

Επεξεργασία και βελτιστοποίηση αιτημάτων στην PostgreSQL

Φροντιστήριο στο μάθημα
«Βάσεις Δεδομένων»

Βάσεις Δεδομένων
Άνοιξη 2018 - Πολυτεχνείο Κρήτης
Νεκτάριος Μουμουτζής

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

Κατάλογος συστήματος

- Καταχώρηση πληροφοριών για κάθε πίνακα και κάθε ευρετήριο καθώς και για τις όψεις, τις συναρτήσεις κ.λ.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

pgAdmin 4 File Object Tools Help

Browser

Servers (2)

TUC MUSIC v10

Databases (209)

aDbForShowingQueryProcessing

Casts

Catalogs (2)

ANSI (information_schem...

PostgreSQL Catalog (pg_c...

Collations

Domains

FTS Configurations

FTS Dictionaries

FTS Parsers

FTS Templates

Foreign Tables

Functions

Materialized Views

Sequences

Tables

Trigger Functions

Types

Views

Event Triggers

Extensions

Foreign Data Wrappers

Languages

Schemas (1)

public

Dashboard Properties SQL Statistics Dependencies Dependents Query - aDbForS...

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

The screenshot displays the pgAdmin 4 web interface. On the left, the 'Browser' pane shows a tree structure of database objects for 'TUC MUSIC v10', including 'Databases (209)', 'aDbForShowingQueryProcessing', 'Casts', 'Catalogs (2)', and various system catalogs. The 'Tables' folder is selected. The main pane shows a SQL query executed on the 'aDbForShowingQueryProcessing' database. The query is:

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

Below the query editor, the 'Data Output' tab is active, displaying the results of the query in a table format. The table has five columns: 'relname', 'relkind', 'reltuples', and 'relpages'. The results are as follows:

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

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

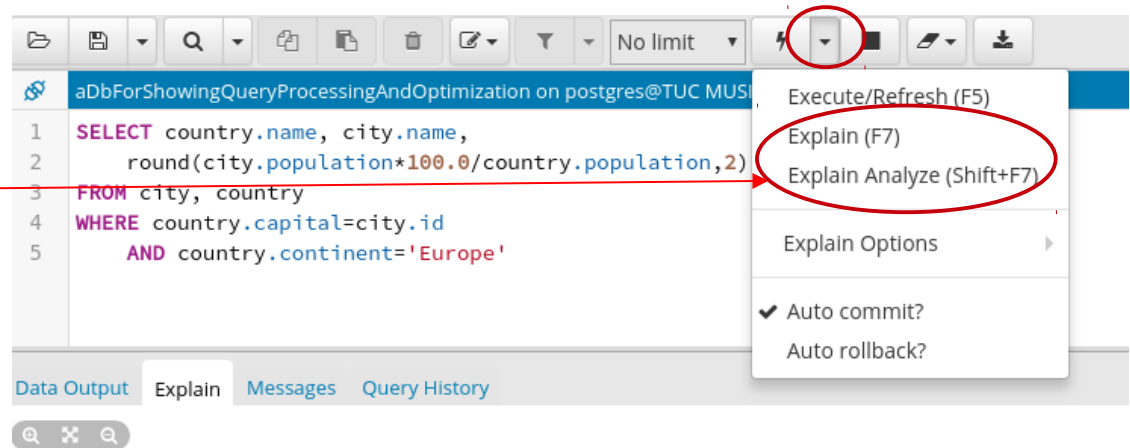
aDbForShowingQueryProcessingAndOptimization on postgres@TUC MUSIC v10	
1	EXPLAIN SELECT country.name, city.name,
2	round(city.population*100.0/country.population,2)
3	FROM city, country
4	WHERE country.capital=city.id
5	AND country.continent='Europe'
Data Output Explain Messages Query History	
QUERY 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:

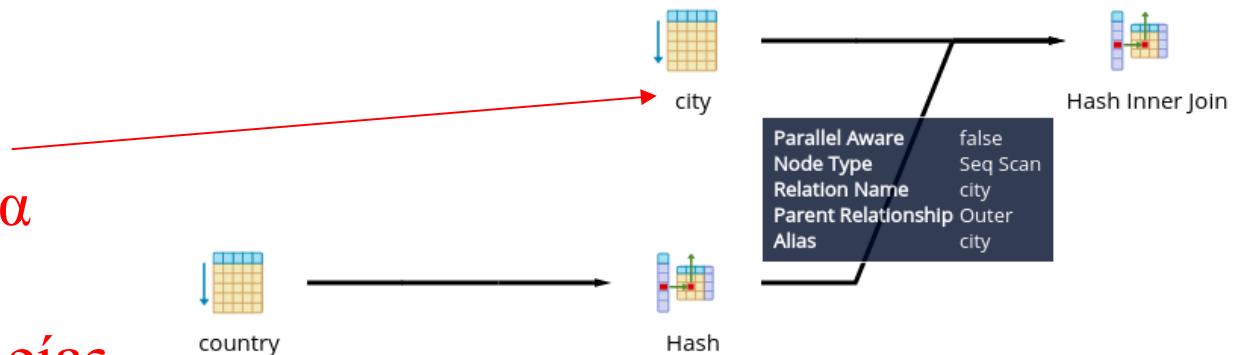
- Χρόνο υπολογισμού πρώτης πλειάδας
- Χρόνο ολοκλήρωσης υπολογισμού
- Πλήθος πλειάδων αποτελέσματος
- Μέγεθος κάθε πλειάδας του αποτελέσματος

Γραφική απεικόνιση σχεδίων

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



Περνώντας με το
ποντίκι πάνω από ένα
κόμβο, βλέπουμε
αναλυτικές πληροφορίες



Η εντολή EXPLAIN ANALYSE

aDbForShowingQueryProcessingAndOptimization on postgres@TUC MUSIC v10	
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
Data Output Explain Messages Query History	
	QUERY PLAN
	text
1	Sort (cost=98.94..99.06 rows=46 width=52) (actual time=7.162..7.194 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=0.254..6.854 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=0.024..3.274 rows=4079 loops=1)
7	-> Hash (cost=7.99..7.99 rows=46 width=19) (actual time=0.184..0.184 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=0.020..0.130 rows=46 loops=1)
10	Filter: (continent = 'Europe'::text)
11	Rows Removed by Filter: 193
12	Planning time: 0.404 ms
13	Execution time: 7.284 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.

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

```

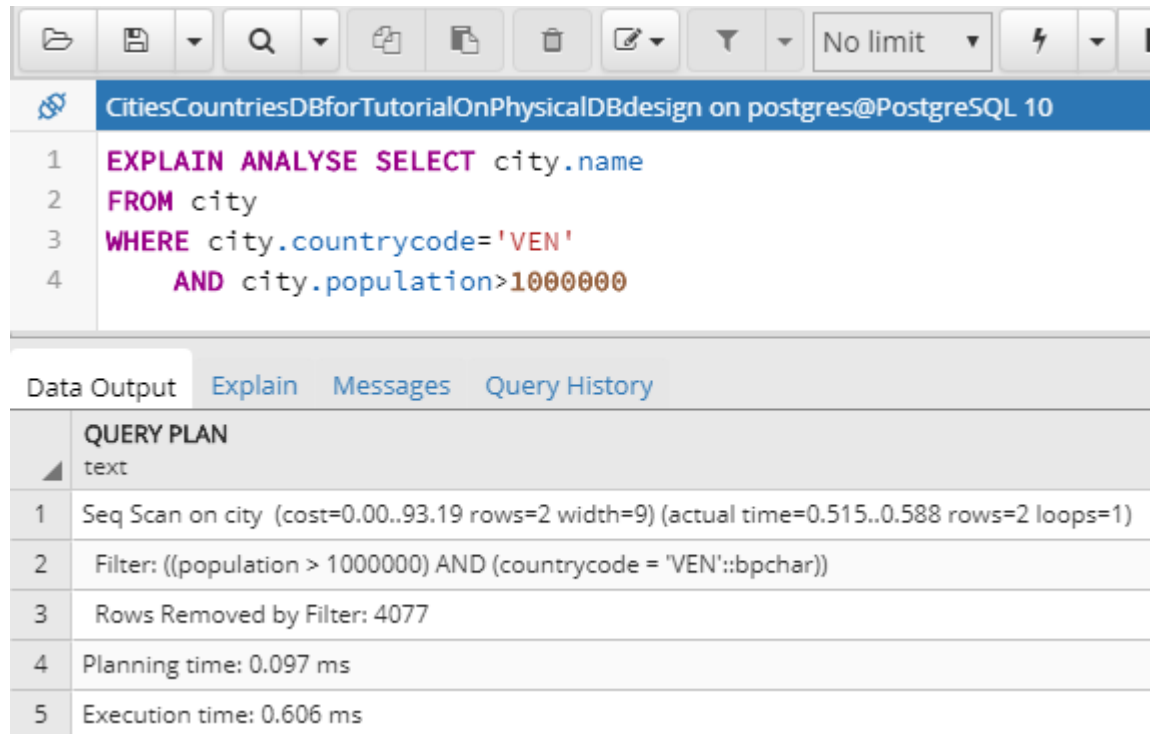
Η επιλογή BUFFERS

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

Data Output	Explain	Messages	Query History
QUERY PLAN text			
1	Sort (cost=98.94..99.06 rows=46 width=52) (actual time=1.468..1.470 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=0.428..1.343 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=0.025..0.411 rows=4079 loops=1)		
9	Buffers: shared hit=32		
10	-> Hash (cost=7.99..7.99 rows=46 width=19) (actual time=0.080..0.080 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=0.014..0.065 rows=46 loops=1)		
14	Filter: (continent = 'Europe'::text)		
15	Rows Removed by Filter: 193		
16	Buffers: shared hit=5		
17	Planning time: 0.952 ms		
18	Execution time: 1.765 ms		

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 top toolbar includes icons for file operations, search, and execution. The query text is as follows:

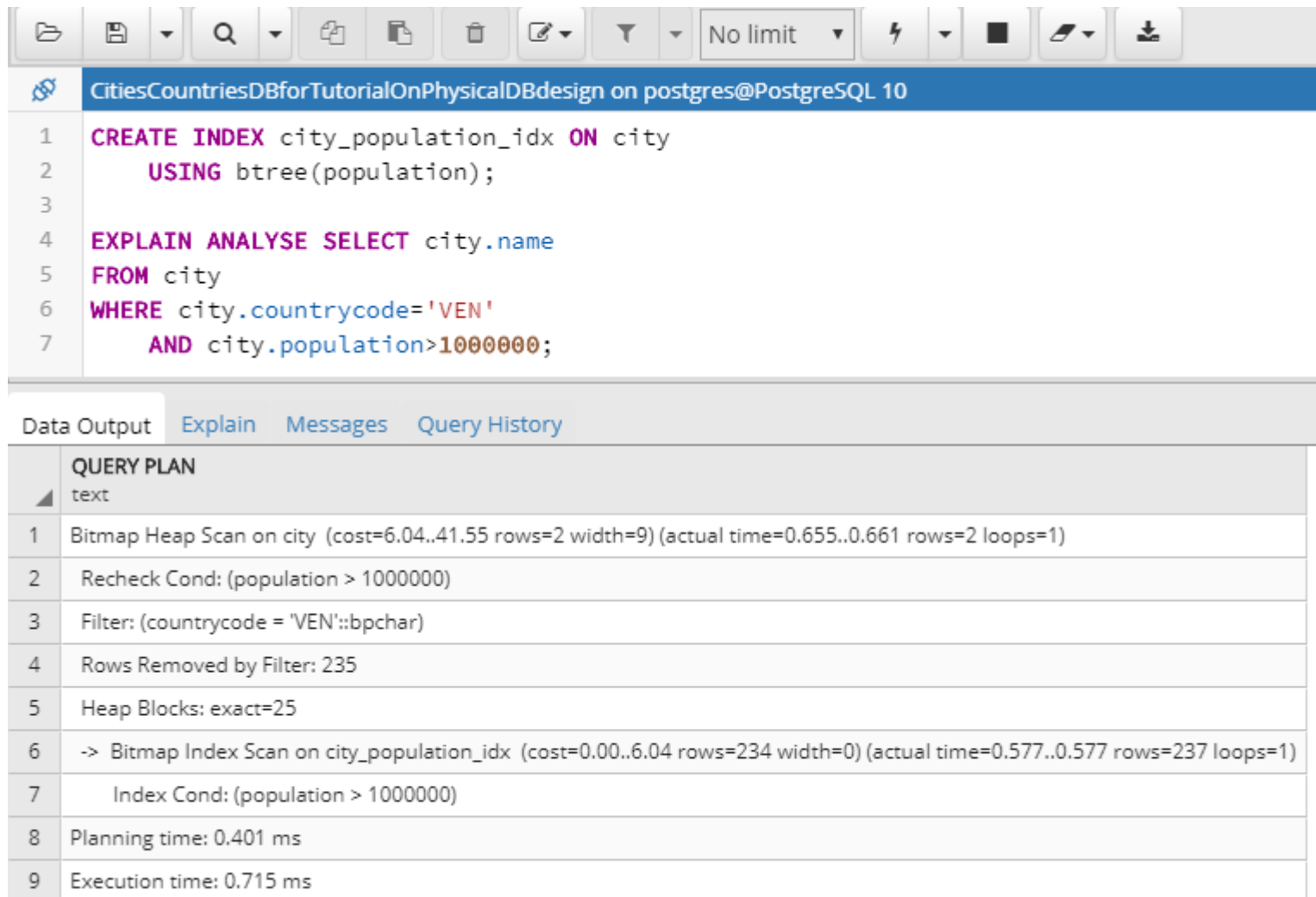
```
1 EXPLAIN ANALYSE SELECT city.name
2 FROM city
3 WHERE city.countrycode='VEN'
4 AND city.population>1000000
```

Below the query editor, there are tabs for "Data Output", "Explain", "Messages", and "Query History". The "Explain" tab is selected, displaying the query plan:

QUERY PLAN	
1	Seq Scan on city (cost=0.00..93.19 rows=2 width=9) (actual time=0.515..0.588 rows=2 loops=1)
2	Filter: ((population > 1000000) AND (countrycode = 'VEN'::bpchar))
3	Rows Removed by Filter: 4077
4	Planning time: 0.097 ms
5	Execution time: 0.606 ms

Δενδρικό ευρετήριο στο population

- Ευρετήριο κατακερματισμού δεν θα χρησίμευε. (Γιατί;)



The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is to 'CitiesCountriesDBforTutorialOnPhysicalDBdesign on postgres@PostgreSQL 10'. The query editor contains the following SQL code:

```
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;
```

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

Step	Operation
1	Bitmap Heap Scan on city (cost=6.04..41.55 rows=2 width=9) (actual time=0.655..0.661 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=0.577..0.577 rows=237 loops=1)
7	Index Cond: (population > 1000000)
8	Planning time: 0.401 ms
9	Execution time: 0.715 ms

Ευρετήριο κατακερμ. στο countrycode

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

CitiesCountriesDBforTutorialOnPhysicalDBdesign on postgres@PostgreSQL 10

```
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;
```

Data Output Explain Messages Query History

	QUERY PLAN
	text
1	Bitmap Heap Scan on city (cost=4.31..38.63 rows=2 width=9) (actual time=0.022..0.030 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=0.012..0.012 rows=41 loops=1)
7	Index Cond: (countrycode = 'VEN'::bpchar)
8	Planning time: 0.357 ms
9	Execution time: 0.067 ms

Επιπλέον εναλλακτικές...

- **Συνδυασμός ευρετηρίων**

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

- **Ευρετήριων πολλών χαρακτηριστικών**

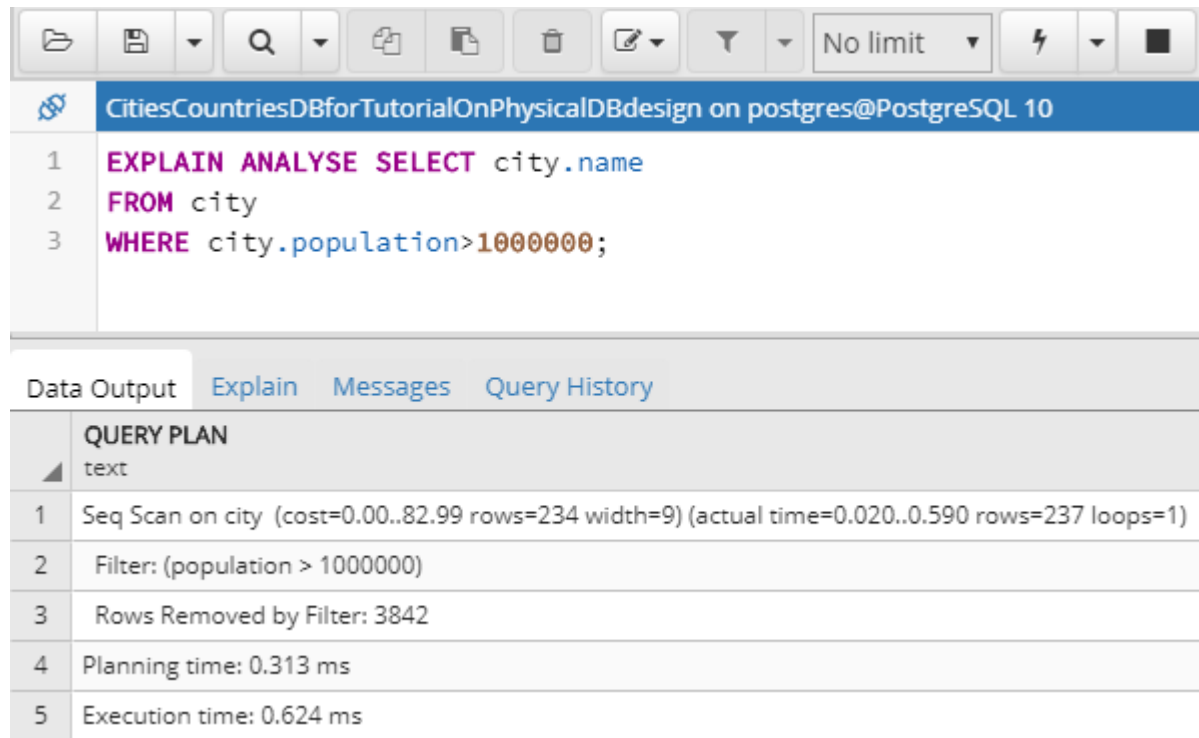
Χρήση δενδρικού (γιατί;) ευρετηρίου στα δύο χαρακτηριστικά countrycode και population (ποιο πρώτο και ποιο δεύτερο;)

- **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. The query being executed is:

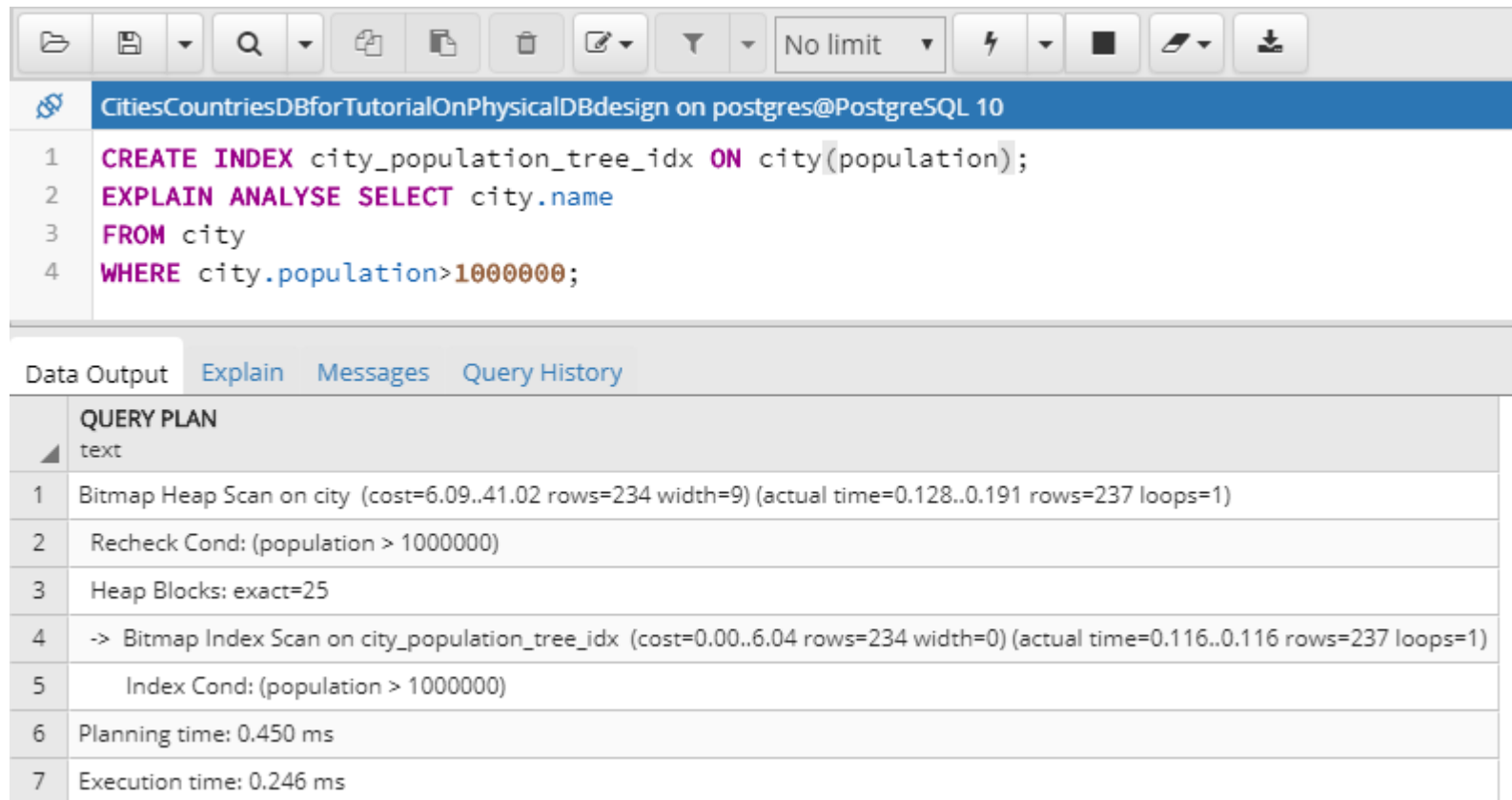
```
1 EXPLAIN ANALYSE SELECT city.name
2 FROM city
3 WHERE city.population > 1000000;
```

Below the query, the 'Data Output' tab is selected, displaying the 'QUERY PLAN' for the 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=0.020..0.590 rows=237 loops=1)
2	Filter: (population > 1000000)
3	Rows Removed by Filter: 3842
4	Planning time: 0.313 ms
5	Execution time: 0.624 ms

Ομαδοποίηση (Clustering) (2/3)

- Δημιουργούμε ένα δενδρικό ευρετήριο στο population



The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is to 'CitiesCountriesDBforTutorialOnPhysicalDBdesign on postgres@PostgreSQL 10'. The query editor contains the following SQL code:

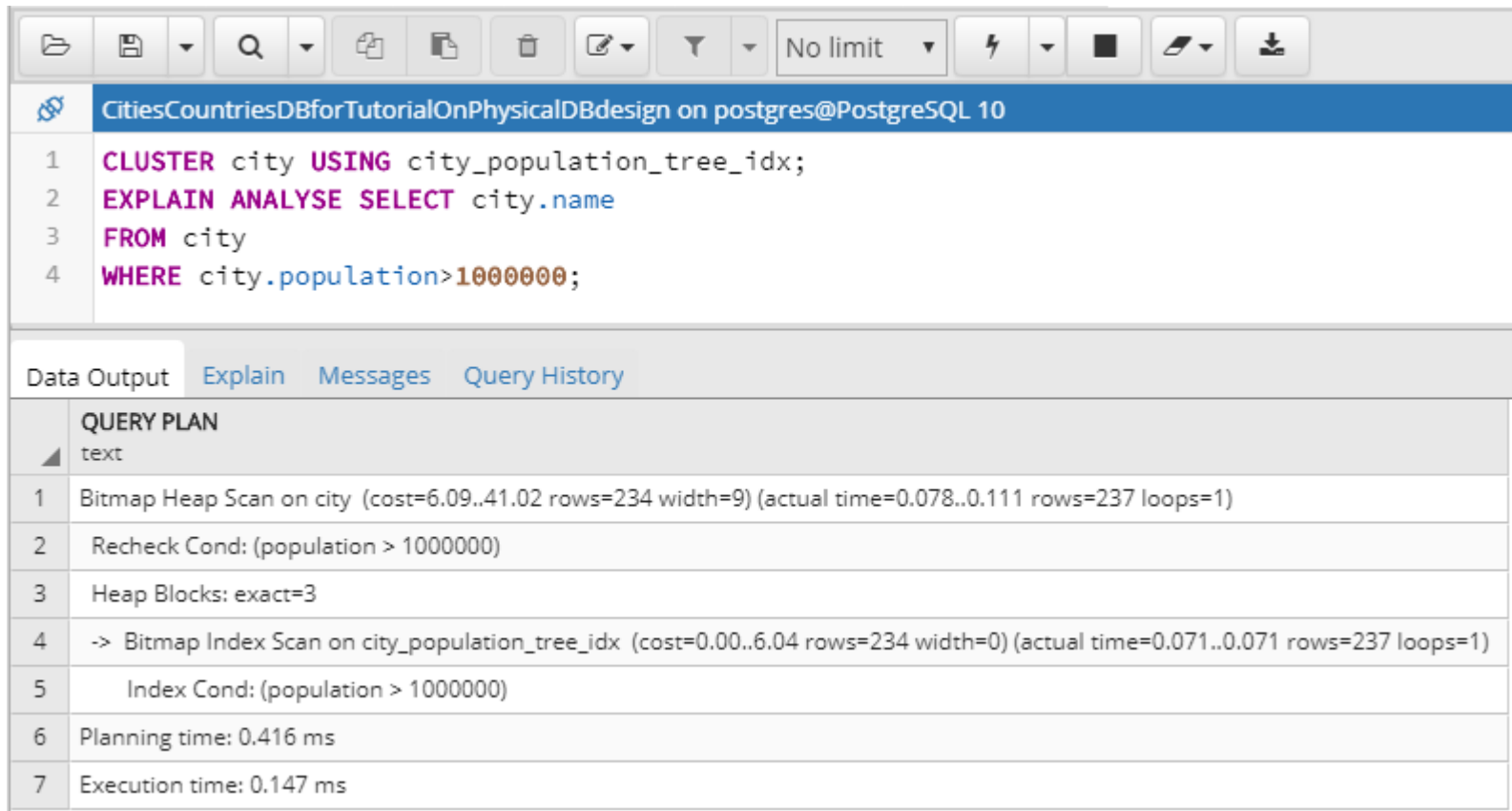
```
1 CREATE INDEX city_population_tree_idx ON city(population);
2 EXPLAIN ANALYSE SELECT city.name
3 FROM city
4 WHERE city.population > 1000000;
```

Below the query editor, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Query History'. The 'Explain' tab is selected, displaying the query plan for the executed query. The plan shows a Bitmap Heap Scan on the 'city' table, followed by a Recheck Cond on the 'population' column, and then a Bitmap Index Scan on the 'city_population_tree_idx' index. The plan also includes timing information for planning and execution.

	QUERY PLAN
1	Bitmap Heap Scan on city (cost=6.09..41.02 rows=234 width=9) (actual time=0.128..0.191 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=0.116..0.116 rows=237 loops=1)
5	Index Cond: (population > 1000000)
6	Planning time: 0.450 ms
7	Execution time: 0.246 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 query being executed is:

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

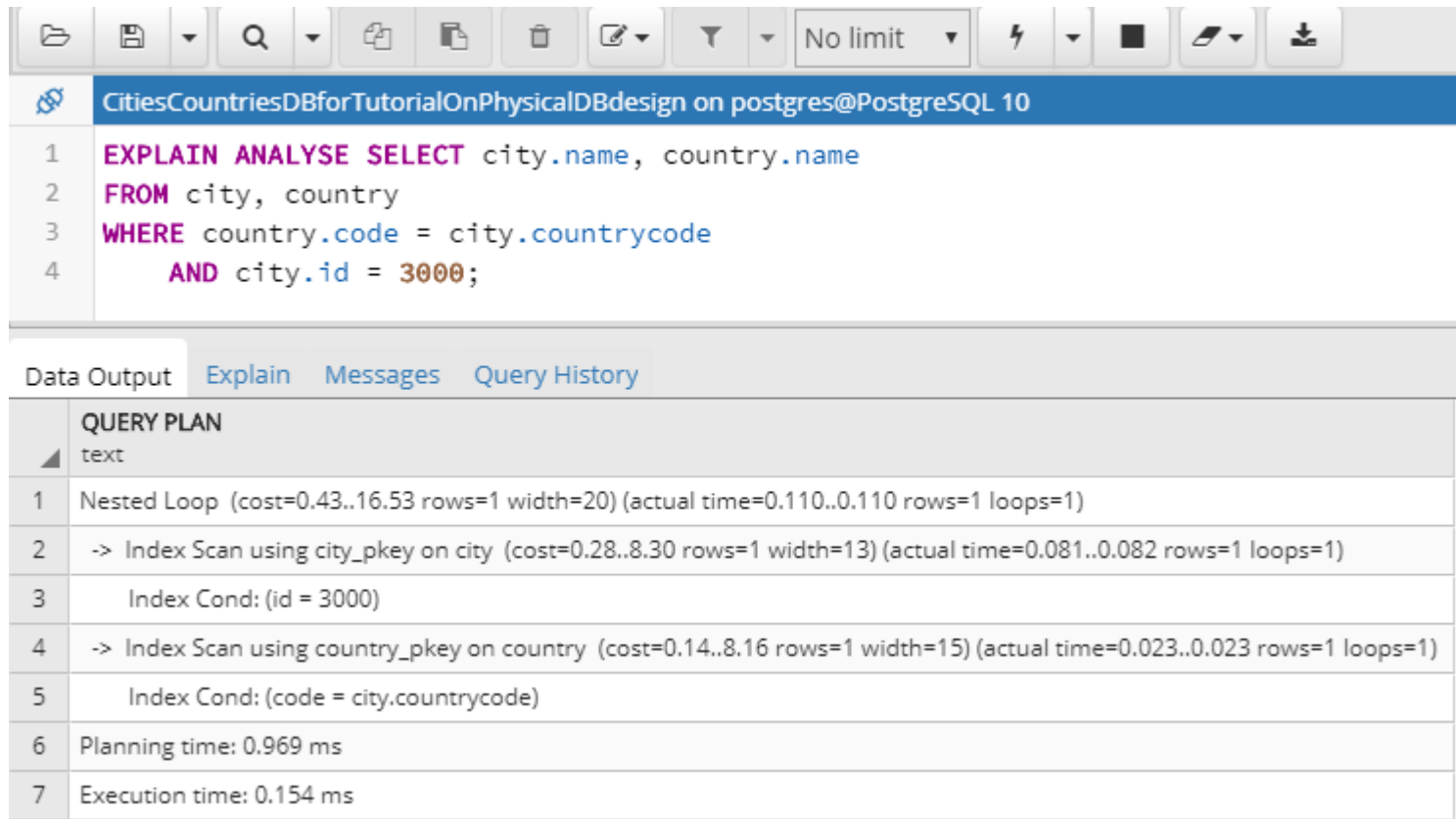
The results tab shows the query plan:

	QUERY PLAN
1	Bitmap Heap Scan on city (cost=6.09..41.02 rows=234 width=9) (actual time=0.078..0.111 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=0.071..0.071 rows=237 loops=1)
5	Index Cond: (population > 1000000)
6	Planning time: 0.416 ms
7	Execution time: 0.147 ms

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

Βελτίωση απόδοσης συνδέσεων (1/3)

- Τα κύρια κλειδιά έχουν ευρετήρια



The screenshot shows a PostgreSQL query editor interface. The query being executed is:

```
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, the 'Data Output' tab is selected, showing the 'QUERY PLAN' for the query. The plan details are as follows:

	QUERY PLAN
1	Nested Loop (cost=0.43..16.53 rows=1 width=20) (actual time=0.110..0.110 rows=1 loops=1)
2	-> Index Scan using city_pkey on city (cost=0.28..8.30 rows=1 width=13) (actual time=0.081..0.082 rows=1 loops=1)
3	Index Cond: (id = 3000)
4	-> Index Scan using country_pkey on country (cost=0.14..8.16 rows=1 width=15) (actual time=0.023..0.023 rows=1 loops=1)
5	Index Cond: (code = city.countrycode)
6	Planning time: 0.969 ms
7	Execution time: 0.154 ms

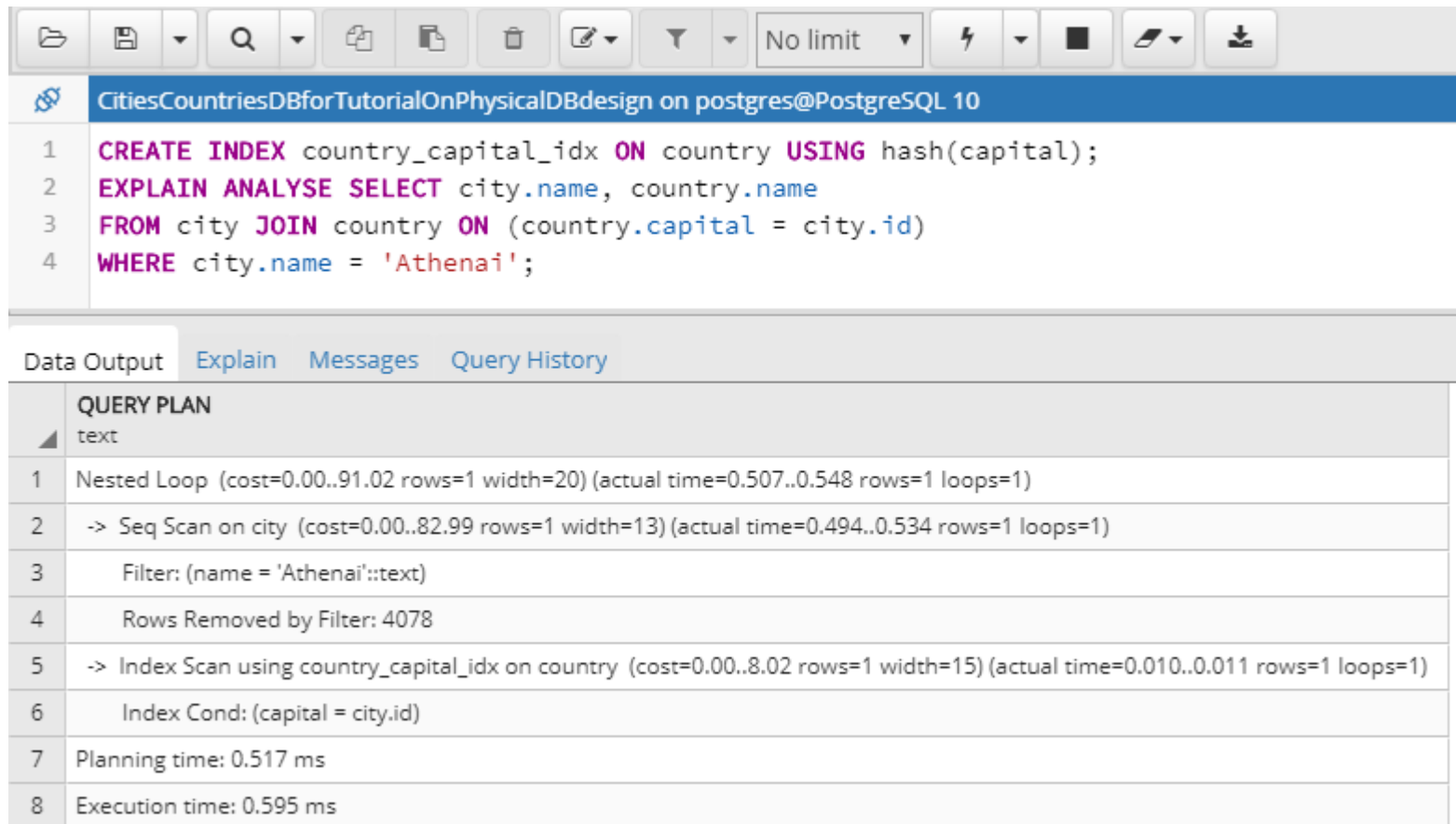
Βελτίωση απόδοσης συνδέσεων (2/3)

- Τα ξένα κλειδιά δεν έχουν ευρετήρια.

CitiesCountriesDBforTutorialOnPhysicalDBdesign on postgres@PostgreSQL 10	
1	EXPLAIN ANALYSE SELECT city.name, country.name
2	FROM city JOIN country ON (country.capital = city.id)
3	WHERE city.name = 'Athenai';
Data Output Explain Messages Query History	
	QUERY PLAN
	text
1	Nested Loop (cost=0.00..93.37 rows=1 width=20) (actual time=0.569..0.666 rows=1 loops=1)
2	Join Filter: (city.id = country.capital)
3	Rows Removed by Join Filter: 238
4	-> Seq Scan on city (cost=0.00..82.99 rows=1 width=13) (actual time=0.514..0.554 rows=1 loops=1)
5	Filter: (name = 'Athenai':text)
6	Rows Removed by Filter: 4078
7	-> Seq Scan on country (cost=0.00..7.39 rows=239 width=15) (actual time=0.017..0.047 rows=239 loops=1)
8	Planning time: 0.347 ms
9	Execution time: 0.697 ms

Βελτίωση απόδοσης συνδέσεων (3/3)

- Δημιουργία νέου ευρετηρίου για το ξένο κλειδί...



The screenshot shows a PostgreSQL query editor interface. The query being executed is:

```
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, the 'Data Output' tab is selected, showing the 'QUERY PLAN' for the query. The plan details are as follows:

	QUERY PLAN
1	Nested Loop (cost=0.00..91.02 rows=1 width=20) (actual time=0.507..0.548 rows=1 loops=1)
2	-> Seq Scan on city (cost=0.00..82.99 rows=1 width=13) (actual time=0.494..0.534 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=0.010..0.011 rows=1 loops=1)
6	Index Cond: (capital = city.id)
7	Planning time: 0.517 ms
8	Execution time: 0.595 ms

Ελέγχοντας τη σειρά των συνδέσεων (1/2)

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

CitiesCountriesDBforTutorialOnPhysicalDBdesign on postgres@PostgreSQL 10	
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';
Data Output Explain Messages Query History	
	QUERY PLAN
	text
1	Nested Loop (cost=4.31..21.06 rows=70 width=17) (actual time=0.067..0.176 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=0.051..0.093 rows=6 loops=1)
4	-> Seq Scan on country (cost=0.00..7.99 rows=1 width=4) (actual time=0.017..0.053 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=0.025..0.030 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=0.018..0.018 rows=6 loops=1)
11	Index Cond: (countrycode = country.code)
12	-> Index Scan using city_country_idx on city (cost=0.00..0.53 rows=18 width=13) (actual time=0.003..0.011 rows=9 loops=6)
13	Index Cond: (countrycode = countrylanguage.countrycode)
14	Planning time: 1.514 ms
15	Execution time: 0.240 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`

CitiesCountriesDBforTutorialOnPhysicalDBdesign on postgres@PostgreSQL 10	
1	<code>set join_collapse_limit = 1;</code>
2	<code>EXPLAIN ANALYSE SELECT city.name, countrylanguage.language</code>
3	<code>FROM (country JOIN city ON (country.code=city.countrycode))</code>
4	<code>NATURAL JOIN countrylanguage</code>
5	<code>WHERE country.name = 'Belgium';</code>
Data Output Explain Messages Query History	
QUERY PLAN	
text	
1	Nested Loop (cost=4.41..48.64 rows=70 width=17) (actual time=0.177..0.319 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=0.050..0.100 rows=9 loops=1)
4	-> Seq Scan on country (cost=0.00..7.99 rows=1 width=4) (actual time=0.019..0.054 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=0.023..0.037 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=0.014..0.014 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=0.019..0.022 rows=6 loops=9)
13	Index Cond: (countrycode = city.countrycode)
14	Heap Fetches: 54
15	Planning time: 0.723 ms
16	Execution time: 0.414 ms

Αλλάζοντας τρόπο υπολ. συνδέσεων (1/3)

- Τρεις βασικοί αλγόριθμοι υπολογισμού συνδέσεων (με ή χωρίς ευρετήριο):
nested loop, hash join, merge join

CitiesCountriesDBforTutorialOnPhysicalDBdesign on postgres@PostgreSQL 10	
1	DROP INDEX IF EXISTS city_country_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;
Data Output Explain Messages Query History	
	QUERY PLAN
	text
1	HashAggregate (cost=10699.22..10750.21 rows=4079 width=21) (actual time=189.134..190.118 rows=4079 loops=1)
2	Group Key: c1.id
3	-> Hash Join (cost=123.78..9682.56 rows=203332 width=13) (actual time=1.678..125.341 rows=308883 loops=1)
4	Hash Cond: (c1.countrycode = c2.countrycode)
5	Join Filter: (c1.population >= c2.population)
6	Rows Removed by Join Filter: 304736
7	-> Seq Scan on city c1 (cost=0.00..72.79 rows=4079 width=21) (actual time=0.042..0.849 rows=4079 loops=1)
8	-> Hash (cost=72.79..72.79 rows=4079 width=8) (actual time=1.603..1.603 rows=4079 loops=1)
9	Buckets: 4096 Batches: 1 Memory Usage: 192kB
10	-> Seq Scan on city c2 (cost=0.00..72.79 rows=4079 width=8) (actual time=0.012..0.822 rows=4079 loops=1)
11	Planning time: 0.612 ms
12	Execution time: 190.768 ms

Αλλάζοντας τρόπο υπολ. συνδέσεων (2/3)

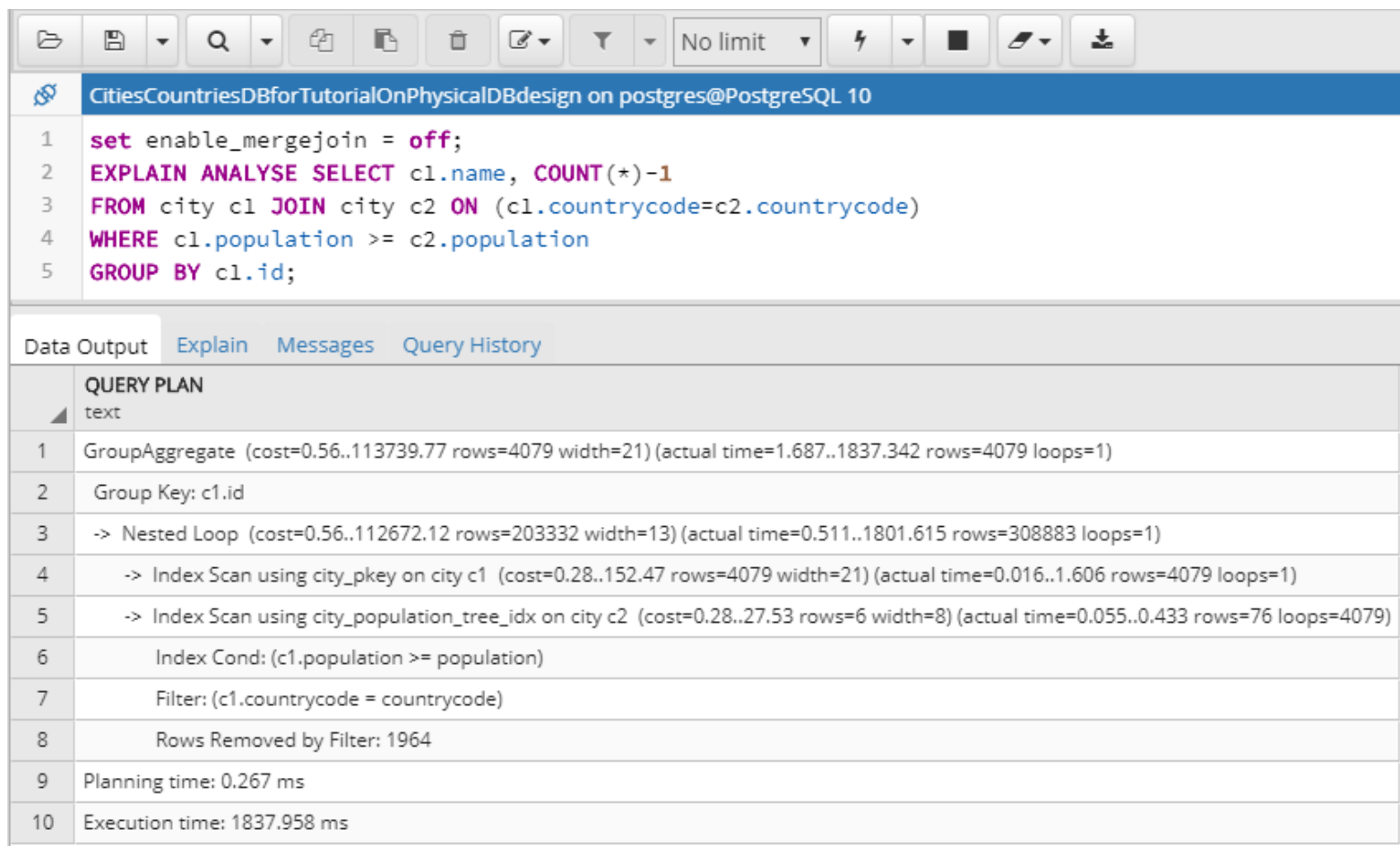
CitiesCountriesDBforTutorialOnPhysicalDBdesign on postgres@PostgreSQL 10	
1	<code>set enable_hashjoin = off;</code>
2	<code>EXPLAIN ANALYSE SELECT c1.name, COUNT(*)-1</code>
3	<code>FROM city c1 JOIN city c2 ON (c1.countrycode=c2.countrycode)</code>
4	<code>WHERE c1.population >= c2.population</code>
5	<code>GROUP BY c1.id;</code>
Data Output Explain Messages Query History	
QUERY PLAN	
text	
1	HashAggregate (cost=12346.80..12397.79 rows=4079 width=21) (actual time=227.229..228.216 rows=4079 loops=1)
2	Group Key: c1.id
3	-> Merge Join (cost=634.82..11330.14 rows=203332 width=13) (actual time=12.331..169.702 rows=308883 loops=1)
4	Merge Cond: (c1.countrycode = c2.countrycode)
5	Join Filter: (c1.population >= c2.population)
6	Rows Removed by Join Filter: 304736
7	-> Sort (cost=317.41..327.61 rows=4079 width=21) (actual time=6.513..7.012 rows=4079 loops=1)
8	Sort Key: c1.countrycode
9	Sort Method: quicksort Memory: 414kB
10	-> Seq Scan on city c1 (cost=0.00..72.79 rows=4079 width=21) (actual time=0.033..0.872 rows=4079 loops=1)
11	-> Sort (cost=317.41..327.61 rows=4079 width=8) (actual time=5.807..35.760 rows=613614 loops=1)
12	Sort Key: c2.countrycode
13	Sort Method: quicksort Memory: 288kB
14	-> Seq Scan on city c2 (cost=0.00..72.79 rows=4079 width=8) (actual time=0.045..0.785 rows=4079 loops=1)
15	Planning time: 0.486 ms
16	Execution time: 229.134 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 PostgreSQL query editor interface. The query being executed is:

```
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;
```

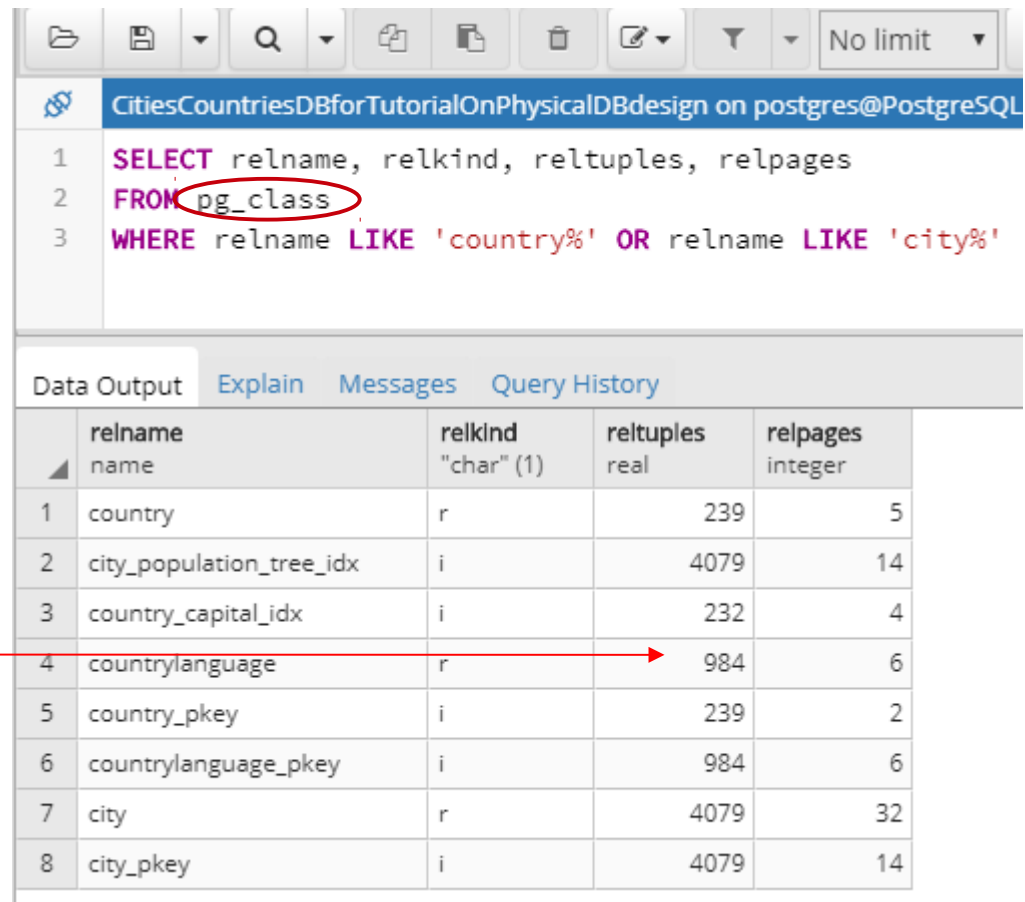
The query plan output is as follows:

	QUERY PLAN
1	GroupAggregate (cost=0.56..113739.77 rows=4079 width=21) (actual time=1.687..1837.342 rows=4079 loops=1)
2	Group Key: c1.id
3	-> Nested Loop (cost=0.56..112672.12 rows=203332 width=13) (actual time=0.511..1801.615 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..1.606 rows=4079 loops=1)
5	-> Index Scan using city_population_tree_idx on city c2 (cost=0.28..27.53 rows=6 width=8) (actual time=0.055..0.433 rows=76 loops=4079)
6	Index Cond: (c1.population >= population)
7	Filter: (c1.countrycode = countrycode)
8	Rows Removed by Filter: 1964
9	Planning time: 0.267 ms
10	Execution time: 1837.958 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 PostgreSQL query window with the following SQL query:

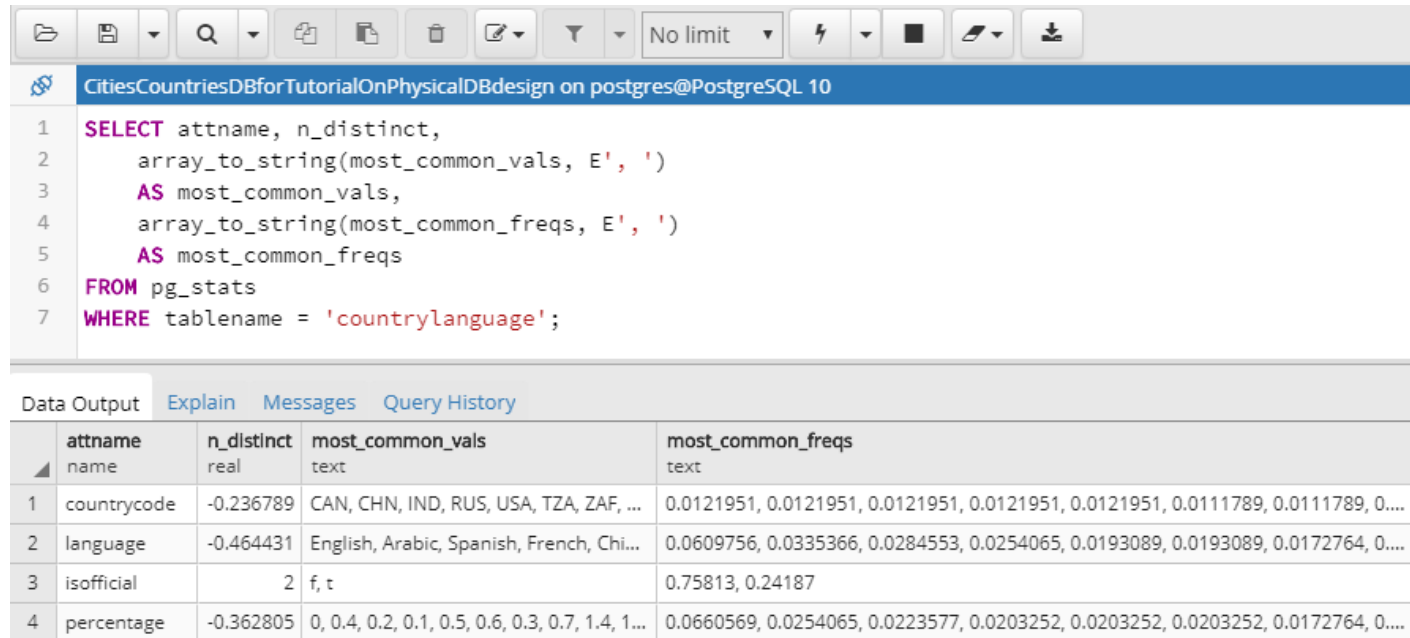
```
1 SELECT relname, relkind, reltuples, relpages
2 FROM pg_class
3 WHERE relname LIKE 'country%' OR relname LIKE 'city%'
```

The query results are displayed in a table with the following columns: relname, relkind, reltuples, and relpages. The results are as follows:

	relname	relkind	reltuples	relpages
1	country	r	239	5
2	city_population_tree_idx	i	4079	14
3	country_capital_idx	i	232	4
4	countrylanguage	r	984	6
5	country_pkey	i	239	2
6	countrylanguage_pkey	i	984	6
7	city	r	4079	32
8	city_pkey	i	4079	14

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.



```
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';
```

	attname name	n_distinct real	most_common_vals text	most_common_freqs text
1	countrycode	-0.236789	CAN, CHN, IND, RUS, USA, TZA, ZAF, ...	0.0121951, 0.0121951, 0.0121951, 0.0121951, 0.0121951, 0.0111789, 0.0111789, 0....
2	language	-0.464431	English, Arabic, Spanish, French, Chi...	0.0609756, 0.0335366, 0.0284553, 0.0254065, 0.0193089, 0.0193089, 0.0172764, 0....
3	isofficial	2	f, t	0.75813, 0.24187
4	percentage	-0.362805	0, 0.4, 0.2, 0.1, 0.5, 0.6, 0.3, 0.7, 1.4, 1...	0.0660569, 0.0254065, 0.0223577, 0.0203252, 0.0203252, 0.0203252, 0.0172764, 0....

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

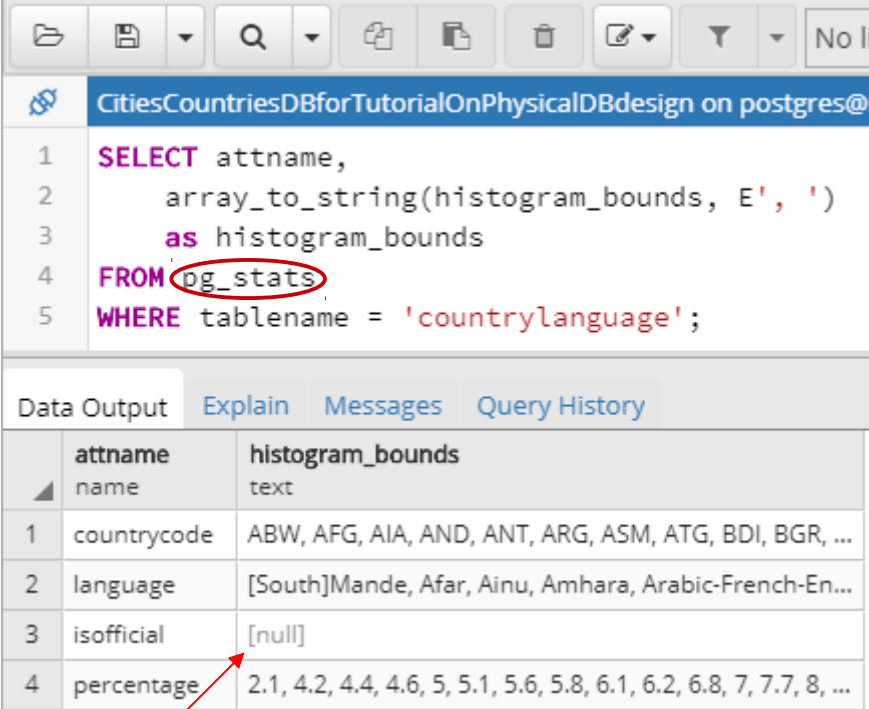
Η όψη καταλόγου `pg_stats` περιέχει πληροφορίες για τις πιο συχνές τιμές κάθε πεδίου

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 editor with the following query:

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

The query results are displayed in a table with the following columns: `attname` and `histogram_bounds`.

	attname	histogram_bounds
1	countrycode	ABW, AFG, AIA, AND, ANT, ARG, ASM, ATG, BDI, BGR, ...
2	language	[South]Mande, Afar, Ainu, Amhara, Arabic-French-En...
3	isofficial	[null]
4	percentage	2.1, 4.2, 4.4, 4.6, 5, 5.1, 5.6, 5.8, 6.1, 6.2, 6.8, 7, 7.7, 8, ...

Red arrows point from the text blocks to the `pg_stats` table name in the query and the `percentage` row in the results table.

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 10)

- **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 68 – 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.)