

# Extending SQL: Such That Clause

ARG  
Aidan Robert Giordano

# Problem And Solution

## Problem:

In standard SQL, multi-dimensional queries, of even the simplest kind, require complex expressions that lead to poor performance.

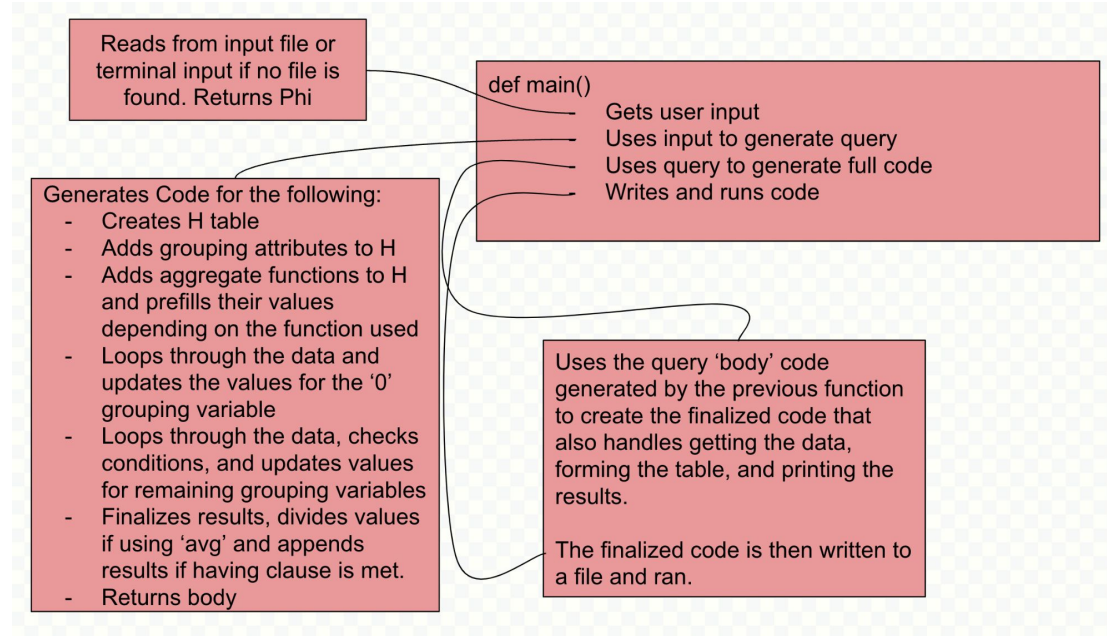
## Solution:

Extending SQL by modifying the 'group by' statement and adding a 'such that' clause that results in an efficient algorithm to process these complex queries.

---

# High Level Architecture

- For each row in the data, the aggregate functions are added to H and their initial values are defined depending on what aggregate used and their key is the grouping attribute tuple.
- For each row in the data, the fields with grouping variable 'O'(no clauses) are updated to reflect true values.
- For each row in the data and for the remaining grouping variables, the such that clause conditions are checked and the corresponding grouping variables are updated using their aggregate functions to reflect true values.
- For each key and value in H, 'avg' values are calculated and results are finalized
- If results meet having clauses, they are appended to the final results that will be made into a table.



# Query Structure

S = List of projected attributes for the query output

N = Number of grouping variables

V = List of grouping attributes

F = List of aggregate functions

$\sigma$  = List of predicates for the grouping variables

G = Predicate for the having clause

S = cust, 1\_avg\_quant, 2\_avg\_quant

N = 2

V = cust

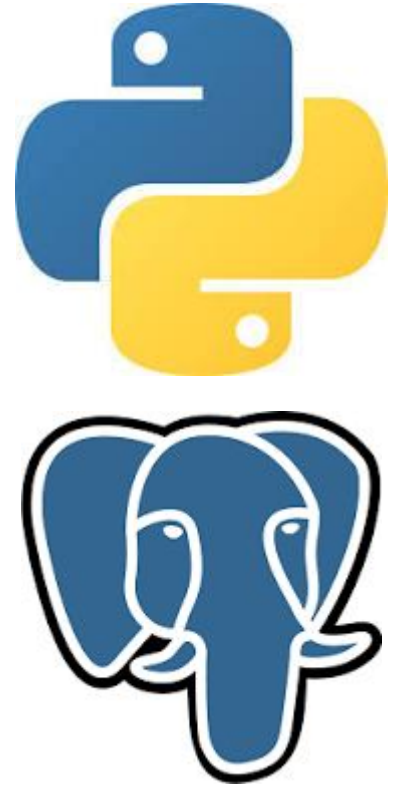
F = 1\_avg\_quant, 2\_avg\_quant

$\sigma$  = 1 year == 2020, 2 year == 2019

G = 1\_avg\_quant > 2\_avg\_quant

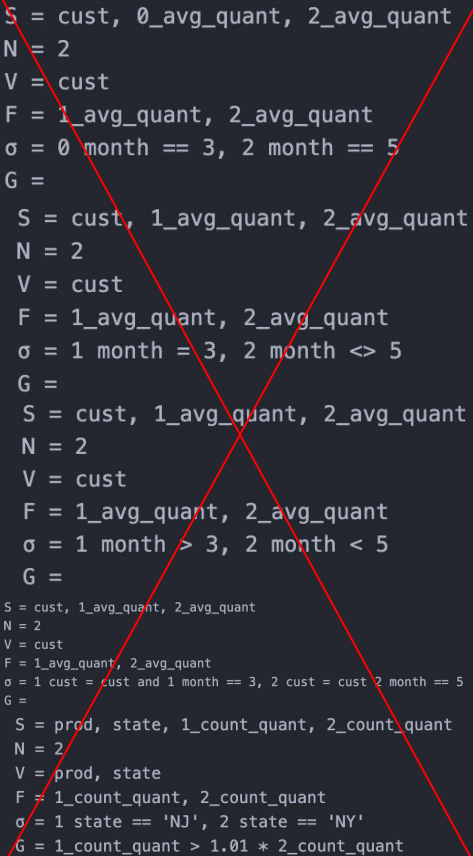
# Technology Stack

- Language:
  - Python
- DBMS:
  - PostgreSQL
- Libraries
  - Dotenv
    - To load DB environment variables
  - Os
    - To read file input and get env. Vars.
  - Psycopg2
    - To connect to the DBMS
  - Tabulate
    - To form the results table



# Technical Limitations

- Grouping variable 'O'
  - 'O' is designated for default queries without such that clauses so it should not have any
- Python Operators
  - Standard SQL operators should not be used and instead replaced with Python Operators
- Such That Clauses
  - Ranges of such that clauses should not intersect
- MF Syntax
  - This program was only built to handle MF Queries and should not be used with EMF Syntax
- Group By
  - Cannot group by an attribute being filtered in the such that clause
- Error Handling
  - Minimal error checking was implemented



```
S = cust, 0_avg_quant, 2_avg_quant
N = 2
V = cust
F = 1_avg_quant, 2_avg_quant
σ = 0 month == 3, 2 month == 5
G =

S = cust, 1_avg_quant, 2_avg_quant
N = 2
V = cust
F = 1_avg_quant, 2_avg_quant
σ = 1 month = 3, 2 month <> 5
G =

S = cust, 1_avg_quant, 2_avg_quant
N = 2
V = cust
F = 1_avg_quant, 2_avg_quant
σ = 1 month > 3, 2 month < 5
G =

S = cust, 1_avg_quant, 2_avg_quant
N = 2
V = cust
F = 1_avg_quant, 2_avg_quant
σ = 1 cust = cust and 1 month == 3, 2 cust = cust 2 month == 5
G =

S = prod, state, 1_count_quant, 2_count_quant
N = 2
V = prod, state
F = 1_count_quant, 2_count_quant
σ = 1 state == 'NJ', 2 state == 'NY'
G = 1_count_quant > 1.01 * 2_count_quant
```

# Demonstration

---

# Conclusion and Future Goals

- In conclusion, this solution greatly simplifies the handling of these multi-dimensional queries and makes them much more efficient.
- This implementation is a solid proof of concept and could be further enhanced to handle more complex queries and be more user friendly.
- In the future, I would implement the ability to handle more complex queries such as EMF. I would also like to make it more user friendly with proper error handling and ability to read input in standard ESQL.

For each product and for sales of 1997, show each month's total sales as percentage of the year-long total sales.

```
select prod, month, sum(x.sale)/sum(y.sale)
```

```
from Sales
```

```
where year=2024
```

```
group by prod,month ; x,y
```

```
such that x.prod = prod and x.month = month, y.prod = prod
```