ini, prompt shell Anda biasanya akan berubah, menandakan bahwa virtual environment aktif.	(.venv) wartadi@Wartadis-MacBook-Pro dbt-demo % Mengaktifkan virtual environment sehingga semua paket Python yang diinstal atau digunakan adalah milik lingkungan terisolasi ini, yang menghindari dampak pada instalasi Python global.
	pip install dbt-postgres	Perintah ini menginstal paket dbt-postgres ke dalam virtual environment yang aktif. dbt-postgres adalah adapter untuk menggunakan DBT (Data Build Tool) dengan basis data PostgreSQL. Dalam kasus ini di computer yang saya gunakan sudah terinstall sebelumya

Create requirements.txt



Untuk menginstall nya cukup jalan kan perintah

pip install –r requirements.txt

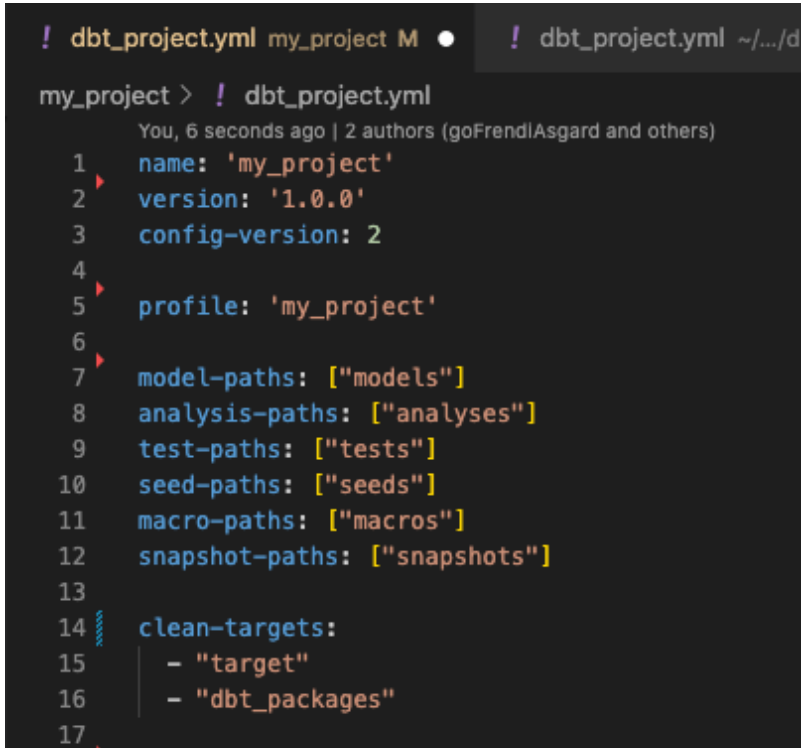
Daftar requirements.txt mencakup paket-paket yang diperlukan untuk bekerja dengan DBT dan PostgreSQL, serta beberapa alat tambahan untuk memperluas fungsionalitas DBT:

dbt-core: Paket inti DBT untuk menjalankan dan mengelola model data.
dbt-extractor: Digunakan untuk mengekstrak metadata dari DBT.
dbt-postgres: Adapter untuk menghubungkan DBT dengan basis data PostgreSQL.
dbt-semantic-interfaces: Menambahkan dukungan untuk antarmuka semantik dalam DBT.

Dengan menginstal paket-paket ini, Anda memastikan bahwa kita memiliki semua alat yang diperlukan untuk menjalankan, mengelola, dan memperluas proyek DBT dengan PostgreSQL sebagai basis data.

Setup Project & Configuration DBT

- **Inisialisasi Proyek DBT**:
  dbt init my_project: Membuat direktori proyek DBT baru bernama **my_project.**
- **Configuration DBT**
  Kemudian setelah itu bisa dilakukan setup konfigurasi dbt project dengan cara membuat file dimana dalam kasus ini diberi nama **dbt_project.yml**. Konfigurasi tersebut bisa dilihat dibawah ini



File dbt_project.yml adalah konfigurasi proyek DBT (Data Build Tool) yang mengatur berbagai aspek dari proyek data tersebut. Berikut adalah penjelasan dari setiap bagian dalam file ini:

1.Informasi Proyek:

1. **name: 'my_project'**: Menentukan nama proyek DBT.
2. **version: '1.0.0'**: Menentukan versi proyek.
3. **config-version: 2**: Menentukan versi konfigurasi yang digunakan. Ini adalah versi skema konfigurasi DBT.

2.Profil Proyek:

1. **profile: 'my_project'**: Menentukan profil DBT yang digunakan untuk proyek ini. Profil ini biasanya berisi informasi koneksi ke database dan pengaturan lainnya yang diperlukan untuk menjalankan proyek.

3.Jalur Pencarian File:

1. **model-paths: ["models"]**: Menentukan direktori di mana DBT mencari file model.
2. **analysis-paths: ["analyses"]**: Menentukan direktori untuk file analisis.
3. **test-paths: ["tests"]**: Menentukan direktori untuk file uji.
4. **seed-paths: ["seeds"]**: Menentukan direktori untuk file seed.
5. **macro-paths: ["macros"]**: Menentukan direktori untuk file makro.
6. **snapshot-paths: ["snapshots"]**: Menentukan direktori untuk file snapshot.

4.Direktori yang Dibersihkan:

1. **clean-targets**: Menentukan direktori yang akan dihapus oleh perintah dbt clean.
   "target" & "dbt_packages"

Setup Project & Configuration DBT

Berikut merupakan lanjutan perintah **dbt_project.yml**

```
17
18  models:
19    my_project:
20      store:
21        _stg:
22          +materialized: view
23          +schema: stg
24          +database: store
25        _int:
26          +materialized: view
27          +schema: int
28          +database: store
29        _fct:      You, 9 seconds ago • Uncommit
30          +materialized: table
31          +schema: fact
32          +database: store
33        _mart:
34          +materialized: table
35          +schema: mart
36          +database: store
```

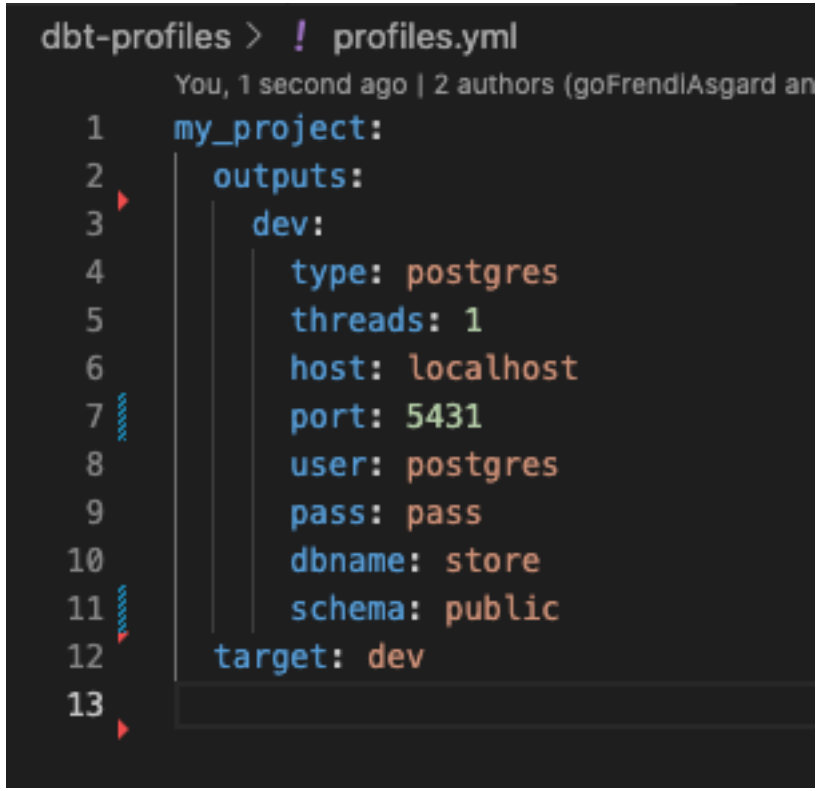
Konfigurasi Model:

•**models: dbt_project**: Bagian ini mengonfigurasi bagaimana model di dalam proyek my_project akan dibangun dan diatur.

- **store**: Menentukan konfigurasi untuk model yang berada di direktori store.
  - **_stg**:
    - +materialized: view: Model ini akan dimaterialisasikan sebagai tampilan.
    - +schema: stg: Model ini akan berada dalam skema stg.
    - +database: store: Model ini akan berada dalam basis data store.
  - **_int**:
    - +materialized: view: Model ini akan dimaterialisasikan sebagai tampilan.
    - +schema: int: Model ini akan berada dalam skema int.
    - +database: store: Model ini akan berada dalam basis data store.
  - **_fct**:
    - +materialized: table: Model ini akan dimaterialisasikan sebagai tabel.
    - +schema: fact: Model ini akan berada dalam skema fact.
    - +database: store: Model ini akan berada dalam basis data store.
  - **_mart**:
    - +materialized: table: Model ini akan dimaterialisasikan sebagai tabel.
    - +schema: mart: Model ini akan berada dalam skema mart.
    - +database: store: Model ini akan berada dalam basis data store.

Setup DBT Profiles

Berikut merupakan konfirgurasi **profiles.yml** yang digunakan dalam project ini



File **profile.yml** adalah konfigurasi penting dalam proyek DBT yang mengatur bagaimana DBT terhubung ke database dan menentukan lingkungan pengembangan(environtment development) yang digunakan. Dengan mengkonfigurasi profil ini, kita bisa memastikan bahwa DBT dapat menjalankan model data dengan benar di lingkungan yang sesuai, serta memungkinkan pengelolaan koneksi ke berbagai lingkungan (seperti dev, test, prod) dengan mudah.

Dalam kasus ini database diberinama store dengan skema public dengn port 5431

- Setelah membuat profiles.yml selanjutnya bisa dilakukan perintah sebagai berikut diterminal

(.venv) wartadi@Wartadis-MacBook-Pro dbt-demo % **export DBT_PROFILES_DIR=$(pwd)/dbt-profiles**

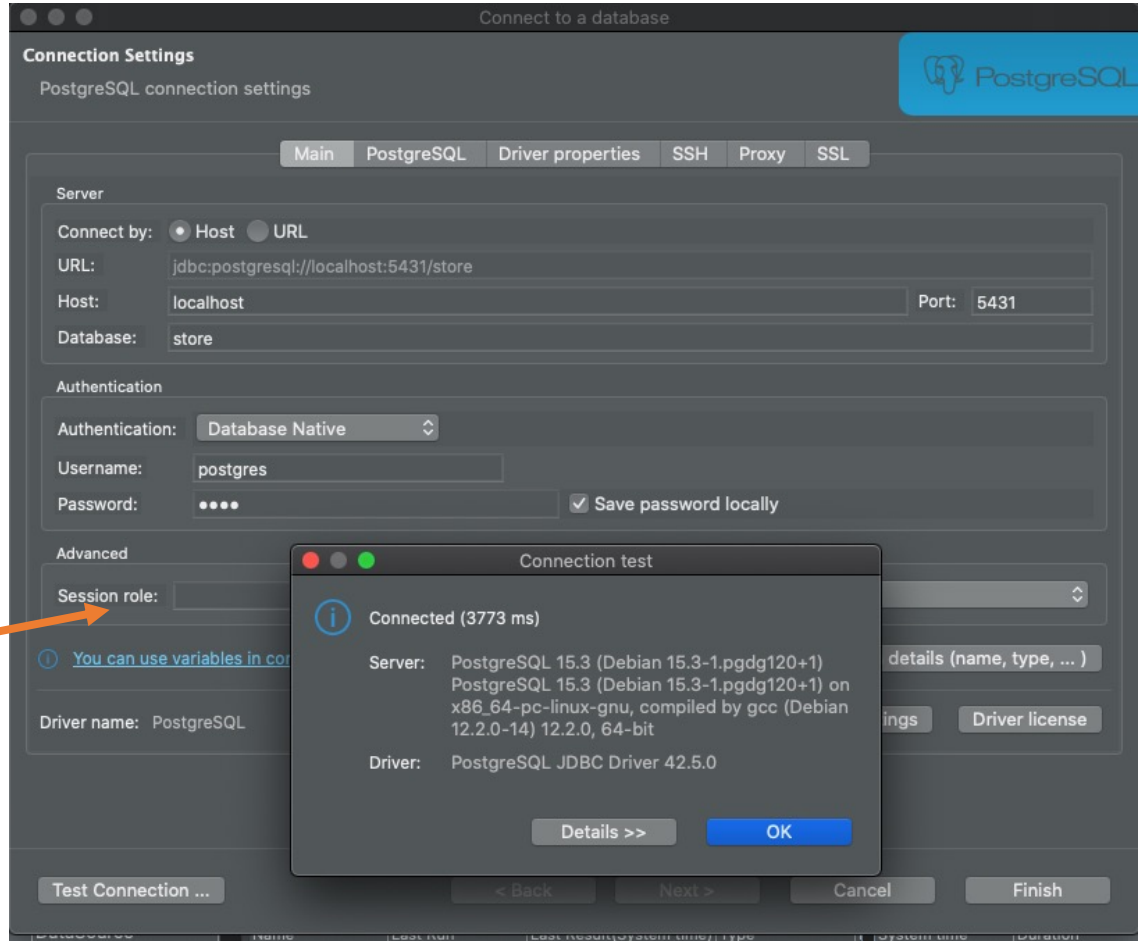
Dimana dalam hal ini DBT akan mencari file profiles.yml di direktori dbt-profiles yang berada di dalam direktori my_project saat ini. Ini memungkinkan DBT untuk menemukan dan menggunakan konfigurasi koneksi yang sesuai untuk proyek tersebut.

Create Postgre Docker Compose

```
(.venv) wartadi@Wartadis-MacBook-Pro dbt-demo % docker-compose up -d
[+] Running 3/3
 ⠿ Container dbt-demo_master    Started
 ⠿ Container dbt-demo_manager   Started
 ⠿ Container dbt-demo-worker-1  Started
```

- Perintah **docker-compose up -d** digunakan untuk menjalankan layanan yang didefinisikan dalam file docker-compose.yml di latar belakang (detached mode). Dalam konteks ini, perintah tersebut telah berhasil menjalankan tiga kontainer: dbt-demo_master, dbt-demo_manager, dan dbt-demo_worker-1. Berikut adalah penjelasan fungsi dari setiap layanan berdasarkan file docker-compose.yml yang diberikan sebelumnya:

- Kemudian untuk gambar disamping adalah proses untuk membuat connection data base dalam hal ini menggunakan PostgreSQL .
- Dengan konfigurasi yang sudah ditentukan sebelumnya baik di profile.yml dan docker compose.yml



Create Postgre Docker Compose

Kesimpulan:

Konfigurasi ini mendefinisikan tiga layanan dalam kluster Citus:

1. **Master**: Layanan utama yang berfungsi sebagai koordinasi dan eksekusi query untuk seluruh kluster.
2. **Worker**: Node tambahan yang melakukan eksekusi query di subset data yang dibagi oleh master.
3. **Manager**: Mengelola keanggotaan dalam kluster Citus dan berinteraksi dengan Docker untuk konfigurasi.

Konfigurasi ini menghubungkan layanan dalam satu jaringan dan menggunakan volume untuk data persistensi serta inisialisasi database. Semua layanan terhubung melalui postgres-network dan dikonfigurasi dengan variabel lingkungan yang serupa untuk konsistensi dan kemudahan pengaturan.

Defining Source

Dalam hal ini kita membuat file dengan nama raw_schema.yml sebagai konfigurasi untuk alat seperti dbt (data build tool) yang mendefinisikan **sumber data (source)** dan tabel-tabel yang ada dalam **skema public** dari **database store**. Konfigurasi ini meliputi **tabel brands**, **products**, **orders**, dan **order_details**. Setiap tabel memiliki kolom-kolom yang didefinisikan dengan deskripsi dan pengujian tertentu. Berikut penjelasannya:

```yaml
version: 2

sources:
  - name: store
    database: store
    schema: public
```

Struktur Umum

- **Version**: Menunjukkan versi konfigurasi, dalam hal ini adalah versi 2.
- **Sources**: Bagian ini mendefinisikan sumber data utama, dalam hal ini database store dengan skema public.

Defining Source

```yaml
tables:
  - name: brands
    columns:
      - name: brand_id
        description: "Unique identifier for each brand"
        tests:
          - unique
          - not_null
      - name: name
        description: "Name of the brand"
        tests:
          - not_null
```

```yaml
  - name: products
    columns:
      - name: product_id
        description: "Unique identifier for each product"
        tests:
          - unique
          - not_null
      - name: brand_id
        description: "Foreign key referencing brands"
        tests:
          - relationships:
              to: source('store', 'brands')
              field: brand_id
      - name: name
        description: "Name of the product"
        tests:
          - not_null
      - name: price
        description: "Price of the product"
        tests:
          - not_null
```

Table: brands

- **brand_id**:
  - Deskripsi: Identifier unik untuk setiap brand.
  - Tests: Harus unik dan tidak boleh null.
- **name**:
  - Deskripsi: Nama dari brand.
  - Tests: Tidak boleh null.

Table: products

- **product_id**:
  - Deskripsi: Identifier unik untuk setiap produk.
  - Tests: Harus unik dan tidak boleh null.
- **brand_id**:
  - Deskripsi: Foreign key yang mereferensikan tabel brands.
  - Tests: Relasi ke kolom brand_id di tabel brands.
- **name**:
  - Deskripsi: Nama dari produk.
  - Tests: Tidak boleh null.
- **price**:
  - Deskripsi: Harga dari produk.
  - Tests: Tidak boleh null.

Defining Source

```yaml
- name: orders
  columns:
    - name: order_id
      description: "Unique identifier for each order"
      tests:
        - unique
        - not_null
    - name: order_date
      description: "Date and time the order was placed"
      tests:
        - not_null
```

```yaml
- name: order_details
  columns:
    - name: order_detail_id
      description: "Unique identifier for each order detail"
      tests:
        - unique
        - not_null
    - name: order_id
      description: "Foreign key referencing orders"
      tests:
        - relationships:
            to: source('store', 'orders')
            field: order_id
    - name: product_id
      description: "Foreign key referencing products"
      tests:
        - relationships:
            to: source('store', 'products')
            field: product_id
    - name: quantity
      description: "Quantity of the product ordered"
      tests:
        - not_null
    - name: price
      description: "Price of the product in the order"
      tests:
        - not_null
```

Table: orders

- **order_id**:
  - Deskripsi: Identifier unik untuk setiap order.
  - Tests: Harus unik dan tidak boleh null.
- **order_date**:
  - Deskripsi: Tanggal dan waktu ketika order dibuat.
  - Tests: Tidak boleh null.

Table: order_details

- **order_detail_id**:
  - Deskripsi: Identifier unik untuk setiap detail order.
  - Tests: Harus unik dan tidak boleh null.
- **order_id**:
  - Deskripsi: Foreign key yang mereferensikan tabel orders.
  - Tests: Relasi ke kolom order_id di tabel orders.
- **product_id**:
  - Deskripsi: Foreign key yang mereferensikan tabel products.
  - Tests: Relasi ke kolom product_id di tabel products.
- **quantity**:
  - Deskripsi: Jumlah produk yang dipesan.
  - Tests: Tidak boleh null.
- **price**:
  - Deskripsi: Harga produk dalam order.
  - Tests: Tidak boleh null.

Create Postgre Docker Compose

Dalam project ini Postgre Docker Compose diberinma **docker-compose.yml. Dimana docker-compose dalam perintahnya adalah** konfigurasi untuk menjalankan kluster Citus, yang merupakan ekstensi untuk PostgreSQL yang memungkinkan pembagian data dan eksekusi query secara paralel di beberapa node.

docker-compose.yml

You, 9 minutes ago | 2 authors (goFrendiAsgard and others)

```yaml
version: '3'

services:

  master:
    container_name: "${COMPOSE_PROJECT_NAME:-citus}_master"
    image: "citusdata/citus:12.0.0"
    ports:
      - "${COORDINATOR_EXTERNAL_PORT:-5431}:5432"
    labels:
      - "com.citusdata.role=Master"
    environment: &AUTH
      - POSTGRES_PASSWORD=pass
      - POSTGRES_USER=postgres
      - POSTGRES_DB=store
      - PGUSER=postgres
      - POSTGRES_HOST_AUTH_METHOD=trust
    networks:
      - postgres-network
    volumes:
      - ./citus-db-data/:/var/lib/postgresql/data/
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
```

docker-compose.yml

```yaml
  worker:
    image: "citusdata/citus:12.0.0"
    labels:
      - "com.citusdata.role=Worker"
    depends_on:
      - manager
    environment: *AUTH
    command: "/wait-for-manager.sh"
    volumes:
      - ./citus-healthcheck:/healthcheck

  manager:
    container_name: "${COMPOSE_PROJECT_NAME:-citus}_manager"
    image: "citusdata/membership-manager:0.3.0"
    volumes:
      - "${DOCKER_SOCK:-/var/run/docker.sock}:/var/run/docker.sock"
      - ./citus-healthcheck:/healthcheck
    depends_on:
      - master
    environment: *AUTH

networks:
  postgres-network:
    driver: bridge
```

Create Macros

Macros di dbt (data build tool) digunakan untuk mendefinisikan fungsi-fungsi yang dapat digunakan kembali dalam SQL dan konfigurasi dbt. Macros memungkinkan kita untuk menulis logika sekali dan menggunakannya di banyak tempat dalam proyek, sehingga meningkatkan keterbacaan, modularitas, dan pemeliharaan kode.

Dalam kasus ini macros yang dibuat untuk menormalisasi nomor telfon yang tidak seragam.

Implementasi pada proyek ini macros dibuayt dan disimpan pada path

/Users/wartadi/Desktop/alta/dbt-demo/my_project/macros/phone_materialization.sql

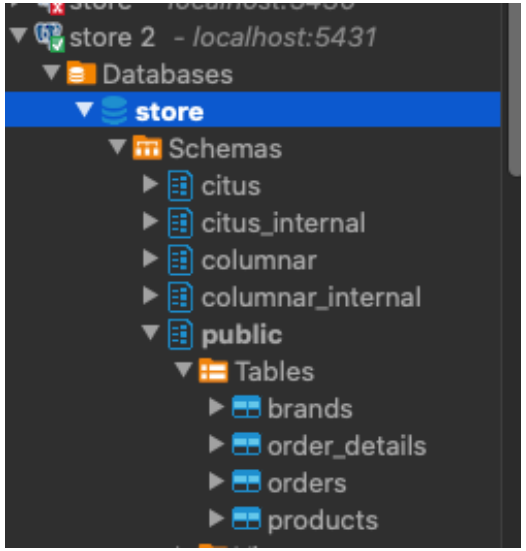
my_project > macros > phone_materialization.sql

```
1  {% macro normalize_phone_number(column_name) %}
2      ltrim({{ column_name }}, '+')
3  {% endmacro %}
```

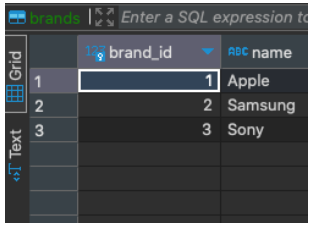
Dalam hal inimacro normalize_phone_number digunakan untuk membersihkan nomor telepon dengan menghapus tanda plus (+) dari awal string. Ini membantu dalam menstandarisasi format nomor telepon dalam data, yang dapat memudahkan pemrosesan dan analisis lebih lanjut.

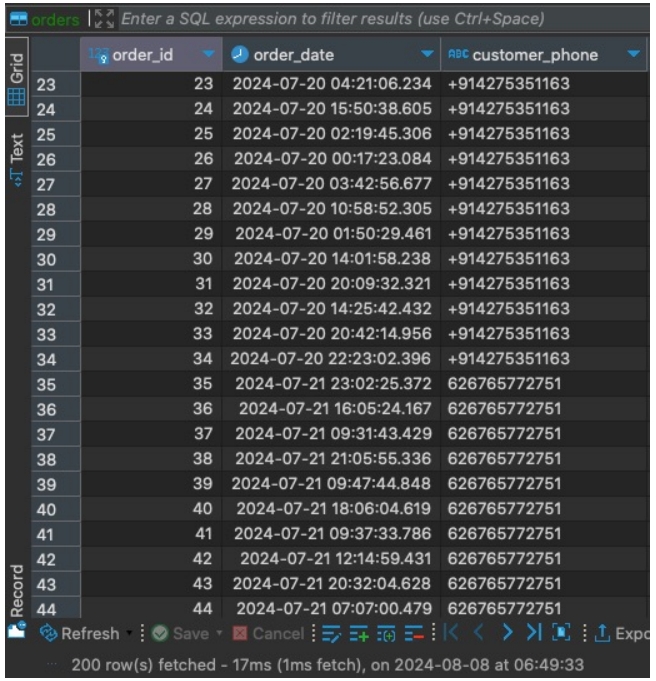
Source Data

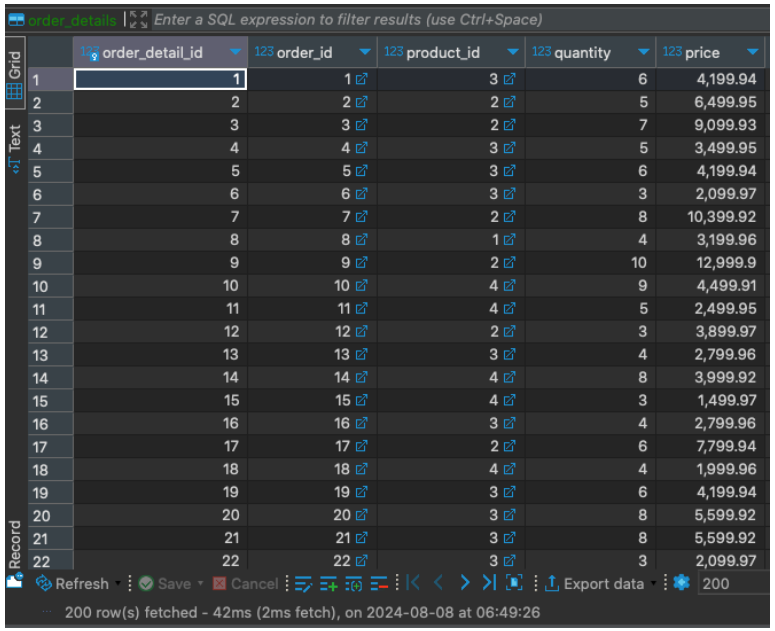
- Kita mempunyai database dengan dimana source disimpan dalam schema public.



- Bisa kita lihat bahwa ditabel order colom costumer_phone data tersebut tidak seragam dimana nomor telepon ada yang menggunakan + dan ada yg tidak.
- Dalam hal ini dilayer selanjutnya(**staging**) costumer_phone harus sudah ada format yang lebih rapih agar bisa digunakan dengan mudah untuk proses berikutnya.



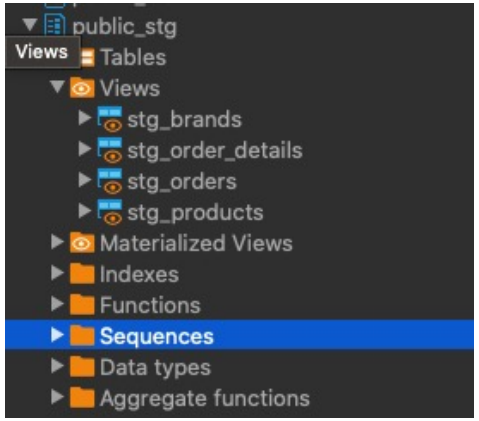




Model/ Layer Staging –Run Models

dbt run untuk run semua model didalam project
dbt run –select models/store/_stg
Perintah ini digunakan untuk running dbt pada model staging saja. Dimana ditunjukkan telah berhasil create 4 view model pada schma **public_stg** pada masing-masing table yang telah dikonfigurasi kan pada model _stg table mana yang mau di staging dari data source.

```
(.venv) wartadi@Wartadis-MacBook-Pro my_project % dbt run --select models/store/_stg
23:38:36  Running with dbt=1.7.0
23:38:37  Registered adapter: postgres=1.7.0
23:38:37  Found 13 models, 40 tests, 4 sources, 0 exposures, 0 metrics, 799 macros, 0 groups, 0 semantic models
23:38:37
23:38:38  Concurrency: 1 threads (target='dev')
23:38:38
23:38:38  1 of 4 START sql view model public_stg.stg_brands ..... [RUN]
23:38:39  1 of 4 OK created sql view model public_stg.stg_brands ..... [CREATE VIEW in 0.39s]
23:38:39  2 of 4 START sql view model public_stg.stg_order_details ..... [RUN]
23:38:39  2 of 4 OK created sql view model public_stg.stg_order_details ..... [CREATE VIEW in 0.29s]
23:38:39  3 of 4 START sql view model public_stg.stg_orders ..... [RUN]
23:38:39  3 of 4 OK created sql view model public_stg.stg_orders ..... [CREATE VIEW in 0.25s]
23:38:39  4 of 4 START sql view model public_stg.stg_products ..... [RUN]
23:38:40  4 of 4 OK created sql view model public_stg.stg_products ..... [CREATE VIEW in 0.18s]
23:38:40
23:38:40  Finished running 4 view models in 0 hours 0 minutes and 2.19 seconds (2.19s).
23:38:40
23:38:40  Completed successfully
23:38:40
23:38:40  Done. PASS=4 WARN=0 ERROR=0 SKIP=0 TOTAL=4
```



Untuk mengkonfirmasi bahwa proses running yang sudah dijalankan berhasil atau tidak bisa juga melihat database. Apakah ada update atau belum.

Dalam kasus ini terlihat dalam **database store** sudah ada pada schema **public_stg** dengan **views stg_brand, stg_order_details, stg_order** dan **stg_products**

Model/ Layer Staging

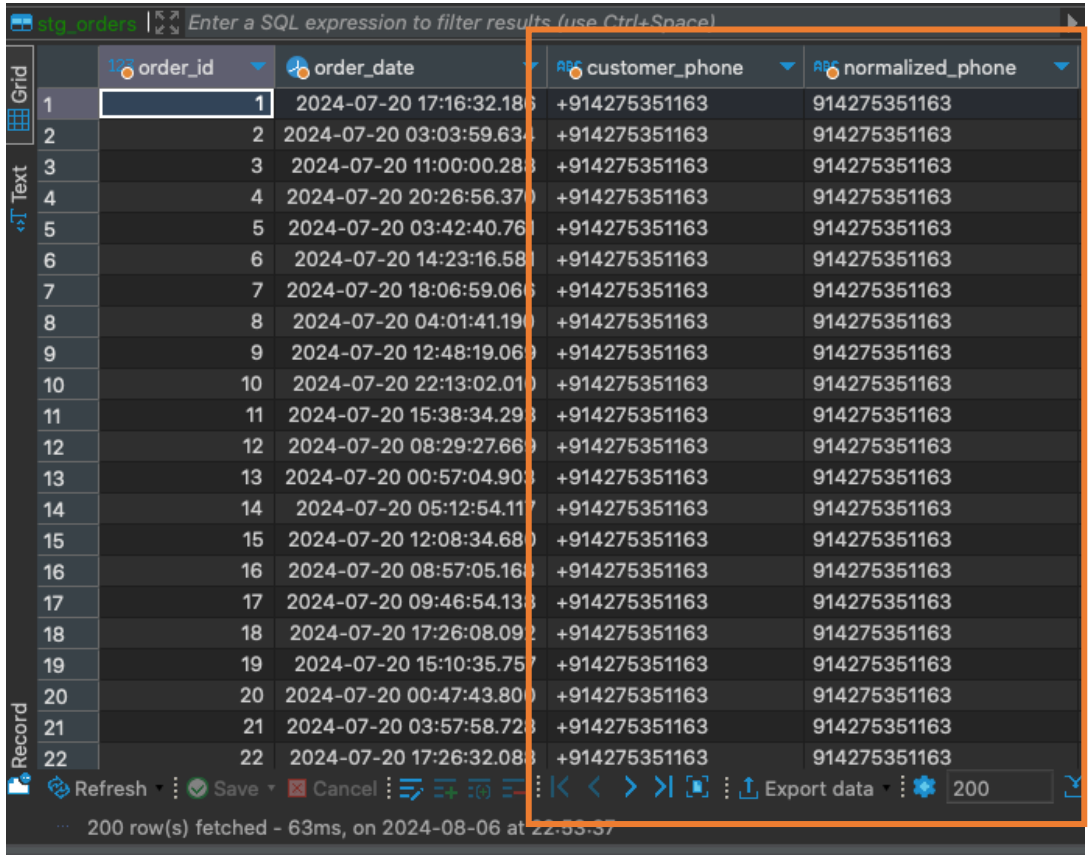
- Pada layer staging tidak banyak melakukan transformasi data Dalam layer ini dilakukan **t**ransformasi dasar seperti pengubahan format tanggal dan normaliasasi data agar konsisten yang mana dalam kasus ini adalah nomor costumer_phone
- Selain itupada layer ini juga membersihkan data mentah yang diambil dari sumber data asli. Ini termasuk mengubah tipe data, menangani nilai null.
- Sebagai perwakilan gambar dibawah ini merupakan konfigurasi digunakan pada **stg_order.sql** dilayer staging.

my_project > models > store > _stg > stg_orders.sql

```
1  select
2      order_id::int as order_id
3      , order_date::timestamp as order_date
4      , customer_phone
5      , {{ normalize_phone_number('orders.customer_phone') }} as normalized_phone
6  from {{ source('store', 'orders') }}
```

Query ini mengambil data dari tabel orders dalam database store dan melakukan beberapa transformasi pada kolom-kolomnya:
•Mengonversi kolom order_id menjadi tipe data integer.
•Mengonversi kolom order_date menjadi tipe data timestamp.
•Mengambil kolom customer_phone tanpa transformasi.
•Menormalkan nomor telepon pada kolom customer_phone menggunakan fungsi **normalize_phone_number** yang memanggil dari **model macros** yang kita buat sebelumnya dan menyimpan hasilnya dalam kolom baru normalized_phone.

Dibawah ini terlihat perbedaan dari table stg_orders bahwa sudah ada normalisasi untuk customer_phone(menghilangkan tanda +) dan kita simpan menjadi kolom baru normalized_phone yang mana sebelumnya tidak ada sebelum dilakukan transformasi (table orders).

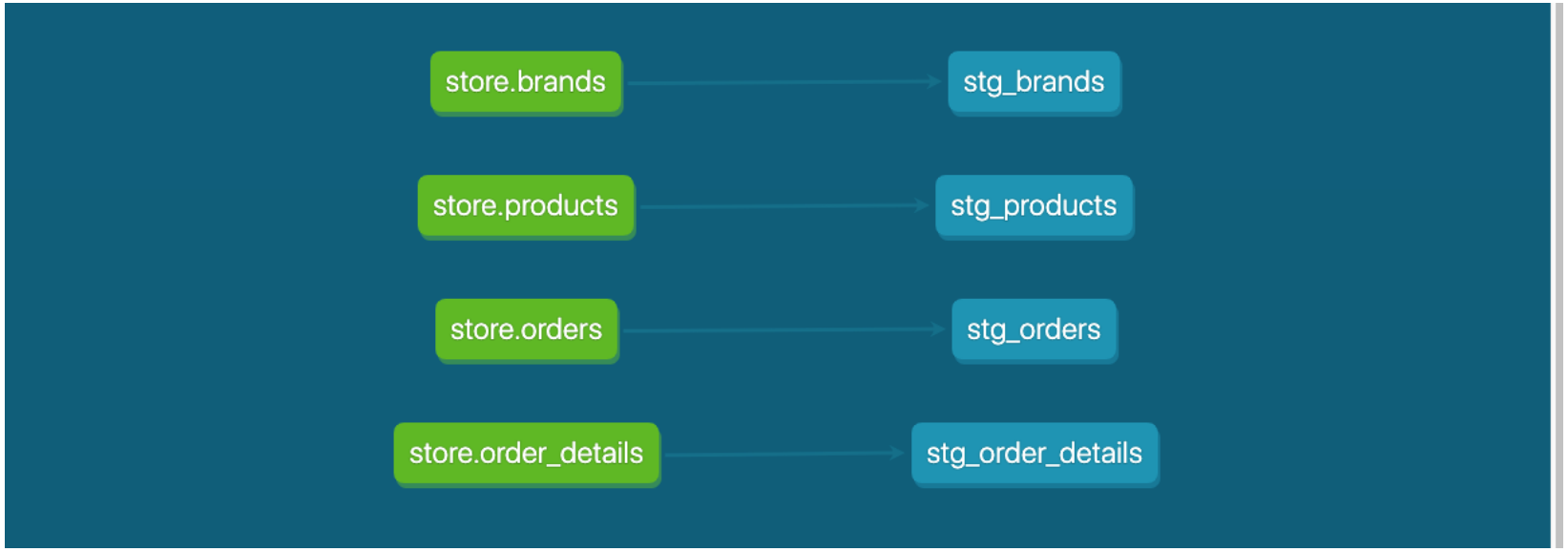


Model/ Layer Staging

Berikut merupakan line graph yang terbentuk setelah running models pada layer staging. Dimana terlihat tidak ada penggabungan data dari antar table. Pada layer layer ini sesuai yang sudah dijelaskan sebelumnya bahwa telah dilakukan transformasi antara lain :

Data yang sudah dinormalisasi dan dikonversi yang mana hal ini akan lebih mudah untuk diolah dalam layer-layer berikutnya (intermediate, fact, dan mart).

Dengan demikian, pada layer ini sangat penting dalam memastikan kualitas data di layer awal dari pipeline ETL (Extract, Transform, Load)



alterra

DATA BUILD TOOL

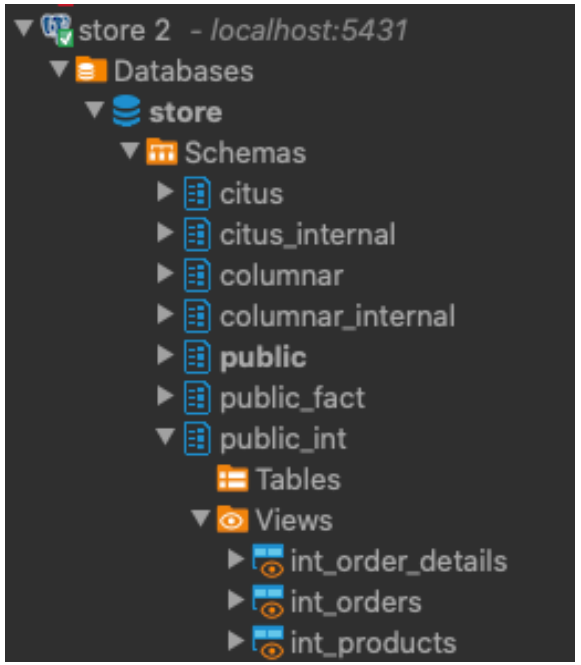
PART 3 – CREATE BDT PROJECT (STAGING INT,FACT,MART)

Model Layer Intermediate

Pada layer intermediate digunakan untuk melakukan transformasi data yang lebih kompleks dibandingkan dengan layer stagingImplementasi pada project layer ini adalah:

- agregasi data dari layer staging.
- Menggabungkan data dari beberapa tabel staging untuk membentuk dataset yang lebih komprehensif.
- Menyiapkan data yang sudah diproses dan diolah untuk digunakan di layer fact atau mart.

```
(.venv) wartadi@Wartadis-MacBook-Pro my_project % dbt run --select models/store/_int
05:43:32  Running with dbt=1.7.0
05:43:33  Registered adapter: postgres=1.7.0
05:43:33  Unable to do partial parsing because a project config has changed
05:43:39  Found 13 models, 40 tests, 4 sources, 0 exposures, 0 metrics, 799 macros, 0 groups, 0 semantic models
05:43:39
05:43:41  Concurrency: 1 threads (target='dev')
05:43:41
05:43:41  1 of 3 START sql view model public_int.int_orders ..... [RUN]
05:43:41  1 of 3 OK created sql view model public_int.int_orders ..... [CREATE VIEW in 0.58s]
05:43:41  2 of 3 START sql view model public_int.int_products ..... [RUN]
05:43:41  2 of 3 OK created sql view model public_int.int_products ..... [CREATE VIEW in 0.15s]
05:43:41  3 of 3 START sql view model public_int.int_order_details ..... [RUN]
05:43:41  3 of 3 OK created sql view model public_int.int_order_details ..... [CREATE VIEW in 0.17s]
05:43:42
05:43:42  Finished running 3 view models in 0 hours 0 minutes and 2.40 seconds (2.40s).
05:43:42
05:43:42  Completed successfully
```



Jika dilihat pada gambar dibawah ini telah berhasil running program pada **models/store/_int** bahwa sudah terbentuk data **int_orders, int_products** dan **int_order_details** pada base store dengan materliazed view dengan schema public_int **(**untuk penyimpanan sendiri merujuk kepada konfigurasi yang kita bangun pada **dbt_project.yml)**

Model/ Layer Intermediate

Berikut konfigurasi model yang dibangun pada int_order_details.sql. Dalam hal ini dilakukan penambahan kolom **“country”** melalui nomor telepon yang sudah dilakukan pada layer staging. Dengan case bahwa pada kolom normalized_phone jyang berawalan **62** adalah **Indonesia** dan **91** adalah **India**.

Pada Query ini bertujuan untuk menggabungkan data dari beberapa tabel staging (**stg_order_details, int_orders, int_products**) dan melakukan beberapa transformasi dan normalisasi data.

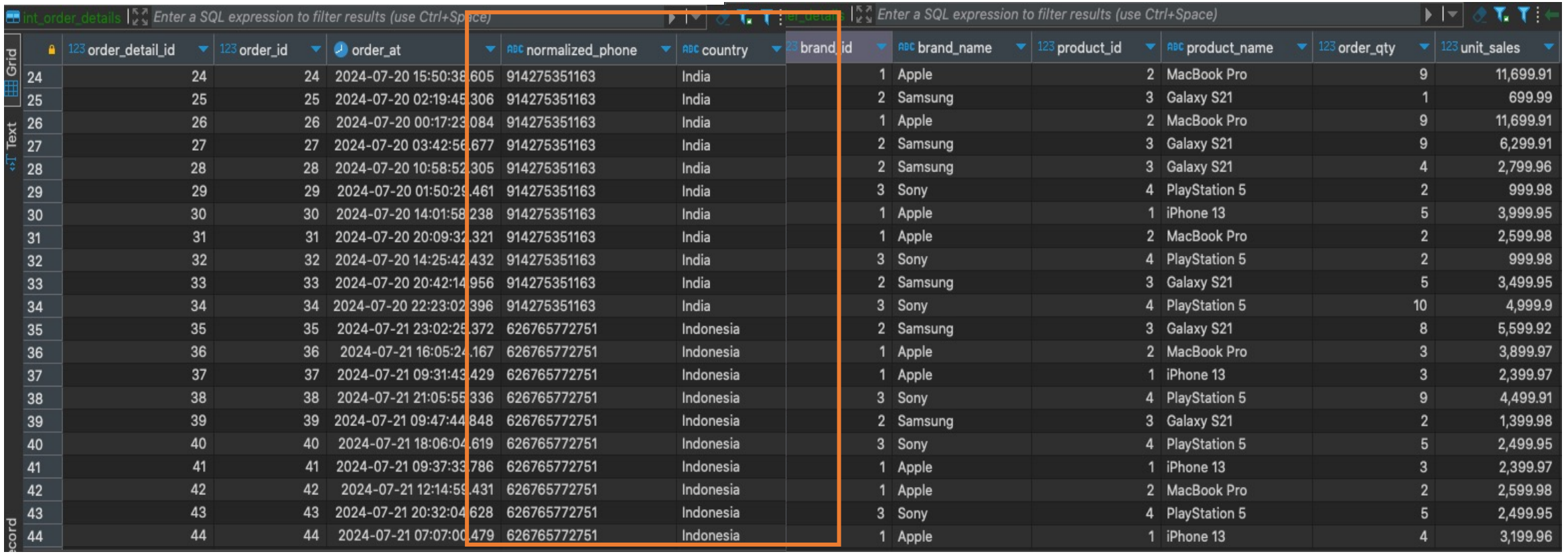
my_project > models > store > _int > int_order_details.sql

```sql
SELECT
    details.order_detail_id,
    details.order_id,
    orders.order_date AS order_at,
    normalized_phone,
    CASE
        WHEN {{ normalize_phone_number('orders.customer_phone') }} LIKE '62%' THEN 'Indonesia'
        WHEN {{ normalize_phone_number('orders.customer_phone') }} LIKE '91%' THEN 'India'
        ELSE 'Unknown'
    END AS country,
    products.brand_id,
    products.brand_name,
    details.product_id,
    details.quantity AS order_qty,
    details.unit_sales
FROM {{ ref('stg_order_details') }} AS details
LEFT JOIN {{ ref('int_orders') }} AS orders
    ON details.order_id = orders.order_id
LEFT JOIN {{ ref('int_products') }} AS products
    ON details.product_id = products.product_id
```

Model/ Layer Intermadiate

Dibawah ini merupakan hasil dari models yang dibangun pada int_order_details.sql. Hasil menunjukkan bahwa table pada int_order_details sudah ada penambahan kolom **country** dan beberapa table yang sudah digabungkan.

Dengan mengimplementasikan query ini di layer intermediate, Anda menciptakan satu set data yang siap untuk analisis lebih lanjut dan pelaporan untuk digunakan di layer fact atau mart.



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

	order_detail_id	order_id	order_at	normalized_phone	country	brand_id	brand_name	product_id	product_name	order_qty	unit_sales
24	24	24	2024-07-20 15:50:38.605	914275351163	India	1	Apple	2	MacBook Pro	9	11,699.91
25	25	25	2024-07-20 02:19:45.306	914275351163	India	2	Samsung	3	Galaxy S21	1	699.99
26	26	26	2024-07-20 00:17:23.084	914275351163	India	1	Apple	2	MacBook Pro	9	11,699.91
27	27	27	2024-07-20 03:42:56.677	914275351163	India	2	Samsung	3	Galaxy S21	9	6,299.91
28	28	28	2024-07-20 10:58:52.305	914275351163	India	2	Samsung	3	Galaxy S21	4	2,799.96
29	29	29	2024-07-20 01:50:29.461	914275351163	India	3	Sony	4	PlayStation 5	2	999.98
30	30	30	2024-07-20 14:01:58.238	914275351163	India	1	Apple	1	iPhone 13	5	3,999.95
31	31	31	2024-07-20 20:09:32.321	914275351163	India	1	Apple	2	MacBook Pro	2	2,599.98
32	32	32	2024-07-20 14:25:42.432	914275351163	India	3	Sony	4	PlayStation 5	2	999.98
33	33	33	2024-07-20 20:42:14.956	914275351163	India	2	Samsung	3	Galaxy S21	5	3,499.95
34	34	34	2024-07-20 22:23:02.396	914275351163	India	3	Sony	4	PlayStation 5	10	4,999.9
35	35	35	2024-07-21 23:02:25.372	626765772751	Indonesia	2	Samsung	3	Galaxy S21	8	5,599.92
36	36	36	2024-07-21 16:05:24.167	626765772751	Indonesia	1	Apple	2	MacBook Pro	3	3,899.97
37	37	37	2024-07-21 09:31:43.429	626765772751	Indonesia	1	Apple	1	iPhone 13	3	2,399.97
38	38	38	2024-07-21 21:05:55.336	626765772751	Indonesia	3	Sony	4	PlayStation 5	9	4,499.91
39	39	39	2024-07-21 09:47:44.848	626765772751	Indonesia	2	Samsung	3	Galaxy S21	2	1,399.98
40	40	40	2024-07-21 18:06:04.619	626765772751	Indonesia	3	Sony	4	PlayStation 5	5	2,499.95
41	41	41	2024-07-21 09:37:33.786	626765772751	Indonesia	1	Apple	1	iPhone 13	3	2,399.97
42	42	42	2024-07-21 12:14:59.431	626765772751	Indonesia	1	Apple	2	MacBook Pro	2	2,599.98
43	43	43	2024-07-21 20:32:04.628	626765772751	Indonesia	3	Sony	4	PlayStation 5	5	2,499.95
44	44	44	2024-07-21 07:07:00.479	626765772751	Indonesia	1	Apple	1	iPhone 13	4	3,199.96

Model/ Layer Intermadiate

Berikut merupakan line graph yang terbentuk setelah running models pada layer Intermedite. Hasil ini adalah sesuai dengan yang sudah kita inginkan melaui konfigurasi model yang kita bangun pada layer intermediate.

Dimana dalam hal ini, **int_product** memberikan data tambahan yang diperoleh layer staging dari model **stg_brands** dan **stg_products**. Layer ini juga menghasilkan data dengan informasi yang lebih lengkap pada **int_oder_details**. Untuk di proses layer fact dan mart untuk pelaporan yang lebih lanjut.

Model/ Layer Fact

my_project > models > store > _fct > fct_sales_daily.sql

```sql
SELECT
    order_id,
    order_at,
    country,
    brand_id,
    brand_name,
    product_name,
    order_qty,
    unit_sales AS daily_unit_sales
FROM {{ ref('int_order_details') }}
```

Dimana sebenarnya data yang lengkap sudah disediakan di **fct_order_details**.

Namun dalam hal ini jika ada sub fungsi bisnis yang membutuhkan laporan harian dan praktik bisa menggunakan model **fct_sales_daily**
Secara detail model tersebut bertujuan untuk :
Data Konsolidasi: Mengambil data dari layer intermediate yang sudah digabungkan dan distandarisasi.
Data Preparation: Menyiapkan data yang siap untuk analisis lebih lanjut atau pelaporan.
Menyediakan Informasi yang Relevan: Kolom yang dipilih menyediakan informasi penting tentang pesanan, produk, brand, kuantitas, dan penjualan harian per unit.

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

	_id	order_at	country	brand_id	brand_name	product_name	order_qty	daily_unit_sales
1	1	2024-07-20 17:16:32.186	India	2	Samsung	Galaxy S21	6	4,199.94
2	2	2024-07-20 03:03:59.634	India	1	Apple	MacBook Pro	5	6,499.95
3	3	2024-07-20 11:00:00.288	India	1	Apple	MacBook Pro	7	9,099.93
4	4	2024-07-20 20:26:56.370	India	2	Samsung	Galaxy S21	5	3,499.95
5	5	2024-07-20 03:42:40.761	India	2	Samsung	Galaxy S21	6	4,199.94
6	6	2024-07-20 14:23:16.581	India	2	Samsung	Galaxy S21	3	2,099.97
7	7	2024-07-20 18:06:59.066	India	1	Apple	MacBook Pro	8	10,399.92
8	8	2024-07-20 04:01:41.190	India	1	Apple	iPhone 13	4	3,199.96
9	9	2024-07-20 12:48:19.069	India	1	Apple	MacBook Pro	10	12,999.9
10	10	2024-07-20 22:13:02.010	India	3	Sony	PlayStation 5	9	4,499.91
11	11	2024-07-20 15:38:34.293	India	3	Sony	PlayStation 5	5	2,499.95
12	12	2024-07-20 08:29:27.669	India	1	Apple	MacBook Pro	3	3,899.97
13	13	2024-07-20 00:57:04.903	India	2	Samsung	Galaxy S21	4	2,799.96
14	14	2024-07-20 05:12:54.117	India	3	Sony	PlayStation 5	8	3,999.92
15	15	2024-07-20 12:08:34.680	India	3	Sony	PlayStation 5	3	1,499.97
16	16	2024-07-20 08:57:05.168	India	2	Samsung	Galaxy S21	4	2,799.96
17	17	2024-07-20 09:46:54.138	India	1	Apple	MacBook Pro	6	7,799.94
18	18	2024-07-20 17:26:08.092	India	3	Sony	PlayStation 5	4	1,999.96
19	19	2024-07-20 15:10:35.757	India	2	Samsung	Galaxy S21	6	4,199.94
20	20	2024-07-20 00:47:43.800	India	2	Samsung	Galaxy S21	8	5,599.92
21	21	2024-07-20 03:57:58.728	India	2	Samsung	Galaxy S21	8	5,599.92

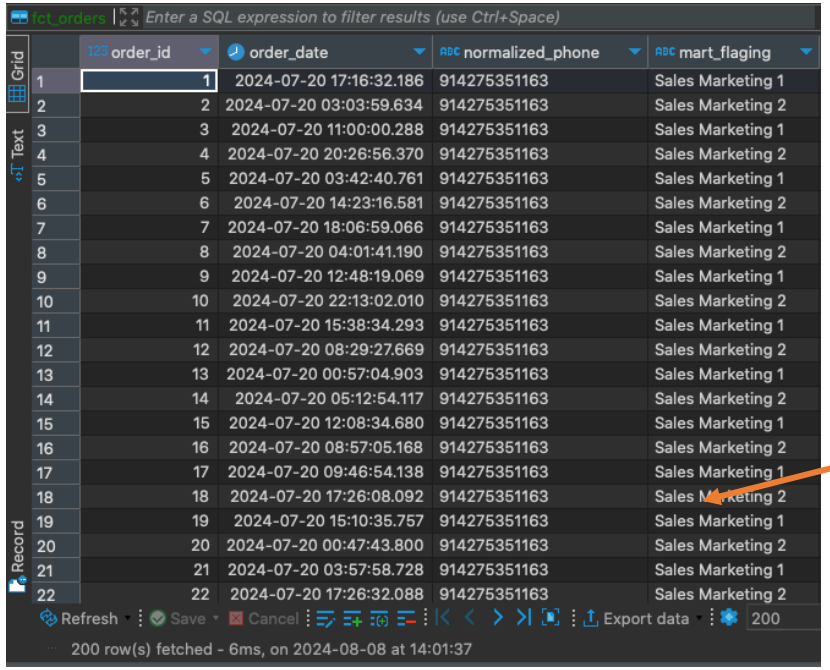
Overview fact_sales_daily

Model/ Layer Fact

my_project > models > store > _fct > fct_orders.sql

```sql
select
    order_id
    , order_date
    , normalized_phone
    , case
        when orders.order_id % 2 != 0 then 'Sales Marketing 1'
        else 'Sales Marketing 2'
    end as mart_flaging

from {{ ref('int_orders') }} as orders
```



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

	order_id	order_date	normalized_phone	mart_flaging
1	1	2024-07-20 17:16:32.186	914275351163	Sales Marketing 1
2	2	2024-07-20 03:03:59.634	914275351163	Sales Marketing 2
3	3	2024-07-20 11:00:00.288	914275351163	Sales Marketing 1
4	4	2024-07-20 20:26:56.370	914275351163	Sales Marketing 2
5	5	2024-07-20 03:42:40.761	914275351163	Sales Marketing 1
6	6	2024-07-20 14:23:16.581	914275351163	Sales Marketing 2
7	7	2024-07-20 18:06:59.066	914275351163	Sales Marketing 1
8	8	2024-07-20 04:01:41.190	914275351163	Sales Marketing 2
9	9	2024-07-20 12:48:19.069	914275351163	Sales Marketing 1
10	10	2024-07-20 22:13:02.010	914275351163	Sales Marketing 2
11	11	2024-07-20 15:38:34.293	914275351163	Sales Marketing 1
12	12	2024-07-20 08:29:27.669	914275351163	Sales Marketing 2
13	13	2024-07-20 00:57:04.903	914275351163	Sales Marketing 1
14	14	2024-07-20 05:12:54.117	914275351163	Sales Marketing 2
15	15	2024-07-20 12:08:34.680	914275351163	Sales Marketing 1
16	16	2024-07-20 08:57:05.168	914275351163	Sales Marketing 2
17	17	2024-07-20 09:46:54.138	914275351163	Sales Marketing 1
18	18	2024-07-20 17:26:08.092	914275351163	Sales Marketing 2
19	19	2024-07-20 15:10:35.757	914275351163	Sales Marketing 1
20	20	2024-07-20 00:47:43.800	914275351163	Sales Marketing 2
21	21	2024-07-20 03:57:58.728	914275351163	Sales Marketing 1
22	22	2024-07-20 17:26:32.088	914275351163	Sales Marketing 2

Refresh | Save | Cancel | Export data | 200

200 row(s) fetched - 6ms, on 2024-08-08 at 14:01:37

Seperti yang kita ketahui sebelumnya bahwa int_orders menyediakan data order date dan nomor telepon maka layer ini akan dimaksimalkan lagi antara lain :

•**Data Penandaan:** Menambahkan kolom penandaan mart_flaging untuk segmentasi data.
•**Segmentasi Data:** Mengelompokkan data pesanan ke dalam dua kategori berdasarkan penandaan, yang bisa digunakan untuk analisis atau strategi pemasaran dalam hal ini untuk '**Sales Marketing 1**' dan '**Sales Marketing 2**' untuk menunjang pekerjaan mereka.
•**Persiapan Data:** Menyiapkan data yang siap untuk analisis lebih lanjut atau pelaporan dengan menambahkan informasi penandaan.
Dengan query ini, bisa digunakan di layer mart, agar masing-masing role bisa melakukan jobdesknya secara maksimal (blasting nomor costumer)

Overview fact_orders

DBT Package(Test)

- Implementasi dbt package untuk menunjang project dengan membuat file **packages.yml** yang berisikan package – package yang akan digunakan dalam project

```
my_project > ! packages.yml
1   packages:
2     - package: dbt-labs/dbt_utils
3       version: 1.2.0
4     - package: calogica/dbt_expectations
5       version: 0.10.3
6
```

- **dbt-labs/dbt_utils**: Paket ini menyediakan berbagai utilitas dan fungsi tambahan untuk meningkatkan produktivitas dalam pengembangan model dbt. Contohnya, ia menyediakan fungsi untuk menangani string, tanggal, dan fungsi logika lainnya yang sering digunakan dalam SQL dbt. Fungsi-fungsi ini mempermudah penulisan kode yang lebih bersih dan lebih efisien.
- **calogica/dbt_expectations**: Paket ini dirancang untuk membantu melakukan validasi dan pengujian data dalam proyek dbt. Ia menyediakan berbagai ekspektasi untuk memastikan bahwa data yang diproses memenuhi kriteria tertentu, seperti rentang nilai atau keunikan.

- Gunakan perintah **dbt dps** untuk install packages

```
(.venv) wartadi@Wartadis-MacBook-Pro my_project % dbt deps
08:03:09  Running with dbt=1.7.0
08:03:10  Installing dbt-labs/dbt_utils
08:03:12  Installed from version 1.2.0
08:03:12  Up to date!
08:03:12  Installing calogica/dbt_expectations
08:03:20  Installed from version 0.10.3
08:03:20  Up to date!
08:03:20  Installing calogica/dbt_date
08:03:21  Installed from version 0.10.1
08:03:21  Up to date!
```

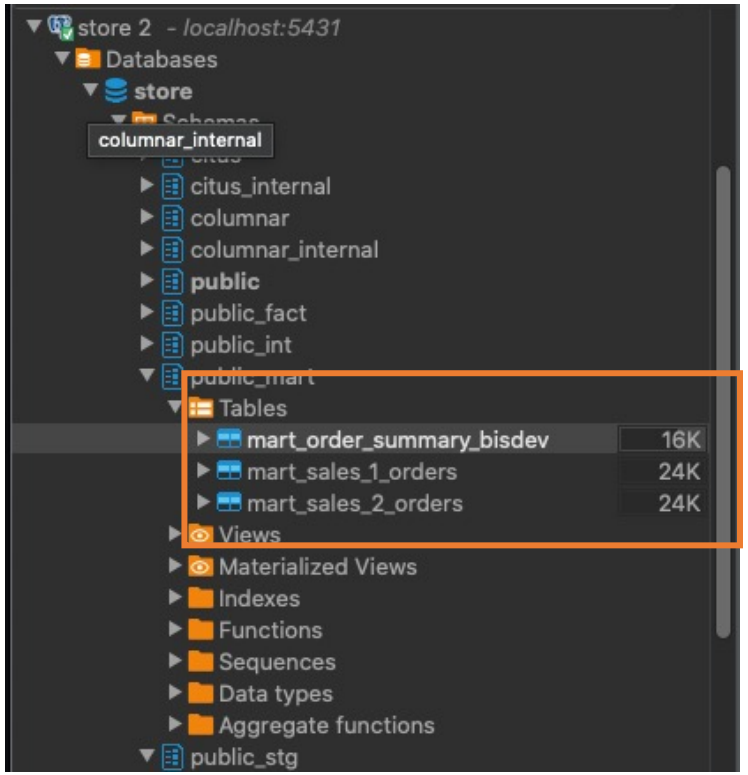
- Unutk mengetahui bahwa test kita berhasil atau tidak bisa dilakukan perintah **dbt test, dbt_expectations_expect_table_row_count_to_equal_other_table**
- Pengujian ini memverifikasi bahwa jumlah baris dalam tabel yang diuji sama dengan jumlah baris dalam tabel referensi int_order_details_ref_stg_order_details dan fct_orders_ref_int_order_details.
- **dbt_utils_unique_combination_of_columns**_fct_order_details_order_id__product_id. Pengujian ini memastikan bahwa kombinasi unik dari kolom order_id dan product_id dalam tabel fct_order_details tidak ada duplikat.

```
(.venv) wartadi@Wartadis-MacBook-Pro my_project % dbt test
10:50:06  Running with dbt=1.7.0
10:50:07  Registered adapter: postgres=1.7.0
10:50:08  Found 11 models, 40 tests, 4 sources, 0 exposures, 0 metrics, 797 macros, 0 groups, 0 semantic models
10:50:08
10:50:09  Concurrency: 1 threads (target='dev')
10:50:09
10:50:09  1 of 40 START test dbt_expectations_expect_table_row_count_to_equal_other_table_fct_orders_ref_int_order_details_  [RUN]
10:50:10  1 of 40 PASS dbt_expectations_expect_table_row_count_to_equal_other_table_fct_orders_ref_int_order_details_  [PASS in 0.46s]
10:50:10  2 of 40 START test dbt_expectations_expect_table_row_count_to_equal_other_table_int_order_details_ref_stg_order_details_  [RUN]
10:50:10  2 of 40 PASS dbt_expectations_expect_table_row_count_to_equal_other_table_int_order_details_ref_stg_order_details_  [PASS in 0.25s]
10:50:10  3 of 40 START test dbt_utils_unique_combination_of_columns_fct_order_details_order_id__product_id  [RUN]
10:50:10  3 of 40 PASS dbt_utils_unique_combination_of_columns_fct_order_details_order_id__product_id  [PASS in 0.23s]
```

Model/ Layer Mart

Berikut merupakan running DBT ketika sudah sampai pembuatan konfiguarasi models di layer Mart. Dimana total yang dihasilkan ada 13 Models

```
(.venv) wartadi@Wartadis-MacBook-Pro my_project % dbt run
08:28:16  Running with dbt=1.7.0
08:28:16  Registered adapter: postgres=1.7.0
08:28:17  Found 13 models, 40 tests, 4 sources, 0 exposures, 0 metrics, 799 macros, 0 groups, 0 semantic models
08:28:17
08:28:18  Concurrency: 1 threads (target='dev')
08:28:18
08:28:18  1 of 13 START sql view model public_stg.stg_brands ............................. [RUN]
08:28:19  1 of 13 OK created sql view model public_stg.stg_brands ........................ [CREATE VIEW in 0.63s]
08:28:19  2 of 13 START sql view model public_stg.stg_order_details ...................... [RUN]
08:28:19  2 of 13 OK created sql view model public_stg.stg_order_details ................. [CREATE VIEW in 0.19s]
08:28:19  3 of 13 START sql view model public_stg.stg_orders ............................. [RUN]
08:28:19  3 of 13 OK created sql view model public_stg.stg_orders ........................ [CREATE VIEW in 0.19s]
08:28:19  4 of 13 START sql view model public_stg.stg_products ........................... [RUN]
08:28:20  4 of 13 OK created sql view model public_stg.stg_products ...................... [CREATE VIEW in 0.24s]
08:28:20  5 of 13 START sql view model public_int.int_orders ............................. [RUN]
08:28:20  5 of 13 OK created sql view model public_int.int_orders ........................ [CREATE VIEW in 0.17s]
08:28:20  6 of 13 START sql view model public_int.int_products ........................... [RUN]
08:28:20  6 of 13 OK created sql view model public_int.int_products ...................... [CREATE VIEW in 0.18s]
08:28:20  7 of 13 START sql table model public_fact.fct_orders ........................... [RUN]
08:28:20  7 of 13 OK created sql table model public_fact.fct_orders ...................... [SELECT 419 in 0.35s]
08:28:20  8 of 13 START sql view model public_int.int_order_details ...................... [RUN]
08:28:21  8 of 13 OK created sql view model public_int.int_order_details ................. [CREATE VIEW in 0.18s]
08:28:21  9 of 13 START sql table model public_mart.mart_sales_1_orders .................. [RUN]
08:28:21  9 of 13 OK created sql table model public_mart.mart_sales_1_orders ............. [SELECT 210 in 0.26s]
08:28:21  10 of 13 START sql table model public_mart.mart_sales_2_orders ................. [RUN]
08:28:21  10 of 13 OK created sql table model public_mart.mart_sales_2_orders ............ [SELECT 209 in 0.32s]
08:28:21  11 of 13 START sql table model public_fact.fct_order_details ................... [RUN]
08:28:22  11 of 13 OK created sql table model public_fact.fct_order_details .............. [SELECT 419 in 0.39s]
08:28:22  12 of 13 START sql table model public_fact.fct_sales_daily ..................... [RUN]
08:28:22  12 of 13 OK created sql table model public_fact.fct_sales_daily ................ [SELECT 419 in 0.37s]
08:28:22  13 of 13 START sql table model public_mart.mart_order_summary_bisdev ........... [RUN]
08:28:22  13 of 13 OK created sql table model public_mart.mart_order_summary_bisdev ...... [SELECT 8 in 0.32s]
08:28:23
08:28:23  Finished running 7 view models, 6 table models in 0 hours 0 minutes and 5.51 seconds (5.51s).
08:28:23
08:28:23  Completed successfully
08:28:23
08:28:23  Done. PASS=13 WARN=0 ERROR=0 SKIP=0 TOTAL=13
```



Model/ Layer Mart

Seperti yang kita ketahui bahwa layer mart adalah tahap akhir dalam arsitektur data di mana data yang telah diproses di layer fact disiapkan untuk pelaporan dan analisis. Di layer ini, data biasanya disesuaikan untuk memenuhi kebutuhan spesifik bisnis dan analisis, membuatnya siap untuk digunakan oleh pengguna akhir dalam keputusan bisnis sehari-hari.

Dalam project ini models yang dihasilkan dari layer mart antara lain adalah **mart_sales_1_orders**, **mart_sales_2_order** dan **mart_order_summary_bisdev**

mart_sales_1_orders

my_project > models > store > _mart > mart_sales_1_orders.sql

```
1  select *
2  from {{ ref('fct_orders') }}
3  where mart_flaging = 'Sales Marketing 1'
```

mart_sales_1_orders | Enter a SQL expression to filter results (use Ctrl+Space)

	order_id	order_date	normalized_phone	mart_flaging
1	1	2024-07-20 17:16:32.186	914275351163	Sales Marketing 1
2	3	2024-07-20 11:00:00.288	914275351163	Sales Marketing 1
3	5	2024-07-20 03:42:40.761	914275351163	Sales Marketing 1
4	7	2024-07-20 18:06:59.066	914275351163	Sales Marketing 1
5	9	2024-07-20 12:48:19.069	914275351163	Sales Marketing 1
6	11	2024-07-20 15:38:34.293	914275351163	Sales Marketing 1

Bisa kita lihat bahwa data yang dihasilkan sudah tersegmentasi bahwa yang nomor telepon konsumen(normalized_phone) yang dengan id ganjil untuk **Sales Marketing 1**.

data ini bertujuan untuk menunjang pekerjaan seperti blasting dan constumer care **untuk Sales Marekting 1** agar bisa efektif kerjanya dan tidak terjadi duplikasi costumer yang ditangani oleh sales lain

Model/ Layer Mart

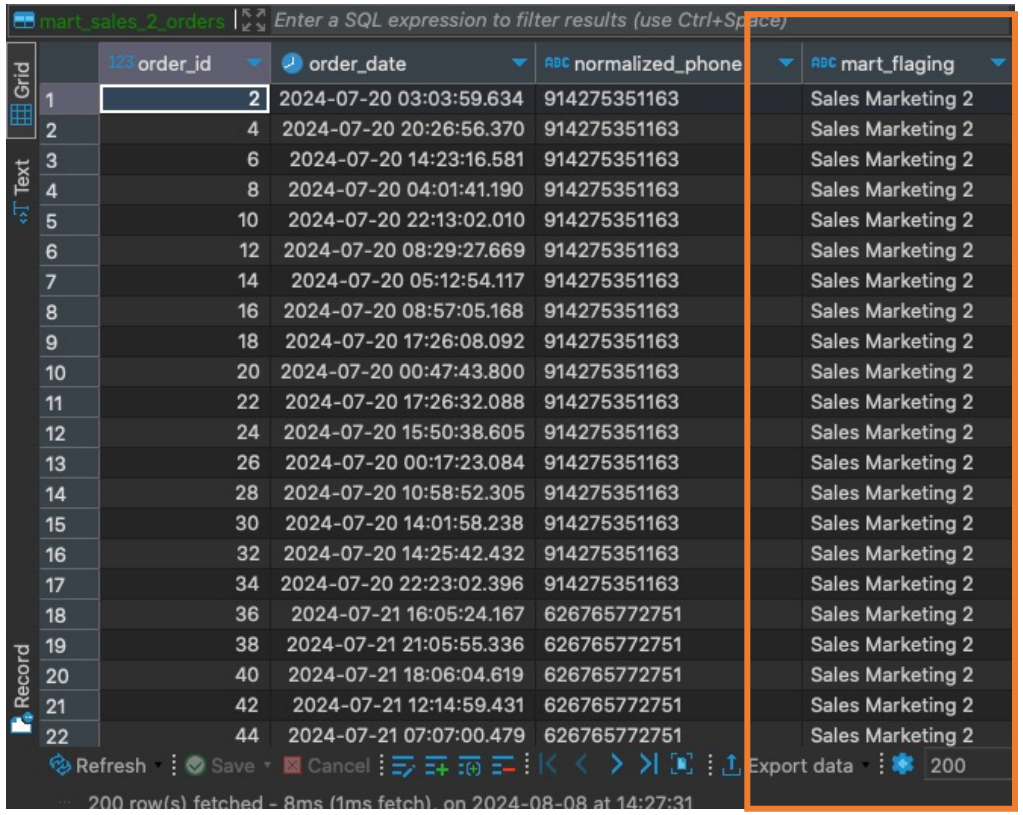
mart_sales_2_orders

my_project > models > store > _mart > mart_sales_2_orders.sql

```
1  select *
2  from {{ ref('fct_orders') }}
3  where mart_flaging = 'Sales Marketing 2'
```

Bisa kita lihat bahwa data yang dihasilkan sudah tersegmentasi bahwa yang nomor telepon konsumen (**normalized_phone**) yang dengan id genapuntuk **Sales Marketing 2**.

data ini bertujuan untuk menunjang pekerjaan **Sales Marekting 2** seperti blasting dan constumer care **untuk** agar bisa efektif kerjanya dan tidak terjadi duplikasi costumer yang ditangani oleh sales lain



Model/ Layer Mart

Dalam kasus ini models **mart_order_summary_ bisdev** sudah siap launching untuk menyediakan data total kuantitas, total unit sales, dan total jumlah penjualan untuk setiap bulan, brand, dan produk, yang berguna untuk laporan bulanan atau analisis performa produk (**Montly Report**).

Laporan bulanan ini sangat berguna untuk tim **Bisnis Development** sebagai acuan untuk menunjang performa dan strategi bisnis kedepannya

mart_order_summary_ bisdev

my_project > models > store > _mart > mart_order_summary_bisdev.sql

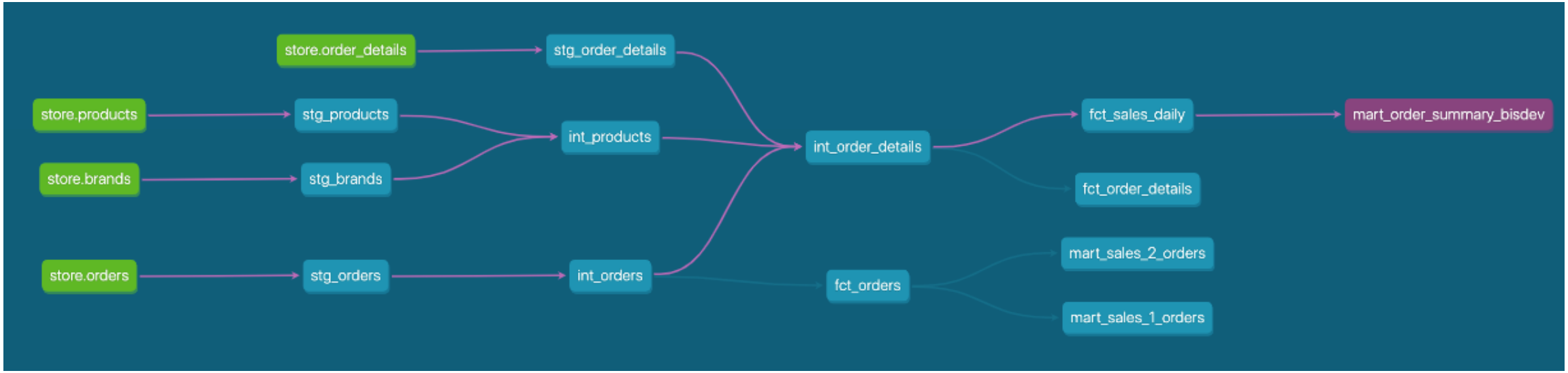
```sql
SELECT
    DATE_TRUNC('month', order_at) AS month,
    brand_id,
    brand_name,
    product_name,
    SUM(order_qty) AS total_order_qty,
    SUM(daily_unit_sales) AS total_unit_sales,
    SUM(daily_unit_sales * order_qty) AS total_amount
FROM {{ ref('fct_sales_daily') }}
GROUP BY 1, 2, 3, 4
ORDER BY 1, 2, 3
```

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

Grid | Text

	month	brand_id	brand_name	product_name	total_order_qty	total_unit_sales	total_amount
1	2024-07-01 00:00:00.000	1	Apple	iPhone 13	548	438,394.52	3,206,359.9199999995
2	2024-07-01 00:00:00.000	1	Apple	MacBook Pro	484	629,195.1600000001	4,583,764.739999999
3	2024-07-01 00:00:00.000	2	Samsung	Galaxy S21	514	359,794.8599999999	2,442,965.099999999
4	2024-07-01 00:00:00.000	3	Sony	PlayStation 5	431	215,495.69	1,439,471.2099999997
5	2024-08-01 00:00:00.000	1	Apple	iPhone 13	125	99,998.75	709,591.13
6	2024-08-01 00:00:00.000	1	Apple	MacBook Pro	142	184,598.58	1,362,389.5199999998
7	2024-08-01 00:00:00.000	2	Samsung	Galaxy S21	63	44,099.37	268,096.17
8	2024-08-01 00:00:00.000	3	Sony	PlayStation 5	106	52,998.94	399,992

Model/ Layer Mart – Lineagh Graph



Lineage graph ini menunjukkan bagaimana data mentah dari berbagai sumber diproses melalui beberapa tahap transformasi:

Staging Models: Membersihkan standarisasi dan mempersiapkan data mentah.
Intermediate: Models: Menggabungkan dan memperkaya data dari beberapa staging models.
Fact Models: Menyimpan data yang telah diringkas dan siap untuk analisis. (daily sales detail dan costumer telepon)
Mart Models: Menyediakan data siap pakai untuk analisis bisnis dan pelaporan.
Setiap langkah dalam alur ini memastikan bahwa data diproses secara bertahap, dari mentah hingga siap digunakan untuk analisis dan pengambilan keputusan. Ini memudahkan pemeliharaan, memastikan konsistensi data, dan meningkatkan efisiensi dalam pemrosesan data. (Data **Laporan Bulanan** dan Pembagian data untuk **Sales Marketing 1** - **Sales Marketing 2**)

alterra

THANK YOU ☺

dbt™