LAPORAN TUTORIAL LAB 3
BASIS DATA

ALDEN LUTHFI
2206028932
KELAS C

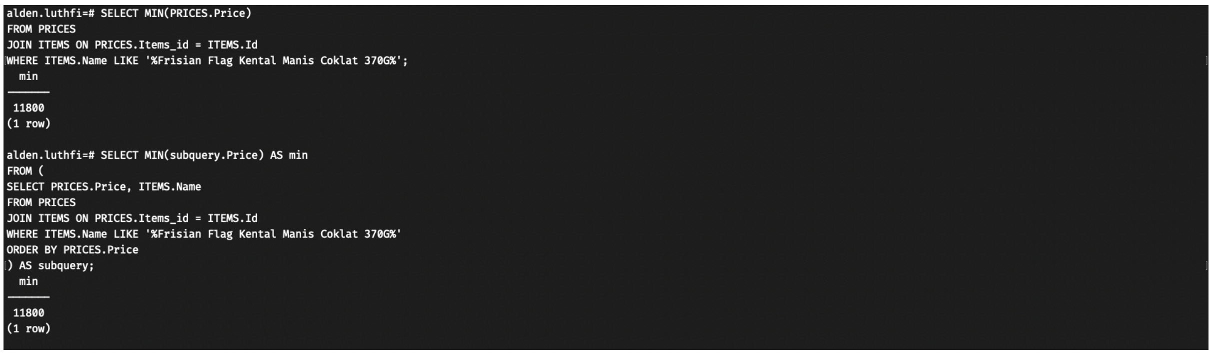
FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
DEPOK
2024

Laporan Tutorial Lab 3
Basis Data
CSGE602070
Semester Genap 2023/2024

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

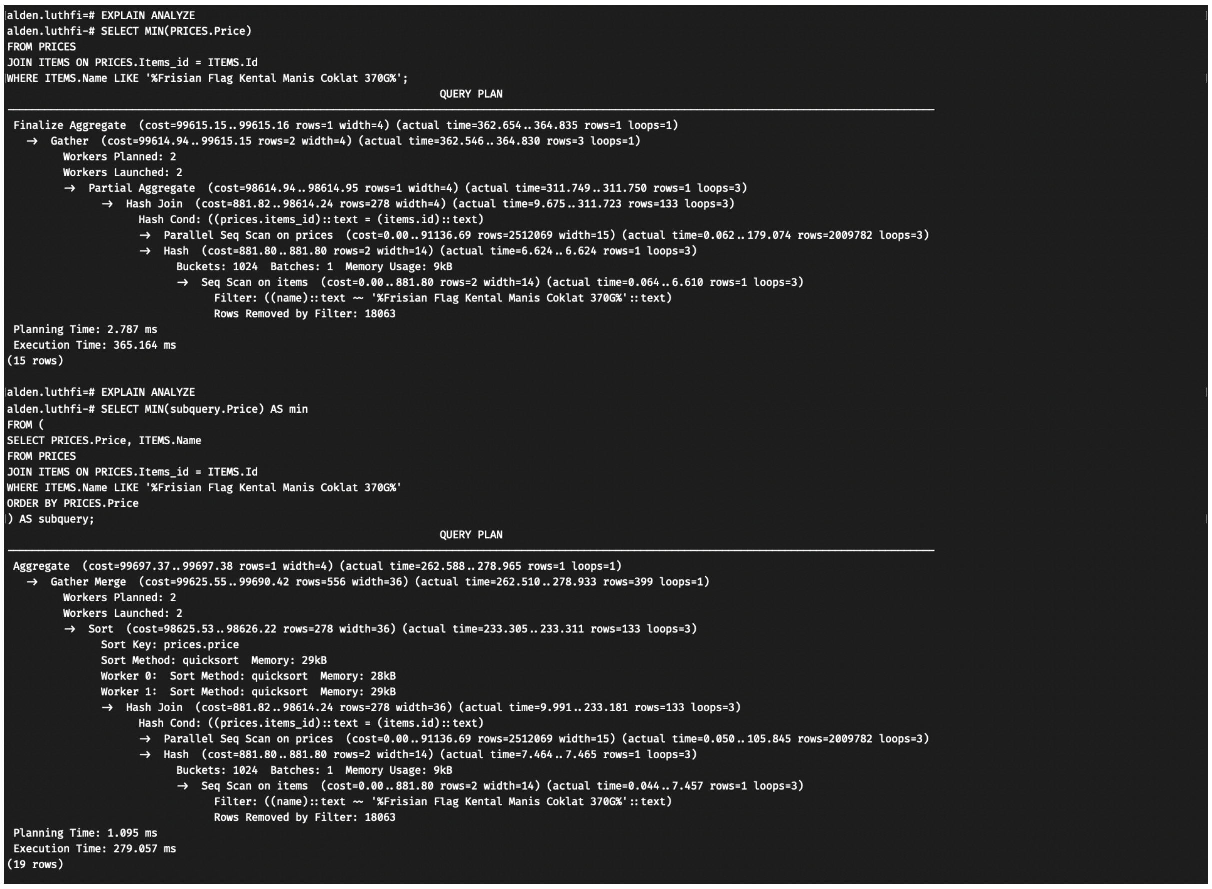
Latihan 1

1. **[SQL]** Jalankan kedua query di atas.



```
alden.luthfi=# SELECT MIN(PRICES.Price)
FROM PRICES
JOIN ITEMS ON PRICES.Items_id = ITEMS.Id
WHERE ITEMS.Name LIKE '%Frisian Flag Kental Manis Coklat 370G%';
  min
───────
 11800
(1 row)

alden.luthfi=# SELECT MIN(subquery.Price) AS min
FROM (
SELECT PRICES.Price, ITEMS.Name
FROM PRICES
JOIN ITEMS ON PRICES.Items_id = ITEMS.Id
WHERE ITEMS.Name LIKE '%Frisian Flag Kental Manis Coklat 370G%'
ORDER BY PRICES.Price
) AS subquery;
  min
───────
 11800
(1 row)
```

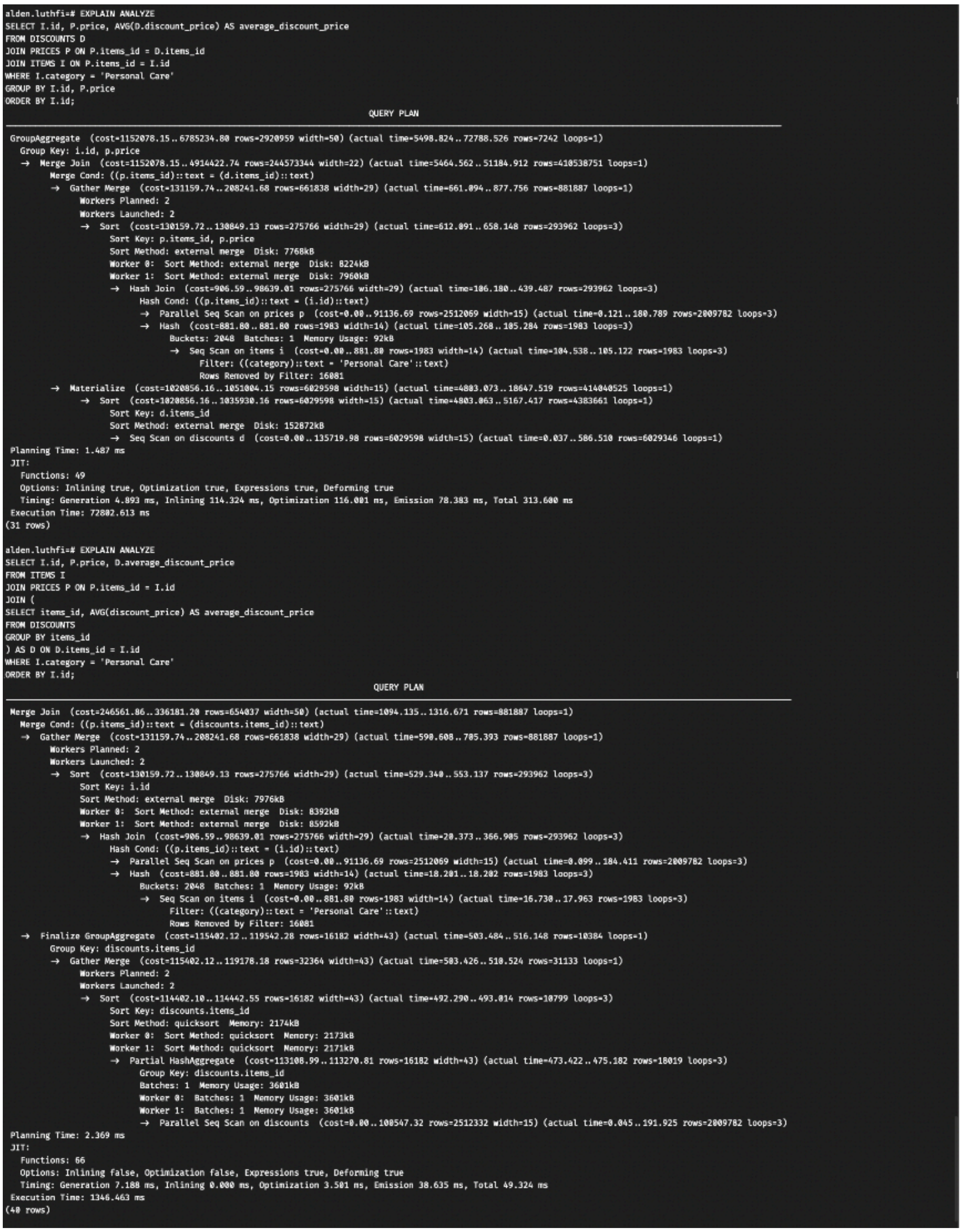
2. **[Trivia]** Tanpa menggunakan index, lakukan analisis terhadap kedua query tersebut.
   Hint: Anda dapat menggunakan perintah EXPLAIN ANALYZE



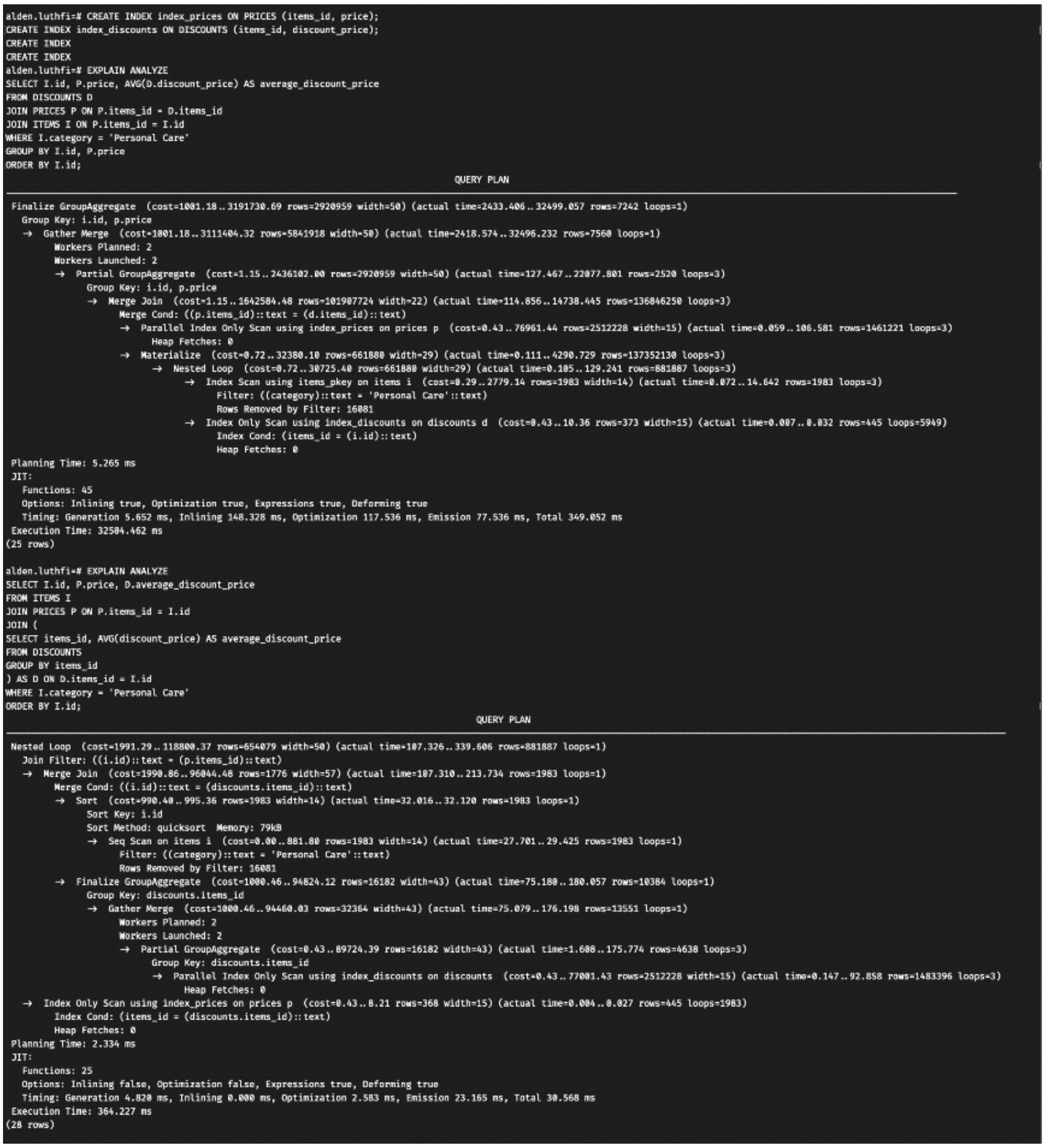
```
alden.luthfi=# EXPLAIN ANALYZE
alden.luthfi-# SELECT MIN(PRICES.Price)
FROM PRICES
JOIN ITEMS ON PRICES.Items_id = ITEMS.Id
WHERE ITEMS.Name LIKE '%Frisian Flag Kental Manis Coklat 370G%';
                                                                    QUERY PLAN
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
 Finalize Aggregate  (cost=99615.15..99615.16 rows=1 width=4) (actual time=362.654..364.835 rows=1 loops=1)
   ->  Gather  (cost=99614.94..99615.15 rows=2 width=4) (actual time=362.546..364.830 rows=3 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Partial Aggregate  (cost=98614.94..98614.95 rows=1 width=4) (actual time=311.749..311.750 rows=1 loops=3)
               ->  Hash Join  (cost=881.82..98614.24 rows=278 width=4) (actual time=9.675..311.723 rows=133 loops=3)
                     Hash Cond: ((prices.items_id)::text = (items.id)::text)
                     ->  Parallel Seq Scan on prices  (cost=0.00..91136.69 rows=2512069 width=15) (actual time=0.062..179.074 rows=2009782 loops=3)
                     ->  Hash  (cost=881.80..881.80 rows=2 width=14) (actual time=6.624..6.624 rows=1 loops=3)
                           Buckets: 1024  Batches: 1  Memory Usage: 9kB
                           ->  Seq Scan on items  (cost=0.00..881.80 rows=2 width=14) (actual time=0.064..6.610 rows=1 loops=3)
                                 Filter: ((name)::text ~~ '%Frisian Flag Kental Manis Coklat 370G%'::text)
                                 Rows Removed by Filter: 18063
 Planning Time: 2.787 ms
 Execution Time: 365.164 ms
(15 rows)

alden.luthfi=# EXPLAIN ANALYZE
alden.luthfi-# SELECT MIN(subquery.Price) AS min
FROM (
SELECT PRICES.Price, ITEMS.Name
FROM PRICES
JOIN ITEMS ON PRICES.Items_id = ITEMS.Id
WHERE ITEMS.Name LIKE '%Frisian Flag Kental Manis Coklat 370G%'
ORDER BY PRICES.Price
) AS subquery;
                                                                    QUERY PLAN
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
 Aggregate  (cost=99697.37..99697.38 rows=1 width=4) (actual time=262.588..278.965 rows=1 loops=1)
   ->  Gather Merge  (cost=99625.55..99690.42 rows=556 width=36) (actual time=262.510..278.933 rows=399 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Sort  (cost=98625.53..98626.22 rows=278 width=36) (actual time=233.305..233.311 rows=133 loops=3)
               Sort Key: prices.price
               Sort Method: quicksort  Memory: 29kB
               Worker 0:  Sort Method: quicksort  Memory: 28kB
               Worker 1:  Sort Method: quicksort  Memory: 29kB
               ->  Hash Join  (cost=881.82..98614.24 rows=278 width=36) (actual time=9.991..233.181 rows=133 loops=3)
                     Hash Cond: ((prices.items_id)::text = (items.id)::text)
                     ->  Parallel Seq Scan on prices  (cost=0.00..91136.69 rows=2512069 width=15) (actual time=0.050..105.845 rows=2009782 loops=3)
                     ->  Hash  (cost=881.80..881.80 rows=2 width=14) (actual time=7.464..7.465 rows=1 loops=3)
                           Buckets: 1024  Batches: 1  Memory Usage: 9kB
                           ->  Seq Scan on items  (cost=0.00..881.80 rows=2 width=14) (actual time=0.044..7.457 rows=1 loops=3)
                                 Filter: ((name)::text ~~ '%Frisian Flag Kental Manis Coklat 370G%'::text)
                                 Rows Removed by Filter: 18063
 Planning Time: 1.095 ms
 Execution Time: 279.057 ms
(19 rows)
```

EXPLAIN ANALYZE menunjukkan perbedaan antara waktu perencanaan (planning time) dan waktu eksekusi (execution time) untuk dua query yang berbeda yang melakukan hal yang sama. Query B memerlukan execution time dan planning time yang sedikit lebih lama dari Query A karena Query B mengambil jalan yang lebih tidak efisien seperti membutuhkan subquery dan sorting dalam pemanggilannya.

3. **[SQL]** Jalankan kedua query di bawah ini dengan dan tanpa index



```
alden.luthfi=# EXPLAIN ANALYZE
SELECT I.id, P.price, AVG(D.discount_price) AS average_discount_price
FROM DISCOUNTS D
JOIN PRICES P ON P.items_id = D.items_id
JOIN ITEMS I ON P.items_id = I.id
WHERE I.category = 'Personal Care'
GROUP BY I.id, P.price
ORDER BY I.id;
                                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=1152078.15..6785234.80 rows=2920959 width=50) (actual time=5498.824..72788.526 rows=7242 loops=1)
   Group Key: i.id, p.price
   ->  Merge Join  (cost=1152078.15..4914422.74 rows=244573344 width=22) (actual time=5464.562..51184.912 rows=410538751 loops=1)
         Merge Cond: ((p.items_id)::text = (d.items_id)::text)
         ->  Gather Merge  (cost=131159.74..208241.68 rows=661838 width=29) (actual time=661.094..877.756 rows=881887 loops=1)
               Workers Planned: 2
               Workers Launched: 2
               ->  Sort  (cost=130159.72..130849.13 rows=275766 width=29) (actual time=612.891..658.148 rows=293962 loops=3)
                     Sort Key: p.items_id, p.price
                     Sort Method: external merge  Disk: 7768kB
                     Worker 0:  Sort Method: external merge  Disk: 8224kB
                     Worker 1:  Sort Method: external merge  Disk: 7960kB
                     ->  Hash Join  (cost=906.59..98639.01 rows=275766 width=29) (actual time=106.180..439.487 rows=293962 loops=3)
                           Hash Cond: ((p.items_id)::text = (i.id)::text)
                           ->  Parallel Seq Scan on prices p  (cost=0.00..91136.69 rows=2512069 width=15) (actual time=0.121..180.789 rows=2009782 loops=3)
                           ->  Hash  (cost=881.80..881.80 rows=1983 width=14) (actual time=105.268..105.284 rows=1983 loops=3)
                                 Buckets: 2048  Batches: 1  Memory Usage: 92kB
                                 ->  Seq Scan on items i  (cost=0.00..881.80 rows=1983 width=14) (actual time=104.538..105.122 rows=1983 loops=3)
                                       Filter: ((category)::text = 'Personal Care'::text)
                                       Rows Removed by Filter: 16081
         ->  Materialize  (cost=1020856.16..1051004.15 rows=6029598 width=15) (actual time=4803.073..18647.519 rows=414040525 loops=1)
               ->  Sort  (cost=1020856.16..1035930.16 rows=6029598 width=15) (actual time=4803.063..5167.417 rows=4383661 loops=1)
                     Sort Key: d.items_id
                     Sort Method: external merge  Disk: 152872kB
                     ->  Seq Scan on discounts d  (cost=0.00..135719.98 rows=6029598 width=15) (actual time=0.037..586.510 rows=6029346 loops=1)
 Planning Time: 1.487 ms
 JIT:
   Functions: 49
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 4.893 ms, Inlining 114.324 ms, Optimization 116.001 ms, Emission 78.383 ms, Total 313.600 ms
 Execution Time: 72802.613 ms
(31 rows)

alden.luthfi=# EXPLAIN ANALYZE
SELECT I.id, P.price, D.average_discount_price
FROM ITEMS I
JOIN PRICES P ON P.items_id = I.id
JOIN (
SELECT items_id, AVG(discount_price) AS average_discount_price
FROM DISCOUNTS
GROUP BY items_id
) AS D ON D.items_id = I.id
WHERE I.category = 'Personal Care'
ORDER BY I.id;
                                                                 QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------
 Merge Join  (cost=246561.86..336181.20 rows=654037 width=50) (actual time=1094.135..1316.671 rows=881887 loops=1)
   Merge Cond: ((p.items_id)::text = (discounts.items_id)::text)
   ->  Gather Merge  (cost=131159.74..208241.68 rows=661838 width=29) (actual time=590.608..785.393 rows=881887 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Sort  (cost=130159.72..130849.13 rows=275766 width=29) (actual time=529.340..553.137 rows=293962 loops=3)
               Sort Key: i.id
               Sort Method: external merge  Disk: 7976kB
               Worker 0:  Sort Method: external merge  Disk: 8392kB
               Worker 1:  Sort Method: external merge  Disk: 8592kB
               ->  Hash Join  (cost=906.59..98639.01 rows=275766 width=29) (actual time=20.373..366.905 rows=293962 loops=3)
                     Hash Cond: ((p.items_id)::text = (i.id)::text)
                     ->  Parallel Seq Scan on prices p  (cost=0.00..91136.69 rows=2512069 width=15) (actual time=0.099..184.411 rows=2009782 loops=3)
                     ->  Hash  (cost=881.80..881.80 rows=1983 width=14) (actual time=18.201..18.202 rows=1983 loops=3)
                           Buckets: 2048  Batches: 1  Memory Usage: 92kB
                           ->  Seq Scan on items i  (cost=0.00..881.80 rows=1983 width=14) (actual time=16.730..17.963 rows=1983 loops=3)
                                 Filter: ((category)::text = 'Personal Care'::text)
                                 Rows Removed by Filter: 16081
   ->  Finalize GroupAggregate  (cost=115402.12..119542.28 rows=16182 width=43) (actual time=503.484..516.148 rows=10384 loops=1)
         Group Key: discounts.items_id
         ->  Gather Merge  (cost=115402.12..119178.18 rows=32364 width=43) (actual time=503.426..510.524 rows=31133 loops=1)
               Workers Planned: 2
               Workers Launched: 2
               ->  Sort  (cost=114402.10..114442.55 rows=16182 width=43) (actual time=492.290..493.014 rows=10799 loops=3)
                     Sort Key: discounts.items_id
                     Sort Method: quicksort  Memory: 2174kB
                     Worker 0:  Sort Method: quicksort  Memory: 2173kB
                     Worker 1:  Sort Method: quicksort  Memory: 2171kB
                     ->  Partial HashAggregate  (cost=113108.99..113270.81 rows=16182 width=43) (actual time=473.422..475.182 rows=10019 loops=3)
                           Group Key: discounts.items_id
                           Batches: 1  Memory Usage: 3601kB
                           Worker 0:  Batches: 1  Memory Usage: 3601kB
                           Worker 1:  Batches: 1  Memory Usage: 3601kB
                           ->  Parallel Seq Scan on discounts  (cost=0.00..100547.32 rows=2512332 width=15) (actual time=0.045..191.925 rows=2009782 loops=3)
 Planning Time: 2.369 ms
 JIT:
   Functions: 66
   Options: Inlining false, Optimization false, Expressions true, Deforming true
   Timing: Generation 7.188 ms, Inlining 0.000 ms, Optimization 3.501 ms, Emission 38.635 ms, Total 49.324 ms
 Execution Time: 1346.463 ms
(40 rows)
```



```
alden.luthfi=# CREATE INDEX index_prices ON PRICES (items_id, price);
CREATE INDEX index_discounts ON DISCOUNTS (items_id, discount_price);
CREATE INDEX
CREATE INDEX
alden.luthfi=# EXPLAIN ANALYZE
SELECT I.id, P.price, AVG(D.discount_price) AS average_discount_price
FROM DISCOUNTS D
JOIN PRICES P ON P.items_id = D.items_id
JOIN ITEMS I ON P.items_id = I.id
WHERE I.category = 'Personal Care'
GROUP BY I.id, P.price
ORDER BY I.id;
                                                              QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------
 Finalize GroupAggregate  (cost=1001.18..3191730.69 rows=2920959 width=50) (actual time=2433.406..32499.057 rows=7242 loops=1)
   Group Key: i.id, p.price
   ->  Gather Merge  (cost=1001.18..3111404.32 rows=5841918 width=50) (actual time=2418.574..32496.232 rows=7560 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Partial GroupAggregate  (cost=1.15..2436102.00 rows=2920959 width=50) (actual time=127.467..22077.801 rows=2520 loops=3)
               Group Key: i.id, p.price
               ->  Merge Join  (cost=1.15..1642584.48 rows=101907724 width=22) (actual time=114.856..14738.445 rows=136846250 loops=3)
                     Merge Cond: ((p.items_id)::text = (d.items_id)::text)
                     ->  Parallel Index Only Scan using index_prices on prices p  (cost=0.43..76961.44 rows=2512228 width=15) (actual time=0.059..106.581 rows=1461221 loops=3)
                           Heap Fetches: 0
                     ->  Materialize  (cost=0.72..32380.10 rows=661880 width=29) (actual time=0.111..4290.729 rows=137352130 loops=3)
                           ->  Nested Loop  (cost=0.72..30725.40 rows=661880 width=29) (actual time=0.105..129.241 rows=881887 loops=3)
                                 ->  Index Scan using items_pkey on items i  (cost=0.29..2779.14 rows=1983 width=14) (actual time=0.072..14.642 rows=1983 loops=3)
                                       Filter: ((category)::text = 'Personal Care'::text)
                                       Rows Removed by Filter: 16081
                                 ->  Index Only Scan using index_discounts on discounts d  (cost=0.43..10.36 rows=373 width=15) (actual time=0.007..0.032 rows=445 loops=5949)
                                       Index Cond: (items_id = (i.id)::text)
                                       Heap Fetches: 0
 Planning Time: 5.265 ms
 JIT:
   Functions: 45
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 5.652 ms, Inlining 148.328 ms, Optimization 117.536 ms, Emission 77.536 ms, Total 349.052 ms
 Execution Time: 32504.462 ms
(25 rows)

alden.luthfi=# EXPLAIN ANALYZE
SELECT I.id, P.price, D.average_discount_price
FROM ITEMS I
JOIN PRICES P ON P.items_id = I.id
JOIN (
SELECT items_id, AVG(discount_price) AS average_discount_price
FROM DISCOUNTS
GROUP BY items_id
) AS D ON D.items_id = I.id
WHERE I.category = 'Personal Care'
ORDER BY I.id;
                                                              QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=1991.29..118800.37 rows=654079 width=50) (actual time=107.326..339.606 rows=881887 loops=1)
   Join Filter: ((i.id)::text = (p.items_id)::text)
   ->  Merge Join  (cost=1990.86..96044.48 rows=1776 width=57) (actual time=107.310..213.734 rows=1983 loops=1)
         Merge Cond: ((i.id)::text = (discounts.items_id)::text)
         ->  Sort  (cost=990.40..995.36 rows=1983 width=14) (actual time=32.016..32.120 rows=1983 loops=1)
               Sort Key: i.id
               Sort Method: quicksort  Memory: 79kB
               ->  Seq Scan on items i  (cost=0.00..881.80 rows=1983 width=14) (actual time=27.701..29.425 rows=1983 loops=1)
                     Filter: ((category)::text = 'Personal Care'::text)
                     Rows Removed by Filter: 16081
         ->  Finalize GroupAggregate  (cost=1000.46..94824.12 rows=16182 width=43) (actual time=75.180..180.057 rows=10384 loops=1)
               Group Key: discounts.items_id
               ->  Gather Merge  (cost=1000.46..94460.03 rows=32364 width=43) (actual time=75.079..176.198 rows=13551 loops=1)
                     Workers Planned: 2
                     Workers Launched: 2
                     ->  Partial GroupAggregate  (cost=0.43..89724.39 rows=16182 width=43) (actual time=1.608..175.774 rows=4638 loops=3)
                           Group Key: discounts.items_id
                           ->  Parallel Index Only Scan using index_discounts on discounts  (cost=0.43..77001.43 rows=2512228 width=15) (actual time=0.147..92.858 rows=1483396 loops=3)
                                 Heap Fetches: 0
   ->  Index Only Scan using index_prices on prices p  (cost=0.43..8.21 rows=368 width=15) (actual time=0.004..0.027 rows=445 loops=1983)
         Index Cond: (items_id = (discounts.items_id)::text)
         Heap Fetches: 0
 Planning Time: 2.334 ms
 JIT:
   Functions: 25
   Options: Inlining false, Optimization false, Expressions true, Deforming true
   Timing: Generation 4.820 ms, Inlining 0.000 ms, Optimization 2.583 ms, Emission 23.165 ms, Total 30.568 ms
 Execution Time: 364.227 ms
(28 rows)
```

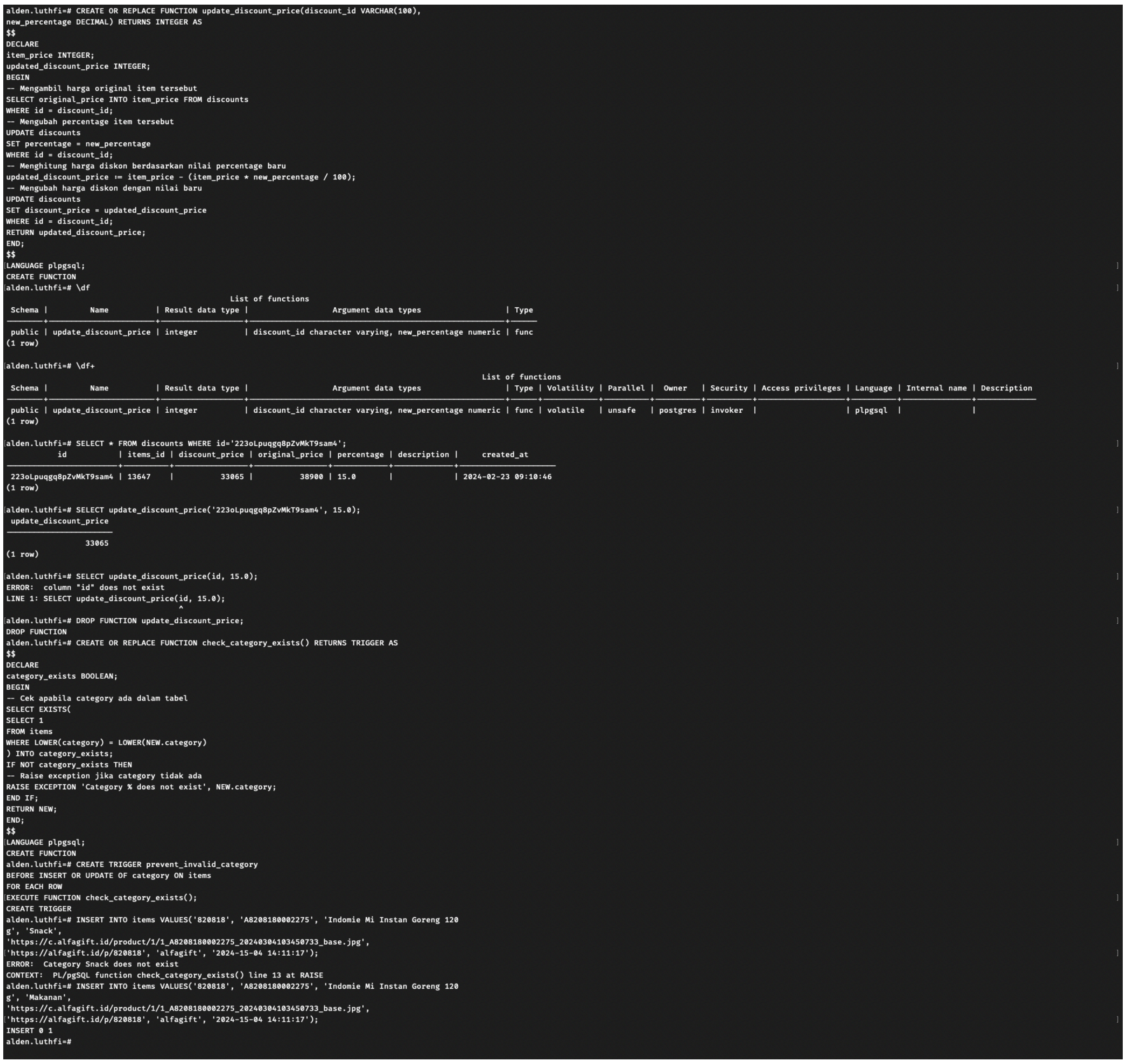
4. **[Trivia]** Analisis perbedaan execution time untuk setiap kondisi pada nomor 3, baik dengan maupun tanpa index. Apa saja faktor yang menyebabkan perbedaan signifikan dalam execution time antar setiap kondisi? Bagaimana pengaruh jumlah row data dan jumlah operasi JOIN pada performa query?

   Index mempercepat waktu eksekusi query karena dengan struktur btree query tidak perlu mengecek semua baris pada relasi. namun, planning time query menjadi lebih lama karena melibatkan pembuatan indeks tersebut. Perbandingan antara struktur

Query A dan Query B dalam hal waktu eksekusi menunjukkan bahwa meskipun Query B didesain untuk lebih efisien dengan mengurangi jumlah baris yang diolah pada join kedua, hasilnya malah menunjukkan kinerja yang lebih lambat dibandingkan Query A. Hal ini kemungkinan disebabkan oleh overhead dari penggunaan subquery dalam Query B yang tidak efisien dalam penggunaan indeks atau memerlukan penyimpanan hasil sementara yang meningkatkan penggunaan memori dan waktu pemrosesan. Selain itu, distribusi data yang tidak merata dan kurangnya efisiensi dalam penggunaan cache dan memori juga dapat memperlambat Query B.

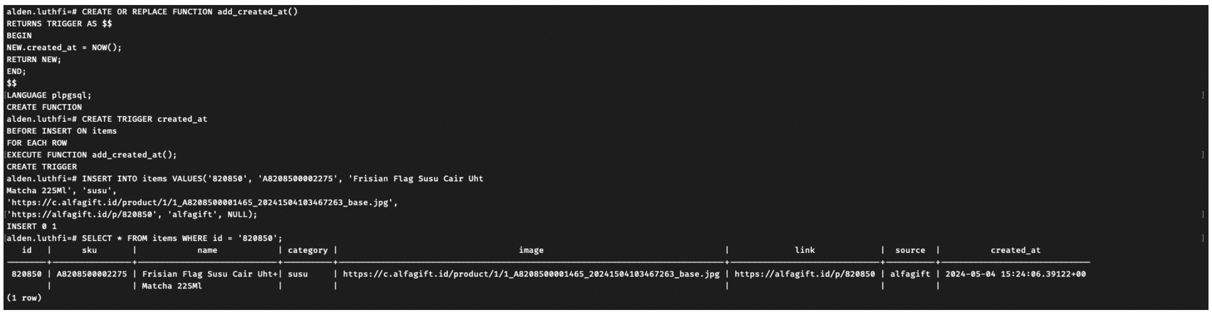
Latihan 2

1. **[SQL]** Jalankan seluruh contoh diatas.



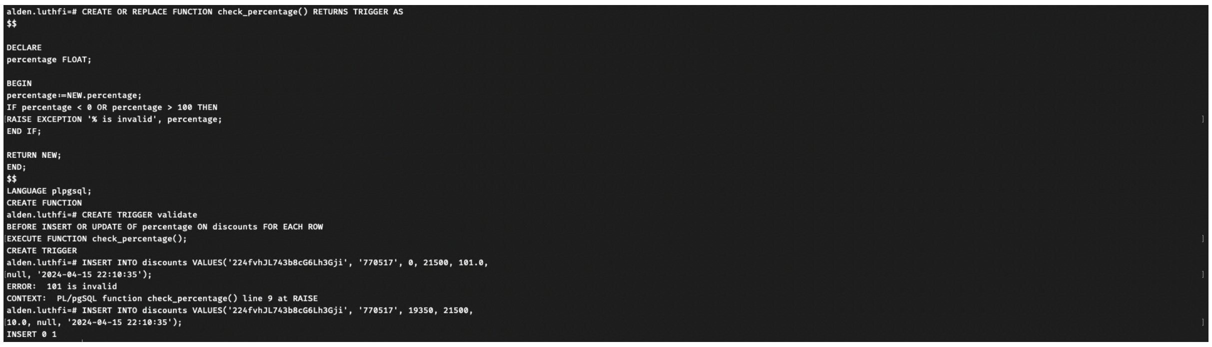
```
alden.luthfi=# CREATE OR REPLACE FUNCTION update_discount_price(discount_id VARCHAR(100),
new_percentage DECIMAL) RETURNS INTEGER AS
$$
DECLARE
item_price INTEGER;
updated_discount_price INTEGER;
BEGIN
-- Mengambil harga original item tersebut
SELECT original_price INTO item_price FROM discounts
WHERE id = discount_id;
-- Mengubah percentage item tersebut
UPDATE discounts
SET percentage = new_percentage
WHERE id = discount_id;
-- Menghitung harga diskon berdasarkan nilai percentage baru
updated_discount_price := item_price - (item_price * new_percentage / 100);
-- Mengubah harga diskon dengan nilai baru
UPDATE discounts
SET discount_price = updated_discount_price
WHERE id = discount_id;
RETURN updated_discount_price;
END;
$$
LANGUAGE plpgsql;
CREATE FUNCTION
alden.luthfi=# \df
                                        List of functions
 Schema |         Name          | Result data type |                  Argument data types                   | Type
--------+-----------------------+------------------+--------------------------------------------------------+------
 public | update_discount_price | integer          | discount_id character varying, new_percentage numeric | func
(1 row)

alden.luthfi=# \df+
                                                                                                  List of functions
 Schema |         Name          | Result data type |                  Argument data types                   | Type | Volatility | Parallel |  Owner   | Security | Access privileges | Language | Internal name | Description
--------+-----------------------+------------------+--------------------------------------------------------+------+------------+----------+----------+----------+-------------------+----------+---------------+-------------
 public | update_discount_price | integer          | discount_id character varying, new_percentage numeric | func | volatile   | unsafe   | postgres | invoker  |                   | plpgsql  |               |
(1 row)

alden.luthfi=# SELECT * FROM discounts WHERE id='223oLpuqgq8pZvMkT9sam4';
          id           | items_id | discount_price | original_price | percentage | description |     created_at
-----------------------+----------+----------------+----------------+------------+-------------+---------------------
 223oLpuqgq8pZvMkT9sam4 | 13647    |          33065 |          38900 | 15.0       |             | 2024-02-23 09:10:46
(1 row)

alden.luthfi=# SELECT update_discount_price('223oLpuqgq8pZvMkT9sam4', 15.0);
 update_discount_price
-----------------------
                 33065
(1 row)

alden.luthfi=# SELECT update_discount_price(id, 15.0);
ERROR:  column "id" does not exist
LINE 1: SELECT update_discount_price(id, 15.0);
                                     ^
alden.luthfi=# DROP FUNCTION update_discount_price;
DROP FUNCTION
alden.luthfi=# CREATE OR REPLACE FUNCTION check_category_exists() RETURNS TRIGGER AS
$$
DECLARE
category_exists BOOLEAN;
BEGIN
-- Cek apabila category ada dalam tabel
SELECT EXISTS(
SELECT 1
FROM items
WHERE LOWER(category) = LOWER(NEW.category)
) INTO category_exists;
IF NOT category_exists THEN
-- Raise exception jika category tidak ada
RAISE EXCEPTION 'Category % does not exist', NEW.category;
END IF;
RETURN NEW;
END;
$$
LANGUAGE plpgsql;
CREATE FUNCTION
alden.luthfi=# CREATE TRIGGER prevent_invalid_category
BEFORE INSERT OR UPDATE OF category ON items
FOR EACH ROW
EXECUTE FUNCTION check_category_exists();
CREATE TRIGGER
alden.luthfi=# INSERT INTO items VALUES('820818', 'A8208180002275', 'Indomie Mi Instan Goreng 120
g', 'Snack',
'https://c.alfagift.id/product/1/1_A8208180002275_20240304103450733_base.jpg',
'https://alfagift.id/p/820818', 'alfagift', '2024-15-04 14:11:17');
ERROR:  Category Snack does not exist
CONTEXT:  PL/pgSQL function check_category_exists() line 13 at RAISE
alden.luthfi=# INSERT INTO items VALUES('820818', 'A8208180002275', 'Indomie Mi Instan Goreng 120
g', 'Makanan',
'https://c.alfagift.id/product/1/1_A8208180002275_20240304103450733_base.jpg',
'https://alfagift.id/p/820818', 'alfagift', '2024-15-04 14:11:17');
INSERT 0 1
alden.luthfi=#
```

2. **[SQL]** Buat suatu trigger yang dapat menambahkan nilai created_at secara otomatis ketika menambahkan item di tabel items dan jalankan beberapa query berikut.



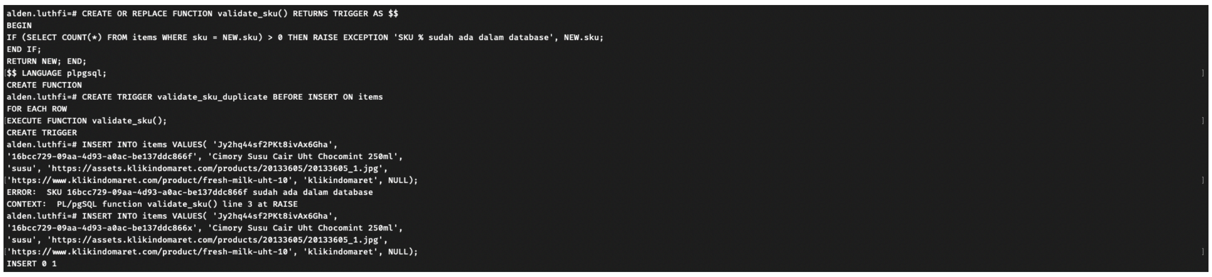
```
alden.luthfi=# CREATE OR REPLACE FUNCTION add_created_at()
RETURNS TRIGGER AS $$
BEGIN
NEW.created_at = NOW();
RETURN NEW;
END;
$$
LANGUAGE plpgsql;
CREATE FUNCTION
alden.luthfi=# CREATE TRIGGER created_at
BEFORE INSERT ON items
FOR EACH ROW
EXECUTE FUNCTION add_created_at();
CREATE TRIGGER
alden.luthfi=# INSERT INTO items VALUES('820850', 'A8208500002275', 'Frisian Flag Susu Cair Uht
Matcha 225Ml', 'susu',
'https://c.alfagift.id/product/1/1_A8208500001465_20241504103467263_base.jpg',
'https://alfagift.id/p/820850', 'alfagift', NULL);
INSERT 0 1
alden.luthfi=# SELECT * FROM items WHERE id = '820850';
   id   |      sku       |            name             | category |                                    image                                     |             link             |  source  |          created_at
--------+----------------+-----------------------------+----------+------------------------------------------------------------------------------+------------------------------+----------+-------------------------------
 820850 | A8208500002275 | Frisian Flag Susu Cair Uht+| susu     | https://c.alfagift.id/product/1/1_A8208500001465_20241504103467263_base.jpg | https://alfagift.id/p/820850 | alfagift | 2024-05-04 15:24:06.39122+00
        |                | Matcha 225Ml                |          |                                                                              |                              |          |
(1 row)
```

3. **[SQL]** Buat suatu trigger yang memicu suatu function yang dapat memvalidasi bahwa nilai percentage tidak boleh di luar rentang 0-100. Jika di luar rentang tersebut, maka raise exception. Trigger tersebut akan dijalankan ketika menambahkan item di tabel discounts dan jalankan beberapa query berikut.



```
alden.luthfi=# CREATE OR REPLACE FUNCTION check_percentage() RETURNS TRIGGER AS
$$

DECLARE
percentage FLOAT;

BEGIN
percentage:=NEW.percentage;
IF percentage < 0 OR percentage > 100 THEN
RAISE EXCEPTION '% is invalid', percentage;
END IF;

RETURN NEW;
END;
$$
LANGUAGE plpgsql;
CREATE FUNCTION
alden.luthfi=# CREATE TRIGGER validate
BEFORE INSERT OR UPDATE OF percentage ON discounts FOR EACH ROW
EXECUTE FUNCTION check_percentage();
CREATE TRIGGER
alden.luthfi=# INSERT INTO discounts VALUES('224fvhJL743b8cG6Lh3Gji', '770517', 0, 21500, 101.0,
null, '2024-04-15 22:10:35');
ERROR:  101 is invalid
CONTEXT:  PL/pgSQL function check_percentage() line 9 at RAISE
alden.luthfi=# INSERT INTO discounts VALUES('224fvhJL743b8cG6Lh3Gji', '770517', 19350, 21500,
10.0, null, '2024-04-15 22:10:35');
INSERT 0 1
```

4. **[SQL]** Buat suatu trigger yang memicu suatu function yang memvalidasi bahwa nilai SKU tidak boleh duplikat sehingga akan menampilkan pesan exception jika SKU item yang akan ditambahkan sudah ada pada tabel items dan jalankan beberapa query berikut.



```
alden.luthfi=# CREATE OR REPLACE FUNCTION validate_sku() RETURNS TRIGGER AS $$
BEGIN
IF (SELECT COUNT(*) FROM items WHERE sku = NEW.sku) > 0 THEN RAISE EXCEPTION 'SKU % sudah ada dalam database', NEW.sku;
END IF;
RETURN NEW; END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
alden.luthfi=# CREATE TRIGGER validate_sku_duplicate BEFORE INSERT ON items
FOR EACH ROW
EXECUTE FUNCTION validate_sku();
CREATE TRIGGER
alden.luthfi=# INSERT INTO items VALUES( 'Jy2hq44sf2PKt8ivAx6Gha',
'16bcc729-09aa-4d93-a0ac-be137ddc866f', 'Cimory Susu Cair Uht Chocomint 250ml',
'susu', 'https://assets.klikindomaret.com/products/20133605/20133605_1.jpg',
'https://www.klikindomaret.com/product/fresh-milk-uht-10', 'klikindomaret', NULL);
ERROR:  SKU 16bcc729-09aa-4d93-a0ac-be137ddc866f sudah ada dalam database
CONTEXT:  PL/pgSQL function validate_sku() line 3 at RAISE
alden.luthfi=# INSERT INTO items VALUES( 'Jy2hq44sf2PKt8ivAx6Gha',
'16bcc729-09aa-4d93-a0ac-be137ddc866x', 'Cimory Susu Cair Uht Chocomint 250ml',
'susu', 'https://assets.klikindomaret.com/products/20133605/20133605_1.jpg',
'https://www.klikindomaret.com/product/fresh-milk-uht-10', 'klikindomaret', NULL);
INSERT 0 1
```