CS 562 – Course Project

EMF Query Processor - Simplifying Complex SQL Aggregations

Presented by,

Name : Jyotsha Kumar	**Name : Yashas Bangalore Mallesh**
CWID : 20031873	**CWID : 20031185**

Project Overview - What We're Presenting

- **Problem:** Standard SQL struggles to express and efficiently run complex OLAP-style queries that compare different groups or time slices (e.g., month-to-month trends).

- **Our Solution:** We built a custom system that extends SQL with a syntax for grouping variables and aggregates, then processes it using an efficient algorithm based on the Phi operator. This allows cleaner, faster query execution without repeated joins or subqueries.

- **What You'll See:**
- The technical problem and how our tool solves it
- Our system's architecture and logic
- The custom query format we accept
- Demo of the working tool with real output
- Technologies used and known limitations

The Technical Problem (Why We Built This)

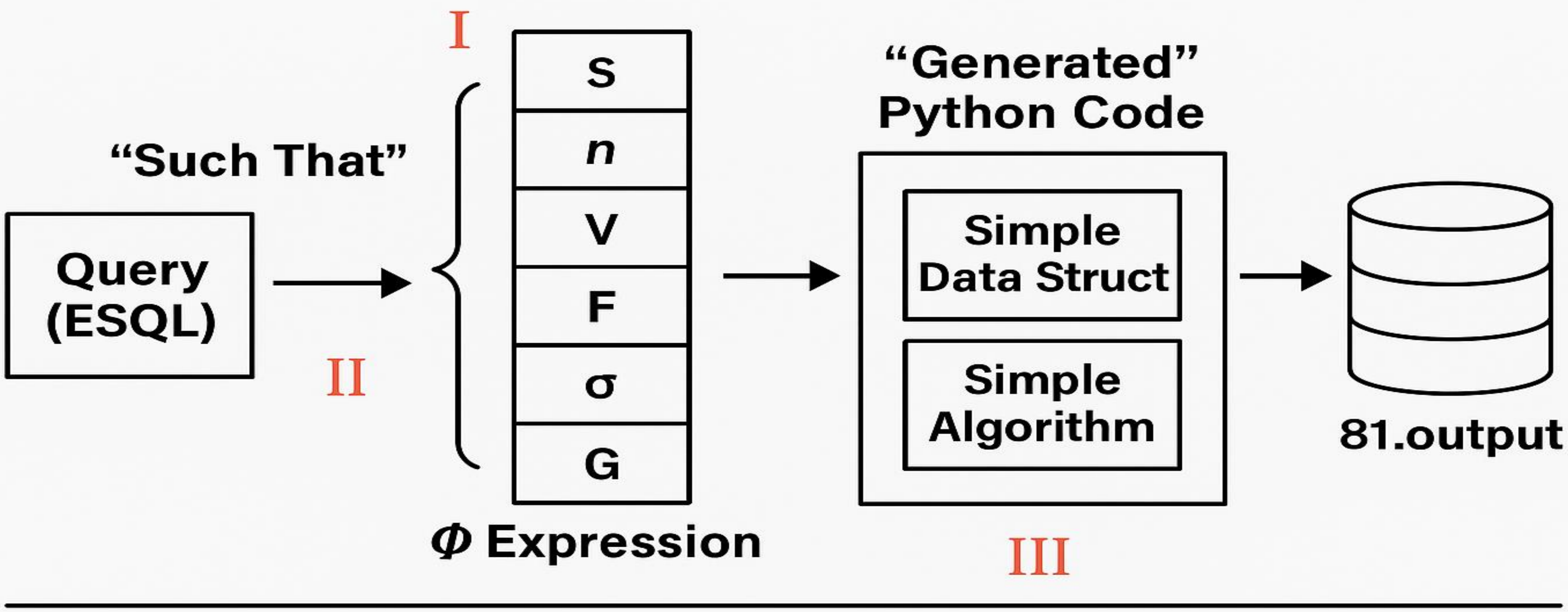
- OLAP queries often need multiple aggregates across dynamic subsets of data (e.g., Jan, Feb, Mar totals per product).
- Standard SQL needs multiple joins, views, and subqueries to do this – it becomes slow and hard to write.
- Even writing simple queries becomes verbose and repetitive.

- **We wanted to:**
- Make it easier to define complex, feature-rich group comparisons.
- Reduce performance overhead by avoiding joins.
- Let the user describe what they want – and let the system figure out how to compute it efficiently.

System Architecture - High Level View

- **Architecture Flow:**
- **Input Layer:**
    - User provides query in Extended SQL (ESQL) or Phi format.
- **Parser:**
    - Breaks it into structured Phi components (S, n, V, F, P, H).
- **mf_struct Builder:**
    - Creates a memory structure (like a custom table) with columns for each group and aggregate.
- **Processing Engine:**
    - Scans the database multiple times (once per dependency layer) and fills `mf_struct` accordingly.
- **Output:**
    - Final grouped results printed to console or file.

Architecture Design



EMF Query Processor

Team contribution – Who did what

- **Team member 1: Yashas Bangalore Mallesh**

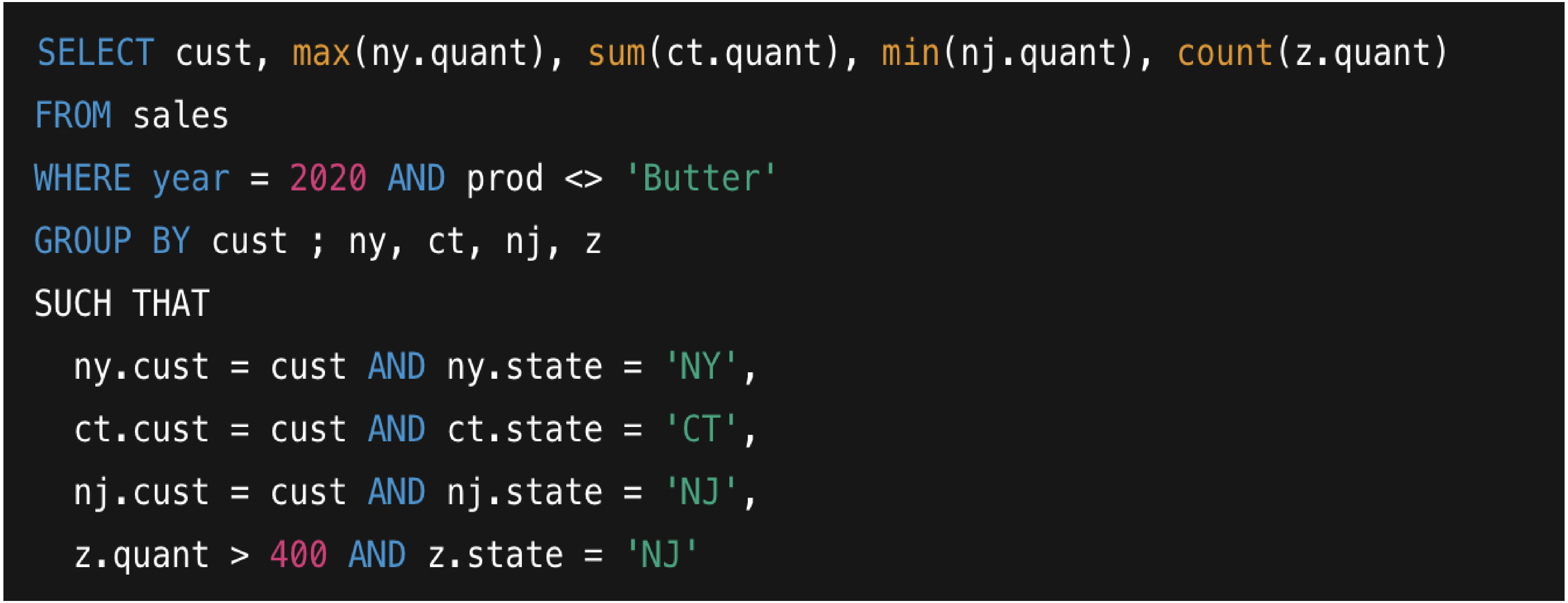
Handled the **MF Query Processor**, including parsing and processing the following Phi components:
S, V, n, P, F
(Selection, Group-by attributes, Grouping variable count, Predicates, Aggregates)

- **Team Member 2** : **Jyotsha Kumar**

Handled the **Extended EMF Query Processor**, which includes all components:
S, V, n, P, H, F
(Selection, Group-by attributes, Grouping variable count, Predicates, Having clause, Aggregates)

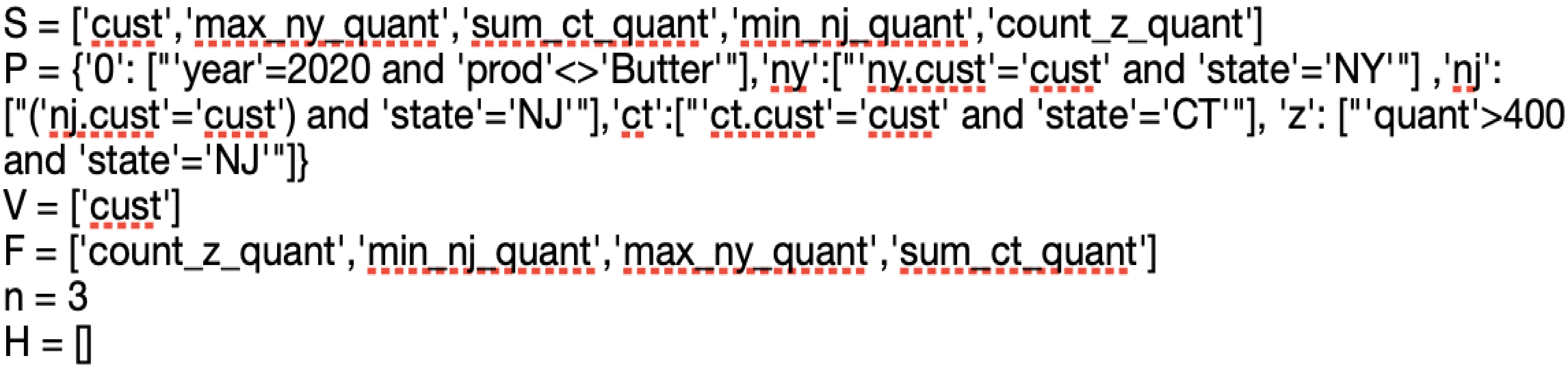
How Queries Work in Our Program

Example Query (EMF style):



```
SELECT cust, max(ny.quant), sum(ct.quant), min(nj.quant), count(z.quant)
FROM sales
WHERE year = 2020 AND prod <> 'Butter'
GROUP BY cust ; ny, ct, nj, z
SUCH THAT
  ny.cust = cust AND ny.state = 'NY',
  ct.cust = cust AND ct.state = 'CT',
  nj.cust = cust AND nj.state = 'NJ',
  z.quant > 400 AND z.state = 'NJ'
```

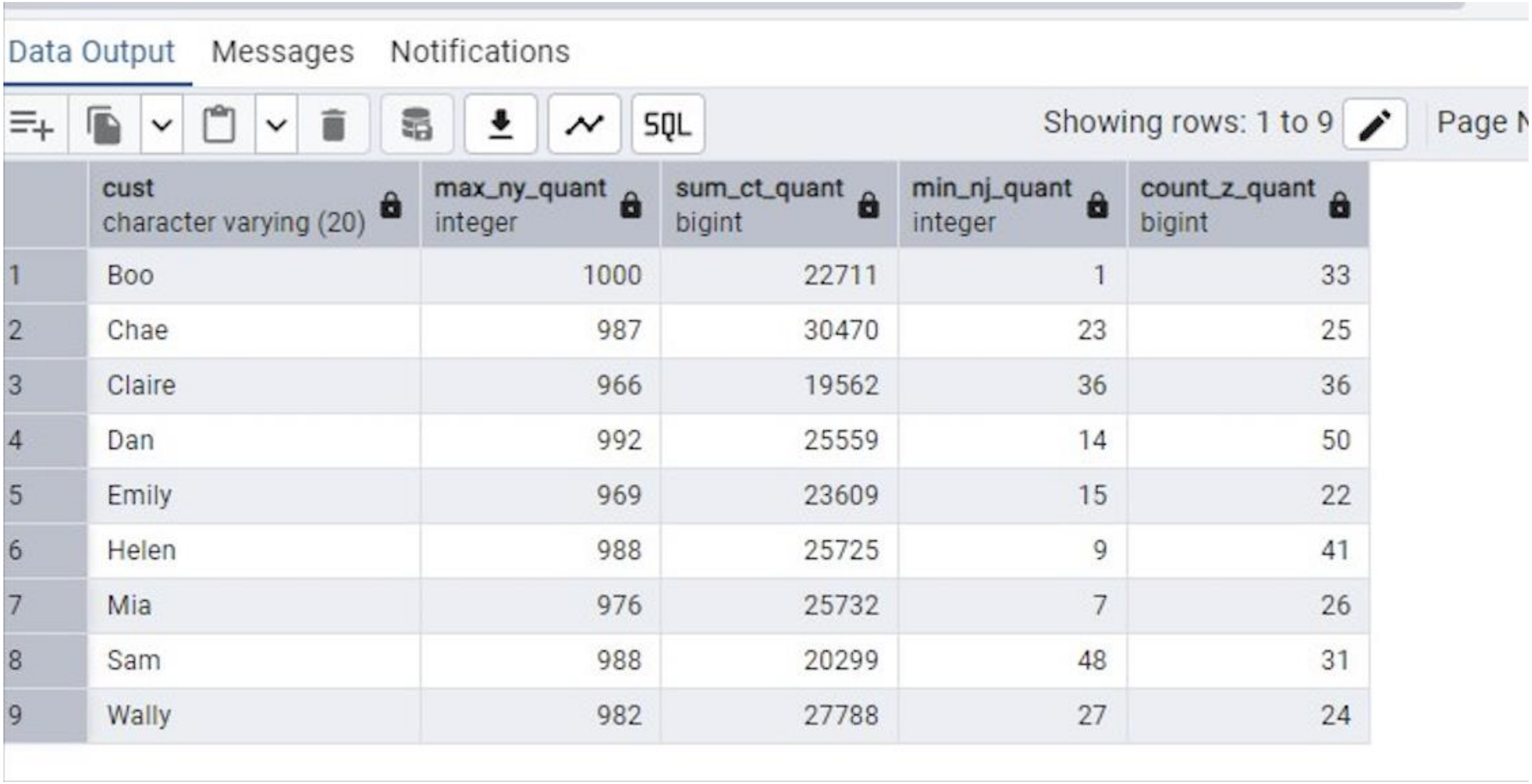
We use converted EMF into Phi form:



```
S = ['cust','max_ny_quant','sum_ct_quant','min_nj_quant','count_z_quant']
P = {'0': ["'year'=2020 and 'prod'<>'Butter'"],'ny':["'ny.cust'='cust' and 'state'='NY'"] ,'nj':
["('nj.cust'='cust') and 'state'='NJ'"],'ct':["'ct.cust'='cust' and 'state'='CT'"], 'z': ["'quant'>400
and 'state'='NJ'"]}
V = ['cust']
F = ['count_z_quant','min_nj_quant','max_ny_quant','sum_ct_quant']
n = 3
H = []
```

This Phi form will give the output table when run by our generated code run_emf_query.py

The generated output



	cust character varying (20)	max_ny_quant integer	sum_ct_quant bigint	min_nj_quant integer	count_z_quant bigint
1	Boo	1000	22711	1	33
2	Chae	987	30470	23	25
3	Claire	966	19562	36	36
4	Dan	992	25559	14	50
5	Emily	969	23609	15	22
6	Helen	988	25725	9	41
7	Mia	976	25732	7	26
8	Sam	988	20299	48	31
9	Wally	982	27788	27	24

Tools and Technology Stack

- **Language:** Python
- **Database:** PostgreSQL
- **Libraries:**
  - `psycopg2` for DB connection
  - `.env` for secure config
  - `tabulate` for nice table output
- **Others:**
  - Plain text file input/output
  - Structured query parsing logic
  - **Limitations:**
- No error messages for invalid queries
- No UI – purely command line

Live Demo

We will:

- Load the sales table
- Show an EMF query and its SQL equivalent
- Run our engine and display the final output table
- Optionally show the internal `mf_struct` generation logic

Recap and Future Scope

- **Summary:**
- We addressed the complexity of OLAP-style SQL queries.
- Using the Phi operator and custom grouping logic, our tool simplifies these queries.

- **Possible Enhancements:**
- Add query validation and error handling.
- Create B+ tree index per PHI operation dynamically for speeding up processing.
- Build a web interface or visualizer for the `mf_struct.`
- Extend support to other types of aggregations (e.g., nested queries).