

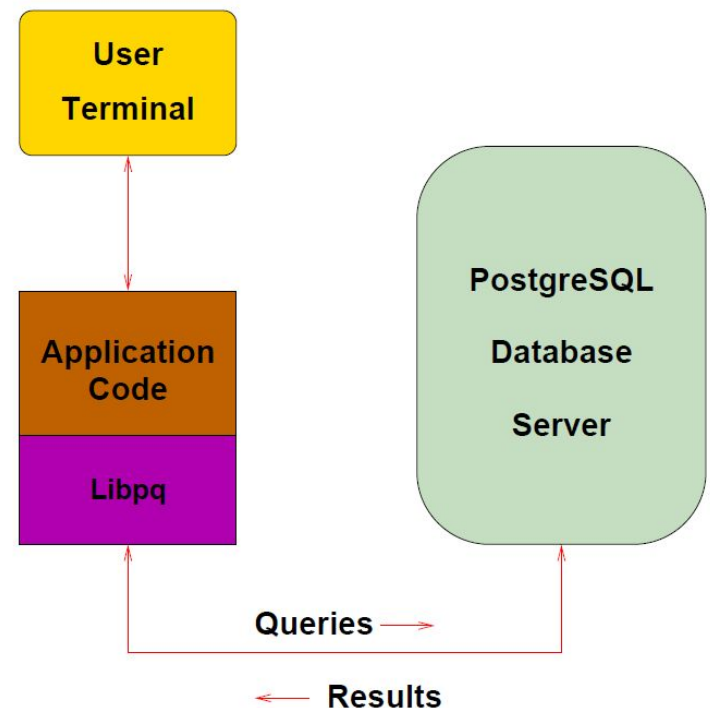
Επεξεργασία και βελτιστοποίηση αιτημάτων στην PostgreSQL

Φροντιστήριο στο μάθημα
«Βάσεις Δεδομένων»

Βάσεις Δεδομένων
Πολυτεχνείο Κρήτης

Περιεχόμενα

- Ο Βελτιστοποιητής αποτελεί τον «εγκέφαλο» ενός ΣΔΒΔ δεδομένων, ερμηνεύει τις ερωτήσεις SQL και καθορίζει την ταχύτερη μέθοδο εκτέλεσής τους.
- Παρουσίαση της χρήσης των εντολών EXPLAIN και EXPLAIN ANALYSE για ναδειχθεί πώς ο βελτιστοποιητής ερμηνεύει τις ερωτήσεις SQL και καθορίζει την ταχύτερη μέθοδο εκτέλεσής τους.



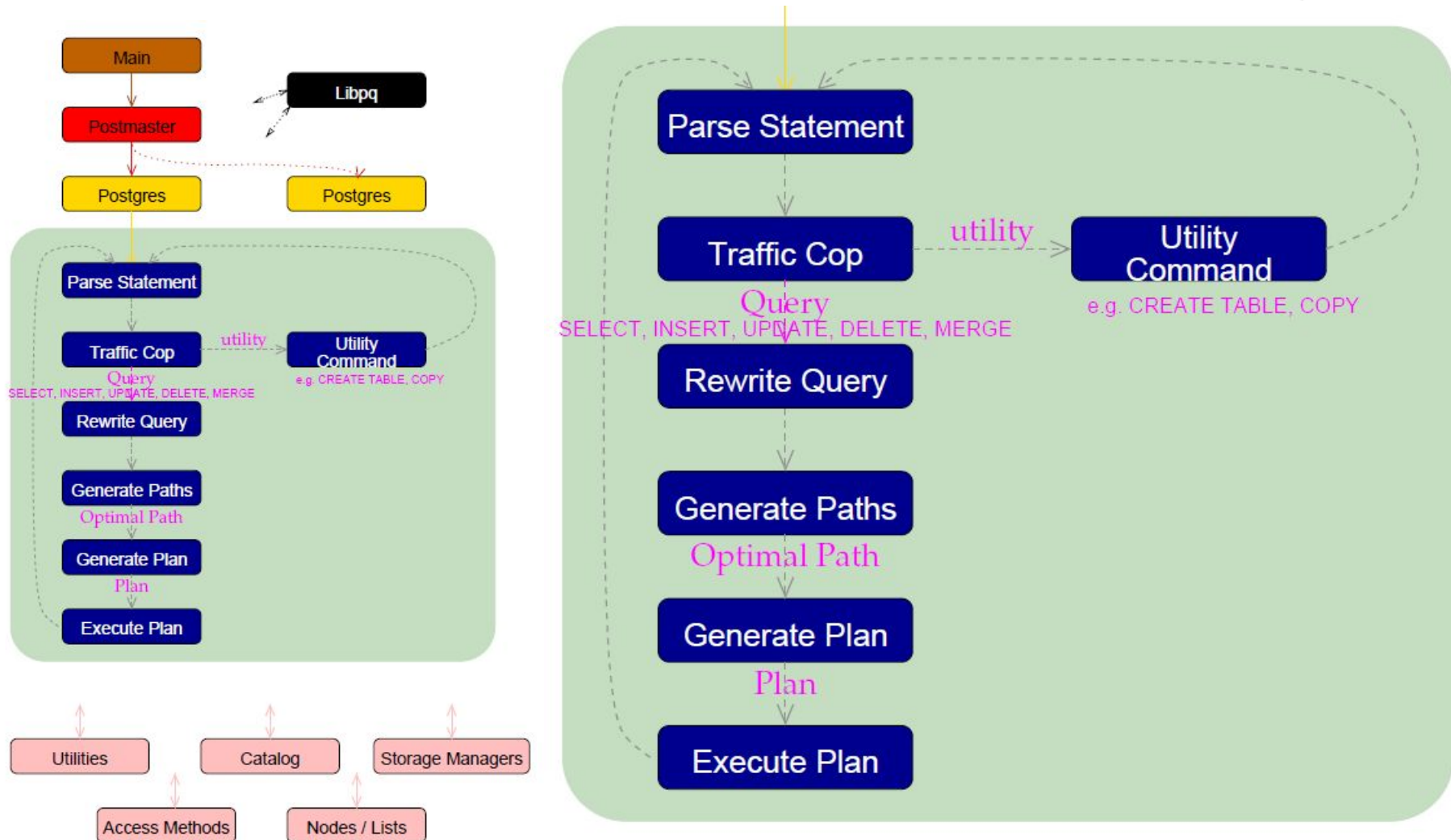
Εναλλακτικά σχέδια

- Υπάρχουν εναλλακτικοί τρόποι επεξεργασίας κάθε αιτήματος με διαφορετικά κόστη
- Η επιλογή του τρόπου εκτέλεσης (σχέδιο - plan) με το μικρότερο κόστος είναι ένα δύσκολο πρόβλημα
- Ο αριθμός των εναλλακτικών τρόπων αυξάνει όσο αυξάνουν οι τελεστές (επί μέρους λειτουργίες) που χρησιμοποιούνται σε ένα ερώτημα
 - Μια ερώτηση είναι ουσιαστικά μια αλγεβρική παράσταση (δέντρο) που περιλαμβάνει τους σχεσιακούς τελεστές

Η λειτουργία του βελτιστοποιητή ερωτήσεων

- Αρχικά γίνεται συντακτική ανάλυση της ερώτησης
- Στη συνέχεια ο βελτιστοποιητής επιχειρεί να προσδιορίσει ένα αποδοτικό σχέδιο επεξεργασίας
 - Παραγωγή **εναλλακτικών σχεδίων**
 - **Εκτίμηση κόστους** χρησιμοποιώντας πληροφορία από τους **καταλόγους** του συστήματος
 - Επιλογή του σχεδίου που έχει το μικρότερο αναμενόμενο κόστος

Η εκτέλεση ερωτήσεων στην Postgres



Κατάλογος συστήματος

- Καταχώρηση πληροφοριών για κάθε πίνακα και κάθε ευρετήριο καθώς και για τις όψεις, τις συναρτήσεις κλ.π.
- Συλλογή πινάκων που ονομάζονται **πίνακες του καταλόγου ή κατάλογος του συστήματος ή κατάλογος ή λεξικό δεδομένων ή μεταδεδομένα**
- Ο βελτιστοποιητής χρησιμοποιεί τον κατάλογο για να υπολογίζει το κόστος των εναλλακτικών σχεδίων καθώς και το μέγεθος των ενδιάμεσων αποτελεσμάτων.

Ο κατάλογος ως συλλογή πινάκων

- Επιτρέπεται η διαχείριση του όπως και στους πίνακες μιας βάσης που ορίζει ο χρήστης
- Ωστόσο το ΣΔΒΔ χειρίζεται με ειδικό τρόπο τους καταλόγους γνωρίζοντας το σχήμα τους.

Κατάλογος στην PostgreSQL

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays the 'PostgreSQL Catalog (pg_catalog)' under the 'Catalogs (2)' section. The 'Tables' and 'Views' items are circled in red. The main pane shows the 'Properties' tab for the 'PostgreSQL Catalog (pg_catalog)', displaying a list of system tables.

Name	Owner	Partitioned Table?
pg_aggregate	postgres	False
pg_am	postgres	False
pg_amop	postgres	False
pg_amproc	postgres	False
pg_attrdef	postgres	False
pg_attribute	postgres	False
pg_auth_members	postgres	False
pg_authid	postgres	False
pg_cast	postgres	False
pg_class	postgres	False
pg_collation	postgres	False
pg_constraint	postgres	False
pg_conversion	postgres	False
pg_database	postgres	False
pg_db_role_setting	postgres	False
pg_default_acl	postgres	False
pg_depend	postgres	False
pg_description	postgres	False
pg_enum	postgres	False
pg_event_trigger	postgres	False
pg_extension	postgres	False
pg_foreign_data_wrapper	postgres	False
pg_foreign_server	postgres	False
pg_foreign_table	postgres	False
pg_index	postgres	False
pg_inherits	postgres	False

Η βάση για τα παραδείγματα που θα χρησιμοποιήσουμε

CITY
(*) id: integer (PK) name: text countrycode: character(3) (FK -> COUNTRY) district: text population: integer

COUNTRYLANGUAGE
(*) countrycode character(3) (FK -> COUNTRY) (*) language text isofficial boolean percentage real

COUNTRY
(*) code character(3) name text continent text region text surfacearea real indepyear smallint population integer lifeexpectancy real gnp numeric(10,2) gnpold numeric(10,2) localname text governmentform text headofstate text capital integer (FK -> CITY) code2 character

Πληροφορίες καταλόγου

pgAdmin 4

File Object Tools Help

Browser

- Servers (5)
 - PostgreSQL 11
 - PostgreSQL 13
 - Databases (5)
 - db_2022_2023_A_with_e
 - lab_8
 - old_project
 - postgres
 - tutorial_5
 - Casts
 - Catalogs
 - Event Triggers
 - Extensions
 - Foreign Data Wrapper
 - Languages
 - Publications
 - Schemas
 - Subscriptions
 - Login/Group Roles
 - Tablespaces
 - isc_server
 - stsp_vm_ubuntu
 - stsp_vm_ubuntu_sec

Dashboard Properties SQL tutorial_5/postgres@PostgreSQL 13*

tutorial_5/postgres@PostgreSQL 13

No limit

Query Query History

```
1 SELECT relname,relkind,reltuples,relpages
2 FROM pg_class
3 WHERE relname LIKE 'country%' OR relname LIKE 'city%';
```

Data Output Messages Notifications

	relname name	relkind "char" (1)	reltuples real	relpages integer
1	city	r	4079	32
2	country	r	239	5
3	countrylanguage	r	984	6
4	city_pkey	i	4079	14
5	country_pkey	i	239	2
6	countrylanguage_pkey	i	984	6

Total rows: 6 of 6 Query complete 00:00:00.045 Ln 3, Col 55

Σχέδια υπολογισμού στην PostgreSQL

- Η εντολή EXPLAIN μας δείχνει το σχέδιο εκτέλεσης μιας ερώτησης
- Η εντολή EXPLAIN ANALYSE παρουσιάζει αναλυτικά τους πραγματικούς χρόνους εκτέλεσης μιας ερώτησης σε αντιπαραβολή με τις αρχικές εκτιμήσεις κόστους του σχεδίου που εφαρμόστηκε.
- Μελετώντας τα σχέδια και τους χρόνους εκτέλεσης μπορούμε να καταλήξουμε σε αποφάσεις που αφορούν το φυσικό σχεδιασμό της βάσης και βελτιώνουν την απόδοση κρίσιμων αιτημάτων.
- Με τη βοήθεια του καταλόγου μπορούμε να εκτιμήσουμε την επιλεκτικότητα (selectivity) κάθε λογικής συνθήκης και να επιλέξουμε το/τα καλύτερο/-α από τα υποψήφια ευρετήρια.
- Μπορούμε ακόμη να δούμε αν η σειρά εκτέλεσης των συνδέσεων είναι βέλτιστη. Αν δεν είναι και θέλουμε, να επιβάλουμε μια ορισμένη σειρά εκτέλεσης των συνδέσεων.
- Αν δεν χρησιμοποιούνται ευρετήρια που υπάρχουν ίσως οφείλεται σε παρωχημένη στατιστική πληροφορία.
 - Με την εντολή VACUUM ANALYSE μπορούμε να ζητήσουμε ρητά την ανανέωση της στατιστικής πληροφορίας για όλη τη βάση ή για κάποιο πίνακα ώστε να βοηθήσουμε το βελτιστοποιητή

Η εντολή EXPLAIN

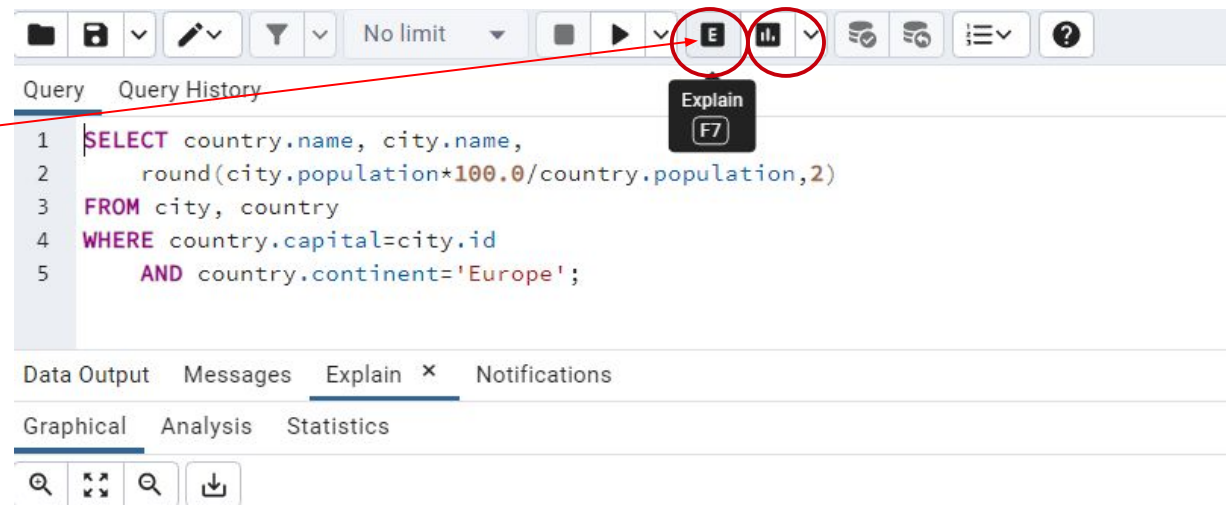
```
1 EXPLAIN SELECT country.name, city.name,  
2     round(city.population*100.0/country.population,2)  
3 FROM city, country  
4 WHERE country.capital=city.id  
5     AND country.continent='Europe';
```

QUERY PLAN	
text	
1	Hash Join (cost=8.56..97.67 rows=46 width=52)
2	Hash Cond: (city.id = country.capital)
3	-> Seq Scan on city (cost=0.00..72.79 rows=4079 width=17)
4	-> Hash (cost=7.99..7.99 rows=46 width=19)
5	-> Seq Scan on country (cost=0.00..7.99 rows=46 width=19)
6	Filter: (continent = 'Europe'::text)

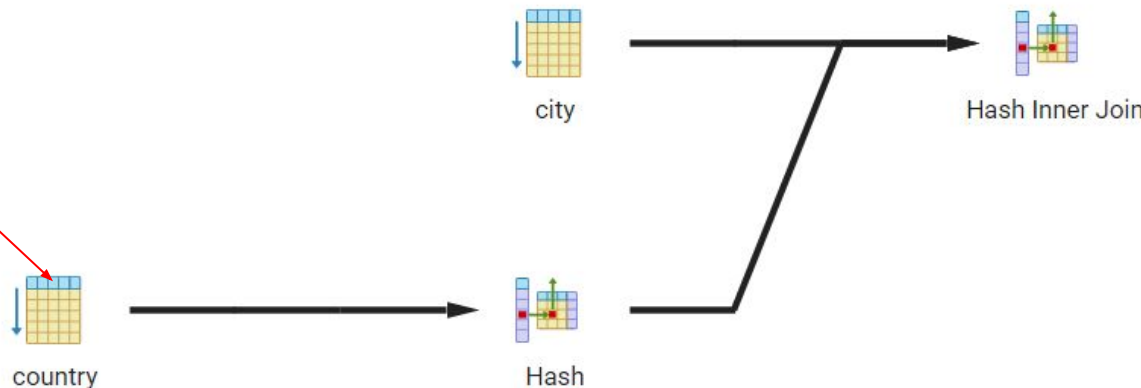
- Για κάθε βήμα (κόμβο) του σχεδίου δίνονται **εκτιμήσεις** για:
- Χρόνο υπολογισμού πρώτης πλειάδας
 - Χρόνο ολοκλήρωσης υπολογισμού
 - Πλήθος πλειάδων αποτελέσματος
 - Μέγεθος κάθε πλειάδας του αποτελέσματος

Γραφική απεικόνιση σχεδίων

Επιλέγουμε “Explain”
ή “Explain Analyse”
(εναλλακτικά πατάμε
F7 ή Shift-F7
αντίστοιχα)



Πατώντας με το ποντίκι
πάνω σε ένα κόμβο,
βλέπουμε αναλυτικές
πληροφορίες



Η εντολή EXPLAIN ANALYSE

```
1 EXPLAIN ANALYSE SELECT country.name, city.name,  
2     round(city.population*100.0/country.population,2)  
3 FROM city, country  
4 WHERE country.capital=city.id  
5     AND country.continent='Europe';
```

	QUERY PLAN text	
1	Hash Join (cost=8.56..97.67 rows=46 width=52) (actual time=0.083..0.581 rows=46 loops=1)	
2	Hash Cond: (city.id = country.capital)	
3	-> Seq Scan on city (cost=0.00..72.79 rows=4079 width=17) (actual time=0.010..0.193 rows=4079 loops=1)	
4	-> Hash (cost=7.99..7.99 rows=46 width=19) (actual time=0.041..0.041 rows=46 loops=1)	
5	Buckets: 1024 Batches: 1 Memory Usage: 11kB	
6	-> Seq Scan on country (cost=0.00..7.99 rows=46 width=19) (actual time=0.008..0.034 rows=46 loops=1)	
7	Filter: (continent = 'Europe'::text)	
8	Rows Removed by Filter: 193	
9	Planning Time: 0.127 ms	
10	Execution Time: 0.599 ms	

Παρατηρήσεις για τους χρόνους

- Οι χρόνοι είναι σε msec ενώ οι εκτιμήσεις κόστους είναι σε αφηρημένες μονάδες με αποτέλεσμα να μην έχει νόημα η αριθμητική σύγκρισή τους.
- Αυτό που έχει νόημα είναι η σύγκριση του αναμενόμενου μεγέθους του αποτελέσματος σε σχέση με το πραγματικό.
- Το loops μας δείχνει πόσες φορές εκτελείται ένας κόμβος καθώς υπάρχουν περιπτώσεις (π.χ. Nested loop join) επαναλαμβανόμενης εκτέλεσης ενός κόμβου. Τα πραγματικά κόστη σε αυτή την περίπτωση είναι μέσοι όροι.

Η εντολή EXPLAIN ANALYSE

```
1 EXPLAIN ANALYSE SELECT country.name, city.name,  
2     round(city.population*100.0/country.population,2)  
3 FROM city, country  
4 WHERE country.capital=city.id  
5     AND country.continent='Europe'  
6 ORDER BY city.name;
```

QUERY PLAN

text



1	Sort (cost=98.94..99.06 rows=46 width=52) (actual time=0.571..0.573 rows=46 loops=1)
2	Sort Key: city.name
3	Sort Method: quicksort Memory: 28kB
4	-> Hash Join (cost=8.56..97.67 rows=46 width=52) (actual time=0.065..0.501 rows=46 loops=1)
5	Hash Cond: (city.id = country.capital)
6	-> Seq Scan on city (cost=0.00..72.79 rows=4079 width=17) (actual time=0.010..0.189 rows=4079 loops=1)
7	-> Hash (cost=7.99..7.99 rows=46 width=19) (actual time=0.043..0.043 rows=46 loops=1)
8	Buckets: 1024 Batches: 1 Memory Usage: 11kB
9	-> Seq Scan on country (cost=0.00..7.99 rows=46 width=19) (actual time=0.009..0.036 rows=46 loops=1)
10	Filter: (continent = 'Europe'::text)
11	Rows Removed by Filter: 193
12	Planning Time: 0.173 ms
13	Execution Time: 0.590 ms

Επιπλέον πληροφορίες

- Οι κόμβοι ταξινόμησης (Sort) δείχνουν επίσης τη μέθοδο που χρησιμοποιήθηκε και το μέγεθος της μνήμης (RAM ή δίσκου) που απαιτήθηκε.
- Οι κόμβοι κατακερματισμού (Hash) δείχνουν πόσοι κάδοι (buckets) και πόσες δέσμες (batches) απαιτήθηκαν καθώς και το μέγιστο μέγεθος κύριας μνήμης που απαιτήθηκε.
 - Αν οι δέσμες είναι περισσότερες από μία, χρησιμοποιήθηκε και χώρος στο δίσκο αλλά αυτό δεν δείχνεται.

```

1 EXPLAIN (ANALYSE,BUFFERS) SELECT country.name, city.name,
2   round(city.population*100.0/country.population,2)
3 FROM city, country
4 WHERE country.capital=city.id
5   AND country.continent='Europe'
6 ORDER BY city.name;

```

Data Output Messages Explain × Notifications

QUERY PLAN		
	text	
1	Sort (cost=98.94..99.06 rows=46 width=52) (actual time=0.555..0.557 rows=46 loops=1)	
2	Sort Key: city.name	
3	Sort Method: quicksort Memory: 28kB	
4	Buffers: shared hit=37	
5	-> Hash Join (cost=8.56..97.67 rows=46 width=52) (actual time=0.062..0.486 rows=46 loops=1)	
6	Hash Cond: (city.id = country.capital)	
7	Buffers: shared hit=37	
8	-> Seq Scan on city (cost=0.00..72.79 rows=4079 width=17) (actual time=0.011..0.182 rows=4079 loops=1)	
9	Buffers: shared hit=32	
10	-> Hash (cost=7.99..7.99 rows=46 width=19) (actual time=0.043..0.043 rows=46 loops=1)	
11	Buckets: 1024 Batches: 1 Memory Usage: 11kB	
12	Buffers: shared hit=5	
13	-> Seq Scan on country (cost=0.00..7.99 rows=46 width=19) (actual time=0.009..0.033 rows=46 loops=1)	
14	Filter: (continent = 'Europe'::text)	
15	Rows Removed by Filter: 193	
16	Buffers: shared hit=5	
17	Planning:	
18	Buffers: shared hit=6	
19	Planning Time: 0.170 ms	
20	Execution Time: 0.579 ms	
Total rows: 20 of 20		Query complete 00:00:00.045

Η επιλογή BUFFERS

Βοηθά στον εντοπισμό των σημείων που έχουν τα μεγαλύτερα κόστη Εισόδου/Εξόδου

Τι αποφάσεις χρειάζεται να πάρει
ο βελτιστοποιητής;

- Μέθοδο σάρωσης (scan method)
- Μέθοδο σύνδεσης (join method)
- Σειρά σύνδεσης (join order)

Τι αποφάσεις χρειάζεται να πάρει ο βελτιστοποιητής;

- Μέθοδοι σάρωσης στην Postgres:

- Sequential Scan
- Bitmap Index Scan
- Index Scan

- Μέθοδοι σύνδεσης στην Postgres:

- Nested Loop
 - With Inner Sequential Scan
 - With Inner Index Scan
- Hash Join
- Merge Join

Ένα παράδειγμα για την επιλογή μεθόδου σάρωσης στην Postgres

Βασισμένο στην παρουσίαση «*Explaining the Postgres Query Optimizer*» του Bruce Momjian,
<https://momjian.us/presentations>

- Δημιουργία ενός temporary table με ένα index

```
Query  Query History
1  CREATE TEMPORARY TABLE sample (form, junk) AS
2      SELECT governmentform, repeat('x', 250)
3      FROM country
4      ORDER BY random(); -- add rows in random order
5
6  CREATE INDEX i_sample on sample (form);
```

- Δημιουργία μιας EXPLAIN function

```
8  CREATE OR REPLACE FUNCTION lookup_form(text) RETURNS SETOF
9  text AS $$
10 BEGIN
11 RETURN QUERY EXECUTE '
12     EXPLAIN SELECT form
13     FROM sample
14     WHERE form = ''' || $1 || ''';
15 END
```

Ένα παράδειγμα για την επιλογή μεθόδου σάρωσης στην Postgres

- Εύρεση της κατανομής των δεδομένων στο *sample* table

```
18 WITH forms (form, count) AS (  
19     SELECT form, COUNT(*)  
20     FROM sample  
21     GROUP BY 1  
22 )  
23 SELECT form, count, (count * 100.0 / (SUM(count) OVER  
24 ()))::numeric(4,1) AS "%"  
25 FROM forms  
26 ORDER BY 2 DESC;
```

Ένα παράδειγμα για την επιλογή μεθόδου σάρωσης στην Postgres

- Εύρεση της κατανομής των δεδομένων στο *sample* table

	form text	count bigint	% numeric (4,1)
1	Republic	122	51.0
2	Constitutional Monarchy	29	12.1
3	Federal Republic	15	6.3
4	Dependent Territory of the UK	12	5.0
5	Monarchy	5	2.1
6	Overseas Department of Fra...	4	1.7
7	Territory of Australia	4	1.7
8	Constitutional Monarchy, Fed...	4	1.7
9	Nonmetropolitan Territory of ...	4	1.7
10	Socialistic Republic	3	1.3
11	Nonmetropolitan Territory of ...	3	1.3
12	US Territory	3	1.3
13	Part of Denmark	2	0.8
14	Monarchy (Sultanate)	2	0.8
15	Islamic Republic	2	0.8

...

20	Nonmetropolitan Territory of ...	2	0.8
21	Administrated by the UN	1	0.4
22	Parlementary Monarchy	1	0.4
23	Emirate Federation	1	0.4
24	Parliamentary Coprincipality	1	0.4
25	Monarchy (Emirate)	1	0.4
26	Socialistic State	1	0.4
27	Co-administrated	1	0.4
28	Independent Church State	1	0.4
29	Occupied by Marocco	1	0.4
30	Federation	1	0.4
31	Constitutional Monarchy (Em...	1	0.4
32	Autonomous Area	1	0.4
33	People'sRepublic	1	0.4
34	Islamic Emirate	1	0.4
35	Dependent Territory of the US	1	0.4

Total rows: 35 of 35 Query complete 00:00:00.064 Rows :

Ένα παράδειγμα για την επιλογή μεθόδου σάρωσης στην Postgres

- Πόσο σημαντική είναι η κατανομή των δεδομένων;

```
30 EXPLAIN SELECT form
31 FROM sample
32 WHERE form = 'Republic';
```

QUERY PLAN		
text		
1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=32)	
2	Index Cond: (form = 'Republic'::text)	

```
34 EXPLAIN SELECT form
35 FROM sample
36 WHERE form = 'Monarchy';
```

QUERY PLAN		
text		
1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=32)	
2	Index Cond: (form = 'Monarchy'::text)	

```
38 EXPLAIN SELECT form
39 FROM sample
40 WHERE form = 'Federation';
```

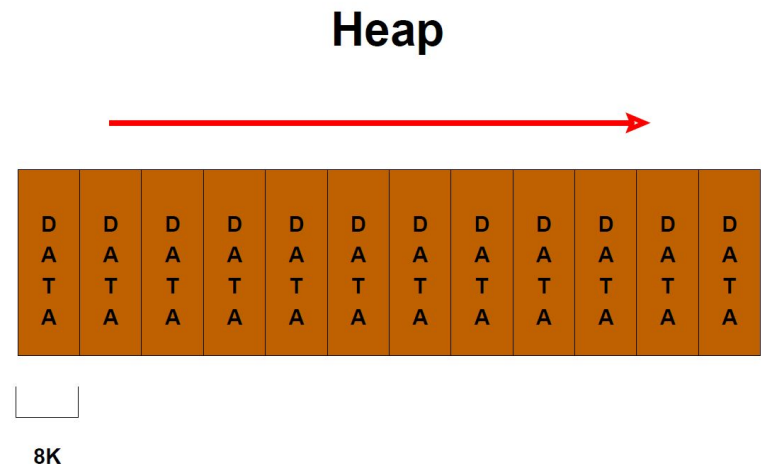
QUERY PLAN		
text		
1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=32)	
2	Index Cond: (form = 'Federation'::text)	

Ένα παράδειγμα για την επιλογή μεθόδου σάρωσης στην Postgres

- Η εκτέλεση της ANALYZE προκαλεί sequential scan για τις πιο συχνές τιμές

```
40 ANALYZE sample;  
41  
42 EXPLAIN SELECT form  
43 FROM sample  
44 WHERE form = 'Republic';
```

QUERY PLAN	
text	
1	Seq Scan on sample (cost=0.00..12.99 rows=122 width=16)
2	Filter: (form = 'Republic'::text)

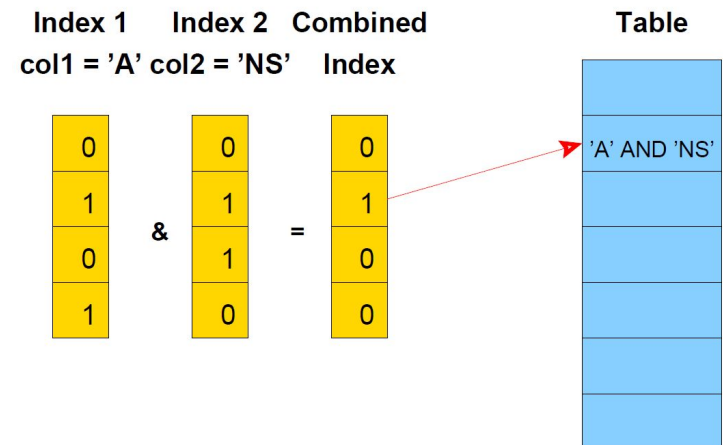


- Η ANALYZE δεν εκτελείται αυτόματα στους προσωρινούς πίνακες γιατί αυτοί είναι ορατοί μόνο στη σύνοδο που τους δημιούργησε

Ένα παράδειγμα για την επιλογή μεθόδου σάρωσης στην Postgres

- Η εκτέλεση της ANALYZE προκαλεί bitmap index scan για τις λιγότερο συχνές τιμές

46	EXPLAIN SELECT form
47	FROM sample
48	WHERE form = 'Monarchy';
	QUERY PLAN text
1	Bitmap Heap Scan on sample (cost=4.18..12.66 rows=5 width=16)
2	Recheck Cond: (form = 'Monarchy'::text)
3	-> Bitmap Index Scan on i_sample (cost=0.00..4.18 rows=5 width=0)
4	Index Cond: (form = 'Monarchy'::text)



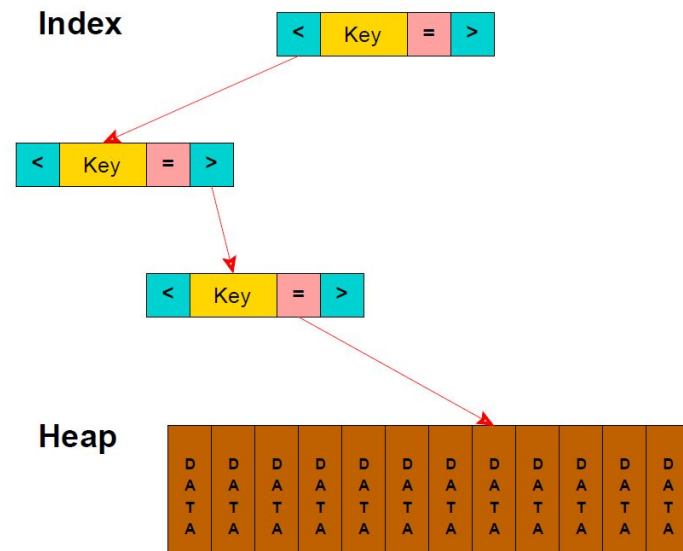
Ένα παράδειγμα για την επιλογή μεθόδου σάρωσης στην Postgres

- Η εκτέλεση της ANALYZE προκαλεί index scan για τις σπάνιες τιμές

```
50 EXPLAIN SELECT form
51 FROM sample
52 WHERE form = 'Federation';
```

	QUERY PLAN
1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=16)
2	Index Cond: (form = 'Federation'::text)

Επιλογή Index/Bitmap
Index scan όταν η
επιλεκτικότητα είναι
περίπου 5-10%



Ένα παράδειγμα για την επιλογή μεθόδου σάρωσης στην Postgres

- Όλες οι τιμές...

```
54 WITH form (form, count) AS (  
55     SELECT form, COUNT(*)  
56     FROM sample  
57     GROUP BY 1  
58 )  
59 SELECT form AS l, count, lookup_form(form)  
60 FROM form  
61 ORDER BY 2 DESC;
```

	l text	count bigint	lookup_form text
1	Republic	122	Seq Scan on sample (cost=0.00..12.99 rows=122 width=16)
2	Republic	122	Filter: (form = 'Republic'::text)
3	Constitutional Monarchy	29	Seq Scan on sample (cost=0.00..12.99 rows=29 width=16)
4	Constitutional Monarchy	29	Filter: (form = 'Constitutional Monarchy'::text)
5	Federal Republic	15	Seq Scan on sample (cost=0.00..12.99 rows=15 width=16)
6	Federal Republic	15	Filter: (form = 'Federal Republic'::text)
7	Dependent Territory of the...	12	Seq Scan on sample (cost=0.00..12.99 rows=12 width=16)
8	Dependent Territory of the...	12	Filter: (form = 'Dependent Territory of the UK'::text)
9	Monarchy	5	Bitmap Heap Scan on sample (cost=4.18..12.66 rows=5 width=16)
10	Monarchy	5	Recheck Cond: (form = 'Monarchy'::text)
11	Monarchy	5	-> Bitmap Index Scan on i_sample (cost=0.00..4.18 rows=5 width=0)
12	Monarchy	5	Index Cond: (form = 'Monarchy'::text)
13	Territory of Australia	4	Bitmap Heap Scan on sample (cost=4.18..12.64 rows=4 width=16)
14	Territory of Australia	4	Recheck Cond: (form = 'Territory of Australia'::text)
15	Territory of Australia	4	-> Bitmap Index Scan on i_sample (cost=0.00..4.17 rows=4 width=0)

Ένα παράδειγμα για την επιλογή μεθόδου σάρωσης στην Postgres

- Παίρνοντας μόνο την πρώτη γραμμή...

```
64 WITH form (form, count) AS (  
65     SELECT form, COUNT(*)  
66     FROM sample  
67     GROUP BY 1  
68 )  
69 SELECT form AS l, count,  
70     (SELECT *  
71     FROM lookup_form(form) AS l2  
72     LIMIT 1) AS lookup_form  
73 FROM form  
74 ORDER BY 2 DESC;
```

Ένα παράδειγμα για την επιλογή μεθόδου σάρωσης στην Postgres

	I text	count bigint	lookup_form text
1	Republic	122	Seq Scan on sample (cost=0.00..12.99 rows=122 width=16)
2	Constitutional Monarchy	29	Seq Scan on sample (cost=0.00..12.99 rows=29 width=16)
3	Federal Republic	15	Seq Scan on sample (cost=0.00..12.99 rows=15 width=16)
4	Dependent Territory of the UK	12	Seq Scan on sample (cost=0.00..12.99 rows=12 width=16)
5	Monarchy	5	Bitmap Heap Scan on sample (cost=4.18..12.66 rows=5 width=16)
6	Territory of Australia	4	Bitmap Heap Scan on sample (cost=4.18..12.64 rows=4 width=16)
7	Nonmetropolitan Territory of ...	4	Bitmap Heap Scan on sample (cost=4.18..12.64 rows=4 width=16)
8	Overseas Department of Fran...	4	Bitmap Heap Scan on sample (cost=4.18..12.64 rows=4 width=16)
9	Constitutional Monarchy, Fed...	4	Bitmap Heap Scan on sample (cost=4.18..12.64 rows=4 width=16)
10	US Territory	3	Bitmap Heap Scan on sample (cost=4.17..11.28 rows=3 width=16)
11	Nonmetropolitan Territory of ...	3	Bitmap Heap Scan on sample (cost=4.17..11.28 rows=3 width=16)
12	Socialistic Republic	3	Bitmap Heap Scan on sample (cost=4.17..11.28 rows=3 width=16)
13	Commonwealth of the US	2	Bitmap Heap Scan on sample (cost=4.16..9.50 rows=2 width=16)
14	Part of Denmark	2	Bitmap Heap Scan on sample (cost=4.16..9.50 rows=2 width=16)
15	Islamic Republic	2	Bitmap Heap Scan on sample (cost=4.16..9.50 rows=2 width=16)
16	Monarchy (Sultanate)	2	Bitmap Heap Scan on sample (cost=4.16..9.50 rows=2 width=16)
17	Special Administrative Regio...	2	Bitmap Heap Scan on sample (cost=4.16..9.50 rows=2 width=16)
18	Dependent Territory of Norway	2	Bitmap Heap Scan on sample (cost=4.16..9.50 rows=2 width=16)
19	Nonmetropolitan Territory of ...	2	Bitmap Heap Scan on sample (cost=4.16..9.50 rows=2 width=16)
20	Territorial Collectivity of Fran...	2	Bitmap Heap Scan on sample (cost=4.16..9.50 rows=2 width=16)

Πολύ συχνές
τιμές

Λιγότερο
συχνές τιμές

Ένα παράδειγμα για την επιλογή μεθόδου σάρωσης στην Postgres

21	Peoples Republic	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
22	Co-administrated	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
23	Parlementary Monarchy	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
24	Administrated by the UN	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
25	Federation	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
26	Emirate Federation	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
27	Constitutional Monarchy (Em...	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
28	Parliamentary Coprincipality	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
29	Autonomous Area	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
30	Monarchy (Emirate)	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
31	Socialistic State	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
32	Islamic Emirate	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
33	Independent Church State	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
34	Dependent Territory of the US	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...
35	Occupied by Marocco	1	Index Only Scan using i_sample on sample (cost=0.14..8.16 rows=1 width=...

Total rows: 35 of 35 Query complete 00:00:00.071

Σπάνιες
τιμές

Τα αποτελέσματα μπορεί να διαφέρουν ανάλογα με το clustering των τιμών στα heap pages

Ένα παράδειγμα για την επιλογή μεθόδου σάρωσης στην Postgres

- Επιβάλλοντας το index scan...

```
76 SET enable_seqscan = false;
77
78 SET enable_bitmapscan = false;
79
80 WITH form (form, count) AS (
81     SELECT form, COUNT(*)
82     FROM sample
83     GROUP BY 1
84 )
85 SELECT form AS l, count,
86     (SELECT *
87      FROM lookup_form(form) AS l2
88      LIMIT 1) AS lookup_form
89 FROM form
90 ORDER BY 2 DESC;
```


Ένα παράδειγμα για την επιλογή μεθόδου σάρωσης στην Postgres

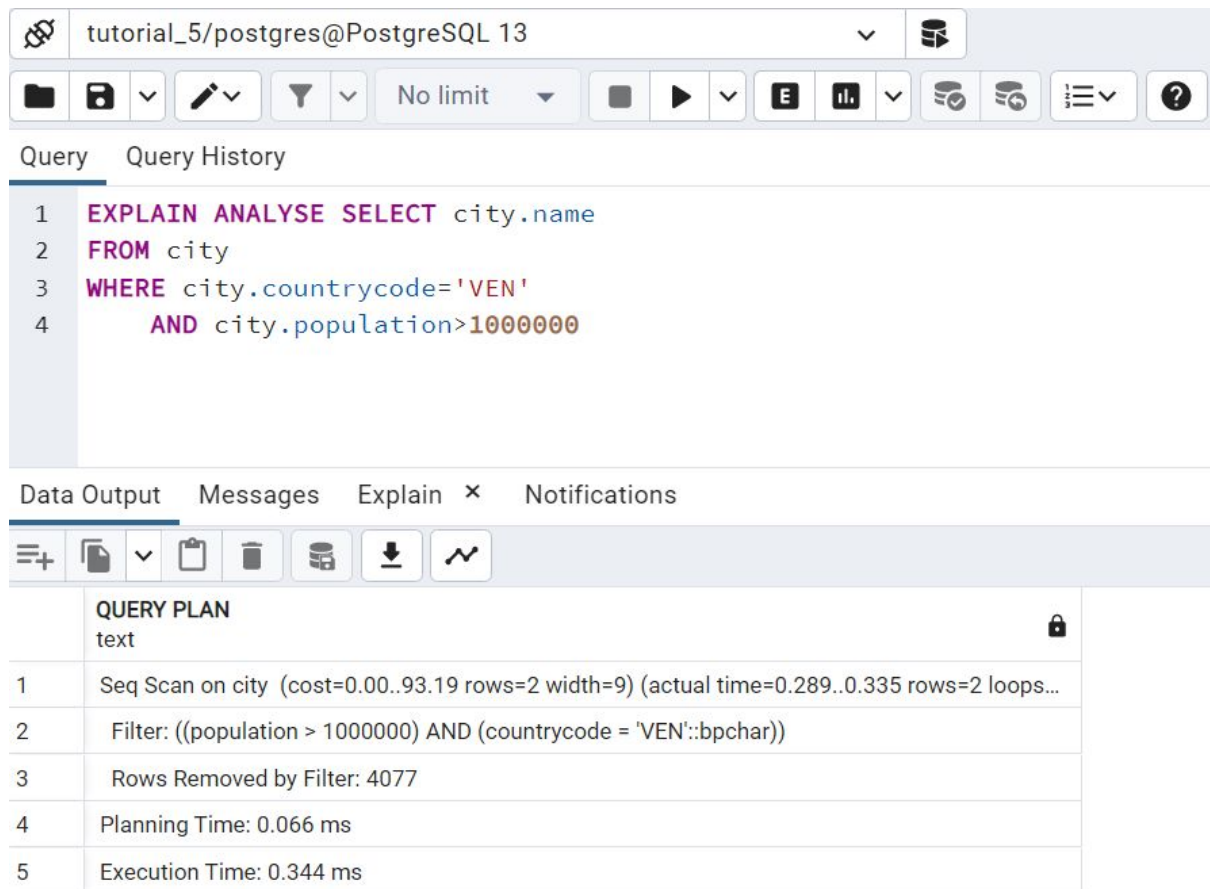
- Επιβάλλοντας το index scan...

Ήταν ~13

	i text	count bigint	lookup_form text
1	Republic	122	Index Only Scan using i_sample on sample (cost=0.14..47.58 rows=122 width=16)
2	Constitutional Monarchy	29	Index Only Scan using i_sample on sample (cost=0.14..41.60 rows=29 width=16)
3	Federal Republic	15	Index Only Scan using i_sample on sample (cost=0.14..37.62 rows=15 width=16)
4	Dependent Territory of the ...	12	Index Only Scan using i_sample on sample (cost=0.14..33.92 rows=12 width=16)
5	Monarchy	5	Index Only Scan using i_sample on sample (cost=0.14..19.19 rows=5 width=16)
6	Nonmetropolitan Territory o...	4	Index Only Scan using i_sample on sample (cost=0.14..19.17 rows=4 width=16)
7	Overseas Department of Fr...	4	Index Only Scan using i_sample on sample (cost=0.14..19.17 rows=4 width=16)
8	Constitutional Monarchy, F...	4	Index Only Scan using i_sample on sample (cost=0.14..19.17 rows=4 width=16)
9	Territory of Australia	4	Index Only Scan using i_sample on sample (cost=0.14..19.17 rows=4 width=16)
10	Socialistic Republic	3	Index Only Scan using i_sample on sample (cost=0.14..15.50 rows=3 width=16)
11	Nonmetropolitan Territory o...	3	Index Only Scan using i_sample on sample (cost=0.14..15.50 rows=3 width=16)
12	US Territory	3	Index Only Scan using i_sample on sample (cost=0.14..15.50 rows=3 width=16)
13	Territorial Collectivity of Fra...	2	Index Only Scan using i_sample on sample (cost=0.14..11.83 rows=2 width=16)
14	Monarchy (Sultanate)	2	Index Only Scan using i_sample on sample (cost=0.14..11.83 rows=2 width=16)
15	Dependent Territory of Nor...	2	Index Only Scan using i_sample on sample (cost=0.14..11.83 rows=2 width=16)

Βελτίωση απόδοσης απλών ερωτήσεων

- Ποιες είναι οι πόλεις της Βενεζουέλας με πληθυσμό άνω του 1.000.000;



The screenshot shows a PostgreSQL query editor interface. The query is as follows:

```
1 EXPLAIN ANALYSE SELECT city.name
2 FROM city
3 WHERE city.countrycode='VEN'
4     AND city.population>1000000
```

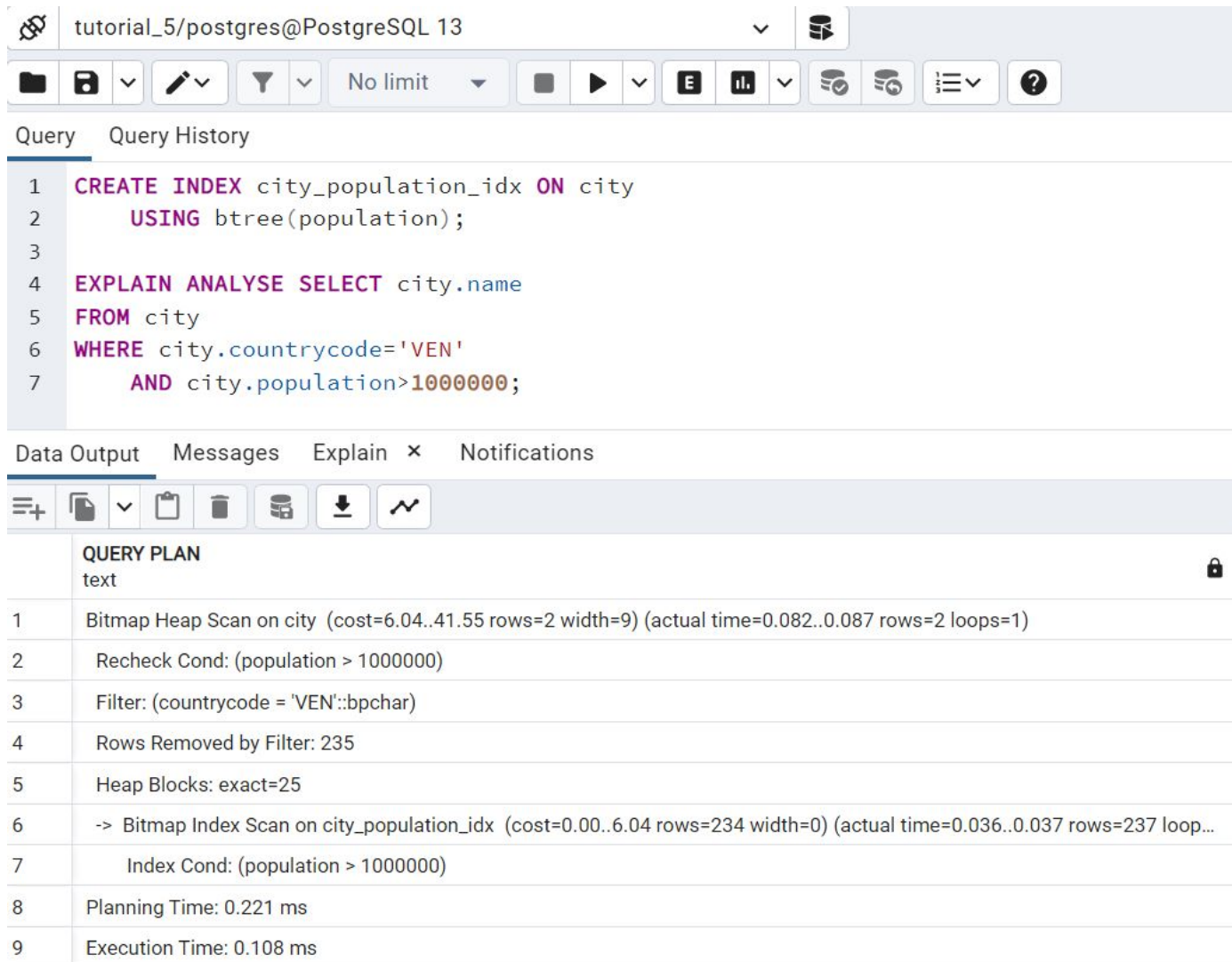
Below the query, the 'Data Output' tab is selected, displaying the 'QUERY PLAN' section. The plan details are as follows:

	QUERY PLAN
1	Seq Scan on city (cost=0.00..93.19 rows=2 width=9) (actual time=0.289..0.335 rows=2 loops...
2	Filter: ((population > 1000000) AND (countrycode = 'VEN'::bpchar))
3	Rows Removed by Filter: 4077
4	Planning Time: 0.066 ms
5	Execution Time: 0.344 ms

Χωρίς
ευρετήριο
αλλά με
χρήση
στατιστικών

Δενδρικό ευρετήριο στο population

- Ευρετήριο κατακερματισμού δεν θα χρησίμευε. (Γιατί;)



The screenshot shows a PostgreSQL client interface with the following components:

- Connection:** tutorial_5/postgres@PostgreSQL 13
- Query Editor:** Contains a SQL query to create an index and explain an analysis.
- Query:**

```
1 CREATE INDEX city_population_idx ON city
2   USING btree(population);
3
4 EXPLAIN ANALYSE SELECT city.name
5 FROM city
6 WHERE city.countrycode='VEN'
7   AND city.population>1000000;
```
- Data Output / Messages / Explain:** The 'Explain' tab is active, showing the query plan.
- Query Plan:**

	QUERY PLAN
1	Bitmap Heap Scan on city (cost=6.04..41.55 rows=2 width=9) (actual time=0.082..0.087 rows=2 loops=1)
2	Recheck Cond: (population > 1000000)
3	Filter: (countrycode = 'VEN'::bpchar)
4	Rows Removed by Filter: 235
5	Heap Blocks: exact=25
6	-> Bitmap Index Scan on city_population_idx (cost=0.00..6.04 rows=234 width=0) (actual time=0.036..0.037 rows=237 loop=1)
7	Index Cond: (population > 1000000)
8	Planning Time: 0.221 ms
9	Execution Time: 0.108 ms

Με δενδρικό
ευρετήριο και
με χρήση
στατιστικών

Ευρετήριο κατακερμ. στο countrycode

- Καλύτερο από δενδρικό για την ερώτηση αυτή (Γιατί;)

The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is 'tutorial_5/postgres@PostgreSQL 13'. Below the toolbar, the 'Query' tab is active, displaying the following SQL query:

```
1 DROP INDEX city_population_idx;  
2 CREATE INDEX city_country_idx ON city  
3   USING hash(countrycode);  
4 EXPLAIN ANALYSE SELECT city.name  
5 FROM city  
6 WHERE city.countrycode='VEN'  
7       AND city.population>1000000;
```

Below the query, the 'Data Output' tab is active, showing the 'QUERY PLAN' for the query. The plan details the execution steps and their costs.

	QUERY PLAN
1	Bitmap Heap Scan on city (cost=4.31..38.63 rows=2 width=9) (actual time=0.014..0.018 rows=2 loops=1)
2	Recheck Cond: (countrycode = 'VEN'::bpchar)
3	Filter: (population > 1000000)
4	Rows Removed by Filter: 39
5	Heap Blocks: exact=1
6	-> Bitmap Index Scan on city_country_idx (cost=0.00..4.31 rows=41 width=0) (actual time=0.009..0.009 rows=41 loop=1)
7	Index Cond: (countrycode = 'VEN'::bpchar)
8	Planning Time: 0.163 ms
9	Execution Time: 0.032 ms

Με hash
ευρετήριο και
με χρήση
στατιστικών

Επιπλέον εναλλακτικές...

- **Συνδυασμός ευρετηρίων**

Χρήση και του δενδρικού ευρετηρίου στο population και του ευρετηρίου κατακερματισμού στο countrycode

- **Ευρετήριων πολλών χαρακτηριστικών**

Χρήση δενδρικού (γιατί;) ευρετηρίου στα δύο χαρακτηριστικά countrycode και population (ποιο πρώτο και ποιο δεύτερο;)

- **Clustering**

Προσοχή: Μόνο ένα ευρετήριο μπορεί να χρησιμοποιηθεί για την ομαδοποίηση των πλειάδων ενός πίνακα.

Ομαδοποίηση (Clustering) (1/3)

- Θέλουμε να βρούμε πόλεις με πληθυσμό άνω του ενός εκατομμυρίου.

The screenshot shows a PostgreSQL query editor interface. At the top, the connection is 'tutorial_5/postgres@PostgreSQL 13'. Below the connection bar is a toolbar with icons for file operations, query execution, and settings. The 'Query' tab is active, displaying the following SQL query:

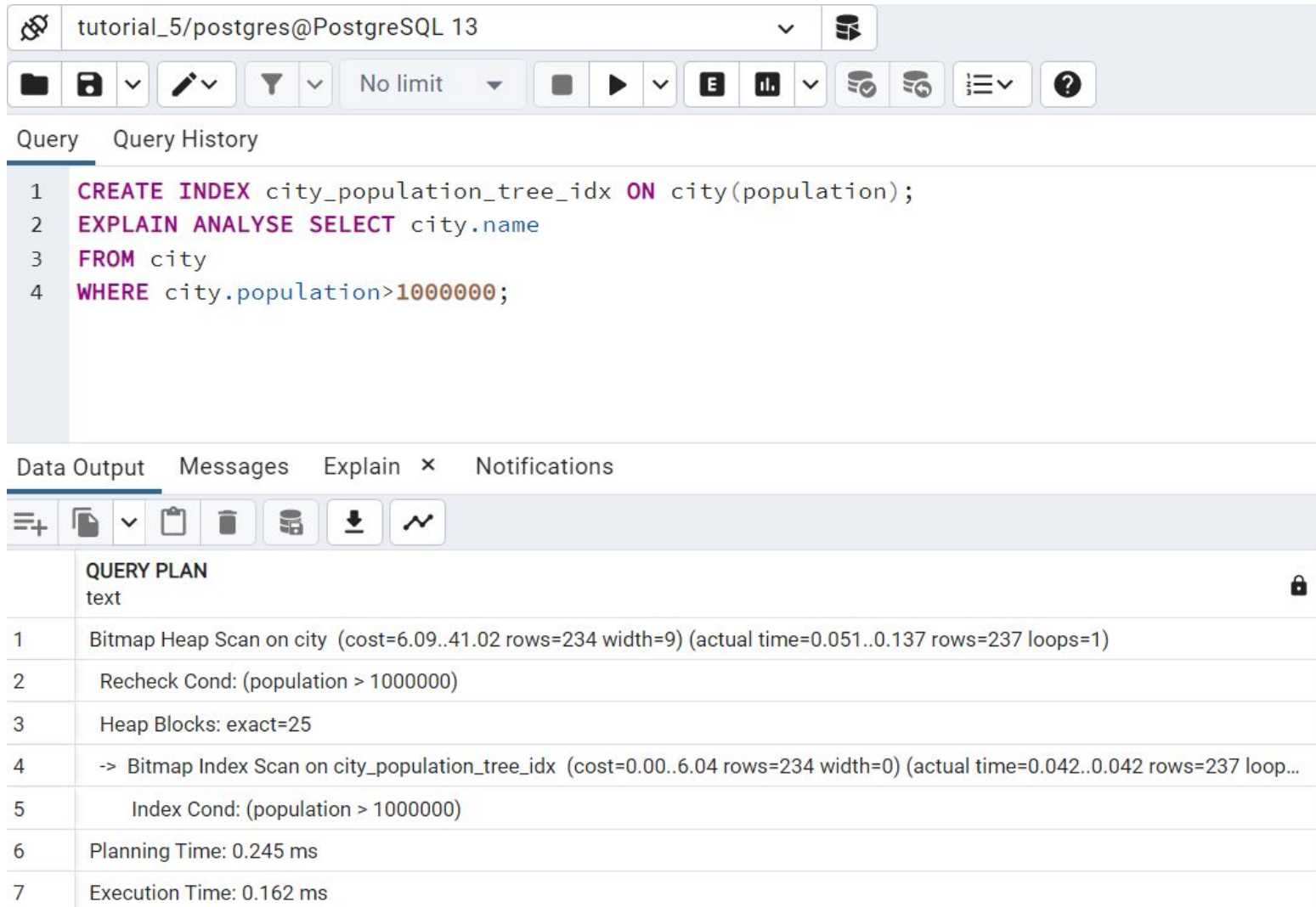
```
1 EXPLAIN ANALYSE SELECT city.name
2 FROM city
3 WHERE city.population > 1000000;
```

Below the query editor, the 'Data Output' tab is active, showing the 'QUERY PLAN' for the executed query. The plan details are as follows:

	QUERY PLAN
1	Seq Scan on city (cost=0.00..82.99 rows=234 width=9) (actual time=0.009..0.277 rows=237 loops=...
2	Filter: (population > 1000000)
3	Rows Removed by Filter: 3842
4	Planning Time: 0.246 ms
5	Execution Time: 0.289 ms

Ομαδοποίηση (Clustering) (2/3)

- Δημιουργούμε ένα δενδρικό ευρετήριο στο population



The screenshot shows a PostgreSQL client interface with the following components:

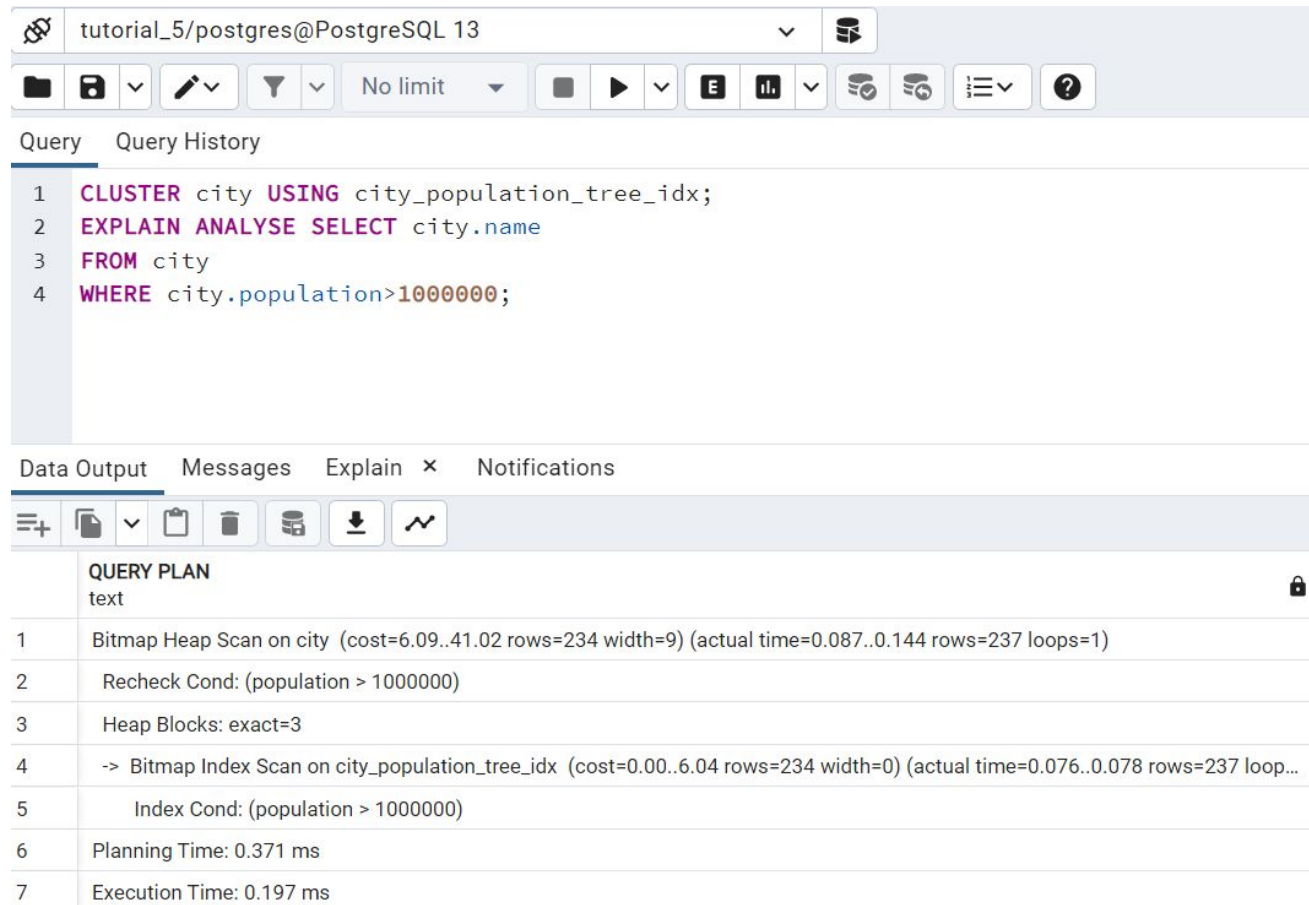
- Connection:** tutorial_5/postgres@PostgreSQL 13
- Toolbar:** Includes icons for file operations, query execution, and settings. A dropdown menu shows "No limit".
- Query Editor:** Contains the following SQL query:

```
1 CREATE INDEX city_population_tree_idx ON city(population);
2 EXPLAIN ANALYSE SELECT city.name
3 FROM city
4 WHERE city.population > 1000000;
```
- Query Plan Output:** Displays the execution plan for the query.

	QUERY PLAN
1	Bitmap Heap Scan on city (cost=6.09..41.02 rows=234 width=9) (actual time=0.051..0.137 rows=237 loops=1)
2	Recheck Cond: (population > 1000000)
3	Heap Blocks: exact=25
4	-> Bitmap Index Scan on city_population_tree_idx (cost=0.00..6.04 rows=234 width=0) (actual time=0.042..0.042 rows=237 loop...)
5	Index Cond: (population > 1000000)
6	Planning Time: 0.245 ms
7	Execution Time: 0.162 ms

Ομαδοποίηση (Clustering) (3/3)

- Ομαδοποιούμε (clustering) τις εγγραφές του πίνακα city με βάση το ευρετήριο που μόλις δημιουργήσαμε.



The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is 'tutorial_5/postgres@PostgreSQL 13'. Below the toolbar, the query editor contains the following SQL code:

```
1 CLUSTER city USING city_population_tree_idx;  
2 EXPLAIN ANALYSE SELECT city.name  
3 FROM city  
4 WHERE city.population > 1000000;
```

Below the query editor, the 'Data Output' tab is active, displaying the 'QUERY PLAN' for the executed query. The plan details are as follows:

Step	Operation
1	Bitmap Heap Scan on city (cost=6.09..41.02 rows=234 width=9) (actual time=0.087..0.144 rows=237 loops=1)
2	Recheck Cond: (population > 1000000)
3	Heap Blocks: exact=3
4	-> Bitmap Index Scan on city_population_tree_idx (cost=0.00..6.04 rows=234 width=0) (actual time=0.076..0.078 rows=237 loop=1)
5	Index Cond: (population > 1000000)
6	Planning Time: 0.371 ms
7	Execution Time: 0.197 ms

- Τι παρατηρείτε; Αλλάζει το σχέδιο; Τι αλλάζει; Γιατί;

Ένα παράδειγμα για την επιλογή μεθόδου σύνδεσης στην Postgres

Βασισμένο στην παρουσίαση «*Explaining the Postgres Query Optimizer*» του Bruce Momjian,
<https://momjian.us/presentations>

- Δημιουργία 2 temporary tables (χωρίς index και χωρίς στατιστικά)

```
1 CREATE TEMPORARY TABLE sample1 (id, junk) AS
2   SELECT city.countrycode, repeat('x',250)
3   FROM city
4   ORDER BY RANDOM();
5
6 CREATE TEMPORARY TABLE sample2 (id, junk) AS
7   SELECT country.code, repeat('x',250)
8   FROM country
9   ORDER BY RANDOM();
```

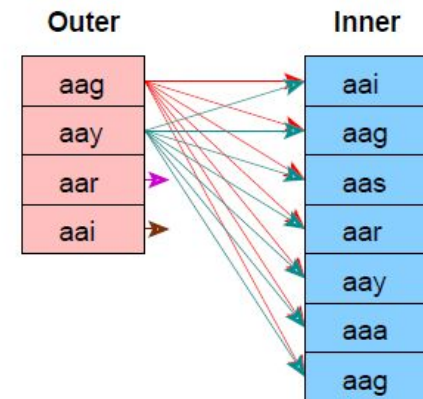
Ένα παράδειγμα για την επιλογή μεθόδου σύνδεσης στην Postgres

- Σύνδεση των δύο tables με αυστηρή συνθήκη επιλογής

```
11 EXPLAIN SELECT sample2.junk
12 FROM sample1 JOIN sample2 ON (sample1.id = sample2.id)
13 WHERE sample1.id = 'BRA'
```

QUERY PLAN	
	text
1	Nested Loop (cost=0.00..383.75 rows=405 width=32)
2	-> Seq Scan on sample1 (cost=0.00..355.30 rows=81 width=16)
3	Filter: (id = 'BRA'::bpchar)
4	-> Materialize (cost=0.00..23.40 rows=5 width=48)
5	-> Seq Scan on sample2 (cost=0.00..23.38 rows=5 width=48)
6	Filter: (id = 'BRA'::bpchar)

Nested Loop Join with Inner Sequential Scan



Επιλογή nested loop join για φόρτους μικρού μεγέθους

No Setup Required

Used For Small Tables

Ένα παράδειγμα για την επιλογή μεθόδου σύνδεσης στην Postgres

- Σύνδεση των δύο tables με πιο χαλαρή συνθήκη επιλογής

```
15 EXPLAIN SELECT sample2.junk
16 FROM sample1 JOIN sample2 ON (sample1.id = sample2.id)
17 WHERE sample1.id > 'BOL' and sample1.id < 'BRA'
```

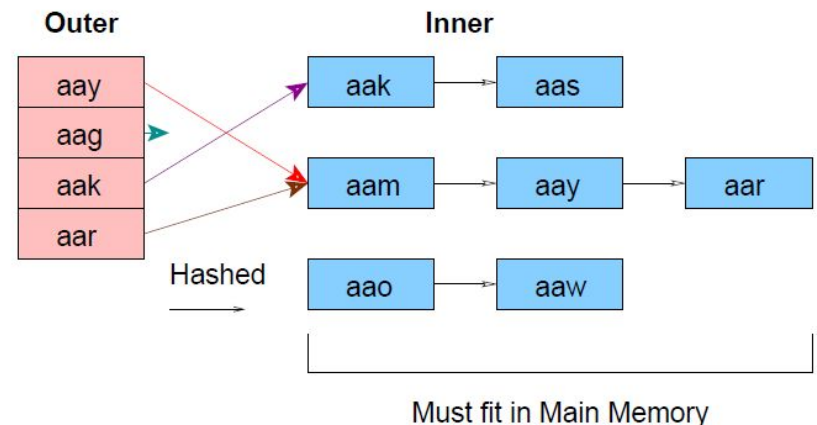
QUERY PLAN

text

1	Hash Join (cost=396.97..435.38 rows=433 width=32)
2	Hash Cond: (sample2.id = sample1.id)
3	-> Seq Scan on sample2 (cost=0.00..20.70 rows=1070 width=48)
4	-> Hash (cost=395.96..395.96 rows=81 width=16)
5	-> Seq Scan on sample1 (cost=0.00..395.96 rows=81 width=16)
6	Filter: ((id > 'BOL'::bpchar) AND (id < 'BRA'::bpchar))

Επιλογή hash join για φόρτους
μεσαίου μεγέθους

Hash Join

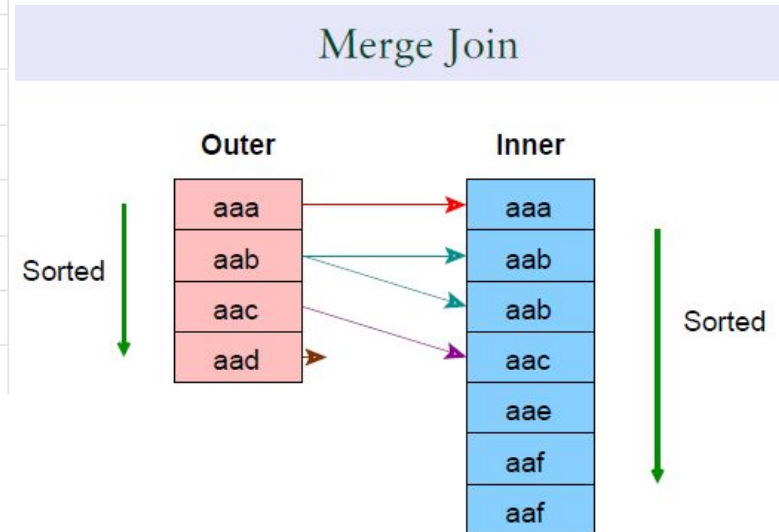


Ένα παράδειγμα για την επιλογή μεθόδου σύνδεσης στην Postgres

- Σύνδεση των δύο tables χωρίς συνθήκη επιλογής

```
19 EXPLAIN SELECT sample2.junk
20 FROM sample1 JOIN sample2 ON (sample1.id = sample2.id)
```

QUERY PLAN		text	
1	Merge Join (cost=1526.80..2837.33 rows=87012 width=32)		
2	Merge Cond: (sample2.id = sample1.id)		
3	-> Sort (cost=74.54..77.21 rows=1070 width=48)		
4	Sort Key: sample2.id		
5	-> Seq Scan on sample2 (cost=0.00..20.70 rows=1070 width=48)		
6	-> Sort (cost=1452.26..1492.92 rows=16264 width=16)		
7	Sort Key: sample1.id		
8	-> Seq Scan on sample1 (cost=0.00..314.64 rows=16264 width=1...)		



Επιλογή merge join για φόρτους
μεγάλου μεγέθους

Ideal for Large Tables
An Index Can Be Used to Eliminate the Sort

Ένα παράδειγμα για την επιλογή μεθόδου σύνδεσης στην Postgres

- Σύνδεση των δύο tables χωρίς συνθήκη επιλογής. Η σειρά της σύνδεσης των δύο tables δεν είναι σημαντική.

```
22 EXPLAIN SELECT sample2.junk
23 FROM sample2 JOIN sample1 ON (sample2.id = sample1.id)
```

	QUERY PLAN	
	text	
1	Merge Join (cost=1526.80..2837.33 rows=87012 width=32)	
2	Merge Cond: (sample2.id = sample1.id)	
3	-> Sort (cost=74.54..77.21 rows=1070 width=48)	
4	Sort Key: sample2.id	
5	-> Seq Scan on sample2 (cost=0.00..20.70 rows=1070 width=48)	
6	-> Sort (cost=1452.26..1492.92 rows=16264 width=16)	
7	Sort Key: sample1.id	
8	-> Seq Scan on sample1 (cost=0.00..314.64 rows=16264 width=1...	

Το μικρότερο sample2 table είναι πάντα στην εξωτερική πλευρά του merge join

Ένα παράδειγμα για την επιλογή μεθόδου σύνδεσης στην Postgres

- Δημιουργία στατιστικών

```
25 ANALYSE sample1;  
26  
27 ANALYSE sample2;  
28  
29 EXPLAIN SELECT sample2.junk  
30 FROM sample1 JOIN sample2 ON (sample1.id = sample2.id)
```

QUERY PLAN		
text		
1	Hash Join (cost=14.38..263.25 rows=4079 width=254)	
2	Hash Cond: (sample1.id = sample2.id)	
3	-> Seq Scan on sample1 (cost=0.00..192.79 rows=4079 width=4)	
4	-> Hash (cost=11.39..11.39 rows=239 width=258)	
5	-> Seq Scan on sample2 (cost=0.00..11.39 rows=239 width=258)	

Ένα παράδειγμα για την επιλογή μεθόδου σύνδεσης στην Postgres

- Τα εξωτερικά joins μπορεί να επηρεάσουν την επιλογή των συνδέσεων από το βελτιστοποιητή

```
32 EXPLAIN SELECT sample1.junk
33 FROM sample1 RIGHT OUTER JOIN sample2 ON (sample1.id = sample2.id);
```

QUERY PLAN	
text	
1	Hash Right Join (cost=14.38..263.25 rows=4079 width=254)
2	Hash Cond: (sample1.id = sample2.id)
3	-> Seq Scan on sample1 (cost=0.00..192.79 rows=4079 width=258)
4	-> Hash (cost=11.39..11.39 rows=239 width=4)
5	-> Seq Scan on sample2 (cost=0.00..11.39 rows=239 width=4)

Ένα παράδειγμα για την επιλογή μεθόδου σύνδεσης στην Postgres

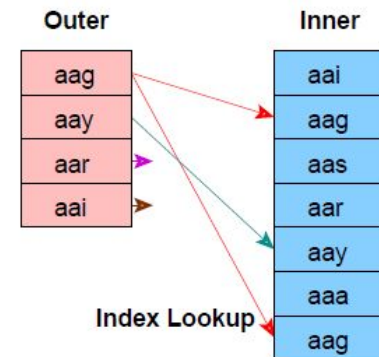
- Δημιουργία ευρετηρίων

```
35 CREATE INDEX i_sample1 on sample1 (id);
36 CREATE INDEX i_sample2 on sample2 (id);
42 EXPLAIN SELECT sample2.junk
43 FROM sample1 JOIN sample2 ON (sample1.id = sample2.id)
44 WHERE sample1.id > 'BOL' and sample1.id < 'BRA'
```

Ήταν ~436 με
hash join

QUERY PLAN	
	text
1	Nested Loop (cost=0.43..16.47 rows=1 width=254)
2	-> Index Only Scan using i_sample1 on sample1 (cost=0.28..8.30 rows=1 width=254)
3	Index Cond: ((id > 'BOL'::bpchar) AND (id < 'BRA'::bpchar))
4	-> Index Scan using i_sample2 on sample2 (cost=0.14..8.16 rows=1 width=254)
5	Index Cond: (id = sample1.id)

Nested Loop Join with Inner Index Scan



No Setup Required

Index Must Already Exist

Ένα παράδειγμα για την επιλογή μεθόδου σύνδεσης στην Postgres

- Η επιλογή των συνθηκών όπως και η χρήση του LIMIT επηρεάζουν την επιλογή του join.

```
46 EXPLAIN SELECT sample2.junk
47 FROM sample1 JOIN sample2 ON (sample1.id = sample2.id)
48 WHERE sample2.junk ~ '^xxx';
```

QUERY PLAN	
text	
1	Hash Join (cost=14.98..263.85 rows=4079 width=254)
2	Hash Cond: (sample1.id = sample2.id)
3	-> Seq Scan on sample1 (cost=0.00..192.79 rows=4079 width=4)
4	-> Hash (cost=11.99..11.99 rows=239 width=258)
5	-> Seq Scan on sample2 (cost=0.00..11.99 rows=239 width=258)
6	Filter: (junk ~ '^xxx'::text)

Επιλέχθηκε hash join γιατί αναμένονται πολλά tuples. Ο μικρότερος πίνακας sample2 πάντα γίνεται hashed.

Ένα παράδειγμα για την επιλογή μεθόδου σύνδεσης στην Postgres

- Χωρίς LIMIT επιλέγεται hash join για συνδέσεις χωρίς συνθήκη επιλογής

```
50 EXPLAIN SELECT sample2.junk
51 FROM sample1 JOIN sample2 ON (sample1.id = sample2.id)
52 ORDER BY 1;
```

QUERY PLAN		
text		
1	Sort (cost=507.87..518.07 rows=4079 width=254)	
2	Sort Key: sample2.junk	
3	-> Hash Join (cost=14.38..263.25 rows=4079 width=254)	
4	Hash Cond: (sample1.id = sample2.id)	
5	-> Seq Scan on sample1 (cost=0.00..192.79 rows=4079 width=4)	
6	-> Hash (cost=11.39..11.39 rows=239 width=258)	
7	-> Seq Scan on sample2 (cost=0.00..11.39 rows=239 width=258)	

Ένα παράδειγμα για την επιλογή μεθόδου σύνδεσης στην Postgres

- Με LIMIT επιλέγεται merge join (μπορεί και nested loop join ανάλογα με το πλήθος) για συνδέσεις χωρίς συνθήκη επιλογής

```
54 EXPLAIN SELECT sample2.id, sample2.junk
55 FROM sample1 JOIN sample2 ON (sample1.id = sample2.id)
56 ORDER BY 1
57 LIMIT 100;
```

QUERY PLAN

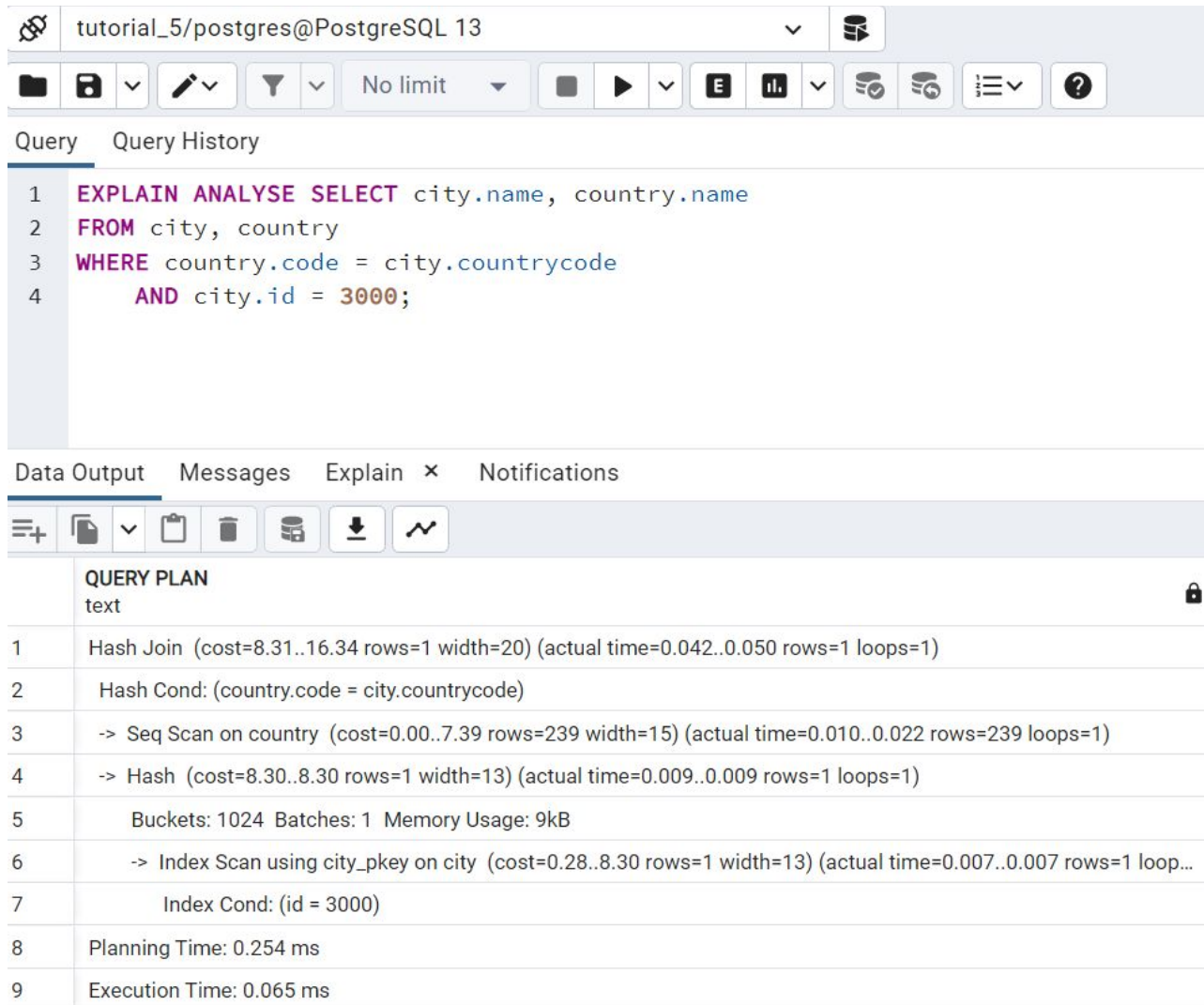
text



1	Limit (cost=0.43..19.90 rows=100 width=258)
2	-> Merge Join (cost=0.43..794.62 rows=4079 width=258)
3	Merge Cond: (sample2.id = sample1.id)
4	-> Index Scan using i_sample2 on sample2 (cost=0.14..47.73 rows=239 width=258)
5	-> Index Only Scan using i_sample1 on sample1 (cost=0.28..695.31 rows=4079 width=4)

Βελτίωση απόδοσης συνδέσεων (1/3)

- Τα κύρια κλειδιά έχουν ευρετήρια



The screenshot shows a PostgreSQL query editor interface. At the top, the connection is 'tutorial_5/postgres@PostgreSQL 13'. Below the connection bar is a toolbar with various icons for file operations, query execution, and settings. The 'Query' tab is active, displaying the following SQL query:

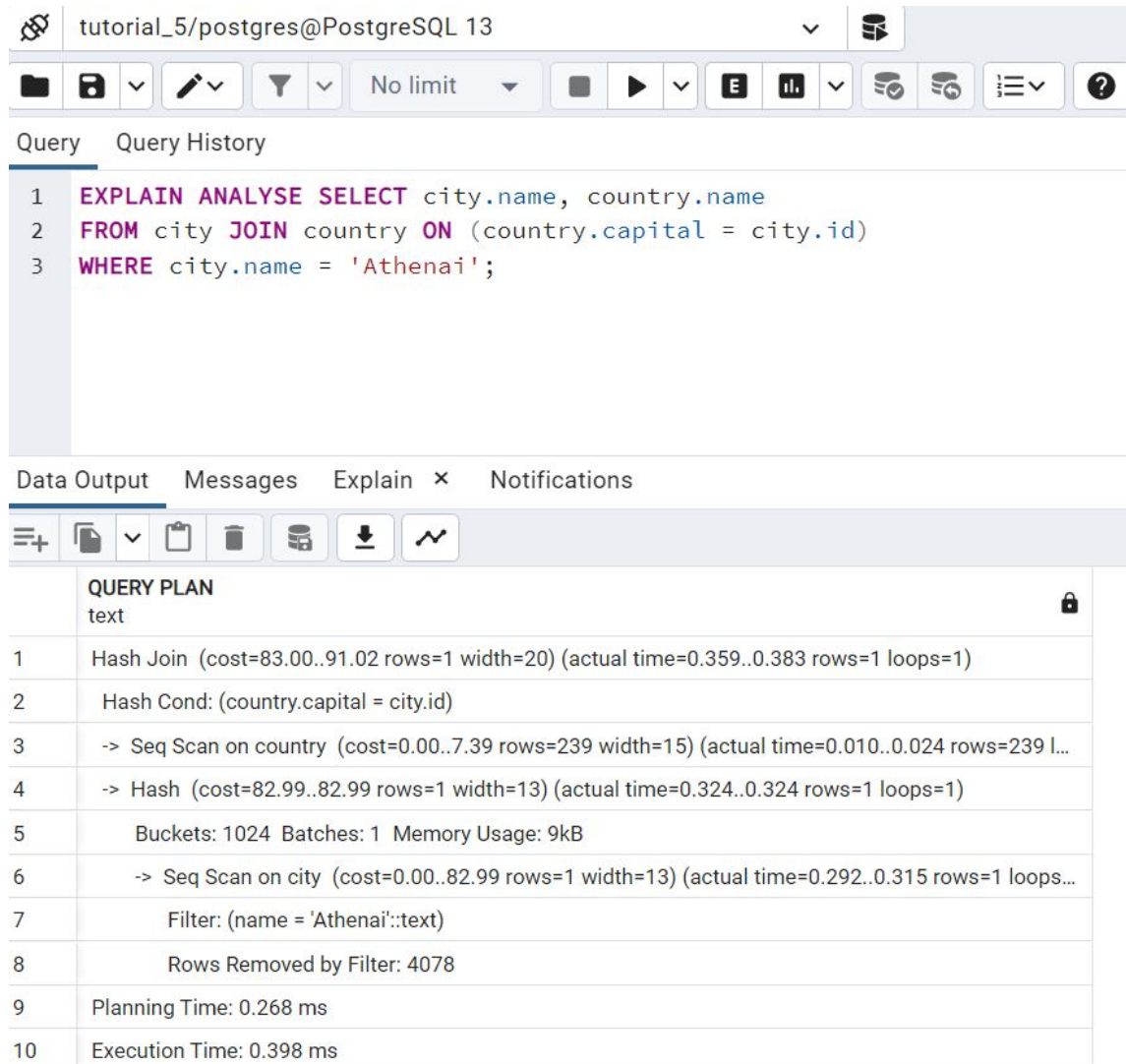
```
1 EXPLAIN ANALYSE SELECT city.name, country.name
2 FROM city, country
3 WHERE country.code = city.countrycode
4     AND city.id = 3000;
```

Below the query editor is a toolbar for the 'Data Output' tab, which includes icons for refreshing, saving, deleting, and other data management actions. The 'Data Output' tab is active, showing the 'QUERY PLAN' section. The plan details the execution strategy for the query, including the cost, rows, and time for each step.

Step	Operation
1	Hash Join (cost=8.31..16.34 rows=1 width=20) (actual time=0.042..0.050 rows=1 loops=1)
2	Hash Cond: (country.code = city.countrycode)
3	-> Seq Scan on country (cost=0.00..7.39 rows=239 width=15) (actual time=0.010..0.022 rows=239 loops=1)
4	-> Hash (cost=8.30..8.30 rows=1 width=13) (actual time=0.009..0.009 rows=1 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Index Scan using city_pkey on city (cost=0.28..8.30 rows=1 width=13) (actual time=0.007..0.007 rows=1 loop=1)
7	Index Cond: (id = 3000)
8	Planning Time: 0.254 ms
9	Execution Time: 0.065 ms

Βελτίωση απόδοσης συνδέσεων (2/3)

- Τα ξένα κλειδιά δεν έχουν ευρετήρια.



The screenshot shows a PostgreSQL query editor interface. At the top, the connection is 'tutorial_5/postgres@PostgreSQL 13'. Below the connection bar is a toolbar with icons for file operations, query execution, and settings. The 'Query' tab is active, displaying the following SQL query:

```
1 EXPLAIN ANALYSE SELECT city.name, country.name
2 FROM city JOIN country ON (country.capital = city.id)
3 WHERE city.name = 'Athenai';
```

Below the query editor, the 'Data Output' tab is active, showing the 'QUERY PLAN' for the executed query. The plan details are as follows:

Step	Operation
1	Hash Join (cost=83.00..91.02 rows=1 width=20) (actual time=0.359..0.383 rows=1 loops=1)
2	Hash Cond: (country.capital = city.id)
3	-> Seq Scan on country (cost=0.00..7.39 rows=239 width=15) (actual time=0.010..0.024 rows=239 l...
4	-> Hash (cost=82.99..82.99 rows=1 width=13) (actual time=0.324..0.324 rows=1 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on city (cost=0.00..82.99 rows=1 width=13) (actual time=0.292..0.315 rows=1 loops=...
7	Filter: (name = 'Athenai':text)
8	Rows Removed by Filter: 4078
9	Planning Time: 0.268 ms
10	Execution Time: 0.398 ms

Βελτίωση απόδοσης συνδέσεων (3/3)

- Δημιουργία νέου ευρετηρίου για το ξένο κλειδί...

The screenshot shows a PostgreSQL query editor interface. At the top, the connection is 'tutorial_5/postgres@PostgreSQL 13'. Below the connection bar is a toolbar with various icons for file operations, query execution, and settings. The 'Query' tab is active, displaying the following SQL query:

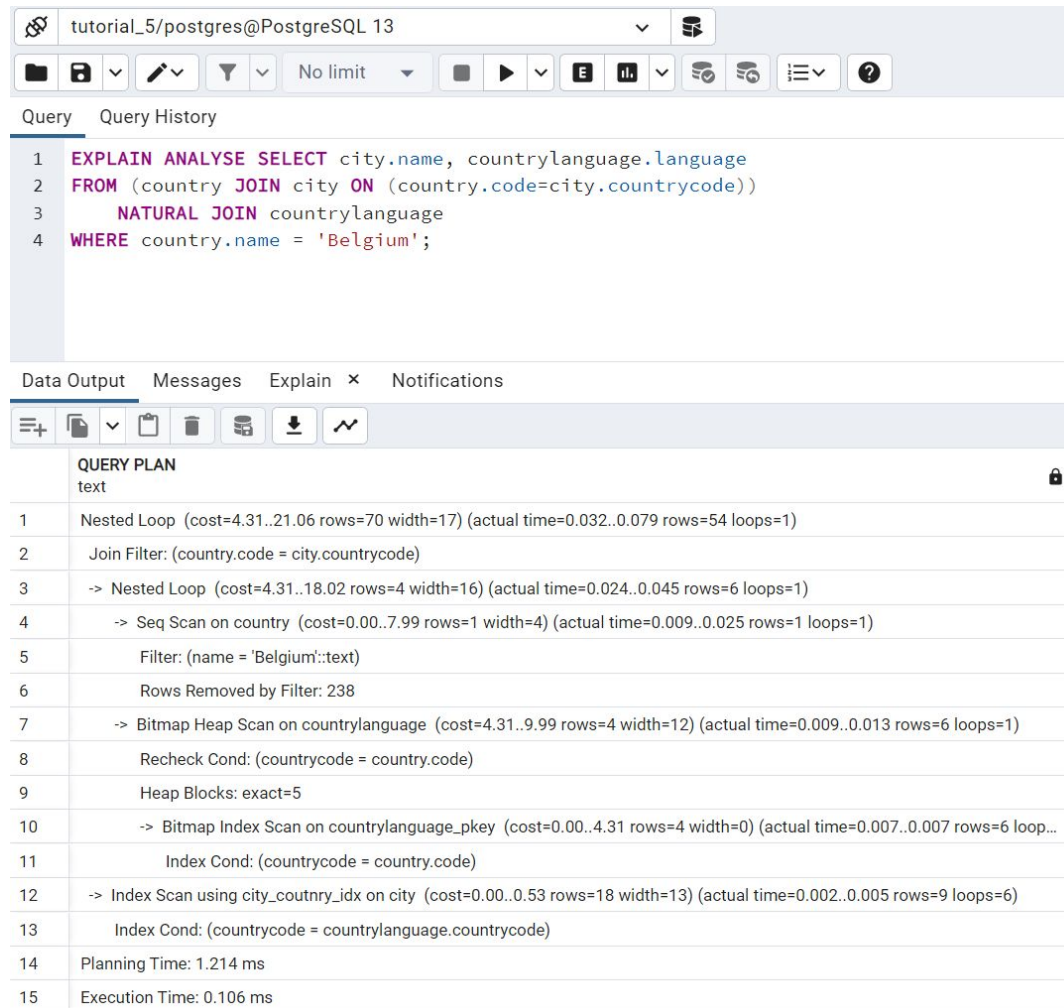
```
1 CREATE INDEX country_capital_idx ON country USING hash(capital);
2 EXPLAIN ANALYSE SELECT city.name, country.name
3 FROM city JOIN country ON (country.capital = city.id)
4 WHERE city.name = 'Athenai';
```

Below the query editor, the 'Data Output' tab is active, showing the 'QUERY PLAN' for the executed query. The plan details the execution strategy, including a nested loop join, a sequential scan on the 'city' table, a filter for 'Athenai', and an index scan on the 'country' table using the newly created 'country_capital_idx'.

	QUERY PLAN
1	Nested Loop (cost=0.00..91.02 rows=1 width=20) (actual time=0.268..0.288 rows=1 loops=1)
2	-> Seq Scan on city (cost=0.00..82.99 rows=1 width=13) (actual time=0.261..0.280 rows=1 loops=1)
3	Filter: (name = 'Athenai':text)
4	Rows Removed by Filter: 4078
5	-> Index Scan using country_capital_idx on country (cost=0.00..8.02 rows=1 width=15) (actual time=0.005..0.006 rows=1 loop...
6	Index Cond: (capital = city.id)
7	Planning Time: 0.266 ms
8	Execution Time: 0.301 ms

Ελέγχοντας τη σειρά των συνδέσεων (1/2)

- Η σειρά των συνδέσεων επηρεάζει την απόδοση του συστήματος. Τη σειρά την αποφασίζει ο βελτιστοποιητής



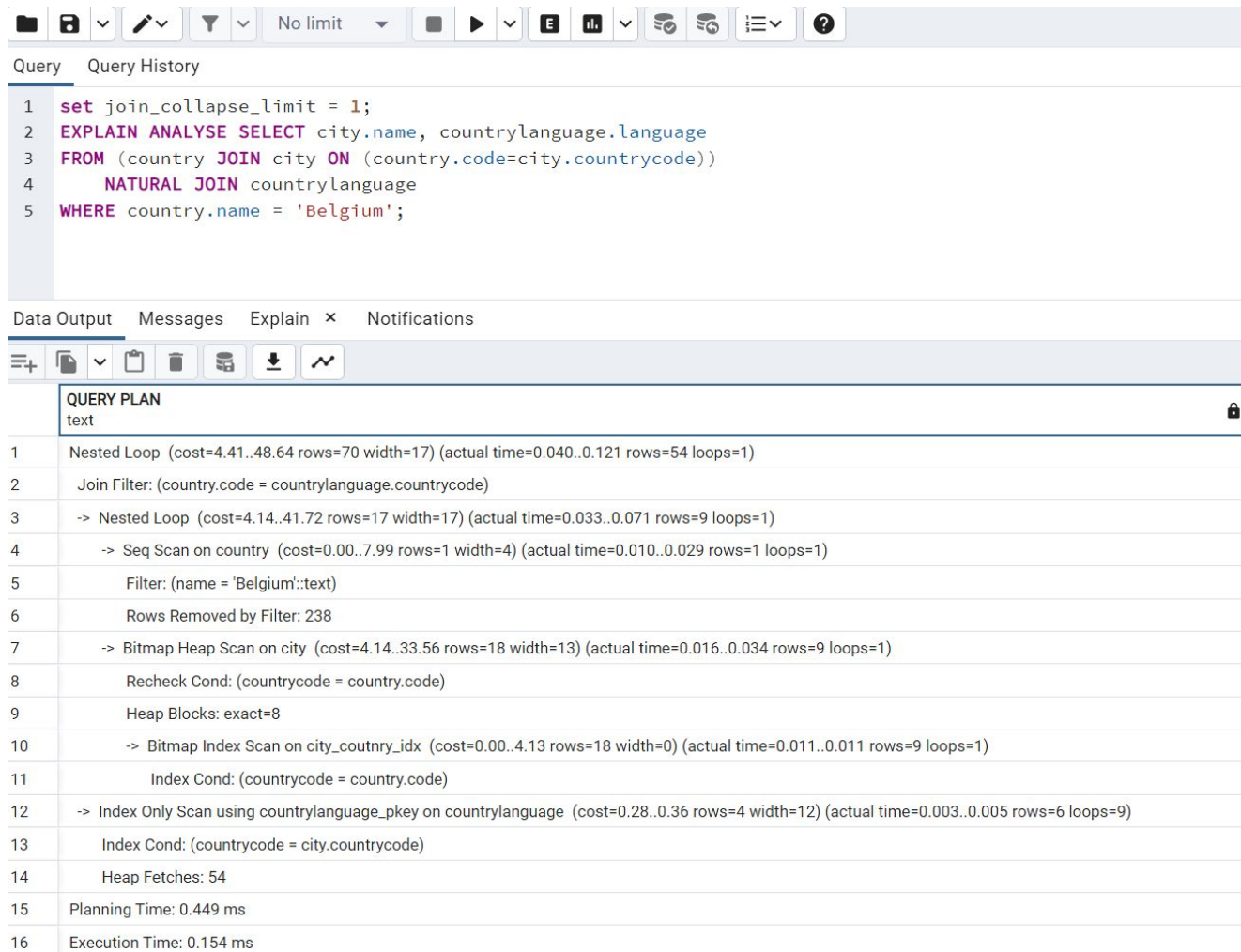
The screenshot shows a PostgreSQL query editor interface. At the top, the connection is 'tutorial_5/postgres@PostgreSQL 13'. Below the toolbar, the query is displayed in a text editor. The query is an `EXPLAIN ANALYSE` statement that selects city names and country languages, joined by country code, and filtered for Belgium. Below the query, the 'Data Output' tab is active, showing the 'QUERY PLAN' in a table format. The plan details the execution strategy, including nested loops, join filters, and various scans (Seq Scan, Bitmap Heap Scan, Bitmap Index Scan, Index Scan) with their respective costs, row counts, and execution times.

```
1 EXPLAIN ANALYSE SELECT city.name, countrylanguage.language
2 FROM (country JOIN city ON (country.code=city.countrycode))
3 NATURAL JOIN countrylanguage
4 WHERE country.name = 'Belgium';
```

	QUERY PLAN
1	Nested Loop (cost=4.31..21.06 rows=70 width=17) (actual time=0.032..0.079 rows=54 loops=1)
2	Join Filter: (country.code = city.countrycode)
3	-> Nested Loop (cost=4.31..18.02 rows=4 width=16) (actual time=0.024..0.045 rows=6 loops=1)
4	-> Seq Scan on country (cost=0.00..7.99 rows=1 width=4) (actual time=0.009..0.025 rows=1 loops=1)
5	Filter: (name = 'Belgium'::text)
6	Rows Removed by Filter: 238
7	-> Bitmap Heap Scan on countrylanguage (cost=4.31..9.99 rows=4 width=12) (actual time=0.009..0.013 rows=6 loops=1)
8	Recheck Cond: (countrycode = country.code)
9	Heap Blocks: exact=5
10	-> Bitmap Index Scan on countrylanguage_pkey (cost=0.00..4.31 rows=4 width=0) (actual time=0.007..0.007 rows=6 loop...
11	Index Cond: (countrycode = country.code)
12	-> Index Scan using city_countrny_idx on city (cost=0.00..0.53 rows=18 width=13) (actual time=0.002..0.005 rows=9 loops=6)
13	Index Cond: (countrycode = countrylanguage.countrycode)
14	Planning Time: 1.214 ms
15	Execution Time: 0.106 ms

Ελέγχοντας τη σειρά των συνδέσεων (2/2)

- Πράξεις σύνδεσης: προσπέλαση πινάκων (FROM) και συνθήκες (WHERE)
- Σε μια συνεδρία (session) μπορούμε να επιβάλουμε τη σειρά συνδέσεων που εμφανίζονται στο FROM θέτοντας `join_collapse_limit=1`



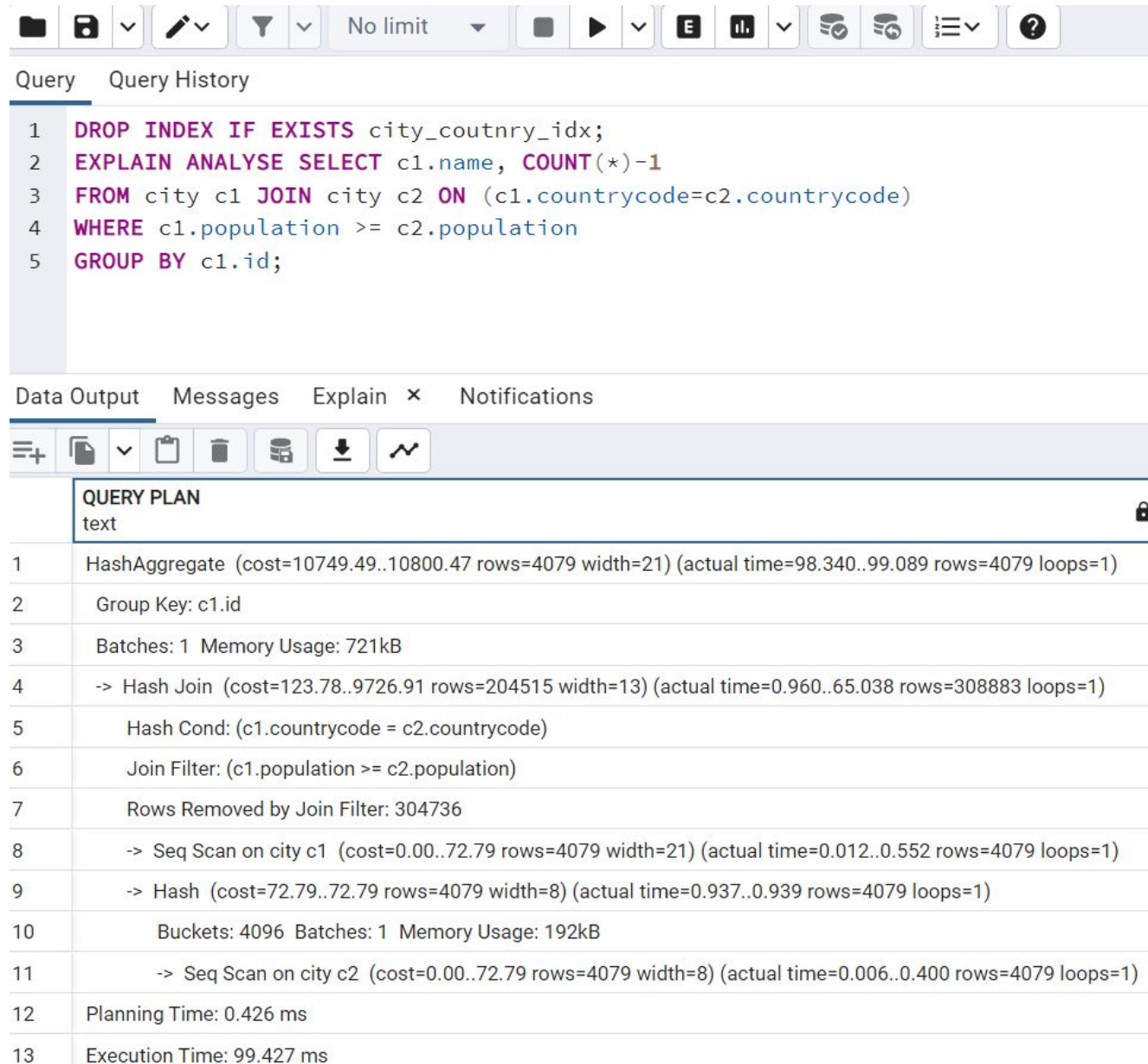
The screenshot displays a PostgreSQL query editor interface. At the top, there's a toolbar with various icons for file operations, query execution, and settings. Below the toolbar, the 'Query' tab is active, showing the following SQL query:

```
1 set join_collapse_limit = 1;
2 EXPLAIN ANALYSE SELECT city.name, countrylanguage.language
3 FROM (country JOIN city ON (country.code=city.countrycode))
4     NATURAL JOIN countrylanguage
5 WHERE country.name = 'Belgium';
```

Below the query, the 'Data Output' tab is active, showing the 'QUERY PLAN' for the query. The plan is as follows:

Step	Operation
1	Nested Loop (cost=4.41..48.64 rows=70 width=17) (actual time=0.040..0.121 rows=54 loops=1)
2	Join Filter: (country.code = countrylanguage.countrycode)
3	-> Nested Loop (cost=4.14..41.72 rows=17 width=17) (actual time=0.033..0.071 rows=9 loops=1)
4	-> Seq Scan on country (cost=0.00..7.99 rows=1 width=4) (actual time=0.010..0.029 rows=1 loops=1)
5	Filter: (name = 'Belgium':text)
6	Rows Removed by Filter: 238
7	-> Bitmap Heap Scan on city (cost=4.14..33.56 rows=18 width=13) (actual time=0.016..0.034 rows=9 loops=1)
8	Recheck Cond: (countrycode = country.code)
9	Heap Blocks: exact=8
10	-> Bitmap Index Scan on city_country_idx (cost=0.00..4.13 rows=18 width=0) (actual time=0.011..0.011 rows=9 loops=1)
11	Index Cond: (countrycode = country.code)
12	-> Index Only Scan using countrylanguage_pkey on countrylanguage (cost=0.28..0.36 rows=4 width=12) (actual time=0.003..0.005 rows=6 loops=9)
13	Index Cond: (countrycode = city.countrycode)
14	Heap Fetches: 54
15	Planning Time: 0.449 ms
16	Execution Time: 0.154 ms

Αλλάζοντας τρόπο υπολ. συνδέσεων (1/3)



The screenshot shows a database query editor interface. At the top, there is a toolbar with icons for file operations, query execution, and settings. Below the toolbar, the 'Query' tab is active, displaying a SQL query. The query is as follows:

```
1 DROP INDEX IF EXISTS city_countrny_idx;
2 EXPLAIN ANALYSE SELECT c1.name, COUNT(*)-1
3 FROM city c1 JOIN city c2 ON (c1.countrycode=c2.countrycode)
4 WHERE c1.population >= c2.population
5 GROUP BY c1.id;
```

Below the query, the 'Data Output' tab is active, showing the 'QUERY PLAN' for the query. The plan is as follows:

	QUERY PLAN
1	HashAggregate (cost=10749.49..10800.47 rows=4079 width=21) (actual time=98.340..99.089 rows=4079 loops=1)
2	Group Key: c1.id
3	Batches: 1 Memory Usage: 721kB
4	-> Hash Join (cost=123.78..9726.91 rows=204515 width=13) (actual time=0.960..65.038 rows=308883 loops=1)
5	Hash Cond: (c1.countrycode = c2.countrycode)
6	Join Filter: (c1.population >= c2.population)
7	Rows Removed by Join Filter: 304736
8	-> Seq Scan on city c1 (cost=0.00..72.79 rows=4079 width=21) (actual time=0.012..0.552 rows=4079 loops=1)
9	-> Hash (cost=72.79..72.79 rows=4079 width=8) (actual time=0.937..0.939 rows=4079 loops=1)
10	Buckets: 4096 Batches: 1 Memory Usage: 192kB
11	-> Seq Scan on city c2 (cost=0.00..72.79 rows=4079 width=8) (actual time=0.006..0.400 rows=4079 loops=1)
12	Planning Time: 0.426 ms
13	Execution Time: 99.427 ms

Αλλάζοντας τρόπο υπολ. συνδέσεων (2/3)

Query

```
1 set enable_hashjoin = off;
2 EXPLAIN ANALYSE SELECT c1.name, COUNT(*)-1
3 FROM city c1 JOIN city c2 ON (c1.countrycode=c2.countrycode)
4 WHERE c1.population >= c2.population
5 GROUP BY c1.id;
```

Data Output Messages Explain × Notifications

QUERY PLAN
text

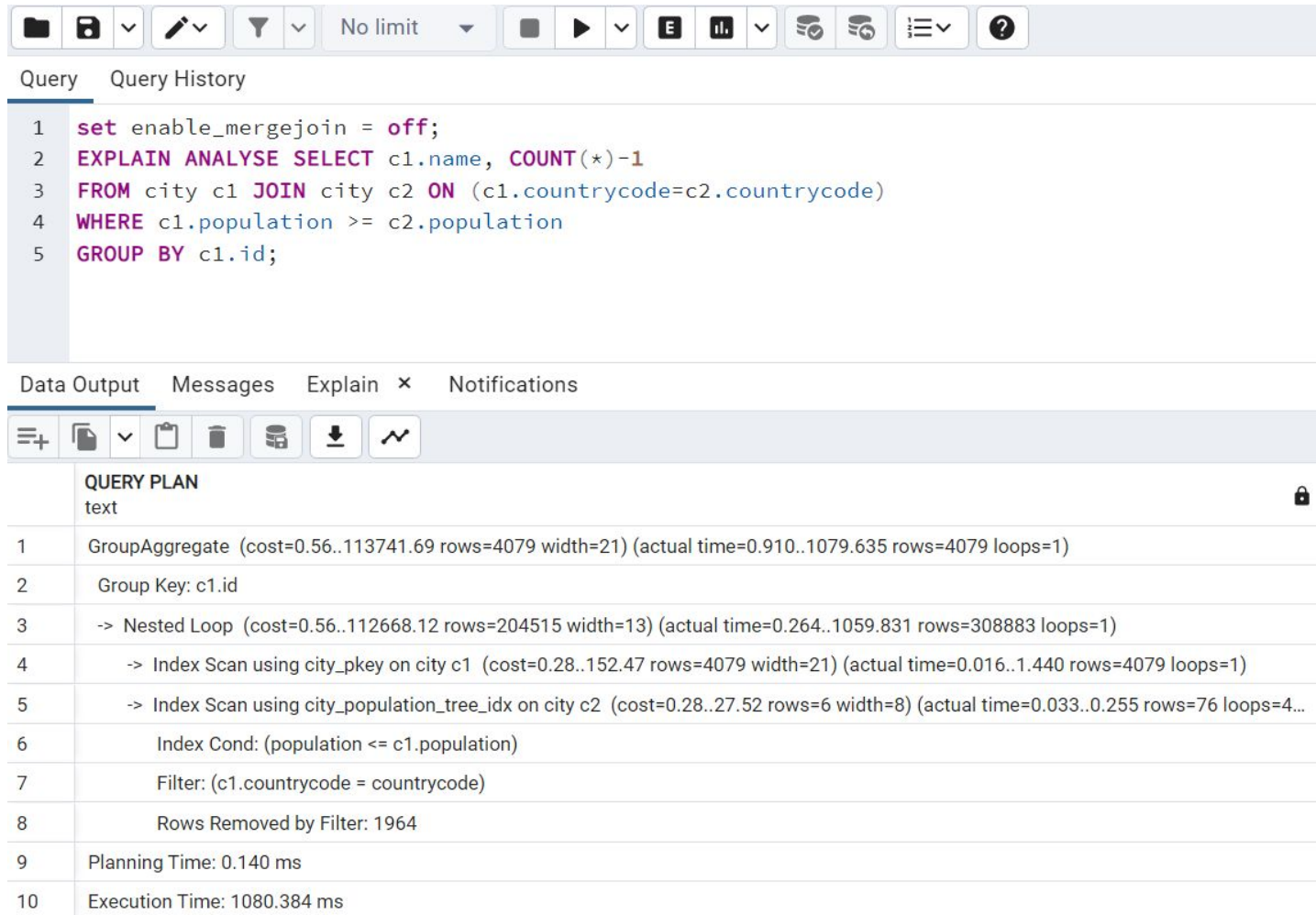
1	HashAggregate (cost=12414.81..12465.79 rows=4079 width=21) (actual time=125.882..126.388 rows=4079 loops=1)
2	Group Key: c1.id
3	Batches: 1 Memory Usage: 721kB
4	-> Merge Join (cost=634.82..11392.23 rows=204515 width=13) (actual time=7.611..94.019 rows=308883 loops=1)
5	Merge Cond: (c1.countrycode = c2.countrycode)
6	Join Filter: (c1.population >= c2.population)
7	Rows Removed by Join Filter: 304736
8	-> Sort (cost=317.41..327.61 rows=4079 width=21) (actual time=3.778..4.488 rows=4079 loops=1)
9	Sort Key: c1.countrycode
10	Sort Method: quicksort Memory: 414kB
11	-> Seq Scan on city c1 (cost=0.00..72.79 rows=4079 width=21) (actual time=0.011..0.418 rows=4079 loops=1)
12	-> Sort (cost=317.41..327.61 rows=4079 width=8) (actual time=3.827..21.890 rows=613614 loops=1)
13	Sort Key: c2.countrycode
14	Sort Method: quicksort Memory: 288kB
15	-> Seq Scan on city c2 (cost=0.00..72.79 rows=4079 width=8) (actual time=0.012..0.521 rows=4079 loops=1)
16	Planning Time: 0.405 ms
17	Execution Time: 126.806 ms

Παράμετροι που ελέγχουν τα είδη της σύνδεσης που μπορεί να χρησιμοποιεί ο βελτιστοποιητής.

Ανάλογη χρήση με το `join_collapse_limit` που είδαμε ήδη

Αλλάζοντας τρόπο υπολ. συνδέσεων (3/3)

- Ας δούμε τι συμβαίνει όταν, μετά την απενεργοποίηση των hash joins απενεργοποιήσουμε και τα merge joins.
- Τι παρατηρείται σχετικά με την απόδοση των τριών αυτών σχεδίων;



The screenshot shows a database query interface. At the top is a toolbar with icons for file operations, query execution, and settings. Below the toolbar are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query:

```
1 set enable_mergejoin = off;
2 EXPLAIN ANALYSE SELECT c1.name, COUNT(*)-1
3 FROM city c1 JOIN city c2 ON (c1.countrycode=c2.countrycode)
4 WHERE c1.population >= c2.population
5 GROUP BY c1.id;
```

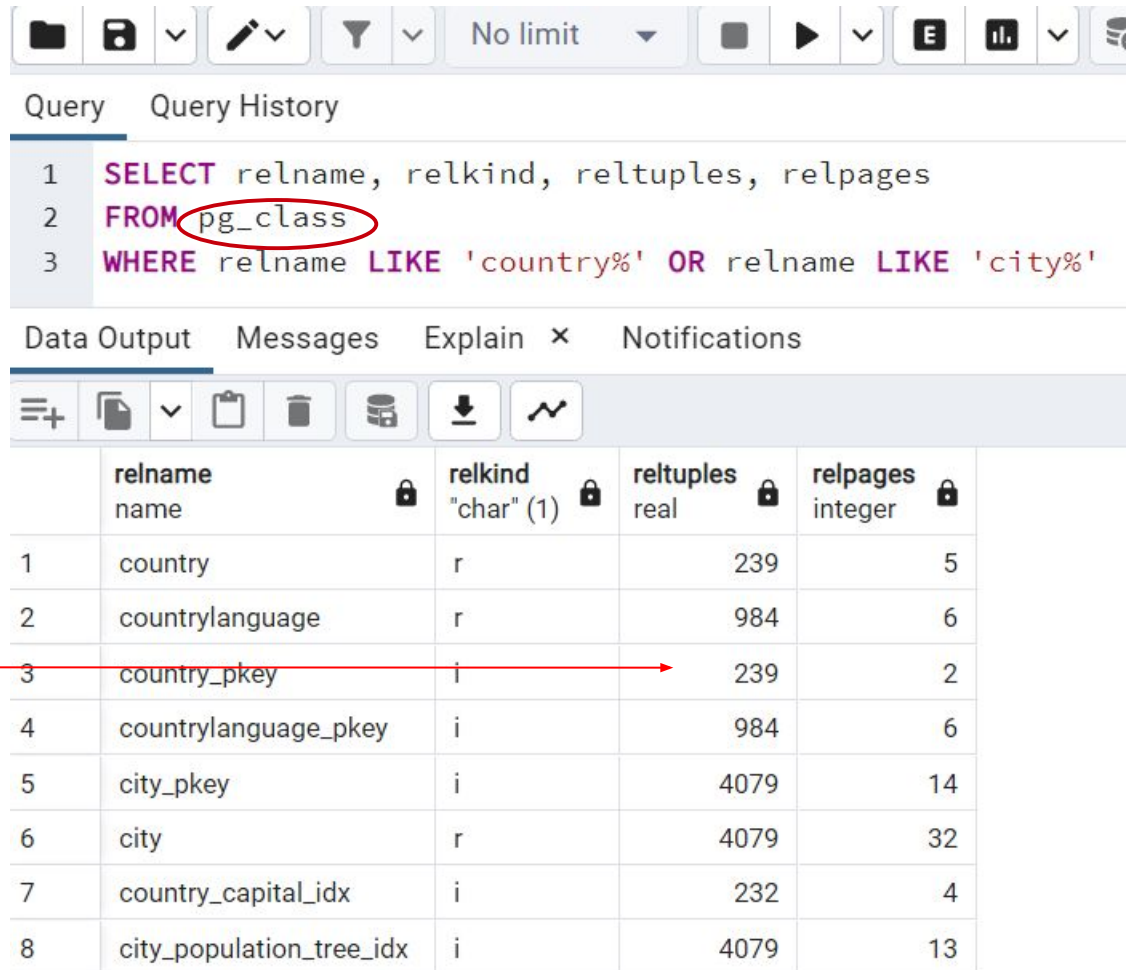
Below the query is a toolbar with icons for data output, messages, explain, and notifications. The 'Data Output' tab is active, displaying the 'QUERY PLAN' section. The query plan is as follows:

	QUERY PLAN
1	GroupAggregate (cost=0.56..113741.69 rows=4079 width=21) (actual time=0.910..1079.635 rows=4079 loops=1)
2	Group Key: c1.id
3	-> Nested Loop (cost=0.56..112668.12 rows=204515 width=13) (actual time=0.264..1059.831 rows=308883 loops=1)
4	-> Index Scan using city_pkey on city c1 (cost=0.28..152.47 rows=4079 width=21) (actual time=0.016..1.440 rows=4079 loops=1)
5	-> Index Scan using city_population_tree_idx on city c2 (cost=0.28..27.52 rows=6 width=8) (actual time=0.033..0.255 rows=76 loops=4...)
6	Index Cond: (population <= c1.population)
7	Filter: (c1.countrycode = countrycode)
8	Rows Removed by Filter: 1964
9	Planning Time: 0.140 ms
10	Execution Time: 1080.384 ms

Χρήση στατιστικών

Ο Κατάλογος κρατά στατιστική πληροφορία την οποία αξιοποιεί ο βελτιστοποιητής για να εκτιμήσει το κόστος κάθε σχεδίου υπολογισμού ενός αιτήματος.

Ο πίνακας καταλόγου
pg_class περιέχει
πληροφορίες για το
πλήθος των πλειάδων
και των σελίδων στο
δίσκο για κάθε
σχέση/πίνακα



The screenshot shows a database query interface. At the top, there is a toolbar with icons for file operations, query execution, and settings. Below the toolbar, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query:

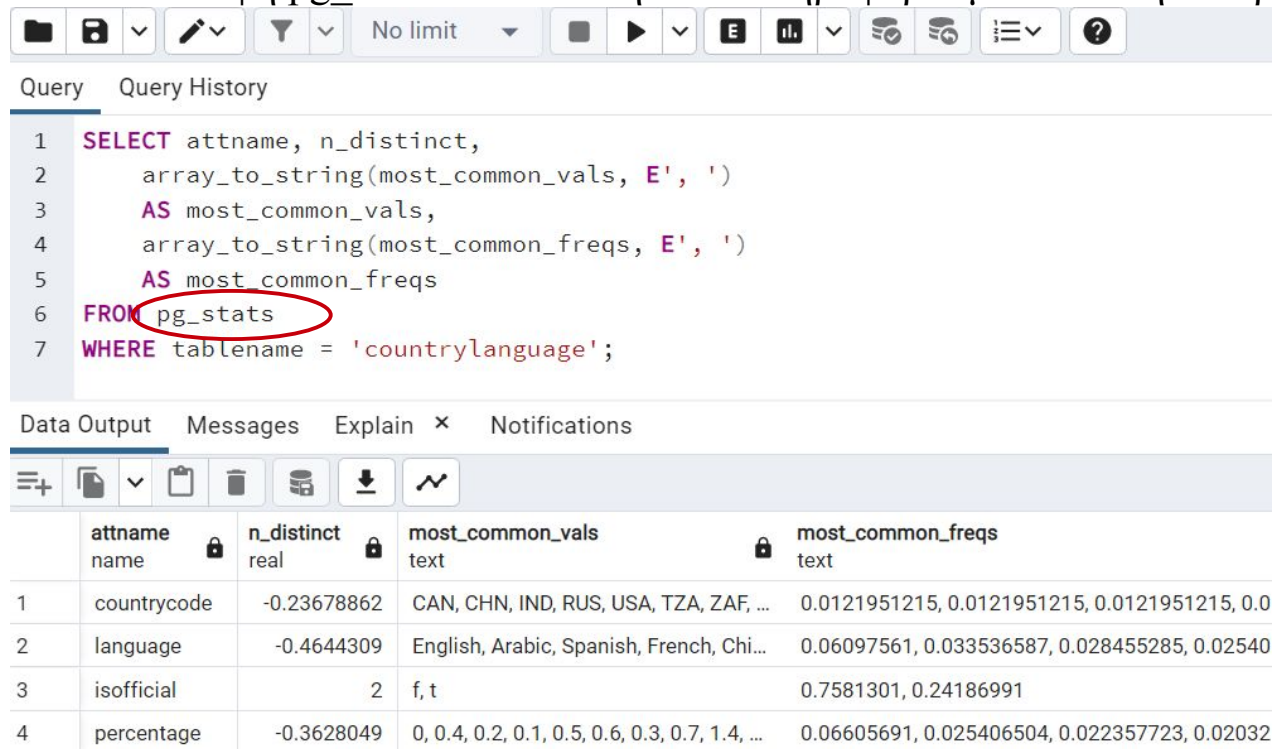
```
1 SELECT relname, relkind, reltuples, relpages
2 FROM pg_class
3 WHERE relname LIKE 'country%' OR relname LIKE 'city%'
```

Below the query, there are tabs for 'Data Output', 'Messages', 'Explain', and 'Notifications'. The 'Data Output' tab is active, displaying a table of results. The table has four columns: 'relname', 'relkind', 'reltuples', and 'relpages'. The 'relname' column is labeled 'name' and has a lock icon. The 'relkind' column is labeled '"char" (1)' and has a lock icon. The 'reltuples' column is labeled 'real' and has a lock icon. The 'relpages' column is labeled 'integer' and has a lock icon. The table contains 8 rows of data. A red arrow points from the text 'country_pkey' in the red text block to the row with 'country_pkey' in the table.

	relname name	relkind "char" (1)	reltuples real	relpages integer
1	country	r	239	5
2	countrylanguage	r	984	6
3	country_pkey	i	239	2
4	countrylanguage_pkey	i	984	6
5	city_pkey	i	4079	14
6	city	r	4079	32
7	country_capital_idx	i	232	4
8	city_population_tree_idx	i	4079	13

Στατιστικά χρήσιμα για εκτίμηση επιλεκτικότητας (1/2)

Στον πίνακα καταλόγου pg_statistic υπάρχουν προσεγγιστικά στατιστικά για τις πιο συχνές τιμές σε κάθε πεδίο κάθε πίνακα. Η όψη pg_stats που δίνει την ίδια πληροφορία με πιο εύληπτο τρόπο.



The screenshot shows a database query interface. At the top is a toolbar with icons for file operations, filters, and execution. Below the toolbar are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query. The query is as follows:

```
1 SELECT attname, n_distinct,  
2     array_to_string(most_common_vals, E', '  
3 AS most_common_vals,  
4     array_to_string(most_common_freqs, E', '  
5 AS most_common_freqs  
6 FROM pg_stats  
7 WHERE tablename = 'countrylanguage';
```

The 'pg_stats' table name in the FROM clause is circled in red. Below the query editor are tabs for 'Data Output', 'Messages', 'Explain', and 'Notifications'. The 'Data Output' tab is active, showing the results of the query in a table format. The table has four columns: attname, n_distinct, most_common_vals, and most_common_freqs. The results are as follows:

	attname name	n_distinct real	most_common_vals text	most_common_freqs text
1	countrycode	-0.23678862	CAN, CHN, IND, RUS, USA, TZA, ZAF, ...	0.0121951215, 0.0121951215, 0.0121951215, 0.0
2	language	-0.4644309	English, Arabic, Spanish, French, Chi...	0.06097561, 0.033536587, 0.028455285, 0.02540
3	isofficial	2	f, t	0.7581301, 0.24186991
4	percentage	-0.3628049	0, 0.4, 0.2, 0.1, 0.5, 0.6, 0.3, 0.7, 1.4, ...	0.06605691, 0.025406504, 0.022357723, 0.02032

Πεδίο n_distinct:

- >0 : εκτιμώμενο πλήθος διακριτών τιμών πεδίου
- <0 : (αρνητικός) λόγος του πλήθους των τιμών του πεδίου προς το πλήθος των εγγραφών του πίνακα

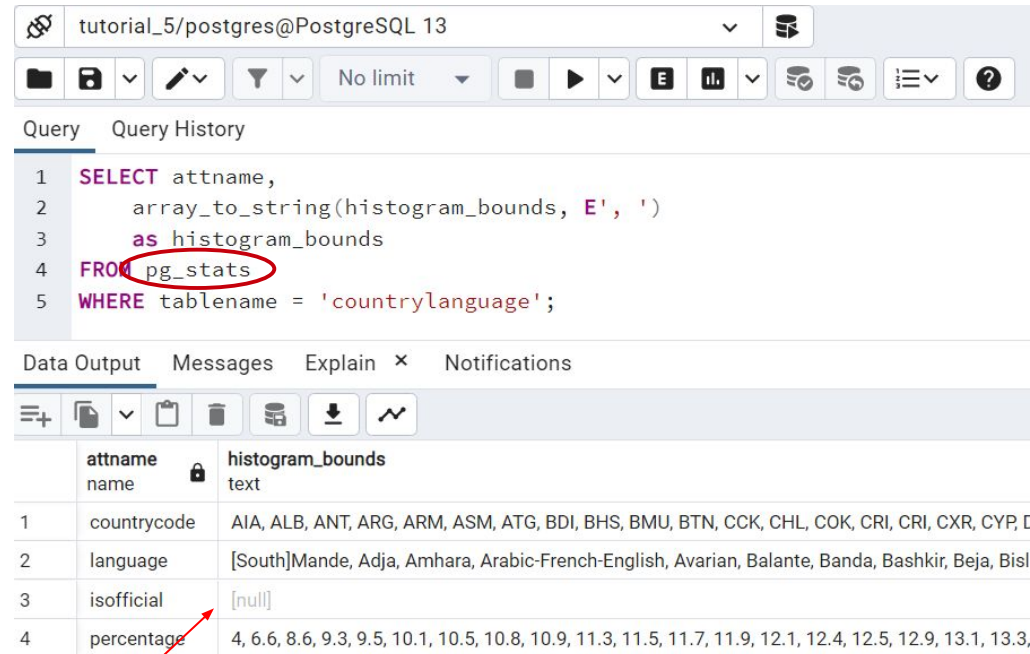
Η όψη καταλόγου pg_stats περιέχει πληροφορίες για τις πιο συχνές τιμές κάθε πεδίου

Στατιστικά χρήσιμα για εκτίμηση επιλεκτικότητας (2/2)

Στον πίνακα καταλόγου `pg_statistic` υπάρχουν και τα προσεγγιστικά ιστογράμματα τιμών για κάθε πεδίο κάθε πίνακα. Η όψη `pg_stats` δίνει και αυτή την πληροφορία με πιο εύληπτο τρόπο.

Η λίστα των τιμών του ιστογράμματος διαμερίζουν το σύνολο των τιμών του πεδίου σε ομάδες με ίδιο πληθυσμό τιμών. Εφόσον υπάρχουν και συχνές τιμές (`most_common_vals`), δεν λαμβάνονται υπόψη στον υπολογισμό του ιστογράμματος.

Παίρνει την τιμή `NULL` αν ο τύπος του πεδίου δεν υποστηρίζει τελεστή `<` καθώς και αν οι συχνές τιμές έχουν ήδη καλύψει όλο τον πληθυσμό τιμών.



The screenshot shows a PostgreSQL query interface with the following query:

```
1 SELECT attname,  
2        array_to_string(histogram_bounds, ' ', ')  
3        as histogram_bounds  
4 FROM pg_stats  
5 WHERE tablename = 'countrylanguage';
```

The results are displayed in a table with two columns: `attname` and `histogram_bounds`.

	attname	histogram_bounds
1	countrycode	AIA, ALB, ANT, ARG, ARM, ASM, ATG, BDI, BHS, BMU, BTN, CCK, CHL, COK, CRI, CRI, CXR, CYP, D
2	language	[South]Mande, Adja, Amhara, Arabic-French-English, Avarian, Balante, Banda, Bashkir, Beja, Bisl
3	isofficial	[null]
4	percentage	4, 6.6, 8.6, 9.3, 9.5, 10.1, 10.5, 10.8, 10.9, 11.3, 11.5, 11.7, 11.9, 12.1, 12.4, 12.5, 12.9, 13.1, 13.3,

Η όψη καταλόγου `pg_stats` περιέχει πληροφορίες για τα ιστογράμματα των τιμών κάθε πεδίου

Ρύθμιση ακρίβειας στατιστικών

- **default_statistics_target**

Μεταβλητή συστήματος (configuration variable) που προσδιορίζει το μέγιστο πλήθος τιμών στα `most_common_vals` και `histogram_bounds`

- Η προκαθορισμένη τιμή αυτής της μεταβλητής είναι 100. Ο διαχειριστής μπορεί να την τροποποιήσει με την εντολή:
`SET default_statistics_target = 10;`

- **ALTER TABLE SET STATISTICS**

Επιτρέπει στο διαχειριστή την αλλαγή του ορίου για συγκεκριμένο πίνακα/πεδίο.

- Έχει αποδειχθεί ότι η χρήση πολύ αναλυτικών στατιστικών δεν βελτιώνει την εκτίμηση κόστους καθώς οι εκτιμήσεις εμπεριέχουν σφάλματα. Πώς μπορείτε να αξιοποιήσετε αυτό το γεγονός για να αποφύγετε την τήρηση περιττών στατιστικών;

Ενημέρωση στατιστικών

Η ενημέρωση των στατιστικών επηρεάζει το παραγόμενο σχέδιο εκτέλεσης από το βελτιστοποιητή

- Για την ενημέρωση των στατιστικών όλης της βάσης:

VACUUM ANALYSE

- Ενημέρωση στατιστικών για συγκεκριμένο πίνακα:

VACUUM ANALYSE city

- Εκτός από την ενημέρωση των στατιστικών γίνεται και συλλογή απορριμμάτων (garbage collection). Για παράδειγμα αποδεσμεύεται χώρος που αφορά πλειάδες που έχουν διαγραφεί.

Σημαντικά κεφάλαια εγχειριδίου αναφοράς PostgreSQL (version 13)

- **Κεφ. 14 – Performance Tips**

Αναλυτική περιγραφή χρήσης του `explain` και του `explain analyse`, γενική αναφορά στη χρήση στατιστικών από το βελτιστοποιητή, έλεγχος σειράς συνδέσεων (`joins`), φόρτωση μεγάλου όγκου δεδομένων.

- **Ενότητα 19.7 – Query Planning**

Περιγράφει όλες τις παραμέτρους (`configuration parameters`) που μπορεί να μεταβάλει ο διαχειριστής μιας βάσης για να αλλάξει το υπολογιζόμενο βέλτιστο σχέδιο υπολογισμού.

- **Κεφάλαιο 71 – How the Planner Uses Statistics**

Περιγράφει πώς από τα στατιστικά υπολογίζεται η εκτίμηση του κόστους υπολογισμού ερωτήσεων (π.χ. Πώς γίνεται η εκτίμηση της επιλεκτικότητας μια συνθήκης, του μεγέθους του αποτελέσματος μια ερώτησης κ.λ.π.)