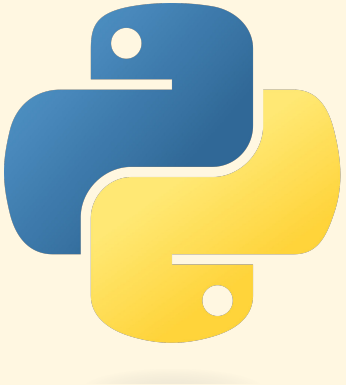
CS 562 Database Management Systems II
Project Presentation & Demo



Steven DeFalco and **Lucas Hope**

Problem & Purpose

“Ad-hoc OLAP queries expressed in SQL...often lead to complex relational algebraic expressions...traditional optimizers do not consider the ‘big picture’.” (Canvas)

⇒ Performing simple aggregates on subsets of data requires subqueries and joins

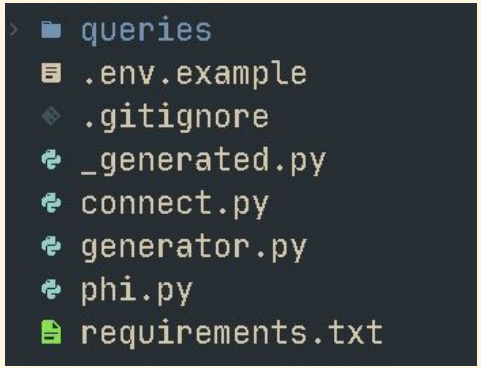
“Extend the group-by statement…(to) take advantage of grouping variables and SUCH THAT to avoid need for using multiple subqueries and joins.” (Canvas)

⇒ Use grouping variables to easily represent subsets of tuples within single query

Demo Outline

1. Project **Architecture**
2. Our **Query Structure**
3. The **Technology** We Use
4. **Technical Limitations** of Our Solution
5. Demo Queries (5)
    a. Query written in **plain SQL**
        i. **Expected output** from PostgreSQL
    b. Query written using **ESQL**
    c. **Input file** for our program

Project Architecture



Generator.py

- Run with ***python3 generatory.py***
  - Prompts user for phi params or file path
- Writes ***_generated.py*** to process query
- Uses ***subprocess*** to run ***_generated.py*** to get result and print it to the terminal

phi.py

- Gets (6) phi operator params from given file
- Returns dict to access each parameter

connect.py

- Establishes connection to postgreSQL DB
- Requires accurate .env file

_generated.py

- Dummy file
- Written by generator.py
- Executed to produce H table

Query Structure

```
SELECT ATTRIBUTE(S):
cust, 1_sum_quant, 2_sum_quant, 3_sum_quant
NUMBER OF GROUPING VARIABLES(n):
3
GROUPING ATTRIBUTES(V):
cust
F-VECT([F]):
1_sum_quant, 1_avg_quant, 2_sum_quant, 3_sum_quant, 3_avg_quant
SELECT CONDITION-VECT([σ]):
1.state='NY'
2.state='NJ'
3.state='CT'
HAVING_CONDITION(G):
1_sum_quant > 2 * 2_sum_quant or 1_avg_quant > 3_avg_quant
```

Technology We Use

- **SQL** to get the entire sales table into Python
- **Text files** (.txt) to hold queries to be executed
- **Python** for all computation, processing, display
  - ↳ **Psycopg2** to access postgreSQL DB from Python script
  - ↳ **Tabulate** to print final H Table *nicely*
  - ↳ **Dotenv** to import environment variables (from .env)
  - ↳ **os** to create accurate file paths in python
  - ↳ **sys** for maxsize (int max) for initialization of H Table fields

Technical Limitations

Minor differences from SQL:

- Only get distinct attributes
  - ↳ can’t perform → SELECT year FROM sales
  - ↳ Instead → SELECT distinct year FROM sales
- Cannot perform SELECT count(*)
  - ↳ Instead → SELECT count(prod) FROM sales
- SELECT * FROM sales is possible
  - ↳ But is very slow

Technical Limitations

Input issues:

- Grouping Attributes determine the GROUP BY order rather than SELECT
- F-Vector determines the order of the aggregates in the SELECT clause
- HAVING input must be properly spaced to be parsed
- Conditions vector and having clause are limited in functionality
  - Can’t do → month in [1,2,3]
- When using min or max, you should check that the value isn’t the initialized min or max

Technical Limitations

Program usage:

- Only can connect to PostgreSQL database
  - ↳ Uses PostgreSQL OIDs to check input
- There is no ability to rename, so column names aren’t indicative of the output data
- There is no where clause, though it could be possible using the conditions vector

Demo Query 01 (demo1.txt)

Find total quantity of each product sold to each customer in NY, NJ, and CT.

```
with ny as
(
    SELECT cust, prod, sum(quant) as ny_sum_quant
    FROM sales
    WHERE state='NY'
    GROUP BY cust, prod
),
nj as
(
    SELECT cust, prod, sum(quant) as nj_sum_quant
    FROM sales
    WHERE state='NJ'
    GROUP BY cust, prod
),
ct as
(
    SELECT cust, prod, sum(quant) as ct_sum_quant
    FROM sales
    WHERE state='CT'
    GROUP BY cust, prod
)
SELECT *
FROM ny natural join nj natural join ct
```

```
SELECT cust, prod, sum(x.quant), sum(y.quant), sum(z.quant)
FROM sales
GROUP BY cust, prod : x,y,z
SUCH THAT
    x.state='NY',
    y.state='NJ',
    z.state='CT'
```

	cust character varying (20)	prod character varying (20)	ny_sum_quant bigint	nj_sum_quant bigint	ct_sum_quant bigint
1	Boo	Apple	13471	14987	14045
2	Boo	Butter	20110	10737	14522
3	Boo	Cherry	14503	14616	9774
4	Boo	Dates	9628	16506	13501
5	Boo	Eggs	13869	17064	17680
6	Boo	Fish	9980	13325	11144
7	Boo	Grapes	12441	10403	15738
8	Boo	Ham	11289	20404	11312
9	Boo	Ice	9460	12209	10993
10	Boo	Jelly	14865	14172	12467
11	Chae	Apple	14112	13342	11453
12	Chae	Butter	14016	17299	15333
13	Chae	Cherry	20850	6037	13739
14	Chae	Dates	17007	11329	13876
15	Chae	Eggs	15890	10018	16200
16	Chae	Fish	15930	13548	12390
17	Chae	Grapes	11980	13888	13953

Demo Query 02 (demo2.txt)

Find the average quantity of each sale of each product for NJ, NY, and CT

```
with ny as
(
    SELECT prod, round(avg(quant),2) as ny_avg_quant
    FROM sales
    WHERE state='NY'
    GROUP BY prod
),
nj as
(
    SELECT prod, round(avg(quant),2) as nj_avg_quant
    FROM sales
    WHERE state='NJ'
    GROUP BY prod
),
ct as
(
    SELECT prod, round(avg(quant),2) as ct_avg_quant
    FROM sales
    WHERE state='CT'
    GROUP BY prod
)
SELECT *
FROM ny natural join nj natural join ct
```

```
SELECT prod, avg(x.quant), avg(y.quant), avg(z.quant)
FROM sales
GROUP BY prod : x,y,z
SUCH THAT
    x.state='NY',
    y.state='NJ',
    z.state='CT'
```

	prod character varying (20)	ny_avg_quant numeric	nj_avg_quant numeric	ct_avg_quant numeric
1	Apple	488.06	512.38	518.43
2	Butter	529.98	506.23	471.07
3	Cherry	513.44	507.25	488.19
4	Dates	465.03	492.50	512.91
5	Eggs	489.47	487.99	494.48
6	Fish	497.85	503.64	503.02
7	Grapes	486.58	457.12	467.55
8	Ham	486.37	504.09	483.48
9	Ice	524.64	541.83	466.98
10	Jelly	503.18	526.99	513.36

Demo Query 03 (demo3.txt)

Find all products that sold better on average and in total quant in NJ than CT

```
WITH nj as
(
    SELECT prod, round(avg(quant),2) nj_avg , sum(quant) nj_sum
    FROM sales
    WHERE state='NJ'
    GROUP BY prod
), ct as
(
    SELECT prod, round(avg(quant),2) ct_avg , sum(quant) ct_sum
    FROM sales
    WHERE state='CT'
    GROUP BY prod
)
SELECT nj.prod, nj.nj_avg, nj.nj_sum, ct.ct_avg, ct.ct_sum
FROM nj natural join ct
WHERE nj.nj_avg > ct.ct_avg and nj.nj_sum > ct.ct_sum
```

```
SELECT prod, avg(x.quant), sum(x.quant), avg(y.quant), sum(z.quant)
FROM sales
GROUP BY prod : x,y
SUCH THAT
    x.state='NJ',
    y.state='CT'
HAVING avg(x.quant) > avg(y.quant) and sum(x.quant) > sum(y.quant)
```

	prod character varying (20)	nj_avg numeric	nj_sum bigint	ct_avg numeric	ct_sum bigint
1	Ice	541.83	132206	466.98	109740
2	Jelly	526.99	124897	513.36	118072
3	Fish	503.64	133464	503.02	118713
4	Ham	504.09	128038	483.48	126671
5	Cherry	507.25	131885	488.19	114724

Demo Query 04 (demo4.txt)

Find all years where NJ had greater average sales
than NY and display both their average and total sale quantity.

```
WITH nj as
(
    SELECT year, round(avg(quant),2) nj_yearAvg, sum(quant) nj_yearSum
    FROM sales
    WHERE state='NJ'
    GROUP BY year
), ny as
(
    SELECT year, round(avg(quant),2) ny_yearAvg, sum(quant) ny_yearSum
    FROM sales
    WHERE state='NY'
    GROUP BY year
)
SELECT * FROM nj natural join ny
WHERE nj_yearAvg > ny_yearAvg
```

```
SELECT year, avg(x.quant), sum(x.quant), avg(y.quant), sum(y.quant)
FROM sales
GROUP BY year
SUCH THAT
    x.state = 'NJ',
    y.state = 'NY'
HAVING avg(x.quant) > avg(y.quant)
```

	year integer	nj_yearavg numeric	nj_yearsum bigint	ny_yearavg numeric	ny_yearsum bigint
1	2016	510.24	264817	482.81	251542
2	2020	533.20	264467	492.35	260945

Demo Query 05 (demo5.txt)

Find each customer and product's maximum quantity for each month and display if that maximum quantity increased from 2016 to 2017 and 2017 to 2018

```
WITH q1 as
(
    SELECT cust, prod, month, max(quant) max1
    FROM sales
    WHERE year=2016
    GROUP BY cust, prod, month
), q2 as
(
    SELECT cust, prod, month, max(quant) max2
    FROM sales
    WHERE year=2017
    GROUP BY cust, prod, month
), q3 as
(
    SELECT cust, prod, month, max(quant) max3
    FROM sales
    WHERE year=2018
    GROUP BY cust, prod, month
)
SELECT q1.cust, q1.prod, q1.month, q1.max1, q2.max2, q3.max3
FROM q1, q2, q3
WHERE q1.cust = q2.cust and q2.cust = q3.cust and
q1.prod = q2.prod and q2.prod = q3.prod and
q1.month = q2.month and q2.month = q3.month and
q1.max1 > 0 and q3.max3 > q2.max2 and q2.max2 > q1.max1
ORDER BY cust, prod, month
```

```
SELECT cust, prod, month, avg(x.quant), avg(y.quant), avg(z.quant)
FROM sales
GROUP BY cust, prod, month
SUCH THAT
    x.year = 2016
    y.year = 2017
    z.year = 2018
HAVING avg(z.quant) > avg(y.quant) and avg(y.quant) > avg(x.quant)
```

	cust character varying (20)	prod character varying (20)	month integer	max1 integer	max2 integer	max3 integer
1	Boo	Apple	12	407	575	639
2	Boo	Butter	7	256	756	837
3	Boo	Butter	9	543	582	593
4	Boo	Dates	2	577	593	983
5	Boo	Ham	2	61	271	470
6	Boo	Ice	5	73	880	990
7	Boo	Jelly	3	489	696	699
8	Boo	Jelly	5	484	508	613
9	Boo	Jelly	6	350	381	801
10	Chae	Apple	2	68	300	684
11	Chae	Apple	3	747	787	921
12	Chae	Apple	10	1	249	903
13	Chae	Butter	12	367	644	789

⋮