Akilesh K

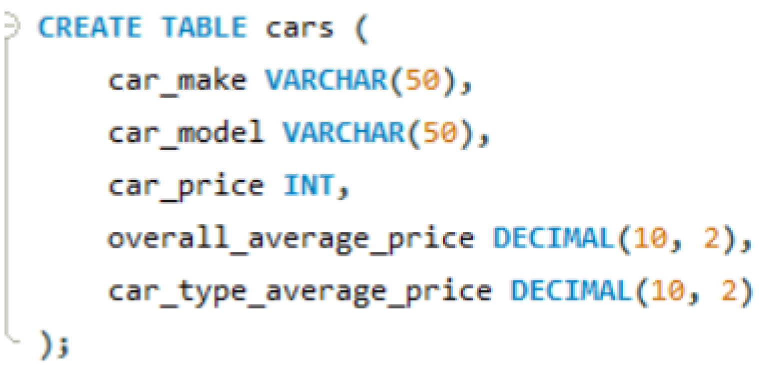
k.akilesh123@gmail.com

Data engineering - Batch 1

Date: 22-01-24

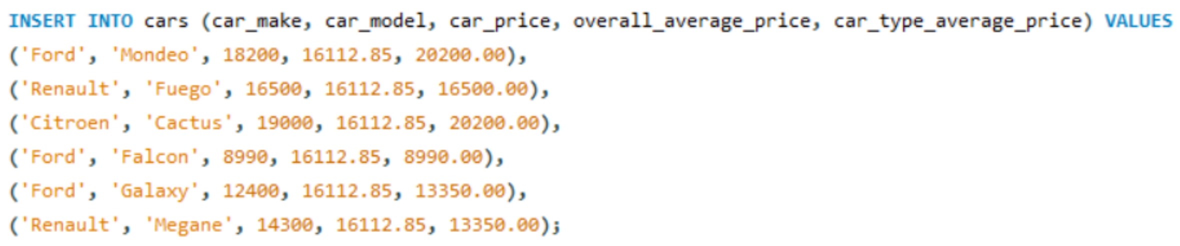
DAY 6-SQL

Total aggregations using sql queries,over and partition by clause in sql queries& total aggregation using over and partition by in sql queries,snowflaking& star schemas,rules and restrictions to group and filter data in sql queries,order of execution of sql queries,how to calculate subtotals in sql queries,regex,materialized view



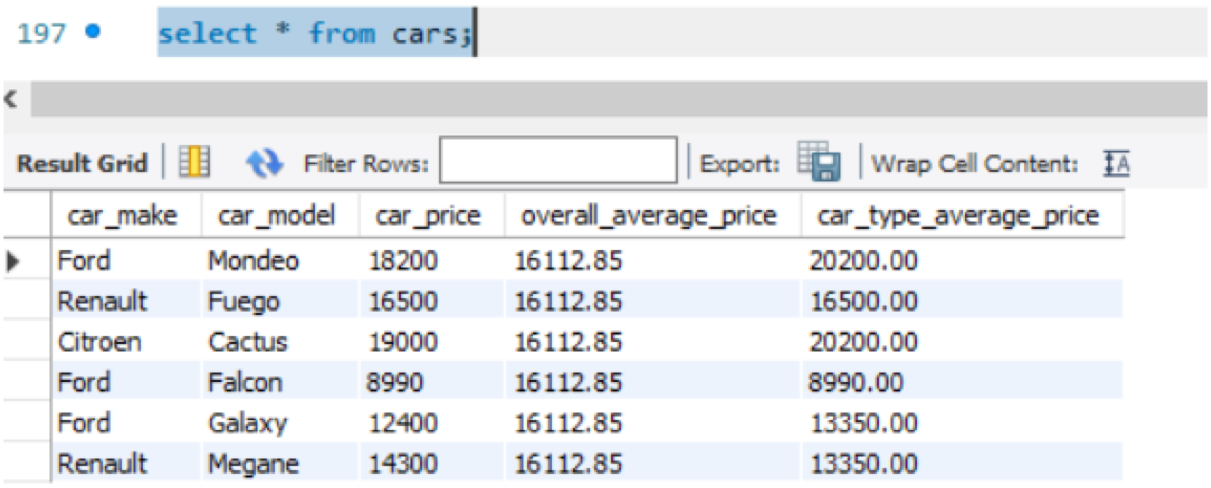
```
CREATE TABLE cars (
    car_make VARCHAR(50),
    car_model VARCHAR(50),
    car_price INT,
    overall_average_price DECIMAL(10, 2),
    car_type_average_price DECIMAL(10, 2)
);
```

Insert



```
INSERT INTO cars (car_make, car_model, car_price, overall_average_price, car_type_average_price) VALUES
('Ford', 'Mondeo', 18200, 16112.85, 20200.00),
('Renault', 'Fuego', 16500, 16112.85, 16500.00),
('Citroen', 'Cactus', 19000, 16112.85, 20200.00),
('Ford', 'Falcon', 8990, 16112.85, 8990.00),
('Ford', 'Galaxy', 12400, 16112.85, 13350.00),
('Renault', 'Megane', 14300, 16112.85, 13350.00);
```

Display cars

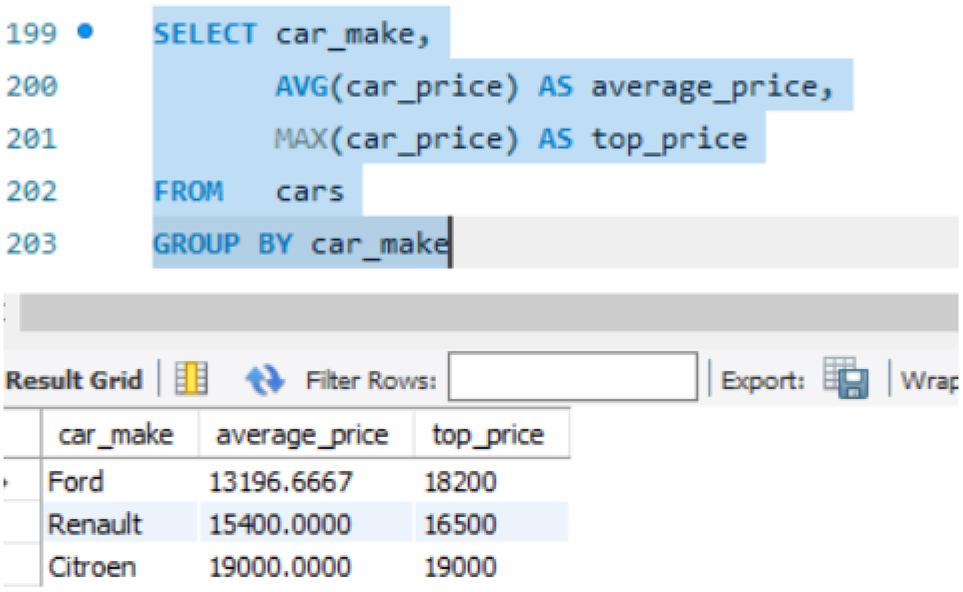


```
select * from cars;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

car_make	car_model	car_price	overall_average_price	car_type_average_price
Ford	Mondeo	18200	16112.85	20200.00
Renault	Fuego	16500	16112.85	16500.00
Citroen	Cactus	19000	16112.85	20200.00
Ford	Falcon	8990	16112.85	8990.00
Ford	Galaxy	12400	16112.85	13350.00
Renault	Megane	14300	16112.85	13350.00

Average and max

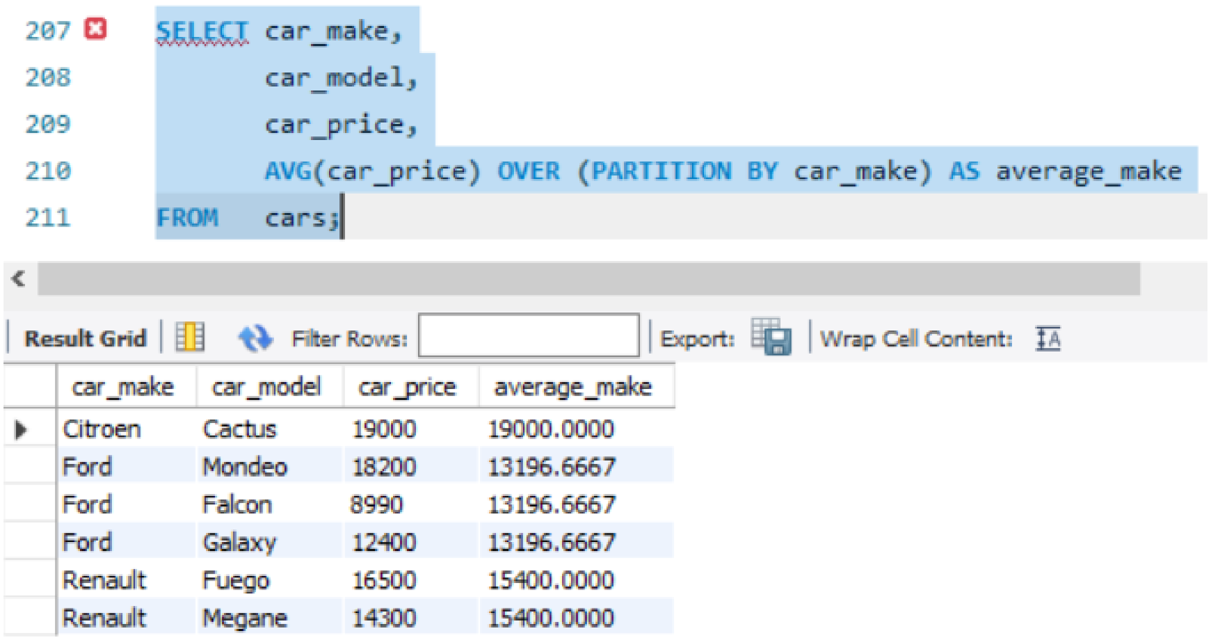


```
SELECT car_make,
       AVG(car_price) AS average_price,
       MAX(car_price) AS top_price
FROM   cars
GROUP BY car_make
```

Result Grid | Filter Rows: | Export: | Wrap

car_make	average_price	top_price
Ford	13196.6667	18200
Renault	15400.0000	16500
Citroen	19000.0000	19000

Using partition to implement average function

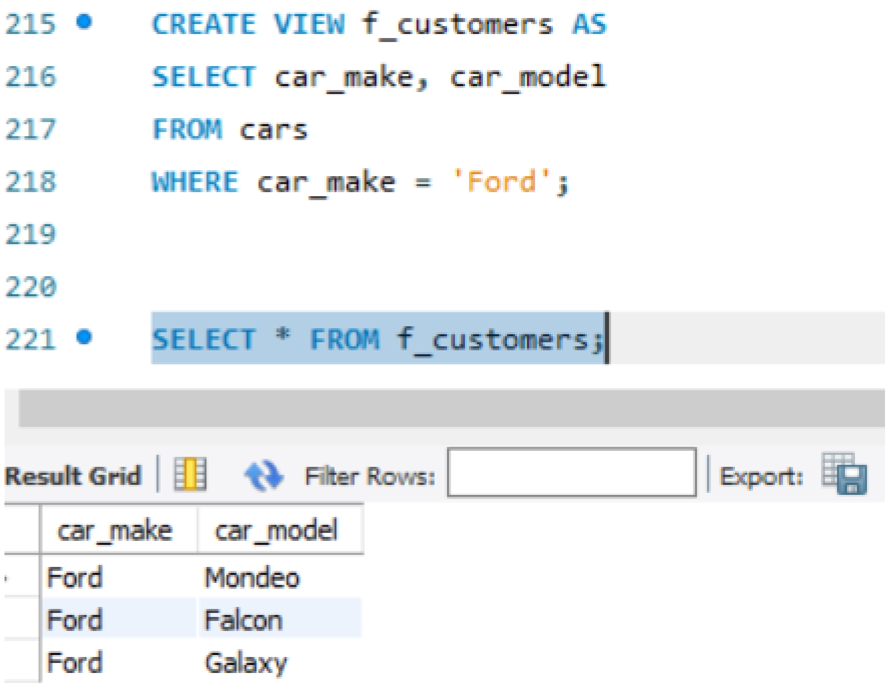


```sql
SELECT car_make,
       car_model,
       car_price,
       AVG(car_price) OVER (PARTITION BY car_make) AS average_make
FROM   cars;
```

car_make	car_model	car_price	average_make
Citroen	Cactus	19000	19000.0000
Ford	Mondeo	18200	13196.6667
Ford	Falcon	8990	13196.6667
Ford	Galaxy	12400	13196.6667
Renault	Fuego	16500	15400.0000
Renault	Megane	14300	15400.0000

VIEW

Creating a view table and displaying it

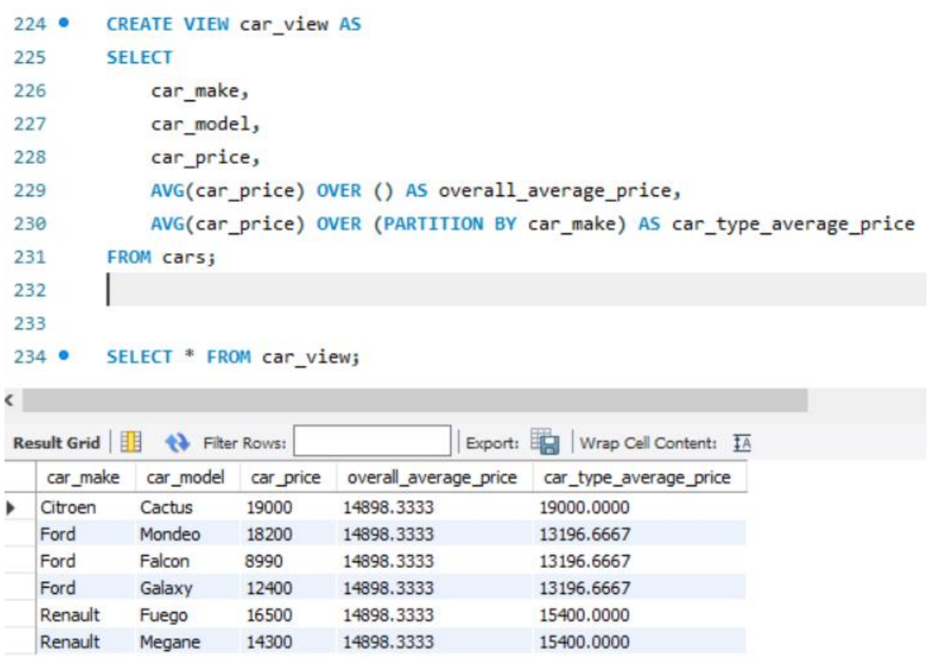


```sql
CREATE VIEW f_customers AS
SELECT car_make, car_model
FROM cars
WHERE car_make = 'Ford';


SELECT * FROM f_customers;
```

car_make	car_model
Ford	Mondeo
Ford	Falcon
Ford	Galaxy

View table using partirion



```sql
CREATE VIEW car_view AS
SELECT
    car_make,
    car_model,
    car_price,
    AVG(car_price) OVER () AS overall_average_price,
    AVG(car_price) OVER (PARTITION BY car_make) AS car_type_average_price
FROM cars;


SELECT * FROM car_view;
```

car_make	car_model	car_price	overall_average_price	car_type_average_price
Citroen	Cactus	19000	14898.3333	19000.0000
Ford	Mondeo	18200	14898.3333	13196.6667
Ford	Falcon	8990	14898.3333	13196.6667
Ford	Galaxy	12400	14898.3333	13196.6667
Renault	Fuego	16500	14898.3333	15400.0000
Renault	Megane	14300	14898.3333	15400.0000

Star Schema

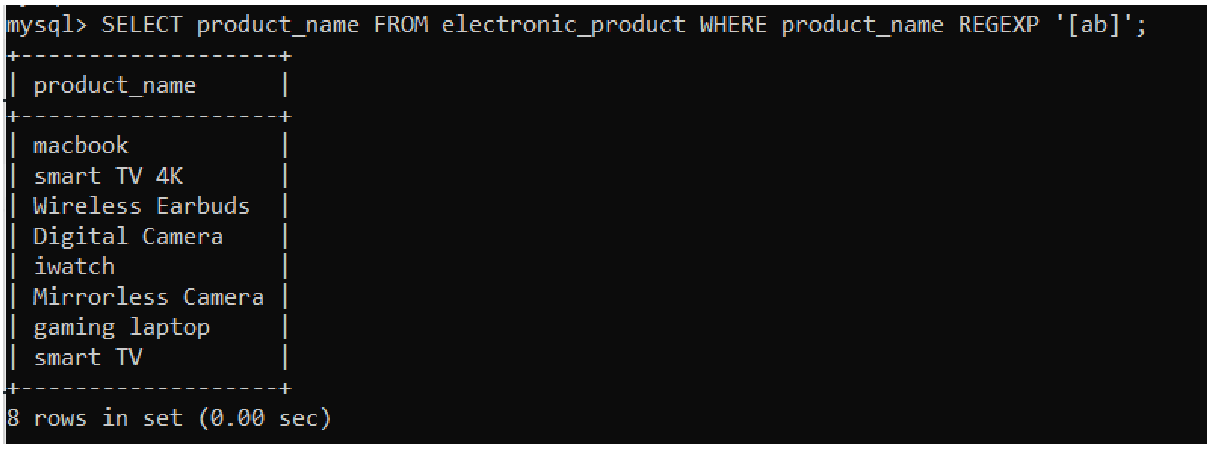
In a Star Schema, there is a central fact table surrounded by dimension tables. The central fact table is connected to dimension tables through foreign key relationships.

Snowflake Schema

In a Snowflake Schema, dimension tables are normalized into multiple related tables, forming a snowflake-like structure. The normalization reduces redundancy by breaking down dimension tables into sub-dimensions.

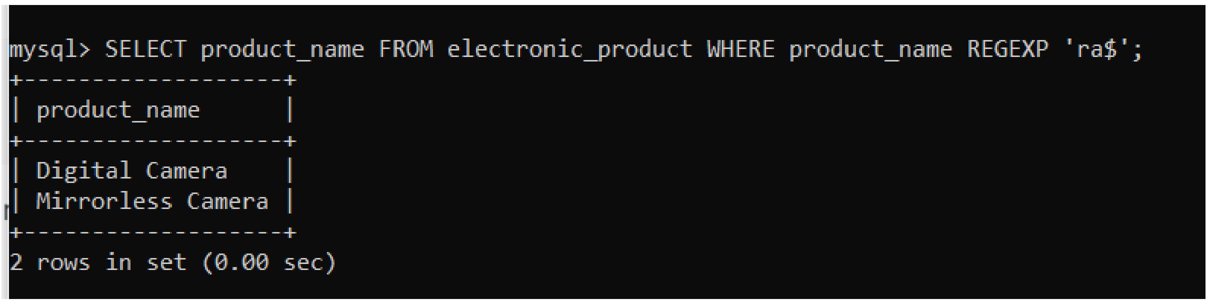
REGEX

Containing string ab



```
mysql> SELECT product_name FROM electronic_product WHERE product_name REGEXP '[ab]';
+--------------------+
| product_name       |
+--------------------+
| macbook            |
| smart TV 4K        |
| Wireless Earbuds   |
| Digital Camera     |
| iwatch             |
| Mirrorless Camera  |
| gaming laptop      |
| smart TV           |
+--------------------+
8 rows in set (0.00 sec)
```

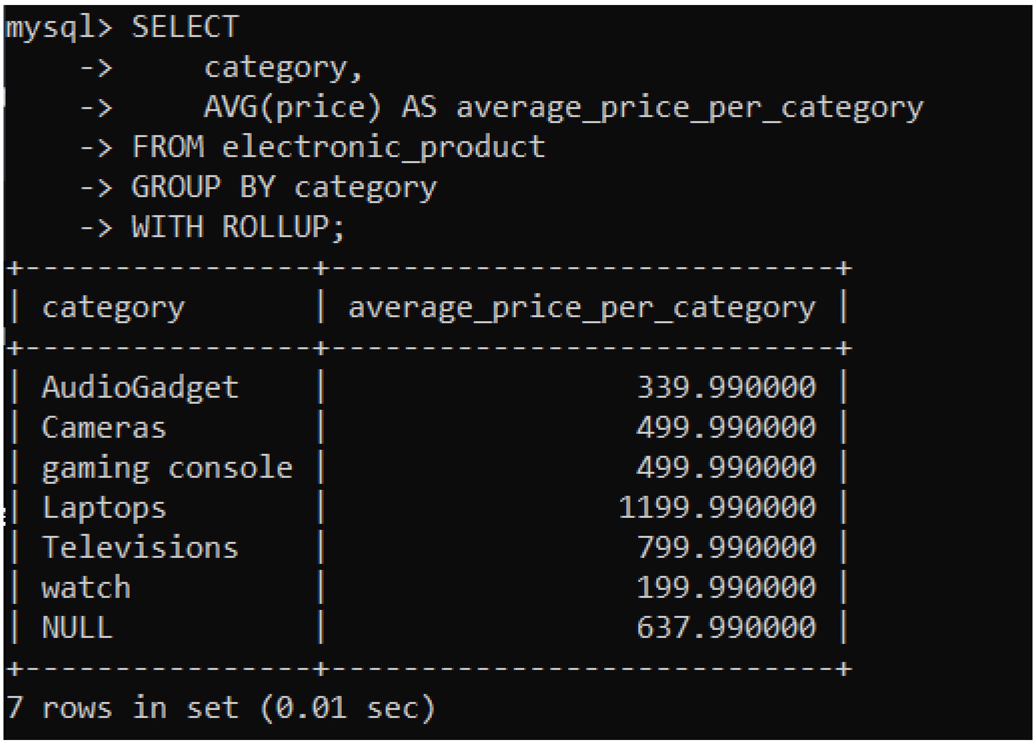
Ending with string ra



```
mysql> SELECT product_name FROM electronic_product WHERE product_name REGEXP 'ra$';
+--------------------+
| product_name       |
+--------------------+
| Digital Camera     |
| Mirrorless Camera  |
+--------------------+
2 rows in set (0.00 sec)
```

How to calculate Subtotals in SQL

WITH ROLLUP is used to include extra rows in the result set to provide subtotals and a grand total.



```
mysql> SELECT
    ->     category,
    ->     AVG(price) AS average_price_per_category
    -> FROM electronic_product
    -> GROUP BY category
    -> WITH ROLLUP;
+---------------+----------------------------+
| category      | average_price_per_category |
+---------------+----------------------------+
| AudioGadget   |                 339.990000 |
| Cameras       |                 499.990000 |
| gaming console|                 499.990000 |
| Laptops       |                1199.990000 |
| Televisions   |                 799.990000 |
| watch         |                 199.990000 |
| NULL          |                 637.990000 |
+---------------+----------------------------+
7 rows in set (0.01 sec)
```

Order of Execution of SQL Queries

SELECT column1, column2

FROM table1

JOIN table2 ON table1.id = table2.id

WHERE condition1 = 'value'

GROUP BY column1

HAVING COUNT(*) > 1

ORDER BY column1 DESC

LIMIT 10;

- FROM Clause: Identifies table1 and table2.
- JOIN Clause: Combines rows from table1 and table2 based on the specified condition.
- WHERE Clause: Filters rows based on the specified condition.
- GROUP BY Clause: Groups rows by column1.
- HAVING Clause: Filters groups where the count is greater than 1.
- SELECT Clause: Specifies the columns to be retrieved.
- ORDER BY Clause: Sorts the result set based on column1 in descending order.
- LIMIT 10: Limits the result set to 10 rows.

Rules and Restrictions to Group and Filter Data in SQL queries

- The HAVING clause is used to filter groups based on aggregated results.
- You cannot use aggregate functions in the WHERE clause
- Conditions in the WHERE clause must result in a boolean (true/false) value