

# CS4221 Tutorial 2

## Tuning in PostgreSQL

Yao LU  
2024 Semester 2

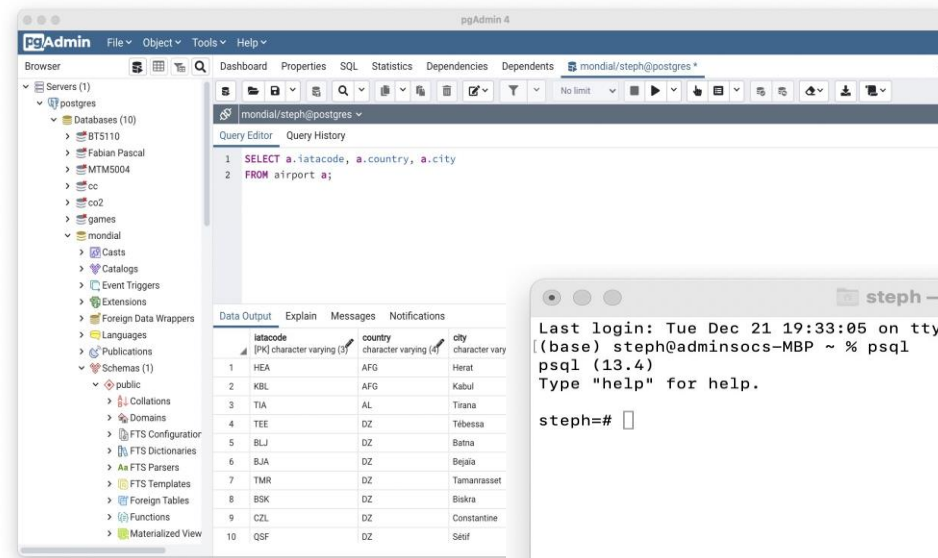
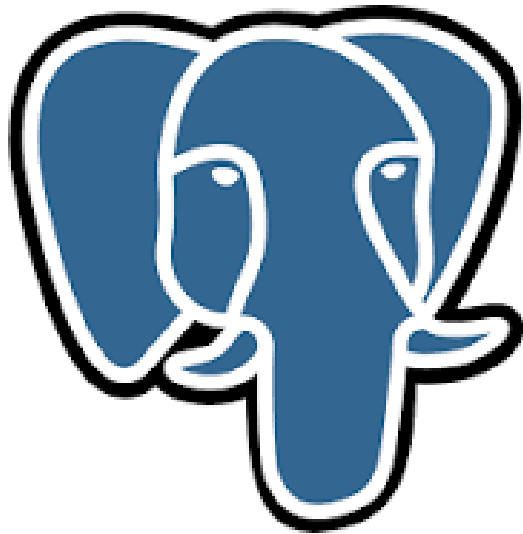
National University of Singapore  
School of Computing

# Goals

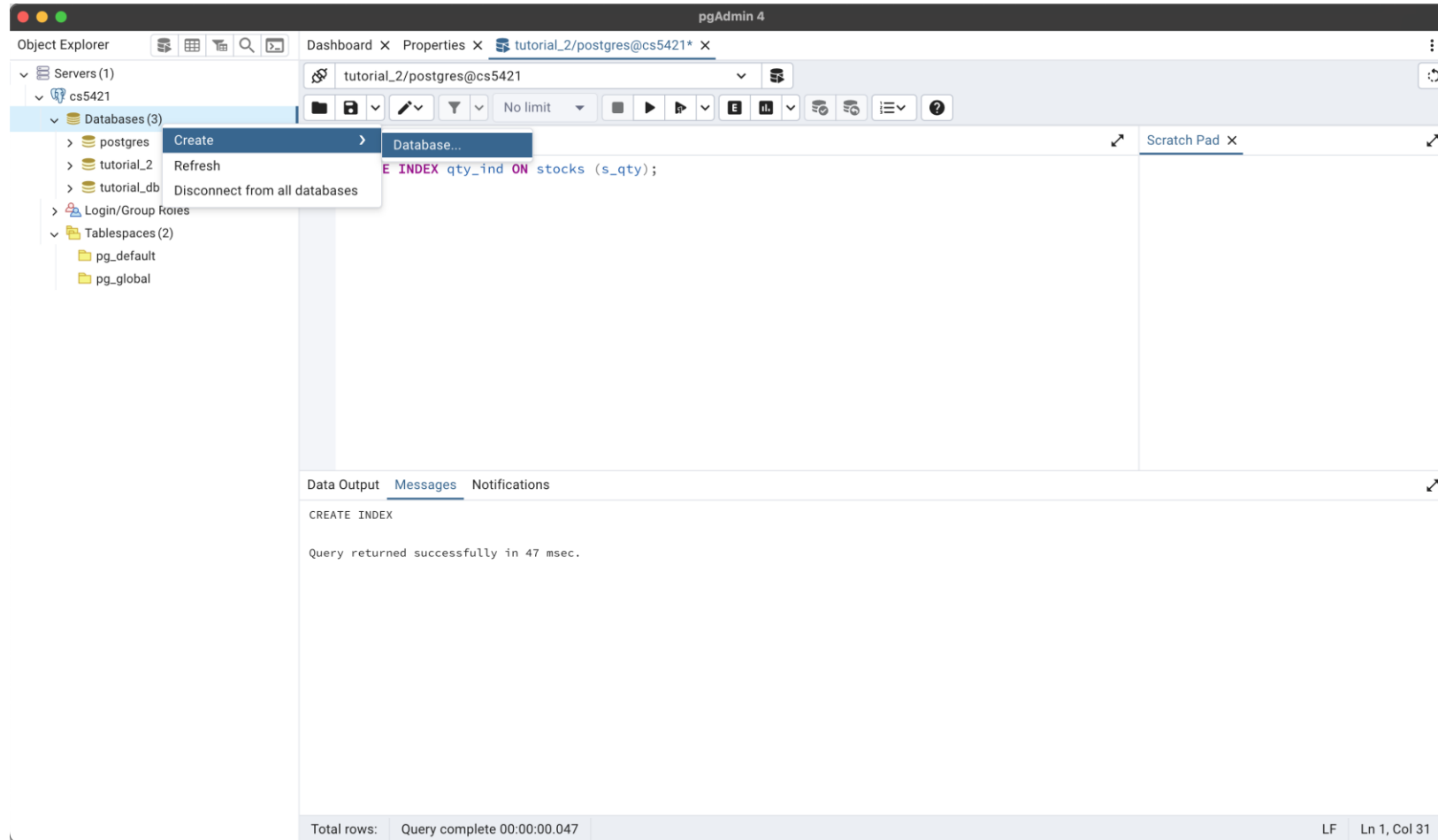
- Demo
  - How to Use pgAdmin 4 with PostgreSQL
- Tuning Strategies

# pgAdmin 4

- We use the relational database management system PostgreSQL with pgAdmin 4.



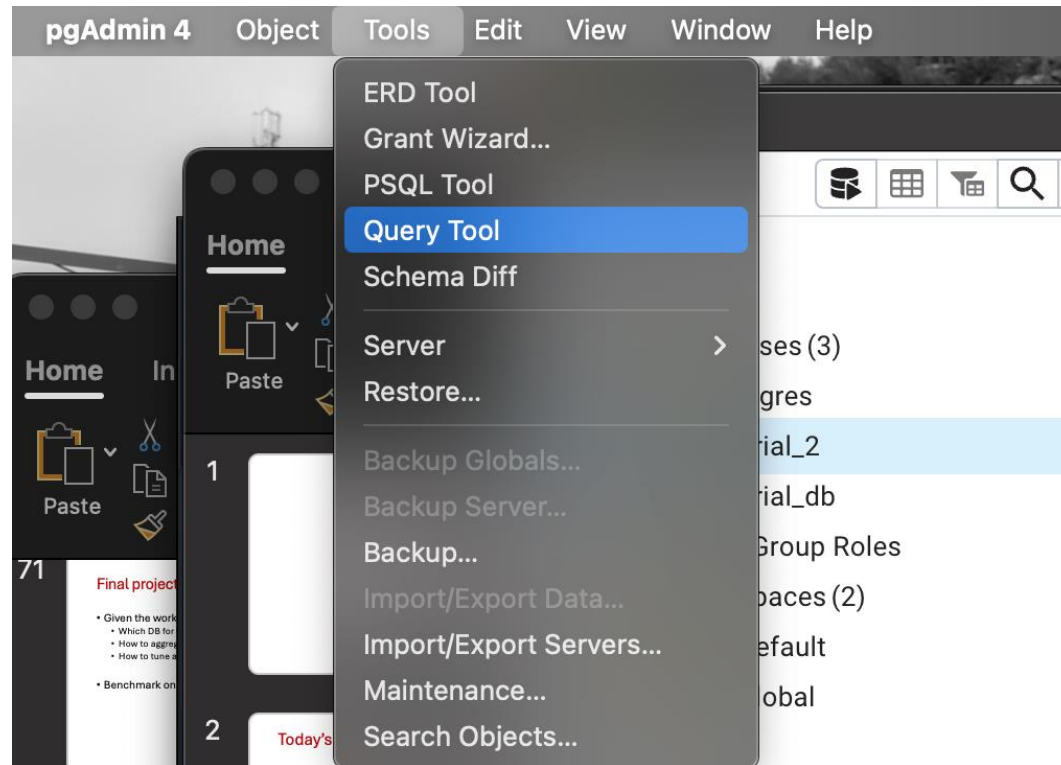
# pgAdmin 4 – Create a new Database



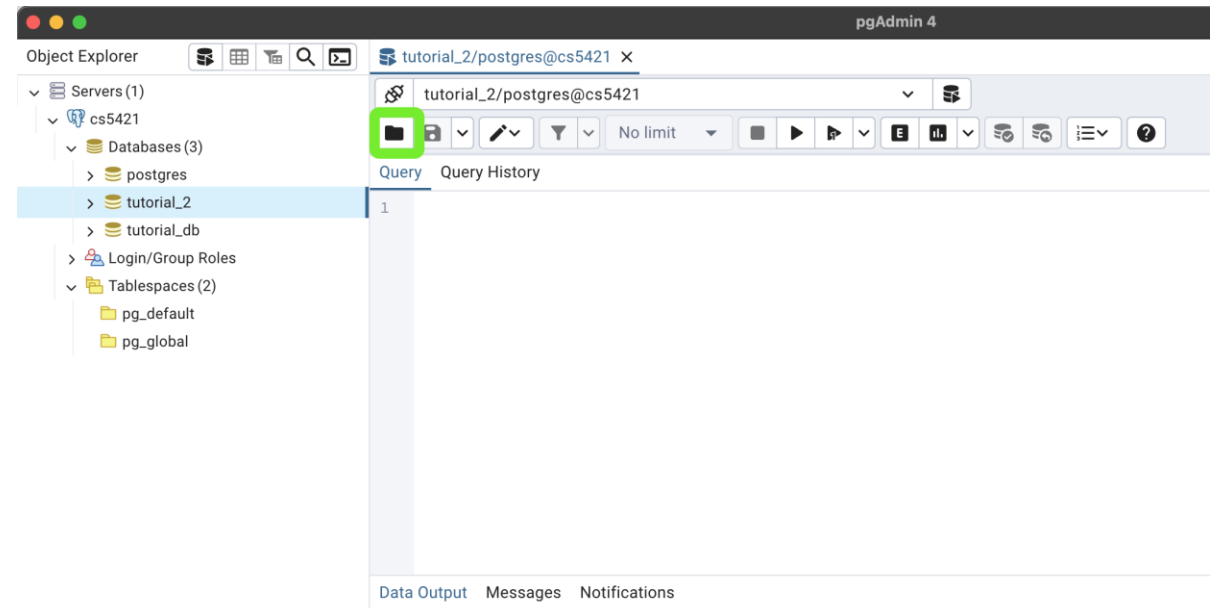
Name it as “tutorial\_2”

# pgAdmin 4 – Import/Edit SQL

## 1. Open Query Tool

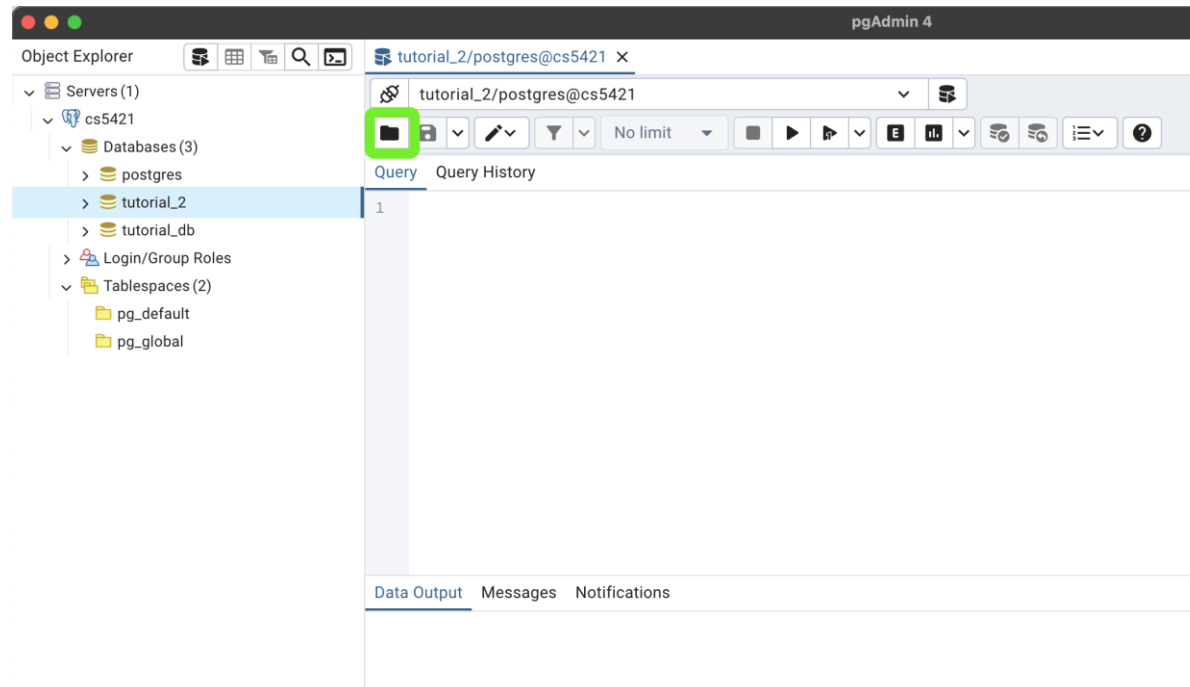


## 2. Select the prepared SQL files and run it



# pgAdmin 4 – Import/Edit SQL

## 2. Select the prepared SQL files and run it

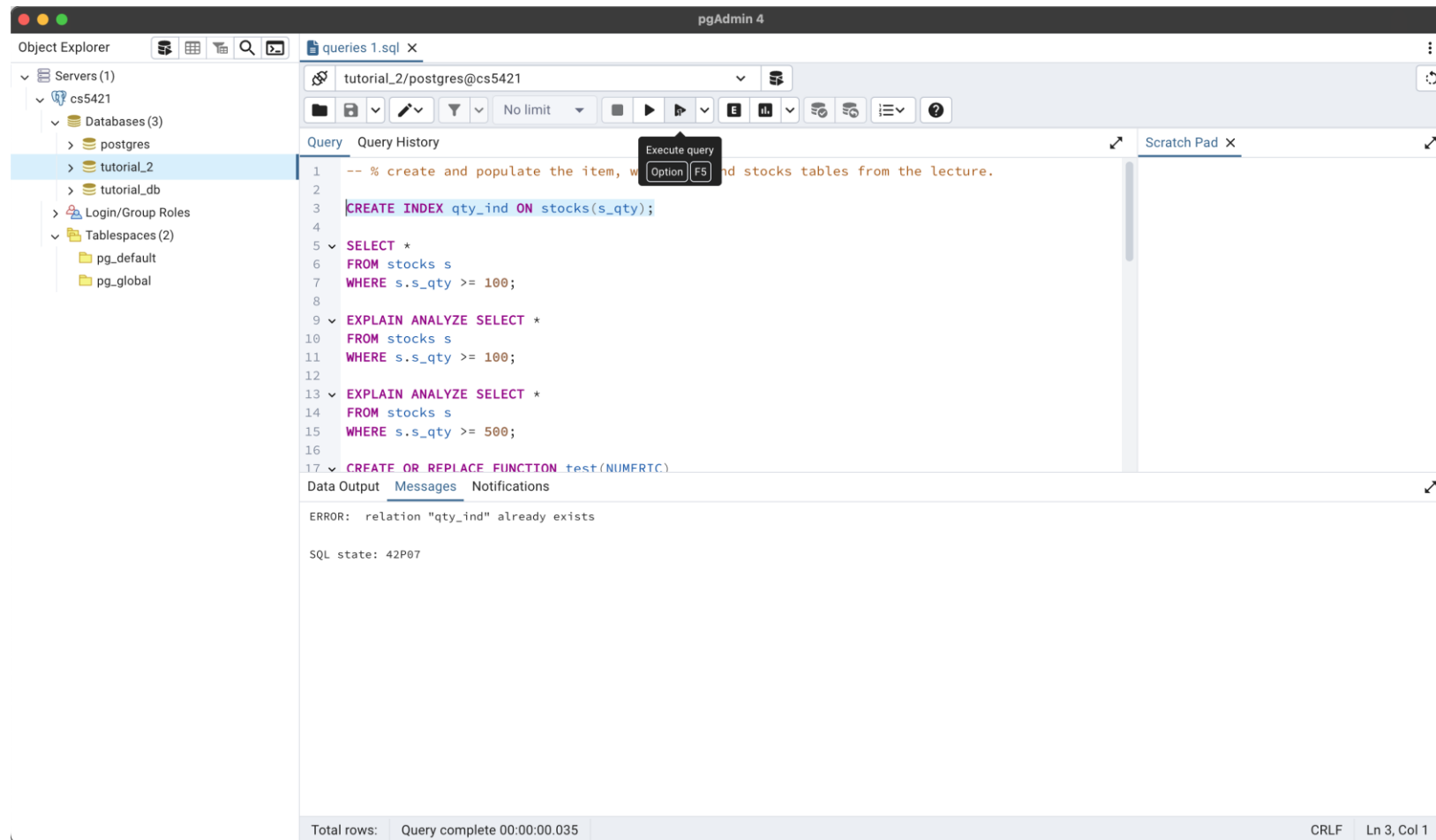


- **Import** the TPC-C data (items, stocks, warehouses) from TPC-C folder
- **Import** the crossword data (crossword.sql).
- Prepare queries 1.sql and queries 2.sql file

# pgAdmin 4 – Run SQL

Let's start with **queries 1.sql**...

Select the desired SQL query and Press “Execute Query” to execute a single query each time



# pgAdmin 4 – Run SQL

**Write the SQL query that finds the items in quantity larger than or equal to 100.**

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the Object Explorer with a tree view of the database structure. The main window shows a SQL query editor with the following code:

```
1 -- % create and populate the item, w and stocks tables from the lecture.
2
3 CREATE INDEX qty_ind ON stocks(s_qty);
4
5 SELECT *
6 FROM stocks s
7 WHERE s.s_qty >= 100;
8
9 EXPLAIN ANALYZE SELECT *
10 FROM stocks s
11 WHERE s.s_qty >= 100;
12
13 EXPLAIN ANALYZE SELECT *
14 FROM stocks s
15 WHERE s.s_qty >= 500;
16
17 CREATE OR REPLACE FUNCTION test(NUMERIC)
```

The query results are displayed in a table with the following columns: w\_id, i\_id, and s\_qty. The table shows 10 rows of data, with the first row having w\_id 301, i\_id 1, and s\_qty 338. The status bar at the bottom indicates "Total rows: 38162" and "Query complete 00:00:00.078".

	w_id [PK] integer	i_id [PK] integer	s_qty smallint
1	301	1	338
2	301	4	938
3	301	5	760
4	301	8	924
5	301	12	454
6	301	13	768
7	301	21	355
8	301	31	700
9	301	36	158
10	301	42	297



# pgAdmin 4 – Run SQL

Print the query plan, planning time, and execution time for queries with different quantities.  
“Use **EXPLAIN ANALYZE**”

s\_qty >= 100:

```
1 -- % create and populate the item, warehouse, and stocks tables from the lecture.
2
3 CREATE INDEX qty_ind ON stocks(s_qty);
4
5 SELECT *
6 FROM stocks s
7 WHERE s.s_qty >= 100;
8
9 EXPLAIN ANALYZE SELECT *
10 FROM stocks s
11 WHERE s.s_qty >= 100;
12
13 EXPLAIN ANALYZE SELECT *
14 FROM stocks s
15 WHERE s.s_qty >= 500;
16
17 CREATE OR REPLACE FUNCTION test(NUMERIC)
```

QUERY PLAN
1 Seq Scan on stocks s (cost=0.00..804.40 rows=38196 width=10) (actual time=0.022..11.086 rows=38162 loops=1)
2 Filter: (s_qty >= 100)
3 Rows Removed by Filter: 6750
4 Planning Time: 0.115 ms
5 Execution Time: 14.054 ms

s\_qty >= 500:

```
1 -- % create and populate the item, warehouse, and stocks tables from the lecture.
2
3 CREATE INDEX qty_ind ON stocks(s_qty);
4
5 SELECT *
6 FROM stocks s
7 WHERE s.s_qty >= 100;
8
9 EXPLAIN ANALYZE SELECT *
10 FROM stocks s
11 WHERE s.s_qty >= 100;
12
13 EXPLAIN ANALYZE SELECT *
14 FROM stocks s
15 WHERE s.s_qty >= 500;
16
17 CREATE OR REPLACE FUNCTION test(NUMERIC)
```

QUERY PLAN
1 Bitmap Heap Scan on stocks s (cost=245.32..754.49 rows=21294 width=10) (actual time=2.172..7.300 rows=21339 loops=1)
2 Recheck Cond: (s_qty >= 500)
3 Heap Blocks: exact=232
4 -> Bitmap Index Scan on qty_ind (cost=0.00..239.99 rows=21294 width=0) (actual time=2.135..2.136 rows=21339 loops=1)
5 Index Cond: (s_qty >= 500)
6 Planning Time: 0.175 ms
7 Execution Time: 9.034 ms

# pgAdmin 4 – Run SQL

**Create a stored function `test(N)` that prints the plan for the query that finds the items in quantity larger than or equal to N.**

```
CREATE OR REPLACE FUNCTION test(NUMERIC)

RETURNS SETOF TEXT AS $$

BEGIN

RETURN QUERY EXECUTE

'EXPLAIN SELECT *

FROM stocks s

WHERE s.s_qty >= ' || $1 ;

END

$$ LANGUAGE plpgsql;
```

# pgAdmin 4 – Run SQL

**Create a stored function `stat(N)` that finds the percentage of items in quantity larger than N, that is the selectivity of the condition.**

```
CREATE OR REPLACE FUNCTION stat(NUMERIC)
RETURNS SETOF NUMERIC AS $$
BEGIN
RETURN QUERY EXECUTE
'SELECT ROUND((COUNT(*)::NUMERIC /(SELECT COUNT(*)
                                FROM stocks))
              *100)
FROM stocks s
WHERE s.s_qty >= ' ||$1 ;
END
$$ LANGUAGE plpgsql;
```

# pgAdmin 4 – Run SQL

**Write an SQL query that tabulates the type of access for varying quantity (in hundreds) and corresponding selectivity.**

```
39
40 ✓ SELECT q.qty, stat(q.qty),
41    regexp_replace((SELECT *
42 FROM test(q.qty) LIMIT 1), '(\\.*)(\\.\\.*', '\\1')
43 FROM (SELECT DISTINCT ROUND(s_qty::NUMERIC/100)*100
44 AS qty FROM stocks) AS q
45 ORDER BY q.qty;
```

Data Output Messages Explain X Notifications

Showing rows: 1 to 11

	qty numeric	stat numeric	regexp_replace text
1	0	100	Seq Scan on stocks s
2	100	85	Seq Scan on stocks s
3	200	76	Seq Scan on stocks s
4	300	66	Seq Scan on stocks s
5	400	57	Seq Scan on stocks s
6	500	47	Bitmap Heap Scan on stocks s
7	600	38	Bitmap Heap Scan on stocks s
8	700	28	Bitmap Heap Scan on stocks s
9	800	19	Bitmap Heap Scan on stocks s
10	900	9	Bitmap Heap Scan on stocks s
11	1000	0	Index Scan using qty_ind on stocks...

# Practice

## Context:

In April 1990, Hal Berghel and Richard Rankin proposed a crossword benchmark in an article titled “A Proposed Standard for Measuring Crossword Compilation Efficiency”

The five rows and the five columns of the crossword are five letter words from a lexicon containing 134 entries. There are 72 solutions.

There are 24 solutions with unique words and not counting transpositions (identical solutions after interchanging rows and columns.)

*The following is an example solution.*

A	A	R	O	N
C	L	I	T	E
A	L	A	T	E
R	O	T	A	L
A	W	A	R	D

```
('R','O','P','E','R'),  
( 'N','A','M','E','R'),  
( 'O','C','H','N','A'),  
( 'R','A','T','E','R'),  
( 'R','O','T','A','L'),  
( 'N','A','N','N','Y'),  
( 'O','M','I','N','A'),  
( 'R','E','B','U','D'),  
( 'R','U','D','A','S'),  
( 'N','A','S','A','L'),  
( 'O','N','S','E','T'),  
( 'R','E','B','U','T'),  
( 'S','A','L','A','L'),  
( 'N','A','T','A','L'),  
( 'O','R','A','O','N'),  
( 'R','E','C','O','N'),  
( 'S','A','L','A','Y'),
```

If the word “AARON” is removed, there is no solution.

The lexicon is stored in the table word created and populated with the SQL script crossword.sql

# Crossword

*Run the simplest possible queries that find the solutions to the crossword problem with and without unique words and transposition.*

***Solution:*** *The following query finds all the solutions.*

```
SELECT * FROM
word r1, word r2, word r3, word r4, word r5,
word c1, word c2, word c3, word c4, word c5
WHERE r1.a1=c1.a1 AND r1.a2=c2.a1 AND r1.a3=c3.a1 AND r1.a4=c4.a1 AND r1.a5=c5.a1
AND r2.a1=c1.a2 AND r2.a2=c2.a2 AND r2.a3=c3.a2 AND r2.a4=c4.a2 AND r2.a5=c5.a2
AND r3.a1=c1.a3 AND r3.a2=c2.a3 AND r3.a3=c3.a3 AND r3.a4=c4.a3 AND r3.a5=c5.a3
AND r4.a1=c1.a4 AND r4.a2=c2.a4 AND r4.a3=c3.a4 AND r4.a4=c4.a4 AND r4.a5=c5.a4
AND r5.a1=c1.a5 AND r5.a2=c2.a5 AND r5.a3=c3.a5 AND r5.a4=c4.a5 AND r5.a5=c5.a5;
```

# Crossword

*Run the simplest possible queries that find the solutions to the crossword problem with and without unique words and transposition.*

**Solution:** *The following query finds the solutions with unique words and no transposition symmetry.*

```
SELECT * FROM
word r1, word r2, word r3, word r4, word r5,
word c1, word c2, word c3, word c4, word c5
      /* answer the crossword */
WHERE r1.a1=c1.a1 AND r1.a2=c2.a1 AND r1.a3=c3.a1 AND r1.a4=c4.a1 AND r1.a5=c5.a1
AND r2.a1=c1.a2 AND r2.a2=c2.a2 AND r2.a3=c3.a2 AND r2.a4=c4.a2 AND r2.a5=c5.a2
AND r3.a1=c1.a3 AND r3.a2=c2.a3 AND r3.a3=c3.a3 AND r3.a4=c4.a3 AND r3.a5=c5.a3
AND r4.a1=c1.a4 AND r4.a2=c2.a4 AND r4.a3=c3.a4 AND r4.a4=c4.a4 AND r4.a5=c5.a4
AND r5.a1=c1.a5 AND r5.a2=c2.a5 AND r5.a3=c3.a5 AND r5.a4=c4.a5 AND r5.a5=c5.a5

/* rows cannot be the same */
AND r1 <> r2 AND r1 <> r3 AND r1 <> r4 AND r1 <> r5
AND r2 <> r3 AND r2 <> r4 AND r2 <> r5
AND r3 <> r4 AND r3 <> r5
AND r4 <> r5

/* columns cannot be the same */
AND c1 <> c2 AND c1 <> c3 AND c1 <> c4 AND c1 <> c5
AND c2 <> c3 AND c2 <> c4 AND c2 <> c5
AND c3 <> c4 AND c3 <> c5
AND c4 <> c5

/* row and column cannot be the same */
AND r1 <> c2 AND r1 <> c3 AND r1 <> c4 AND r1 <> c5
AND r2 <> c1 AND r2 <> c2 AND r2 <> c3 AND r2 <> c4 AND r2 <> c5
AND r3 <> c1 AND r3 <> c2 AND r3 <> c3 AND r3 <> c4 AND r3 <> c5
AND r4 <> c1 AND r4 <> c2 AND r4 <> c3 AND r4 <> c4 AND r4 <> c5
AND r5 <> c1 AND r5 <> c2 AND r5 <> c3 AND r5 <> c4 AND r5 <> c5

/* break the symmetry */
AND r1 < c1;
```

# Crossword

## Task 1.1 (submit the results later to CANVAS):

Print the query plan, planning time, and execution time for the two queries in the previous slides. Compare the results.

### Query 1

```
SELECT * FROM
word r1, word r2, word r3, word r4, word r5,
word c1, word c2, word c3, word c4, word c5
WHERE r1.a1=c1.a1 AND r1.a2=c2.a1 AND r1.a3=c3.a1 AND r1.a4=c4.a1 AND r1.a5=c5.a1
AND r2.a1=c1.a2 AND r2.a2=c2.a2 AND r2.a3=c3.a2 AND r2.a4=c4.a2 AND r2.a5=c5.a2
AND r3.a1=c1.a3 AND r3.a2=c2.a3 AND r3.a3=c3.a3 AND r3.a4=c4.a3 AND r3.a5=c5.a3
AND r4.a1=c1.a4 AND r4.a2=c2.a4 AND r4.a3=c3.a4 AND r4.a4=c4.a4 AND r4.a5=c5.a4
AND r5.a1=c1.a5 AND r5.a2=c2.a5 AND r5.a3=c3.a5 AND r5.a4=c4.a5 AND r5.a5=c5.a5;
```

### Query 2

```
SELECT * FROM
word r1, word r2, word r3, word r4, word r5,
word c1, word c2, word c3, word c4, word c5
/* answer the crossword */
WHERE r1.a1=c1.a1 AND r1.a2=c2.a1 AND r1.a3=c3.a1 AND r1.a4=c4.a1 AND r1.a5=c5.a1
AND r2.a1=c1.a2 AND r2.a2=c2.a2 AND r2.a3=c3.a2 AND r2.a4=c4.a2 AND r2.a5=c5.a2
AND r3.a1=c1.a3 AND r3.a2=c2.a3 AND r3.a3=c3.a3 AND r3.a4=c4.a3 AND r3.a5=c5.a3
AND r4.a1=c1.a4 AND r4.a2=c2.a4 AND r4.a3=c3.a4 AND r4.a4=c4.a4 AND r4.a5=c5.a4
AND r5.a1=c1.a5 AND r5.a2=c2.a5 AND r5.a3=c3.a5 AND r5.a4=c4.a5 AND r5.a5=c5.a5

/* rows cannot be the same */
AND r1 <> r2 AND r1 <> r3 AND r1 <> r4 AND r1 <> r5
AND r2 <> r3 AND r2 <> r4 AND r2 <> r5
AND r3 <> r4 AND r3 <> r5
AND r4 <> r5

/* columns cannot be the same */
AND c1 <> c2 AND c1 <> c3 AND c1 <> c4 AND c1 <> c5
AND c2 <> c3 AND c2 <> c4 AND c2 <> c5
AND c3 <> c4 AND c3 <> c5
AND c4 <> c5

/* row and column cannot be the same */
AND r1 <> c2 AND r1 <> c3 AND r1 <> c4 AND r1 <> c5
AND r2 <> c1 AND r2 <> c2 AND r2 <> c3 AND r2 <> c4 AND r2 <> c5
AND r3 <> c1 AND r3 <> c2 AND r3 <> c3 AND r3 <> c4 AND r3 <> c5
AND r4 <> c1 AND r4 <> c2 AND r4 <> c3 AND r4 <> c4 AND r4 <> c5
AND r5 <> c1 AND r5 <> c2 AND r5 <> c3 AND r5 <> c4 AND r5 <> c5

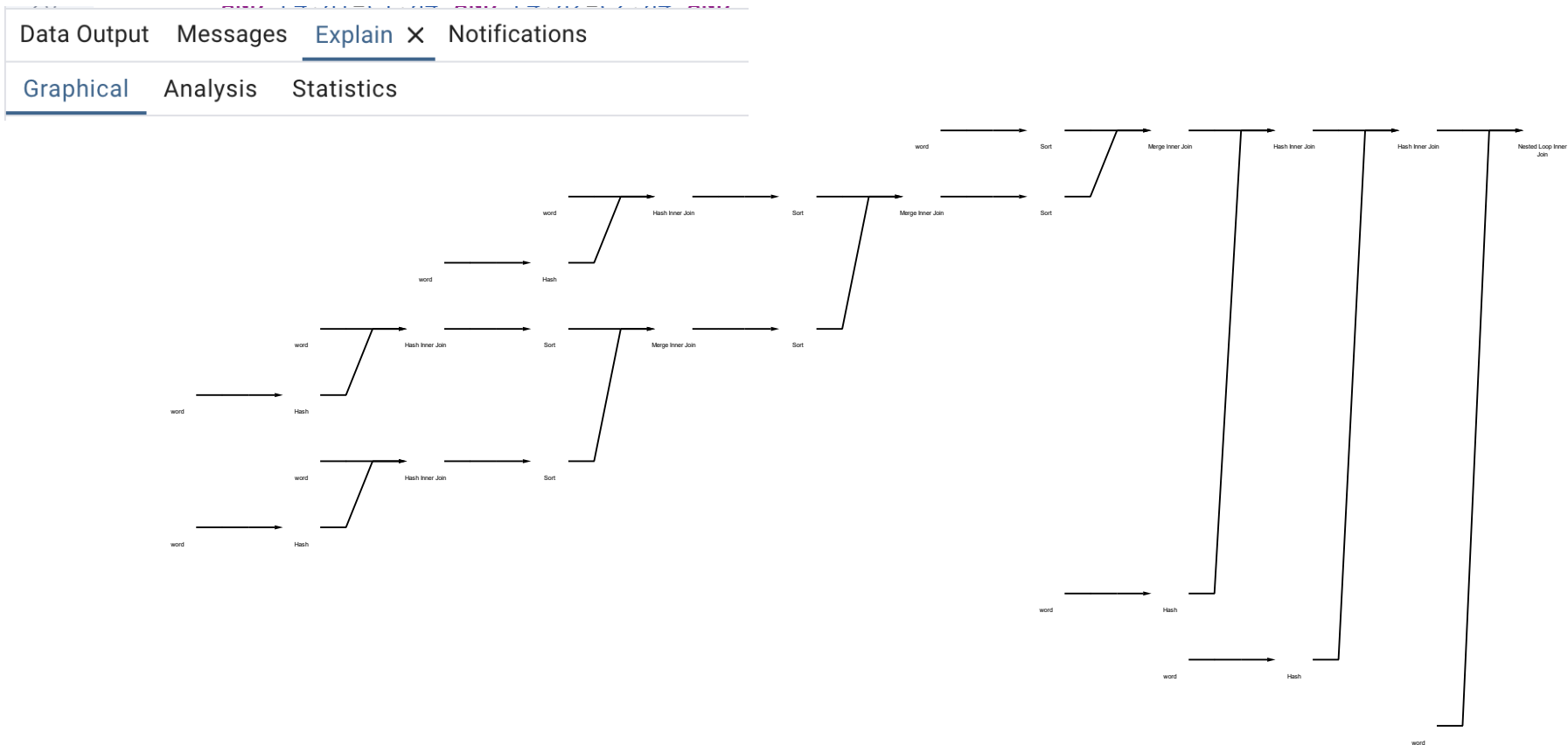
/* break the symmetry */
AND r1 < c1 ;
```



# Crossword

## Task 1.2 (submit the results later to CANVAS):

Using the Explain (F7) option of pgAdmin 4, print the two queries' query plans in JSON, save the tree representation of the query plan in SVG, read and save the analysis and statistics.



# Crossword

**Task 2: Try out several possible methods to improve efficiency**

*Prepare the queries and find their planning and execution time.*

```
 /***** 2d *****/
PREPARE q1 AS SELECT * FROM
word r1, word r2, word r3, word r4, word r5,
word c1, word c2, word c3, word c4, word c5
WHERE r1.a1=c1.a1 AND r1.a2=c2.a1 AND r1.a3=c3.a1
AND r1.a4=c4.a1 AND r1.a5=c5.a1
AND r2.a1=c1.a2 AND r2.a2=c2.a2 AND r2.a3=c3.a2
AND r2.a4=c4.a2 AND r2.a5=c5.a2
AND r3.a1=c1.a3 AND r3.a2=c2.a3 AND r3.a3=c3.a3
AND r3.a4=c4.a2 AND r3.a5=c5.a3
AND r4.a1=c1.a4 AND r4.a2=c2.a4 AND r4.a3=c3.a4
AND r4.a4=c4.a4 AND r4.a5=c5.a4
AND r5.a1=c1.a5 AND r5.a2=c2.a5 AND r5.a3=c3.a5
AND r5.a4=c4.a5 AND r5.a5=c5.a5;

EXPLAIN ANALYZE EXECUTE q1;

DEALLOCATE q1;
```

Output Messages Explain X Notifications

Showing rows: 1

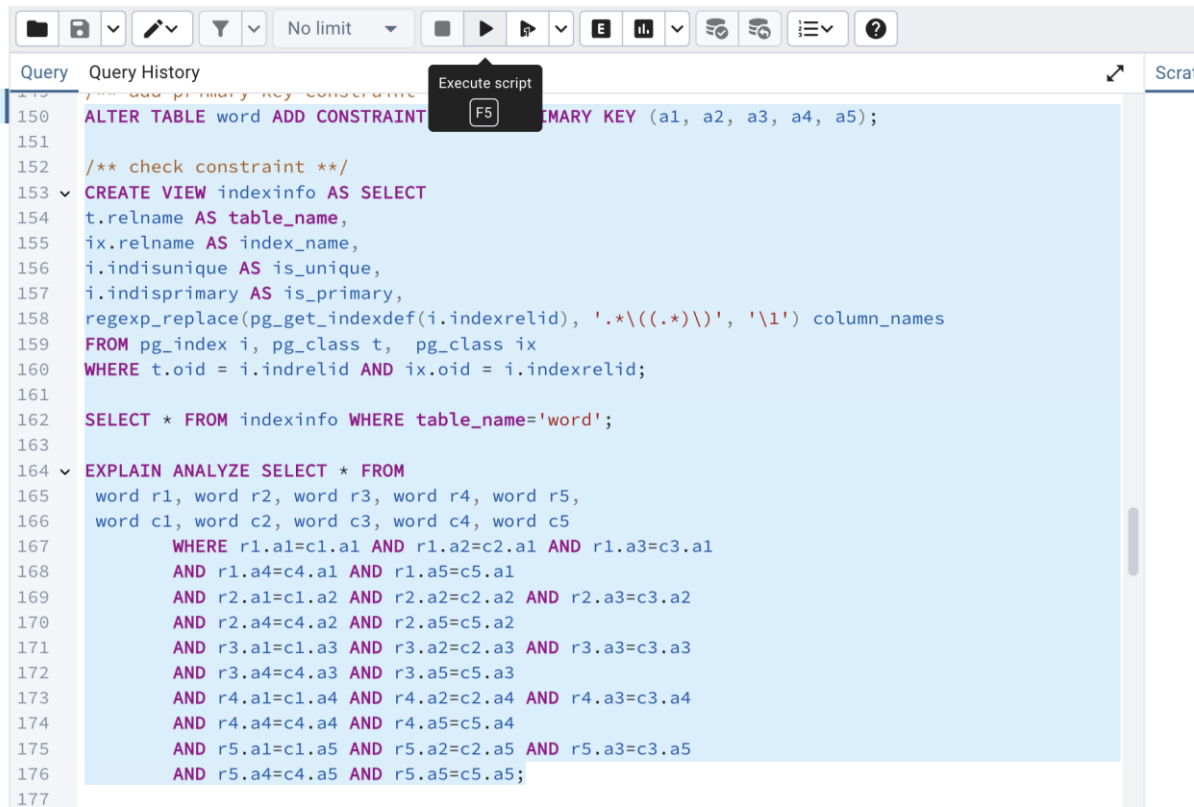
QUERY PLAN
text
-> Index Only Scan using alls on word c2 (cost=0.14..0.25 rows=1 width=10) (actual time=0.001..0.0...
Index Cond: ((a1 = r1.a2) AND (a2 = r2.a2))
Heap Fetches: 42
-> Index Only Scan using alls on word r5 (cost=0.14..0.25 rows=1 width=10) (actual time=0.001..0.001 r...
Index Cond: ((a1 = c1.a5) AND (a2 = c2.a5))
Heap Fetches: 5
Planning Time: 3986.730 ms
Execution Time: 1.138 ms

# Crossword

**Task 2: Try out several possible methods to improve efficiency**

*Create indexes and see whether they improve the performance.*

We try a primary key index first.



```
150 ALTER TABLE word ADD CONSTRAINT PRIMARY KEY (a1, a2, a3, a4, a5);
151
152 /** check constraint **/
153 CREATE VIEW indexinfo AS SELECT
154 t.relname AS table_name,
155 ix.relname AS index_name,
156 i.indisunique AS is_unique,
157 i.indisprimary AS is_primary,
158 regexp_replace(pg_get_indexdef(i.indexrelid), '.*\((.*)\)', '\1') column_names
159 FROM pg_index i, pg_class t, pg_class ix
160 WHERE t.oid = i.indrelid AND ix.oid = i.indexrelid;
161
162 SELECT * FROM indexinfo WHERE table_name='word';
163
164 EXPLAIN ANALYZE SELECT * FROM
165 word r1, word r2, word r3, word r4, word r5,
166 word c1, word c2, word c3, word c4, word c5
167 WHERE r1.a1=c1.a1 AND r1.a2=c2.a1 AND r1.a3=c3.a1
168 AND r1.a4=c4.a1 AND r1.a5=c5.a1
169 AND r2.a1=c1.a2 AND r2.a2=c2.a2 AND r2.a3=c3.a2
170 AND r2.a4=c4.a2 AND r2.a5=c5.a2
171 AND r3.a1=c1.a3 AND r3.a2=c2.a3 AND r3.a3=c3.a3
172 AND r3.a4=c4.a3 AND r3.a5=c5.a3
173 AND r4.a1=c1.a4 AND r4.a2=c2.a4 AND r4.a3=c3.a4
174 AND r4.a4=c4.a4 AND r4.a5=c5.a4
175 AND r5.a1=c1.a5 AND r5.a2=c2.a5 AND r5.a3=c3.a5
176 AND r5.a4=c4.a5 AND r5.a5=c5.a5;
177
```

**QUERY PLAN**

**"Nested Loop**

**( cost =209.70 .. 1869.69 rows=1 width=100)**

**( actual time =6.307. .43.911 rows=72**

**loops=1)"**

**. . . [ 62 lines ]**

**"Planning Time : 10339.931 ms"**

**"Execution Time : 44.285 ms"**

**The planning is still very slow.**

# Crossword

## ***Task 2: Try out several possible methods to improve efficiency***

*Create indexes and see whether they improve the performance.*

Alternatively, it is possible to add a compound index on all columns .

-- Drop primary key index first

```
ALTER TABLE word DROP CONSTRAINT pkword;
```

```
CREATE INDEX alls ON word (a1, a2, a3, a4, a5);
```

-- Run this query again

```
EXPLAIN ANALYZE SELECT * FROM
word r1, word r2, word r3, word r4, word r5,
word c1, word c2, word c3, word c4, word c5
  WHERE r1.a1=c1.a1 AND r1.a2=c2.a1 AND r1.a3=c3.a1
  AND r1.a4=c4.a1 AND r1.a5=c5.a1
  AND r2.a1=c1.a2 AND r2.a2=c2.a2 AND r2.a3=c3.a2
  AND r2.a4=c4.a2 AND r2.a5=c5.a2
  AND r3.a1=c1.a3 AND r3.a2=c2.a3 AND r3.a3=c3.a3
  AND r3.a4=c4.a3 AND r3.a5=c5.a3
  AND r4.a1=c1.a4 AND r4.a2=c2.a4 AND r4.a3=c3.a4
  AND r4.a4=c4.a4 AND r4.a5=c5.a4
  AND r5.a1=c1.a5 AND r5.a2=c2.a5 AND r5.a3=c3.a5
  AND r5.a4=c4.a5 AND r5.a5=c5.a5;
```

```
DROP INDEX a11;
```

## **Task 2.1**

***Compare the results with previous methods.  
Is this method better or similar?***

# Crossword

**Task 2: Try out several possible methods to improve efficiency**

*We can try out the B+ Tree and Hash indexes.*

```
4 CREATE INDEX bi1 ON word USING BTREE (a1);
5 CREATE INDEX bi2 ON word USING BTREE (a2);
6 CREATE INDEX bi3 ON word USING BTREE (a3);
7 CREATE INDEX bi4 ON word USING BTREE (a4);
8 CREATE INDEX bi5 ON word USING BTREE (a5);
9
0 EXPLAIN ANALYZE SELECT * FROM
1 word r1, word r2, word r3, word r4, word r5,
2 word c1, word c2, word c3, word c4, word c5
3 WHERE r1.a1=c1.a1 AND r1.a2=c2.a1 AND r1.a3=c3.a1
4 AND r1.a4=c4.a1 AND r1.a5=c5.a1
5 AND r2.a1=c1.a2 AND r2.a2=c2.a2 AND r2.a3=c3.a2
6 AND r2.a4=c4.a2 AND r2.a5=c5.a2
7 AND r3.a1=c1.a3 AND r3.a2=c2.a3 AND r3.a3=c3.a3
8 AND r3.a4=c4.a3 AND r3.a5=c5.a3
9 AND r4.a1=c1.a4 AND r4.a2=c2.a4 AND r4.a3=c3.a4
0 AND r4.a4=c4.a4 AND r4.a5=c5.a4
1 AND r5.a1=c1.a5 AND r5.a2=c2.a5 AND r5.a3=c3.a5
2 AND r5.a4=c4.a5 AND r5.a5=c5.a5;
```

Output Messages Explain X Notifications

QUERY PLAN
text
Heap Fetches: 2361
-> Index Only Scan using alls on word c2 (cost=0.14..0.25 rows=1 width=10) (actual time=0.001..0.001 rows=5 l...
Index Cond: ((a1 = r1.a2) AND (a2 = r2.a2))
Heap Fetches: 989
Planning Time: 5255.290 ms
Execution Time: 24.656 ms

```
CREATE INDEX hi1 ON word USING HASH (a1);
CREATE INDEX hi2 ON word USING HASH (a2);
CREATE INDEX hi3 ON word USING HASH (a3);
CREATE INDEX hi4 ON word USING HASH (a4);
CREATE INDEX hi5 ON word USING HASH (a5);
EXPLAIN ANALYZE SELECT * FROM
word r1, word r2, word r3, word r4, word r5,
word c1, word c2, word c3, word c4, word c5
WHERE r1.a1=c1.a1 AND r1.a2=c2.a1 AND r1.a3=c3.a1
AND r1.a4=c4.a1 AND r1.a5=c5.a1
AND r2.a1=c1.a2 AND r2.a2=c2.a2 AND r2.a3=c3.a2
AND r2.a4=c4.a2 AND r2.a5=c5.a2
AND r3.a1=c1.a3 AND r3.a2=c2.a3 AND r3.a3=c3.a3
AND r3.a4=c4.a3 AND r3.a5=c5.a3
AND r4.a1=c1.a4 AND r4.a2=c2.a4 AND r4.a3=c3.a4
AND r4.a4=c4.a4 AND r4.a5=c5.a4
AND r5.a1=c1.a5 AND r5.a2=c2.a5 AND r5.a3=c3.a5
AND r5.a4=c4.a5 AND r5.a5=c5.a5;
```

Output Messages Explain X Notifications

QUERY PLAN
text
-> Index Scan using hi4 on word r2 (cost=0.00..0.25 rows=6 width=10) (actual time=0.000..0.001 rows=15 lo...
Index Cond: (a4 = c4.a2)
-> Index Only Scan using alls on word c2 (cost=0.14..0.25 rows=1 width=10) (actual time=0.001..0.001 rows=5 l...
Index Cond: ((a1 = r1.a2) AND (a2 = r2.a2))
Heap Fetches: 989
Planning Time: 4158.621 ms
Execution Time: 26.220 ms

***And find there is no gain!***

# Crossword

**Create a table of letters, add foreign key constraints referencing this table and see the consequences on insertions, deletions and updates.**

```
CREATE TABLE alpha (letter CHAR(1) PRIMARY KEY);

INSERT INTO alpha (
  select a1 from word
  union select a2 from word
  union select a3 from word
  union select a4 from word
  union select a5 from word);

CREATE TABLE word2 (
  a1 CHAR(1) REFERENCES alpha(letter),
  a2 CHAR(1) REFERENCES alpha(letter),
  a3 CHAR(1) REFERENCES alpha(letter),
  a4 CHAR(1) REFERENCES alpha(letter),
  a5 CHAR(1) REFERENCES alpha(letter)
);

EXPLAIN ANALYZE INSERT INTO word2 SELECT * FROM word;
```

Output Messages Explain X Notifications

Showing r

QUERY PLAN
text
insert on word2 (cost=0.00..2.34 rows=0 width=0) (actual time=0.366..0.367 rows=0 loops=1)
-> Seq Scan on word (cost=0.00..2.34 rows=134 width=10) (actual time=0.009..0.021 rows=134 loop...)
Planning Time: 0.079 ms
Trigger for constraint word2_a1_fkey: time=1.270 calls=134
Trigger for constraint word2_a2_fkey: time=1.176 calls=134
Trigger for constraint word2_a3_fkey: time=0.960 calls=134
Trigger for constraint word2_a4_fkey: time=0.939 calls=134
Trigger for constraint word2_a5_fkey: time=0.957 calls=134
Execution Time: 5.749 ms

We notice that insertion (and updates) will now require to check the constraints (using triggers) and will be time consuming. Most of the time is spent in the triggers.

# Crossword

**Create a table of letters, add foreign key constraints referencing this table and see the consequences on insertions, deletions and updates.**

The same thing happens for deletions.

```
INSERT INTO alpha VALUES ('J');  
EXPLAIN ANALYZE DELETE FROM alpha WHERE letter = 'J';
```

Output Messages Explain X Notifications

Showing rows: 1 to 10

QUERY PLAN
text
-> Index Scan using alpha_pkey on alpha (cost=0.15..8.17 rows=1 width=6) (actual time=0.010..0.011 rows=1 loop...)
Index Cond: (letter = 'J'::bpchar)
Planning Time: 0.050 ms
Trigger for constraint word2_a1_fkey: time=0.297 calls=1
Trigger for constraint word2_a2_fkey: time=0.087 calls=1
Trigger for constraint word2_a3_fkey: time=0.076 calls=1
Trigger for constraint word2_a4_fkey: time=0.072 calls=1
Trigger for constraint word2_a5_fkey: time=0.071 calls=1
Execution Time: 0.636 ms

Notice the Index Scan on the primary key of alpha.

Delete on alpha (cost=0.15..8.17 rows=0 width=0) (actual time=0.022..0.022 rows=0 loops=1)
-> Index Scan using alpha_pkey on alpha (cost=0.15..8.17 rows=1 width=6) (actual time=0.010..0.011 rows=1 loop...)
Index Cond: (letter = 'J'::bpchar)

# Submission

**Create** a document and submit the following items:

**Task 1.1** Print the query plan, planning time, and execution time for the two queries in the previous slides. Compare the results. (Page 16)

**Task 1.2** Using the Explain (F7) option of pgAdmin 4, print the two queries' query plans in JSON, save the tree representation of the query plan in SVG, read and save the analysis and statistics (Page 17)

**Task 2.1** Compare the results of adding compound index with methods of adding primary key index (by print and analysis the statistics). Is this method better or similar?