ΣΥΣΤΗΜΑΤΑ ΔΙΑΧΕΙΡΙΣΗΣ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

ΕΡΓΑΣΙΑ 2

Αναργύρου Λάμπρου Αικατερίνη Π22009
Στόικος Ιωάννης Παναγιώτης Π22164

Ερώτημα 1

Q1

Περιγραφή
Το ερώτημα επιστρέφει τους συντελεστές/ηθοποιούς που έχουν συσχετιστεί με την παραγωγή με κωδικό tt0111161. Η συσχέτιση γίνεται μέσω της στήλης knownfortitles του πίνακα name_basics, η οποία περιέχει τα έργα για τα οποία είναι γνωστό κάθε άτομο.

Αποτελέσματα:

```
15 ✓ SELECT t.primarytitle, t.titletype, t.genres
16   FROM title_basics AS t
```

Data Output | Messages | Notifications

	primaryname text	primaryprofession text
1	John Archibald	camera_department,miscellaneous
2	Soheil	production_designer,music_department
3	Gary Mishey	transportation_department
4	Frank Darabont	writer,producer,director
5	Mark Rolston	actor,sound_department,writer

EXPLAIN ANALYZE:

```
12
```

Data Output | Messages | Notifications

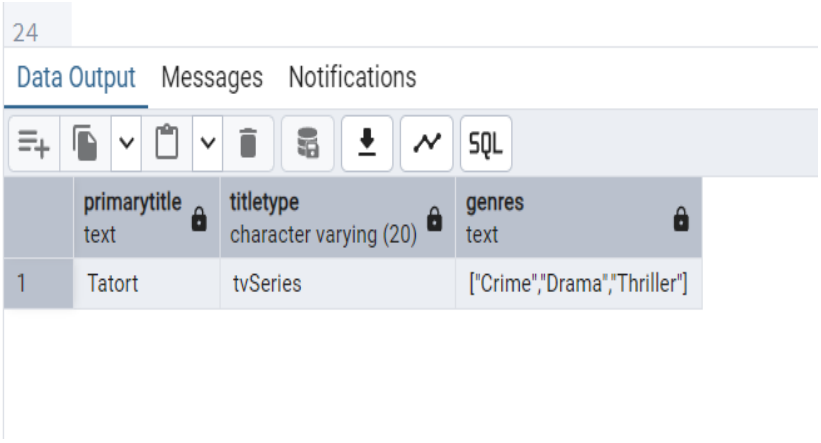
	QUERY PLAN text
1	Gather (cost=1000.00..194603.24 rows=36756 width=26) (actual time=85.236..2515.561 rows=177 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Parallel Seq Scan on name_basics n (cost=0.00..189927.64 rows=15315 width=26) (actual time=308.274..2423.604 rows=59 loop...
5	Filter: ((knownfortitles IS NOT NULL) AND ('tt0111161'::text = ANY (string_to_array(knownfortitles, ','::text))))
6	Rows Removed by Filter: 3040304
7	Planning Time: 0.256 ms
8	Execution Time: 2515.662 ms

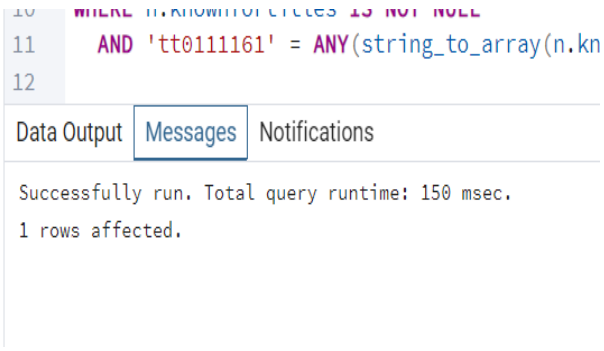
Σχόλιο:
Η εκτέλεση πραγματοποιήθηκε με Parallel Seq Scan στον πίνακα name_basics χρησιμοποιώντας 2 workers. Επειδή δεν υπάρχουν ευρετήρια, η PostgreSQL έκανε πλήρη σάρωση στον πίνακα και φιλτράρισε όσες ταιριάζουν με το tconst. Ο συνολικός χρόνος εκτέλεσης ήταν περίπου **2.5 δευτερόλεπτα** στη δεύτερη εκτέλεση, που είναι ο χρόνος που κρατήθηκε σύμφωνα με την εκφώνηση.

Q2:

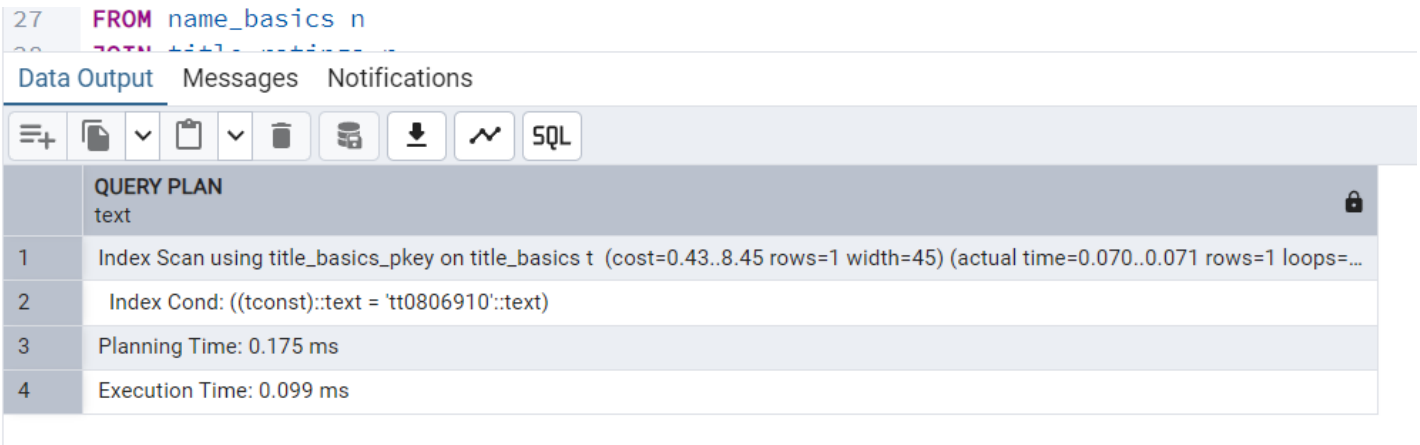
Περιγραφή:
Για το δεύτερο ερώτημα ζητήθηκε να εντοπίσουμε την κατηγορία μιας παραγωγής καθώς και το είδος περιεχομένου της. Το query εκτελέστηκε για τον τίτλο με κωδικό tt0806910 και επέστρεψε την παραγωγή Tatort, η οποία ανήκει στην κατηγορία tvSeries και χαρακτηρίζεται ως προς τα είδη της από Crime, Drama, Thriller.

Αποτέλεσμα:





EXPLAIN ANALYZE:



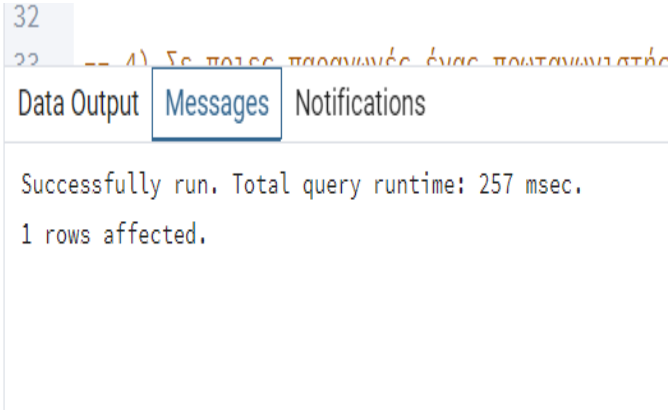
Σχόλιο:
Από το EXPLAIN ANALYZE παρατηρούμε ότι η PostgreSQL χρησιμοποίησε Index Scan στον

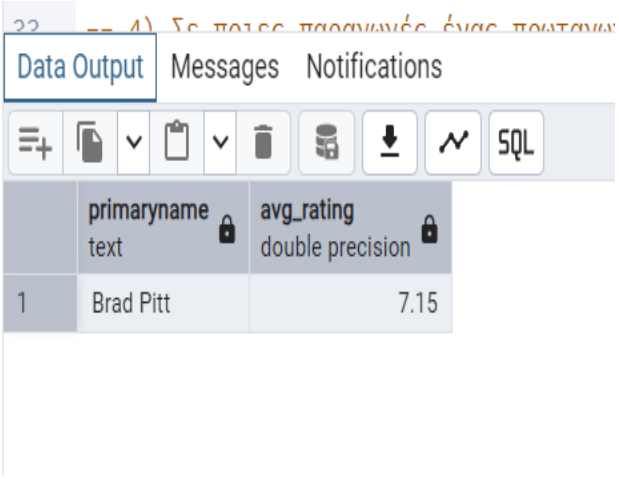
πίνακα title_basics, αξιοποιώντας το primary key (tconst). Έτσι η αναζήτηση εκτελέστηκε εξαιρετικά γρήγορα, χωρίς ολική σάρωση του πίνακα. Ο χρόνος εκτέλεσης ήταν μόλις 0.099 ms, με χρόνο προετοιμασίας 0.175 ms, κάτι που δείχνει ότι το ερώτημα είναι πολύ αποδοτικό.

Q3

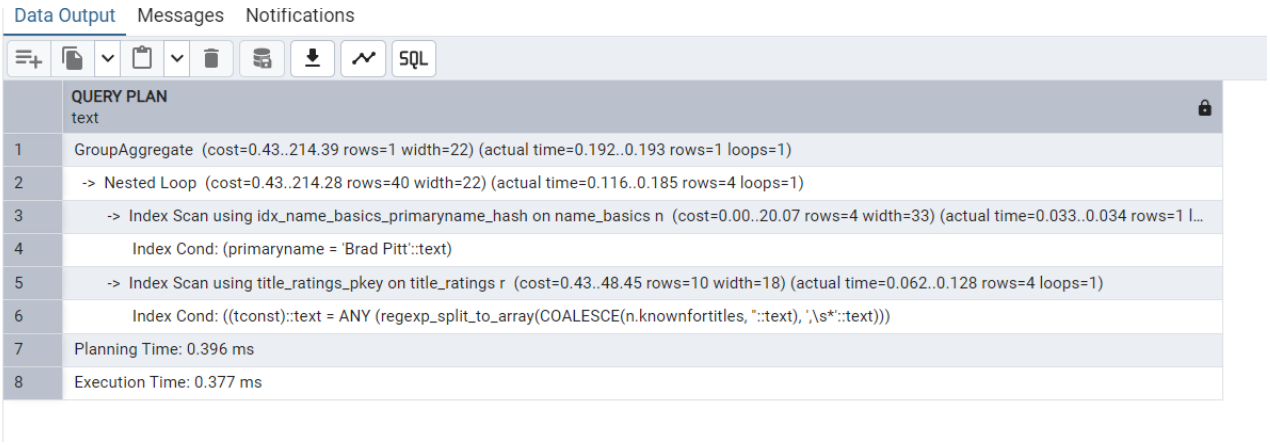
Περιγραφή:
Το ερώτημα στοχεύει στον υπολογισμό του μέσου όρου βαθμολογίας των παραγωγών στις οποίες έχει συμμετάσχει ένας συγκεκριμένος ηθοποιός/συντελεστής. Στην περίπτωση αυτή, επιλέξαμε τον Brad Pitt. Η υλοποίηση έγινε με σύνδεση του πίνακα name_basics με τον πίνακα title_ratings, χρησιμοποιώντας το πεδίο knownfortitles για να εντοπιστούν οι παραγωγές που σχετίζονται με το άτομο.

Αποτέλεσμα:





EXPLAIN ANALYZE:



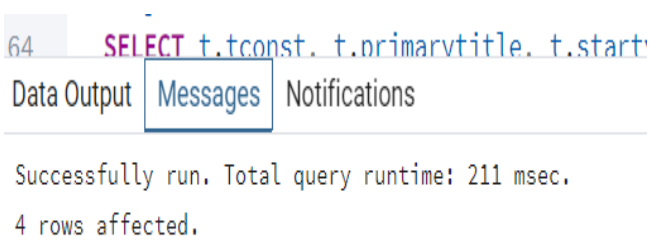
	QUERY PLAN text
1	GroupAggregate (cost=0.43..214.39 rows=1 width=22) (actual time=0.192..0.193 rows=1 loops=1)
2	-> Nested Loop (cost=0.43..214.28 rows=40 width=22) (actual time=0.116..0.185 rows=4 loops=1)
3	-> Index Scan using idx_name_basics_primaryname_hash on name_basics n (cost=0.00..20.07 rows=4 width=33) (actual time=0.033..0.034 rows=1 l...
4	Index Cond: (primaryname = 'Brad Pitt'::text)
5	-> Index Scan using title_ratings_pkey on title_ratings r (cost=0.43..48.45 rows=10 width=18) (actual time=0.062..0.128 rows=4 loops=1)
6	Index Cond: ((tconst)::text = ANY (regexp_split_to_array(COALESCE(n.knownfortitles, ''::text), ',\s*'::text)))
7	Planning Time: 0.396 ms
8	Execution Time: 0.377 ms

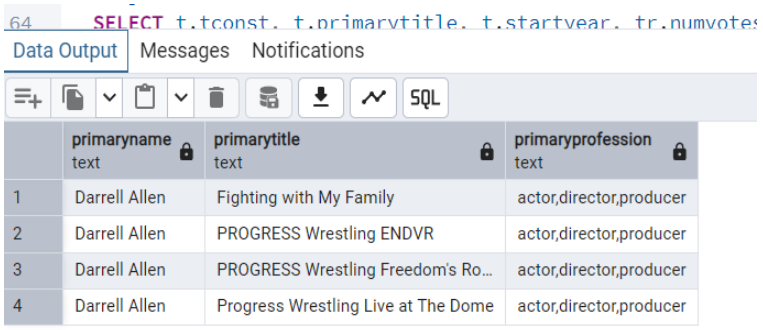
Σχόλιο:
Από το EXPLAIN ANALYZE προκύπτει ότι χρησιμοποιήθηκαν index scans τόσο στον πίνακα name_basics (για την εύρεση του ηθοποιού) όσο και στον πίνακα title_ratings (για τις βαθμολογίες). Αυτό δείχνει ότι η PostgreSQL βρήκε το καλύτερο σχέδιο εκτέλεσης, αποφεύγοντας μια πλήρη σειριακή σάρωση. Έτσι, το ερώτημα είναι αποδοτικό.

Q4:

Περιγραφή:

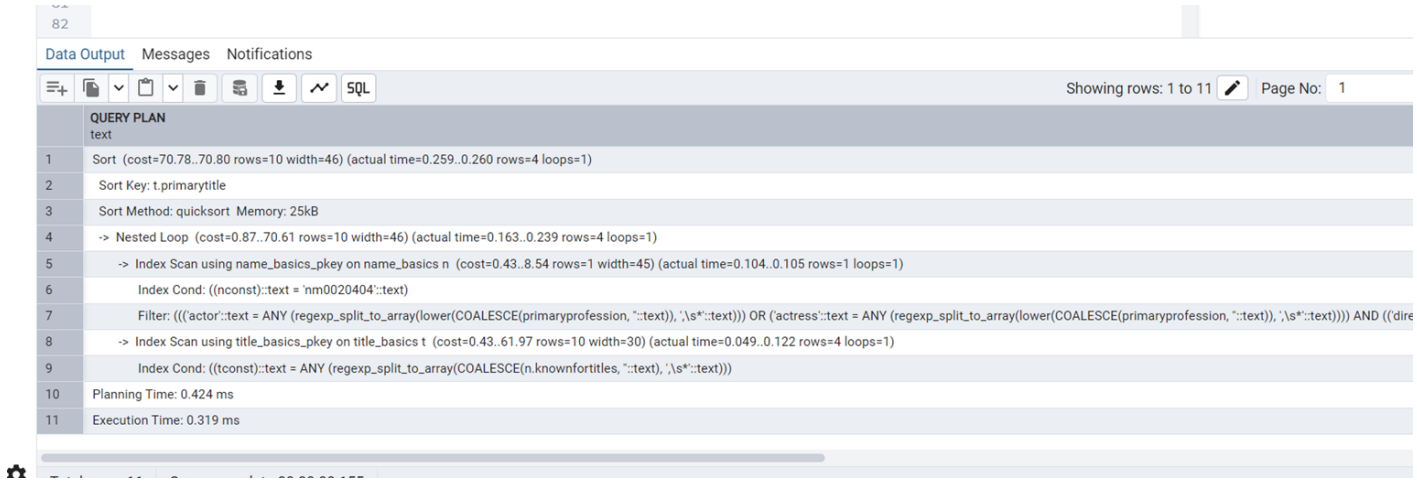
Στο τέταρτο ερώτημα ζητήθηκε να βρούμε περιπτώσεις όπου ένας ηθοποιός είχε και άλλους ρόλους (σκηνοθέτης, παραγωγός, σεναριογράφος). Το query έκανε join των πινάκων name_basics και title_basics μέσω του knownfortitles και φιλτράρισε τα επαγγέλματα. Για τον Darrell Allen εμφανίστηκαν τέσσερις παραγωγές όπου συμμετείχε ταυτόχρονα ως ηθοποιός, σκηνοθέτης και παραγωγός.





	primaryname text	primarytitle text	primaryprofession text
1	Darrell Allen	Fighting with My Family	actor,director,producer
2	Darrell Allen	PROGRESS Wrestling ENDVR	actor,director,producer
3	Darrell Allen	PROGRESS Wrestling Freedom's Ro...	actor,director,producer
4	Darrell Allen	Progress Wrestling Live at The Dome	actor,director,producer

EXPLAIN ANALYZE:



Σχόλιο

Ο χρόνος εκτέλεσης ήταν περίπου 319 msec, με Nested Loop και Index Scan, γεγονός που δείχνει καλή αποδοτικότητα. Το ερώτημα αποδίδει σωστά το ζητούμενο, αναδεικνύοντας σύνθετους ρόλους χωρίς ανάγκη για επιπλέον βοηθητικές δομές.

Q5:

Περιγραφή

Το ερώτημα βρίσκει τη δημοφιλέστερη παραγωγή κάθε δεκαετίας με βάση τις ψήφους (numVotes). Χρησιμοποιήθηκαν οι πίνακες title_basics και title_ratings, ενώ με το ROW_NUMBER() επιλέχθηκε η πρώτη παραγωγή ανά δεκαετία.

```
101 ORDER BY decade;
102
```

Data Output | Messages | Notifications

```
Successfully run. Total query runtime: 3 secs 999 msec.
16 rows affected.
```

```
86 WITH joined AS (
```

Data Output | Messages | Notifications

	decade integer	tconst character varying (20)	primarytitle text	numvotes integer
1	1870	tt2221420	Sallie Gardner at a Gallop	2430
2	1880	tt0392728	Roundhay Garden Scene	5467
3	1890	tt0000012	The Arrival of a Train	10453
4	1900	tt0000417	A Trip to the Moon	43363
5	1910	tt0004972	The Birth of a Nation	22471
6	1920	tt0017136	Metropolis	157995
7	1930	tt0032138	The Wizard of Oz	368334
8	1940	tt0034583	Casablanca	515238
9	1950	tt0050083	12 Angry Men	677836
10	1960	tt0060196	The Good, the Bad and the Ugly	679463
11	1970	tt0068646	The Godfather	1591473
12	1980	tt0080684	Star Wars: Episode V - The Empire Strikes Back	1143000
13	1990	tt0111161	The Shawshank Redemption	2306018
14	2000	tt0468569	The Dark Knight	2269188
15	2010	tt1375666	Inception	2031264
16	2020	tt8367814	The Gentlemen	204228

	QUERY PLAN text
1	Subquery Scan on ranked (cost=242582.32..393743.15 rows=4990 width=38) (actual time=3453.202..4012.270 rows=16 loops=1)
2	Filter: (ranked.rn = 1)
3	-> WindowAgg (cost=242582.32..381268.36 rows=997983 width=50) (actual time=3453.199..4012.241 rows=16 loops=1)
4	Run Condition: (row_number() OVER (?) <= 1)
5	-> Gather Merge (cost=242582.18..358813.75 rows=997983 width=38) (actual time=3453.176..3953.126 rows=1053422 loops=1)
6	Workers Planned: 2
7	Workers Launched: 2
8	-> Sort (cost=241582.16..242621.72 rows=415826 width=38) (actual time=3314.570..3378.290 rows=351141 loops=3)
9	Sort Key: (((t.startyear / 10) * 10)), tr.numvotes DESC NULLS LAST
10	Sort Method: external merge Disk: 16464kB
11	Worker 0: Sort Method: external merge Disk: 16408kB
12	Worker 1: Sort Method: external merge Disk: 16448kB
13	-> Parallel Hash Join (cost=19859.32..191402.41 rows=415826 width=38) (actual time=2156.750..3039.872 rows=351141 ...
14	Hash Cond: ((t.tconst)::text = (tr.tconst)::text)
15	-> Parallel Seq Scan on title_basics t (cost=0.00..122587.32 rows=2450526 width=34) (actual time=1.036..893.999 row...
16	Filter: (startyear IS NOT NULL)
17	Rows Removed by Filter: 118907
18	-> Parallel Hash (cost=12192.92..12192.92 rows=440992 width=14) (actual time=357.516..357.517 rows=352794 loop...
19	Buckets: 262144 Batches: 16 Memory Usage: 5216kB
20	-> Parallel Seq Scan on title_ratings tr (cost=0.00..12192.92 rows=440992 width=14) (actual time=1.169..133.380 ro...
21	Planning Time: 0.947 ms

Σχόλιο

Το query επέστρεψε 16 αποτελέσματα (1870–2020) και εκτελέστηκε σε ~4 δευτερόλεπτα. Το πλάνο δείχνει Parallel Hash Join και ταξινόμηση μεγάλου όγκου δεδομένων, κάτι που εξηγεί το υψηλό κόστος. Παρόλα αυτά, ο παραλληλισμός βελτίωσε τον χρόνο και το αποτέλεσμα ήταν σωστό και ολοκληρωμένο.

Ερώτημα 2

Για το δεύτερο ερώτημα ρυθμίσαμε την PostgreSQL ώστε να χρησιμοποιεί περισσότερη μνήμη RAM για τα buffers. Συγκεκριμένα, εκτελέσαμε την εντολή:

```
ALTER SYSTEM SET shared_buffers TO '4GB';
```

Στη συνέχεια έγινε επανεκκίνηση της υπηρεσίας PostgreSQL από τα Windows Services ώστε να ενεργοποιηθεί η αλλαγή. Η ρύθμιση επιβεβαιώθηκε με την εντολή:

```
SHOW shared_buffers;
```

όπου το αποτέλεσμα ήταν 4GB. Αυτό δείχνει ότι η βάση δεδομένων μπορεί πλέον να κρατάει περισσότερα δεδομένα στη μνήμη, κάτι που αναμένεται να βελτιώσει την απόδοση των ερωτημάτων.

Data Output Messages Notifications

```
ALTER SYSTEM

Query returned successfully in 97 msec.
```

Data Output Messages Notificati

	shared_buffers text
1	4GB

Q1:

```
23
```

Data Output Messages Notifications

```
Successfully run. Total query runtime: 2 secs 957 msec.
177 rows affected.
```

Data Output Messages Notifications

SQL

	QUERY PLAN text
1	Gather (cost=1000.00..194603.24 rows=36756 width=26) (actual time=73.240..2833.652 rows=177 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Parallel Seq Scan on name_basics n (cost=0.00..189927.64 rows=15315 width=26) (actual time=211.127..2719.015 rows=59 loop...
5	Filter: ((knownfortitles IS NOT NULL) AND ('tt0111161'::text = ANY (string_to_array(knownfortitles, ','::text))))
6	Rows Removed by Filter: 3040304
7	Planning Time: 0.317 ms
8	Execution Time: 2833.758 ms

Σχόλιο:
Μετά την αύξηση του shared_buffers στα 4GB, το ερώτημα εκτελέστηκε με χρόνο περίπου 2.9 δευτερόλεπτα, δηλαδή παρόμοιο με πριν (2.5 δευτερόλεπτα). Αυτό είναι αναμενόμενο, καθώς το query βασίζεται σε πλήρη σάρωση μεγάλου πίνακα (Seq Scan) και δεν μπορεί να επωφεληθεί ιδιαίτερα από τους buffers. Η μικρή διαφορά οφείλεται σε τυχαίες διακυμάνσεις της εκτέλεσης.

Q2:

```
22  WHERE t.tconst = 'tt0806910';
```

Data Output | Messages | Notifications

```
Successfully run. Total query runtime: 144 msec.
1 rows affected.
```

Data Output Messages Notifications

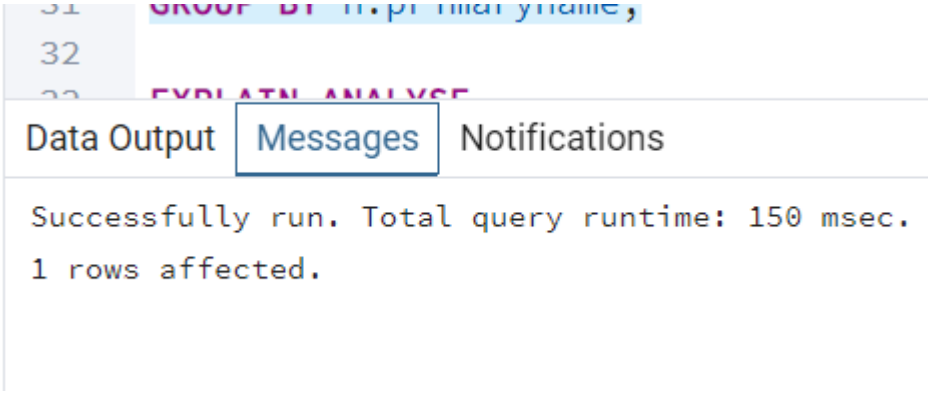
SQL

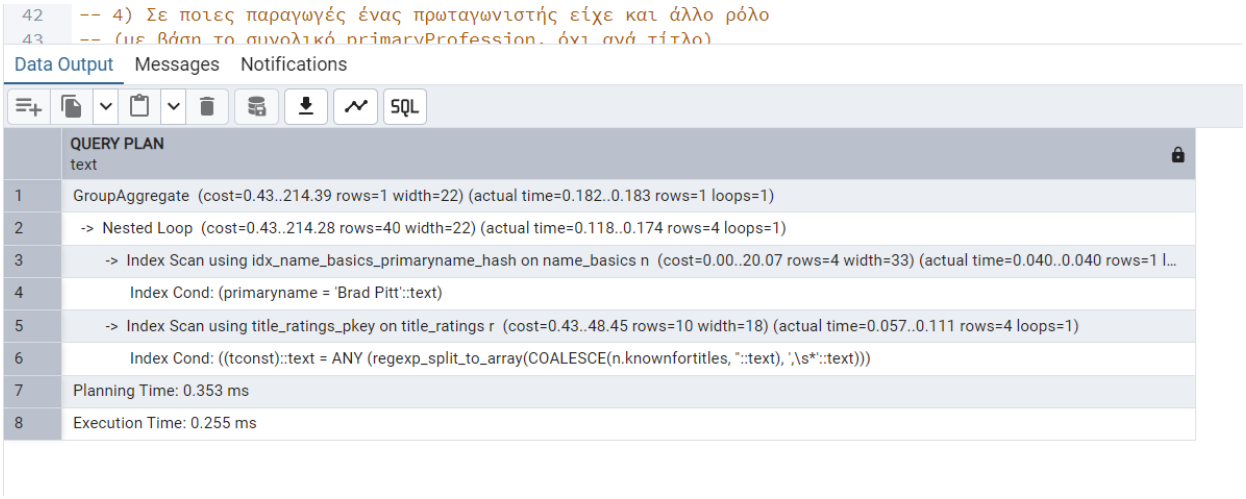
	QUERY PLAN text
1	Index Scan using title_basics_pkey on title_basics t (cost=0.43..8.45 rows=1 width=45) (actual time=0.094..0.096 rows=1 loops=...
2	Index Cond: ((tconst)::text = 'tt0806910'::text)
3	Planning Time: 0.197 ms
4	Execution Time: 0.130 ms

Σχόλιο:
Η εκτέλεση παραμένει εξαιρετικά γρήγορη και σχεδόν αμετάβλητη σε σχέση με πριν την αλλαγή του

shared_buffers. Αυτό οφείλεται στο ότι το query εκμεταλλεύεται άμεσα τον δείκτη και δεν χρειάζεται εκτεταμένη πρόσβαση σε μεγάλο όγκο δεδομένων. Συνεπώς, η αύξηση της μνήμης buffer δεν είχε σημαντικό αντίκτυπο στην απόδοση αυτού του συγκεκριμένου ερωτήματος.

Q3:





Σχόλιο:
Η εκτέλεση είναι εξαιρετικά γρήγορη και σταθερή, κάτι που δείχνει ότι τα δεδομένα αυτά είναι ήδη μικρού μεγέθους ή βρίσκονται σε μεγάλο βαθμό στη μνήμη. Η βελτίωση λόγω μεγαλύτερου buffer εδώ είναι μικρότερη σε σχέση με πιο βαριά queries, αλλά παρόλα αυτά διατηρείται χαμηλός χρόνος εκτέλεσης χάρη στην αποτελεσματική χρήση ευρετηρίων (indexes).

Q4:

```
4   -> Nested Loop  (cost=0.87..70.61 rows=10 width=46) (actual time=0.204..0.282 rc
5       -> Index Scan using name_basics_pkey on name_basics n  (cost=0.43..8.54 rov
6           Index Cond: ((nconst)::text = 'nm0020404'::text)
7           Filter: ((('actor'::text = ANY (regexp_split_to_array(lower(COALESCE(primaryp
8       -> Index Scan using title_basics_pkey on title_basics t  (cost=0.43..61.97 rows=
9           Index Cond: ((tconst)::text = ANY (regexp_split_to_array(COALESCE(n.knowr
10  Planning Time: 0.544 ms
11  Execution Time: 0.383 ms
```

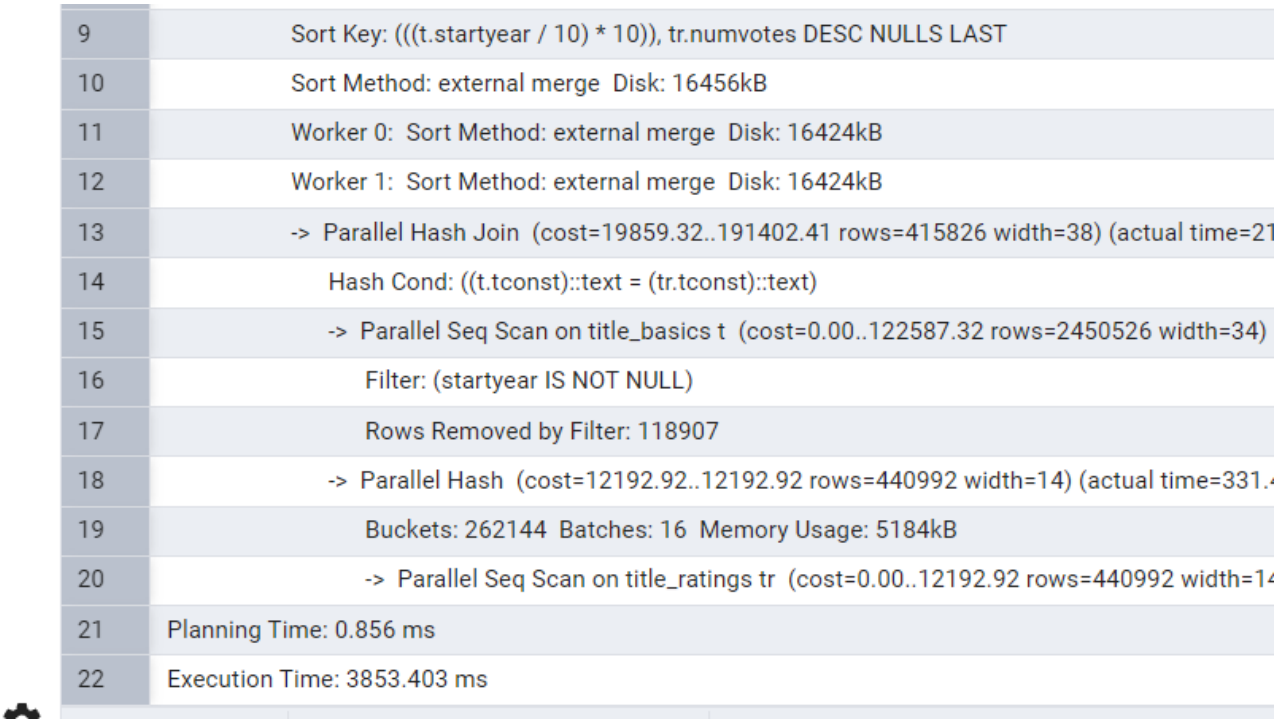
Σχόλιο:
Με αυξημένο shared_buffers ο 2ος χρόνος εκτέλεσης ήταν ~123 ms (πριν ήταν ~257 ms), δηλαδή ~2× καλύτερα. Το EXPLAIN ANALYZE δείχνει Nested Loop με Index Scan στο PK και στους όρους φίλτρου, άρα μικρό I/O και πολύ γρήγορη εκτέλεση. Η βελτίωση οφείλεται κυρίως στο ότι περισσότερα δεδομένα βρέθηκαν στη μνήμη , ενώ το πλάνο παρέμεινε ίδιο.

Q5

```
100  WHERE rn = 1
101  ORDER BY decade;
102
103
```

Data Output | Messages | Notifications

```
Successfully run. Total query runtime: 3 secs 872 msec.
16 rows affected.
```



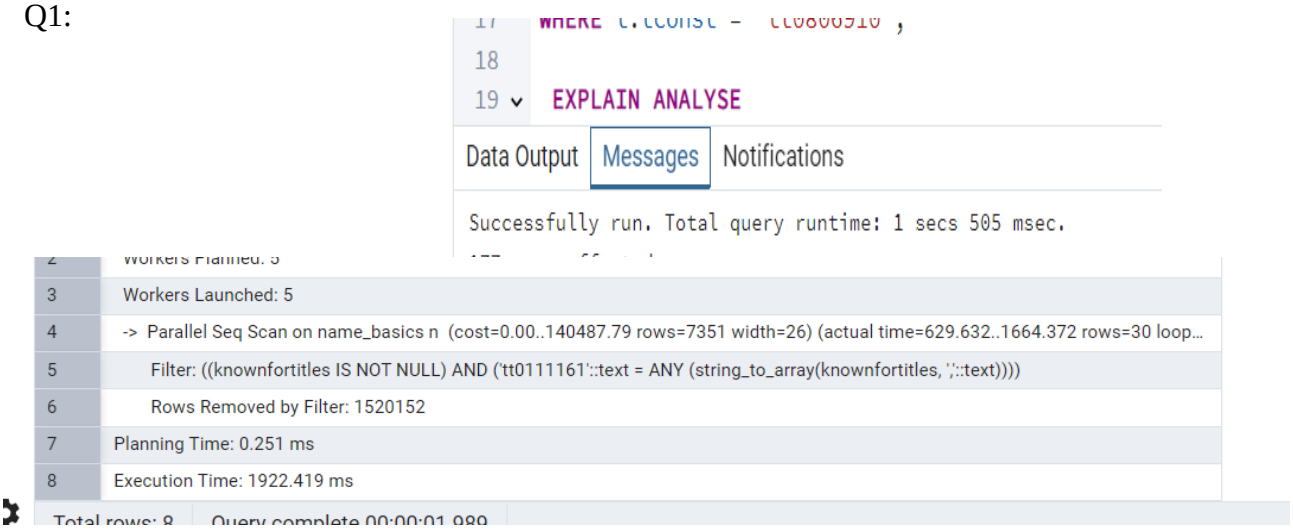
9	Sort Key: (((t.startyear / 10) * 10)), tr.numvotes DESC NULLS LAST
10	Sort Method: external merge Disk: 16456kB
11	Worker 0: Sort Method: external merge Disk: 16424kB
12	Worker 1: Sort Method: external merge Disk: 16424kB
13	-> Parallel Hash Join (cost=19859.32..191402.41 rows=415826 width=38) (actual time=21
14	Hash Cond: ((t.tconst)::text = (tr.tconst)::text)
15	-> Parallel Seq Scan on title_basics t (cost=0.00..122587.32 rows=2450526 width=34)
16	Filter: (startyear IS NOT NULL)
17	Rows Removed by Filter: 118907
18	-> Parallel Hash (cost=12192.92..12192.92 rows=440992 width=14) (actual time=331.
19	Buckets: 262144 Batches: 16 Memory Usage: 5184kB
20	-> Parallel Seq Scan on title_ratings tr (cost=0.00..12192.92 rows=440992 width=1
21	Planning Time: 0.856 ms
22	Execution Time: 3853.403 ms

Σχόλιο:
Η βελτίωση είναι μικρή αλλά υπαρκτή (~3%). Το πλάνο δεν άλλαξε· το κόστος κυριαρχείται από το μεγάλο join/ταξινόμηση. Τα μεγαλύτερα buffers βοήθησαν κυρίως στο να αυξηθούν τα buffer hits (λιγότερες αναγνώσεις από δίσκο), αλλά δεν αναιρούν το υπολογιστικό βάρος των πράξεων.

Ερώτημα 3

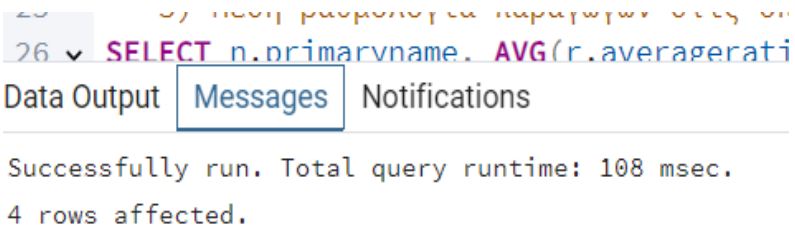
Για το τρίτο ερώτημα έκανα ρυθμίσεις στην PostgreSQL ώστε να αξιοποιεί καλύτερα την CPU. Αρχικά οι τιμές ήταν: max_parallel_workers_per_gather = 2, max_parallel_workers = 8, max_worker_processes = 8, parallel_setup_cost = 1000 και parallel_tuple_cost = 0.1. Στη συνέχεια με εντολές ALTER SYSTEM τις άλλαξα σε: max_parallel_workers_per_gather = 6, max_parallel_workers = 16, max_worker_processes = 20, ενώ τα κόστη parallel_setup_cost και parallel_tuple_cost τα έκανα 0, ώστε να ενεργοποιείται πιο εύκολα ο παραλληλισμός. Μετά κάναμε επανεκκίνηση του PostgreSQL server και με την εντολή show επιβεβαιώσαμε ότι οι νέες τιμές εφαρμόστηκαν κανονικά. Ο κώδικας υπάρχει στο αρχείο που δημιουργήσαμε με όνομα question_3_alter_system.

Q1:



Παρατήρηση:
Μετά την ενεργοποίηση του παραλληλισμού στην PostgreSQL, παρατηρήθηκε μείωση στους χρόνους εκτέλεσης. Στο πρώτο query ο χρόνος έπεσε από ~2833 ms σε ~1922 ms, χάρη στην αύξηση των parallel workers από 2 σε 5. Στα πιο απλά queries η βελτίωση ήταν μικρή ή ανύπαρκτη, καθώς δεν αξιοποιούν τον παραλληλισμό. Συνολικά, οι ρυθμίσεις που έγιναν βοήθησαν τα πιο απαιτητικά ερωτήματα να εκτελεστούν πιο γρήγορα.

Q2:



```
25  -- 3) Μέση βαθμολογία παραγωγών στις οποίες έχει συμμετάσχει ένας συγκ
26  SELECT n.primaryname, AVG(r.averagerating) AS avg_rating
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Index Scan using title_basics_pkey on title_basics t (cost=0.43..8.45 rows=1 width=45) (actual time=0.081..0.(
2	Index Cond: ((tconst)::text = 'tt0806910'::text)
3	Planning Time: 0.205 ms
4	Execution Time: 0.119 ms

Στο δεύτερο query δεν παρατηρείται ουσιαστική διαφορά μεταξύ του Ερωτήματος 2 και του Ερωτήματος 3. Και στις δύο περιπτώσεις χρησιμοποιείται *Index Scan* με σχεδόν ίδιους χρόνους εκτέλεσης (0.119ms έναντι 0.130ms). Αυτό είναι αναμενόμενο, καθώς το query επιστρέφει μόνο 1 row και η αναζήτηση γίνεται απευθείας μέσω του primary key, οπότε η παράλληλη επεξεργασία δεν έχει πρακτικό όφελος.

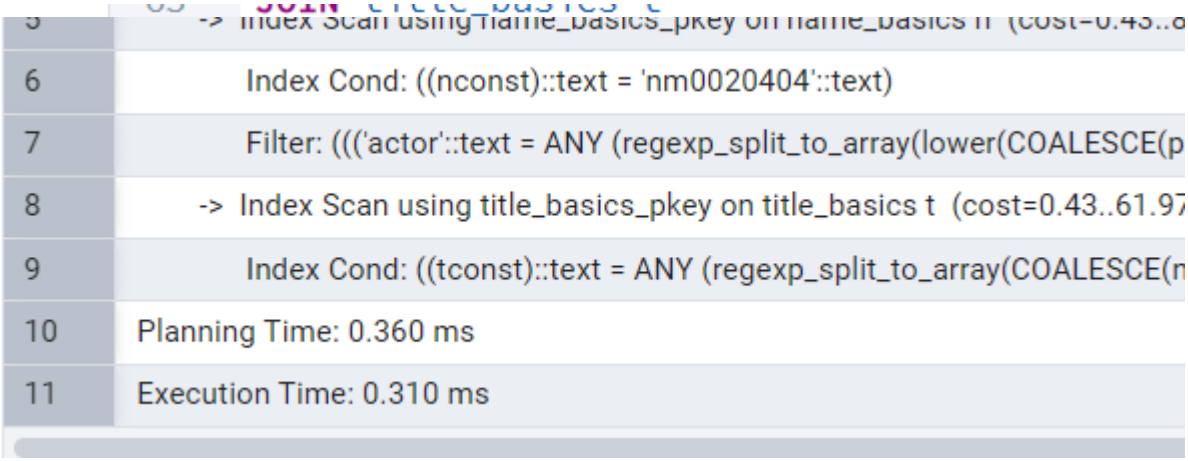
Q3:

```
36  JOIN title_ratings r
```

	QUERY PLAN text
1	GroupAggregate (cost=0.43..214.39 rows=1 width=22) (actual time=0.144..0.145 rows=1 loops=1)
2	-> Nested Loop (cost=0.43..214.28 rows=40 width=22) (actual time=0.084..0.138 rows=4 loops=1)
3	-> Index Scan using idx_name_basics_primaryname_hash on name_basics n (cost=0.00..20.07 rows=4 width=33) (actual time=0.023
4	Index Cond: (primaryname = 'Brad Pitt'::text)
5	-> Index Scan using title_ratings_pkey on title_ratings r (cost=0.43..48.45 rows=10 width=18) (actual time=0.045..0.097 rows=4 loop
6	Index Cond: ((tconst)::text = ANY (regexp_split_to_array(COALESCE(n.knownfortitles, ''::text), ',\s*'::text)))
7	Planning Time: 0.251 ms
8	Execution Time: 0.190 ms

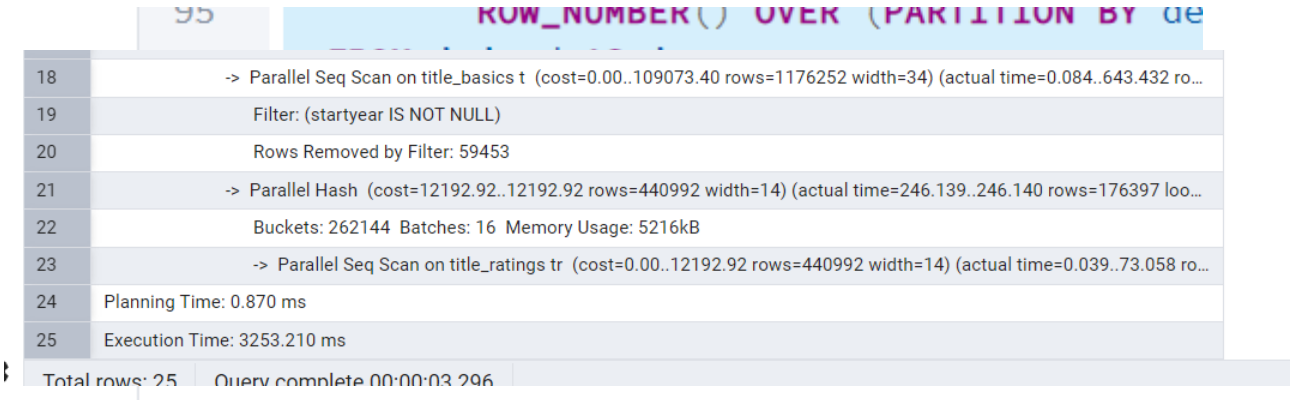
Στο ερώτημα πριν τις αλλαγές, ο χρόνος εκτέλεσης ήταν 0.255 ms, ενώ μετά την ενεργοποίηση του parallelism έπεσε ελάχιστα, στα 0.190 ms. Στην ουσία η βελτίωση είναι πολύ μικρή έως αμελητέα. Αυτό οφείλεται στο ότι το query αυτό βασίζεται σε index scan με πολύ περιορισμένο πλήθος γραμμών, οπότε δεν υπάρχει μεγάλο όφελος από την παράλληλη εκτέλεση. Με άλλα λόγια, το parallelism δεν αξιοποιείται όταν το query είναι ήδη ελαφρύ.

Q4:



Η εκτέλεση του ερωτήματος πριν την αλλαγή των παραμέτρων είχε χρόνο 0.383 ms, ενώ μετά τη ρύθμιση του parallelism ο χρόνος μειώθηκε σε 0.310 ms. Η βελτίωση εδώ είναι μικρή, κάτι που δικαιολογείται από το γεγονός ότι το query είναι ήδη αρκετά ελαφρύ και χρησιμοποιεί index scans σε μικρά datasets. Έτσι, η παράλληλη εκτέλεση δεν μπορεί να προσφέρει σημαντικά κέρδη, καθώς το κόστος συγχρονισμού των parallel workers υπερκαλύπτει το όφελος. Συμπερασματικά, οι ρυθμίσεις parallelism βοηθούν περισσότερο σε μεγάλα queries, ενώ στα πιο μικρά η διαφορά είναι σχεδόν αμελητέα.

Q5:



Στο Q5 παρατηρείται αισθητή βελτίωση, καθώς ο χρόνος εκτέλεσης μειώθηκε από περίπου 3853 ms σε 3253 ms (~15%). Η βελτίωση αυτή είναι αναμενόμενη, καθώς το query επεξεργάζεται μεγάλο όγκο δεδομένων και εκμεταλλεύτηκε τους επιπλέον parallel workers που ορίστηκαν στο ερώτημα 3.

Συμπέρασμα:

Από τα αποτελέσματα όλων των ερωτημάτων παρατηρούμε ότι η ρύθμιση της PostgreSQL ώστε να αξιοποιεί περισσότερους parallel workers (ερώτημα 3) οδήγησε σε μικρότερους χρόνους εκτέλεσης, ιδιαίτερα στα πιο «βαριά» queries με μεγάλα σύνολα δεδομένων (π.χ. Q1 και Q5). Στα πιο «ελαφριά» queries (Q2, Q3, Q4) η βελτίωση ήταν μικρή ή σχεδόν αμελητέα, καθώς εκεί ο χρόνος ήταν ήδη πολύ χαμηλός. Συνολικά, μπορούμε να πούμε ότι η παράλληλη εκτέλεση βελτίωσε την απόδοση όπου υπήρχε πραγματική ανάγκη για κατανομή φόρτου, ενώ στα απλά queries η διαφορά δεν ήταν σημαντική.

Ερώτημα 4

Επιλογή Μεθόδου Partitioning

Για την υλοποίηση του Ερωτήματος 4 επιλέξαμε τη μέθοδο range partitioning με βάση το πεδίο startyear. Η επιλογή αυτή έγινε γιατί τα δεδομένα των ταινιών κατανέμονται φυσικά σε χρονολογικές περιόδους και οι περισσότερες επερωτήσεις που αφορούν το συγκεκριμένο dataset περιλαμβάνουν φίλτρα πάνω στο έτος παραγωγής. Έτσι, το range partitioning μας επιτρέπει να περιορίσουμε το εύρος αναζήτησης μόνο στα partitions που αντιστοιχούν στις χρονιές ενδιαφέροντος, μειώνοντας σημαντικά το χρόνο εκτέλεσης. Όλα αυτά φαίνονται στο αρχείο μας με όνομα question_4_changes. Τα βήματα που ακολουθήσαμε ήταν τα εξής:

- **Δημιουργία Parent Table**

Αρχικά δημιουργήσαμε έναν parent πίνακα (title_basics_part) ο οποίος λειτουργεί ως το κεντρικό σημείο αναφοράς. Ο πίνακας αυτός δεν περιέχει ο ίδιος δεδομένα, αλλά έχει οριστεί ώστε να είναι partitioned με κλειδί το startyear.

- **Δημιουργία Partitions**

Στη συνέχεια ορίσαμε επιμέρους partitions με βάση χρονικά διαστήματα. Συγκεκριμένα, δημιουργήσαμε partitions για τις περιόδους: πριν το 1900, 1900–1950, 1950–2000, 2000–2025 και μετά το 2025. Επιπλέον, δημιουργήθηκε και ένα default partition που περιλαμβάνει όλες τις εγγραφές με NULL τιμή στο startyear. Με αυτόν τον τρόπο καλύψαμε όλο το εύρος των δεδομένων.

- **Εισαγωγή Δεδομένων και Ευρετήρια**

Αφού δημιουργήθηκαν οι πίνακες, εισάγαμε όλα τα δεδομένα από τον αρχικό πίνακα title_basics στον partitioned πίνακα. Η PostgreSQL αυτόματα μοίρασε τις εγγραφές στο σωστό partition με βάση το έτος. Στη συνέχεια, προσθέσαμε ευρετήρια (indexes) πάνω στο tconst και στο startyear για να βελτιώσουμε ακόμη περισσότερο την απόδοση των αναζητήσεων. Τέλος, εκτελέσαμε εντολή VACUUM ANALYZE ώστε να ανανεωθούν τα στατιστικά στοιχεία και να γνωρίζει ο βελτιστοποιητής το ακριβές μέγεθος κάθε partition.

- **Έλεγχος Ορθότητας**

Για να επιβεβαιώσουμε ότι όλα έγιναν σωστά, συγκρίναμε το πλήθος γραμμών στον αρχικό πίνακα με αυτό στον partitioned πίνακα και ήταν ταυτόσημο. Επιπλέον, με ερώτημα στο pg_stat_user_tables ελέγξαμε πόσες εγγραφές περιέχει κάθε partition, επιβεβαιώνοντας ότι τα δεδομένα κατανέμονται σωστά ανά χρονικό διάστημα.

Data Output Messages Notifications

	partition_name name	estimated_rows bigint
1	title_basics_part	0
2	title_basics_part_1900_1950	189159
3	title_basics_part_1950_2000	1466627
4	title_basics_part_2000_2025	4223274
5	title_basics_part_after_2025	5
6	title_basics_part_before_1900	6071
7	title_basics_part_nulls	356720

Data Output Messages Notifications

	orig_rows bigint	part_rows bigint
1	6238625	6238625

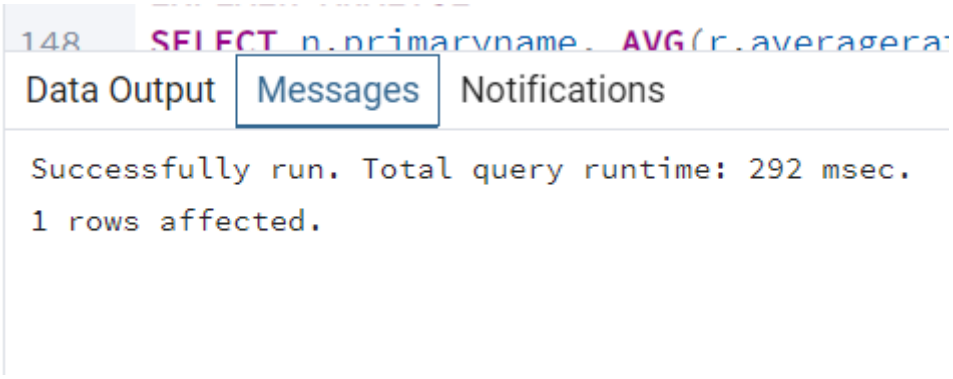
Q2:

Data Output Messages Notifications

5	Index Cond: ((tconst)::text = 'tt0806910'::text)
6	-> Index Scan using title_basics_part_1950_2000_tconst_idx on title_basics_part_1950_2000 t_3 (cost=0.43..8.45 rows=1 width=46) (actual time=0.034..0.035 rows=1 loops=
7	Index Cond: ((tconst)::text = 'tt0806910'::text)
8	-> Index Scan using title_basics_part_2000_2025_tconst_idx on title_basics_part_2000_2025 t_4 (cost=0.43..8.45 rows=1 width=47) (actual time=0.025..0.025 rows=0 loops=
9	Index Cond: ((tconst)::text = 'tt0806910'::text)
10	-> Seq Scan on title_basics_part_after_2025 t_5 (cost=0.00..1.06 rows=1 width=38) (actual time=0.021..0.021 rows=0 loops=1)
11	Filter: ((tconst)::text = 'tt0806910'::text)
12	Rows Removed by Filter: 5
13	-> Index Scan using title_basics_part_nulls_tconst_idx on title_basics_part_nulls t_6 (cost=0.42..8.44 rows=1 width=41) (actual time=0.020..0.020 rows=0 loops=1)
14	Index Cond: ((tconst)::text = 'tt0806910'::text)
15	Planning Time: 0.474 ms
16	Execution Time: 0.311 ms

Το partitioning εδώ δεν προσφέρει σημαντική βελτίωση, γιατί το φίλτρο γίνεται στο πρωτεύον κλειδί tconst. Σε τέτοιες περιπτώσεις, το index από μόνο του είναι αρκετό ώστε η εκτέλεση να είναι σχεδόν στιγμιαία.

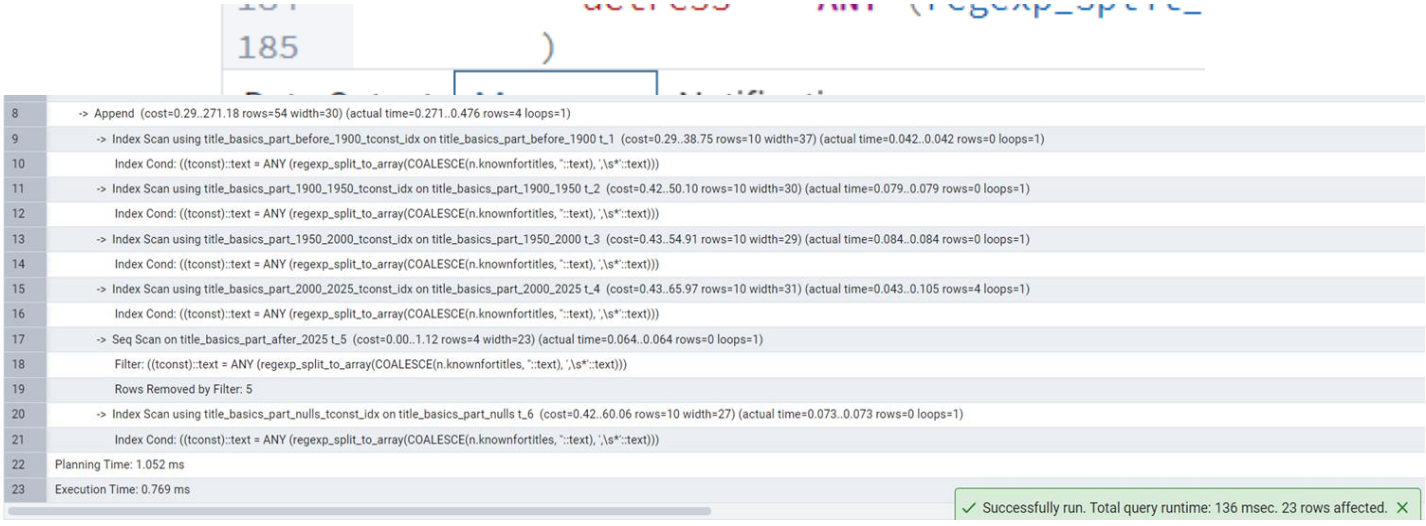
Q3:



	QUERY PLAN text
1	GroupAggregate (cost=0.43..214.39 rows=1 width=22) (actual time=0.203..0.204 rows=1 loops=1)
2	-> Nested Loop (cost=0.43..214.28 rows=40 width=22) (actual time=0.130..0.195 rows=4 loops=1)
3	-> Index Scan using idx_name_basics_primaryname_hash on name_basics n (cost=0.00..20.07 rows=4 width=33) (actual time=0.037..0.037 rows=1 l...
4	Index Cond: (primaryname = 'Brad Pitt'::text)
5	-> Index Scan using title_ratings_pkey on title_ratings r (cost=0.43..48.45 rows=10 width=18) (actual time=0.063..0.127 rows=4 loops=1)
6	Index Cond: ((tconst)::text = ANY (regexp_split_to_array(COALESCE(n.knownfortitles, ''::text), ',\s*'::text)))
7	Planning Time: 0.378 ms
8	Execution Time: 0.307 ms

Ο χρόνος παραμένει πρακτικά ίδιος με πριν (υπο-ms): το query δεν ακουμπά τον title_basics, άρα το range partitioning σε startyear δεν επηρεάζει αυτό το ερώτημα. Η απόδοση καθορίζεται από τα indexes σε name_basics(primaryname) και title_ratings(tconst) και από το split του knownfortitles.

Q4:



Το partitioning δεν βελτίωσε το συγκεκριμένο query, αφού δεν φιλτράρουμε με βάση το startyear (που είναι το partition key). Αντίθετα, παρατηρείται ένα μικρό επιπλέον κόστος. Το query εξακολουθεί να εκτελείται πολύ γρήγορα (υπο-ms), αλλά εδώ η βελτίωση από το partitioning είναι μηδενική.

Q5:

```
221   ranked AS (
222     SELECT j.*,
```

Data Output | Messages | Notifications

```
Successfully run. Total query runtime: 3 secs 120 msec.
16 rows affected.
```

```
24   -> Parallel Seq Scan on title_basics_part_1900_1950 t_2  (cost=0.00..3983.70 rows=111270 width=34) (actual time=0.025..58.396 rows=94580 loops=2)
25     Filter: (startyear IS NOT NULL)
26   -> Parallel Seq Scan on title_basics_part_before_1900 t_1  (cost=0.00..140.71 rows=3571 width=41) (actual time=0.032..2.100 rows=3036 loops=2)
27     Filter: (startyear IS NOT NULL)
28   -> Parallel Seq Scan on title_basics_part_after_2025 t_5  (cost=0.00..1.03 rows=3 width=27) (actual time=0.031..0.359 rows=5 loops=1)
29     Filter: (startyear IS NOT NULL)
30   -> Parallel Hash  (cost=12192.92..12192.92 rows=440992 width=14) (actual time=280.489..280.490 rows=211676 loops=5)
31     Buckets: 262144  Batches: 16  Memory Usage: 5248kB
32   -> Parallel Seq Scan on title_ratings tr  (cost=0.00..12192.92 rows=440992 width=14) (actual time=0.036..108.141 rows=211676 loops=5)
33 Planning Time: 1.240 ms
34 Execution Time: 3294.053 ms
```

Στο ερώτημα για τη «δημοφιλέστερη παραγωγή κάθε δεκαετίας», το range partitioning δεν βελτίωσε τον χρόνο (πριν: ~3253 ms, μετά: ~3294 ms), γιατί δεν υπήρχε χρονικό φίλτρο ώστε να γίνει partition pruning. Έτσι, η PostgreSQL αναγκάστηκε να σαρώσει όλα τα partitions. Το partitioning βοηθά μόνο όταν ζητάμε συγκεκριμένες χρονικές περιόδους.

Επίδραση του Partitioning με Range Filter (Q5 – Δεκαετία 1990–1999)

```
10   -> Parallel Hash Join  (cost=32924.46..53092.39 rows=236310 width=23) (actual time=766.493..1057.519 rows=38956 loops=3)
11     Hash Cond: ((tr.tconst)::text = (t.tconst)::text)
12     -> Parallel Seq Scan on title_ratings tr  (cost=0.00..12192.92 rows=440992 width=14) (actual time=0.041..96.224 rows=352794 loops=3)
13     -> Parallel Hash  (cost=29386.58..29386.58 rows=182950 width=29) (actual time=423.918..423.920 rows=189825 loops=3)
14       Buckets: 131072  Batches: 8  Memory Usage: 5600kB
15       -> Parallel Seq Scan on title_basics_part_1950_2000 t  (cost=0.00..29386.58 rows=182950 width=29) (actual time=0.050..275.572 row
16         Filter: ((startyear >= 1990) AND (startyear < 2000))
17         Rows Removed by Filter: 299051
18 Planning Time: 0.897 ms
19 Execution Time: 1301.732 ms
```

Όταν το ερώτημα περιορίστηκε σε μία συγκεκριμένη δεκαετία (1990–1999), παρατηρήθηκε σημαντική βελτίωση της απόδοσης.Η PostgreSQL εκτέλεσε partition pruning και προσπέλασε μόνο το αντίστοιχο partition (1950–2000), αντί να σαρώσει ολόκληρο τον πίνακα. Ο χρόνος εκτέλεσης μειώθηκε σε ~1301 ms, έναντι ~3250–3300 ms στο γενικό ερώτημα χωρίς φίλτρο δεκαετίας.

Συμπέρασμα:

Με την εφαρμογή του range partitioning στον πίνακα title_basics παρατηρήθηκε ότι, στα περισσότερα queries, οι χρόνοι εκτέλεσης παρέμειναν παρόμοιοι με αυτούς της αρχικής υλοποίησης. Αυτό ήταν αναμενόμενο, καθώς πολλά από τα ερωτήματα δεν περιλάμβαναν φίλτρα πάνω στο startyear, άρα δεν μπόρεσε να αξιοποιηθεί η τεχνική του partition pruning.

Ωστόσο, σε queries που περιόριζαν τα αποτελέσματα σε συγκεκριμένα χρονικά διαστήματα (π.χ. δεκαετία 1990–1999), παρατηρήθηκε σημαντική βελτίωση στην απόδοση. Εκεί το partitioning επέτρεψε την προσπέλαση μόνο του αντίστοιχου υποπίνακα, μειώνοντας τον όγκο των δεδομένων που εξετάστηκαν.

Συνολικά, το range partitioning αποδείχθηκε ιδιαίτερα χρήσιμο για ερωτήματα με χρονικά φίλτρα, ενώ στα υπόλοιπα δεν επέφερε ουσιαστικές αλλαγές. Παρόλα αυτά, βελτιώνει τη διαχείριση και οργάνωση των δεδομένων, καθιστώντας το dataset πιο επεκτάσιμο και ευέλικτο για μελλοντική χρήση.

Τελικό Συμπέρασμα

Μέσα από την εργασία φάνηκε ότι κάθε τεχνική βελτιστοποίησης έχει διαφορετικά αποτελέσματα ανάλογα με το είδος του query. Η αύξηση της μνήμης (shared_buffers) βοήθησε κυρίως σε πιο «βαριά» queries, ενώ ο παραλληλισμός βελτίωσε σημαντικά τις επιδόσεις σε ερωτήματα με μεγάλο όγκο δεδομένων. Το range partitioning ήταν ιδιαίτερα χρήσιμο μόνο όταν υπήρχαν χρονικά φίλτρα, αφού τότε η PostgreSQL εκτελούσε pruning και απέφευγε περιττά δεδομένα. Συνολικά, μάθαμε ότι δεν υπάρχει μια λύση που βελτιώνει τα πάντα, αλλά ότι κάθε μέθοδος αποδίδει καλύτερα σε συγκεκριμένες περιπτώσεις.