Advanced Databases

Liljana Dueva 231524

Anja Spasevski 231518

Eli Panova 231528

View 1: v_lab_request_details

1. Примарен филтер

- Примарен филтер за овој поглед е колоната request_id. Бидејќи секое лабораториско барање има единствен идентификатор, најчест случај на употреба е пребарување на конкретно барање според неговиот идентификациски број. Дополнително, погледот може да се користи за добивање информации за пациентот кој го поднел барањето и лабораторијата во која е поднесено барањето.

2. Намена на погледот

- Погледот v_lab_request_details е креиран со цел да обезбеди централизирано место за преглед на сите важни информации поврзани со едно лабораториско барање. Наместо апликацијата да прави повеќе JOIN операции помеѓу табелите lab_request, patient и lab секогаш кога е потребен приказ на деталите за барањето, овие податоци се веќе обединети во еден поглед.

  Погледот содржи информации за:

  идентификатор и датум на барањето;
  статус на барањето;
  податоци за пациентот;
  податоци за лабораторијата во која е поднесено барањето.

- Овој поглед би се користел во корисничкиот интерфејс за приказ на детали за барање, пребарување на барања и административен преглед на лабораториските активности.

3. Тестирање на перформансите

   За анализа на перформансите беше извршен следниот прашалник:

   ```
   EXPLAIN ANALYZE
   SELECT *
   FROM v_lab_request_details
   WHERE request_id = 100;
   ```

   Резултатите покажаа дека времето на извршување изнесува 115.793 ms.

   Со оглед на тоа што системот работи со голем број записи (10 милиони пациенти и 1 милион лабораториски барања), ова време се смета за прифатливо за потребите на апликацијата.

4. Анализа на планот на извршување

   При анализата на execution plan беше забележано дека PostgreSQL користи Index Scan операции за сите табели кои учествуваат во погледот.

   Се користат следните индекси:

   - lab_request_pkey за табелата lab_request;
   - patient_pkey за табелата patient;
   - lab_pkey за табелата lab.

   Планот не содржи Seq Scan операции. Тоа значи дека системот не ги скенира целосно табелите, туку директно пристапува до потребните записи преку постоечките индекси. Ова значително го намалува времето на извршување и овозможува поефикасна работа со големи количини на податоци.

5. Проверка на индексите

   Дополнително беше извршена проверка на индексите за табелата lab_request со користење на системскиот каталог pg_indexes.

   Резултатите покажаа дека табелата располага со примарен индекс:

   lab_request_pkey

   Овој индекс е креиран како B-Tree индекс над колоната request_id и автоматски се користи од PostgreSQL при пребарување според примарниот клуч.

6. Потреба од дополнителна оптимизација

   По извршената анализа беше утврдено дека нема потреба од дополнително индексирање или преуредување на прашалникот. Сите JOIN операции користат постоечки индекси, а времето на извршување е во прифатливи граници.

7. Заклучок

   Погледот v_lab_request_details е добро оптимизиран и ефикасно ги користи постоечките индекси во базата на податоци. Анализата покажа дека PostgreSQL користи Index Scan за сите релевантни табели, без појава на Seq Scan операции. Поради тоа не беше потребно додавање на нови индекси или модификација на прашалникот. Времето на извршување од 115.793 ms е задоволително и погледот е подготвен за употреба во апликацијата.

View 2: v_invoice_payment_summary

1. Примарен филтер

   Примарен филтер за овој поглед е колоната invoice_id. Погледот се користи за прикажување на информации за фактурите, извршените плаќања, преостанатиот износ за наплата и бројот на извршени плаќања за секоја фактура.

2. Намена на погледот

   Погледот v_invoice_payment_summary ги обединува податоците од табелите invoice и payment и овозможува централизирано следење на финансиските информации поврзани со фактурите. Во рамки на апликацијата овој поглед би се користел за финансиски извештаи, следење на статусот на плаќањата и анализа на приходите.

3. Анализа пред оптимизација

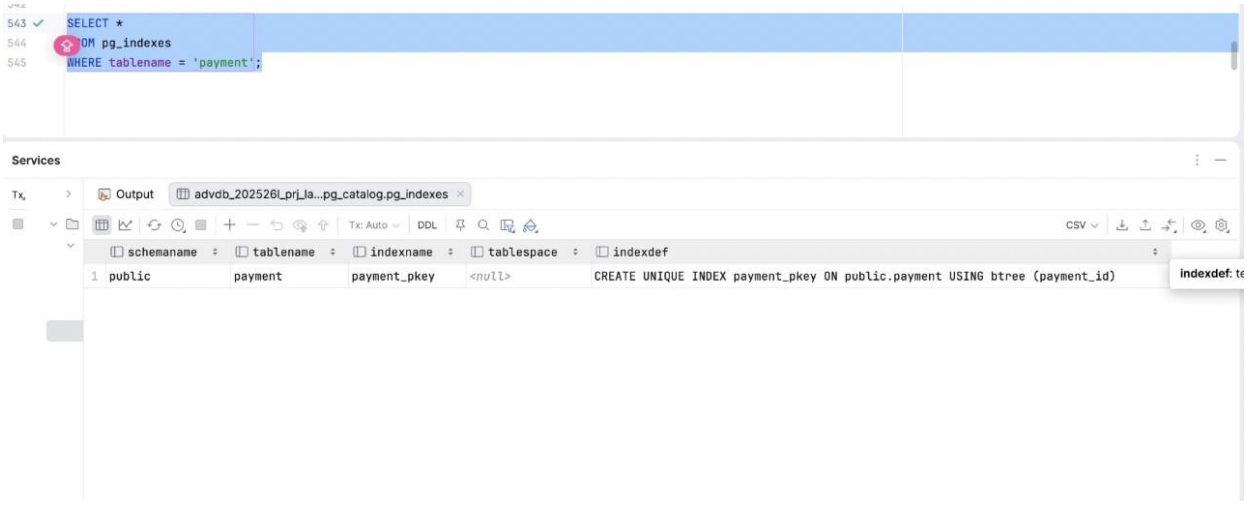
   За анализа на перформансите беше извршен следниот прашалник:

   ```
   EXPLAIN ANALYZE
   SELECT *
   FROM v_invoice_payment_summary
   WHERE invoice_id = 100;
   ```

Измереното време на извршување изнесуваше 235.668 ms.

Од анализата на execution plan беше забележано користење на операцијата Parallel Seq Scan врз табелата payment. Ова значи дека PostgreSQL извршува секвенцијално скенирање на голем дел од табелата за да ги пронајде записите кои одговараат на условот invoice_id = 100.

Бидејќи табелата payment содржи околу 10 милиони записи, недостатокот на индекс над колоната invoice_id предизвикуваше непотребно читање на голем број редови. Во execution plan беше забележано дека голем број записи биле отстранети преку филтерот пред да се пронајдат потребните резултати.



4. Оптимизација

   По анализата беше утврдено дека главната причина за забавеното извршување е недостатокот на индекс над колоната invoice_id во табелата payment.

   За подобрување на перформансите беше креиран следниот индекс:

```
CREATE INDEX idx_payment_invoice_id
    ON payment(invoice_id);
```

   Овој индекс овозможува PostgreSQL директно да пристапи до потребните записи наместо да скенира голем дел од табелата.

5. Анализа по оптимизација

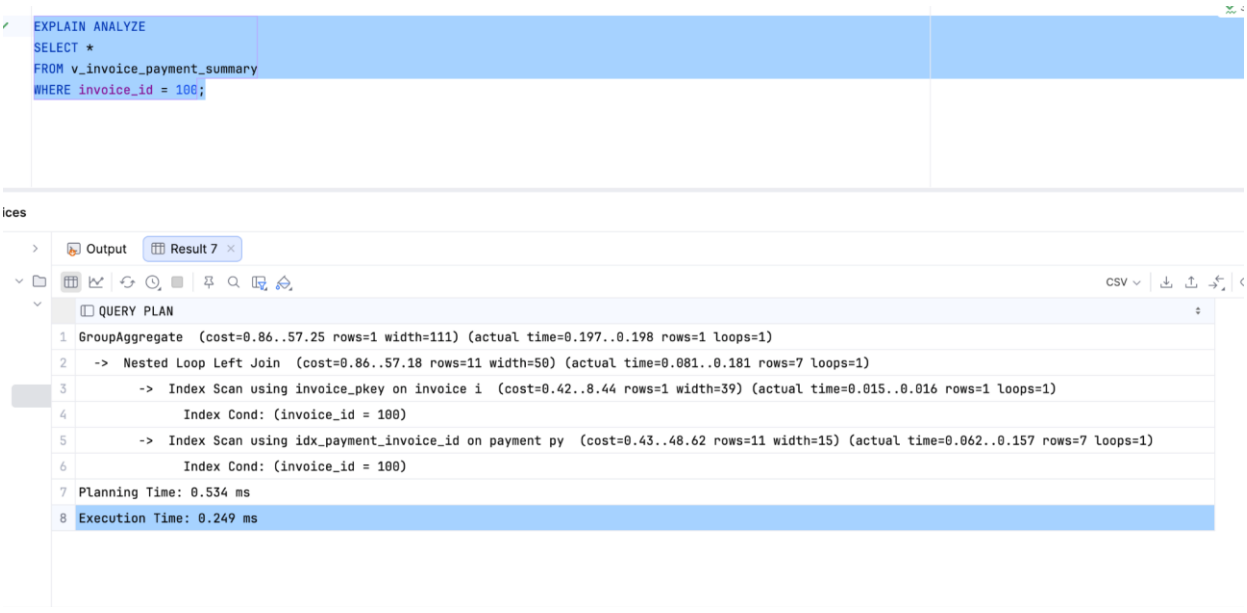
   По креирањето на индексот повторно беше извршен истиот прашалник со EXPLAIN ANALYZE.

   Новото време на извршување изнесува 0.249 ms.

   Execution plan покажува дека PostgreSQL повеќе не користи Parallel Seq Scan, туку користи:

   Index Scan using idx_payment_invoice_id

   Со ова пребарувањето се извршува директно преку индексот, без потреба од читање на милиони редови од табелата payment.



6. Заклучок

   Додавањето на индексот idx_payment_invoice_id значително ги подобри перформансите на погледот. Времето на извршување беше намалено од 235.668 ms на 0.249 ms, што претставува подобрување од приближно 946 пати. Анализата потврди дека индексот успешно го елиминираше Sequential Scan и овозможи значително поефикасно извршување на прашалникот.

3.View 3: v_sample_details

1. Примарен филтер

   Примарен филтер за овој поглед е колоната sample_id. Погледот овозможува пребарување на конкретен примерок и прикажување на дополнителни информации поврзани со него.

2. Намена на погледот

   Погледот v_sample_details ги обединува податоците од табелите sample, sample_type и employee. Неговата цел е да обезбеди детален преглед на примероците, нивниот тип и вработениот кој го извршил земањето на примерокот.

   Во рамки на апликацијата овој поглед може да се користи за следење на примероци, проверка на нивниот статус и идентификација на вработениот кој е одговорен за нивното собирање.

3. Анализа на перформансите

   За анализа на перформансите беше извршен следниот прашалник:

   ```
   EXPLAIN ANALYZE
   SELECT *

   FROM v_sample_details
   WHERE sample_id = 100;
   ```

   Измереното време на извршување изнесува 33.276 ms.

4. Анализа на планот на извршување

   Од execution plan беше утврдено дека PostgreSQL користи Index Scan операции за сите табели кои учествуваат во погледот:

   - Index Scan using sample_pkey
   - Index Scan using sample_type_pkey
   - Index Scan using employee_pkey

Сите поврзувања помеѓу табелите се извршуваат преку постоечки индекси, без потреба од целосно скенирање на табелите.

Дополнително, не се забележани Seq Scan или Parallel Seq Scan операции, што укажува на ефикасен пристап до податоците.

5. Оптимизација

   По анализата беше утврдено дека нема потреба од дополнително индексирање или промена на прашалникот. PostgreSQL веќе ги користи примарните индекси на сите вклучени табели.

6. Заклучок

   Погледот v_sample_details е добро оптимизиран и обезбедува брзо извршување дури и при работа со табелата sample која содржи околу 10 милиони записи. Времето на извршување од 33.276 ms се смета за одличен резултат и не беше потребна дополнителна оптимизација.

4.View 4: v_test_result_details

1. Примарен филтер

   Примарен филтер за овој поглед е колоната test_result_id. Погледот овозможува преглед на конкретен лабораториски резултат заедно со параметрите, примерокот и вработениот кој го извршил тестот.

2. Намена

Погледот ги обединува податоците од табелите test_result, result_parameter, sample и employee. Неговата намена е да обезбеди детален приказ на лабораториските резултати и сите релевантни информации поврзани со нив.

3. Анализа на перформансите

Извршен е следниот прашалник:

EXPLAIN ANALYZE
SELECT *
FROM

v_test_result_details

WHERE test_result_id =

100; Време на

извршување:

100.958.1

4. Анализа на планот на извршување

PostgreSQL користи:

Index Scan using test_result_pkey
Index Scan using result_parameter_pkey
Index Scan using sample_pkey
Index Scan using employee_pkey

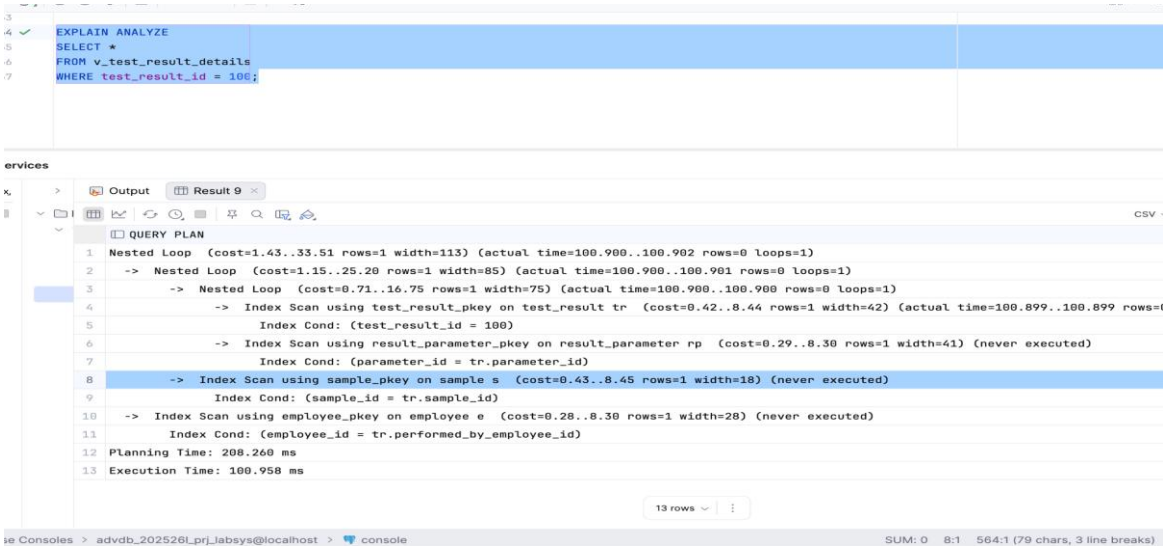
Сите JOIN операции користат индекси и не се забележани Sequential Scan операции.

5. Оптимизација

Не беше потребна дополнителна оптимизација бидејќи PostgreSQL веќе ги користи примарните индекси на сите вклучени табели.

6. Заклучок

Погледот е добро оптимизиран и овозможува брз пристап до податоците. И покрај тоа што комбинира повеќе табели преку JOIN операции, времето на извршување останува ниско и не беше потребна дополнителна интервенција.



5.View 5: v_patient_statistics

1. Примарен филтер:

   Погледот нема филтер бидејќи прикажува агрегирани статистички информации за сите пациенти во системот.

2. Намена:

   Погледот обезбедува статистички преглед на пациентите според пол и статус на активност. Може да се користи во административни панели, dashboard извештаи и аналитички прегледи.

3. Тестирање:

   EXPLAIN ANALYZE
   SELECT *
   FROM

   v_patient_statistics;

   Време на извршување:

   1019.278 ms

   План на

   извршување:

PostgreSQL користи:

Parallel Seq Scan on patient

Операцијата е очекувана бидејќи погледот извршува агрегирање (COUNT) над сите 10 милиони записи од табелата patient.

4. Оптимизација:

   Не беше извршена дополнителна оптимизација. Бидејќи прашалникот мора да ги обработи сите записи за да ја пресмета статистиката, индексите не би донеле значително подобрување на перформансите.

5. Заклучок:

   И покрај тоа што се обработуваат околу 10 милиони записи, времето на извршување изнесува приближно 1 секунда, што е во рамките на поставеното барање (под 2 секунди). Поради тоа не беше потребна дополнителна оптимизација.

```
11              Worker 2:  Sort Method: quicksort  Memory: 25kB
12              Worker 3:  Sort Method: quicksort  Memory: 25kB
13              ->  Partial HashAggregate  (cost=223405.04..223405.06 rows=2 width=26) (actual time=943.649..943.651
14                    Group Key: patient.gender
15                    Batches: 1  Memory Usage: 24kB
16                    Worker 0:  Batches: 1  Memory Usage: 24kB
17                    Worker 1:  Batches: 1  Memory Usage: 24kB
18                    Worker 2:  Batches: 1  Memory Usage: 24kB
19                    Worker 3:  Batches: 1  Memory Usage: 24kB
20                    ->  Parallel Seq Scan on patient  (cost=0.00..198403.52 rows=2500152 width=3) (actual time=0.0
21  Planning Time: 0.972 ms
22  JIT:
23    Functions: 33
24    Options: Inlining false, Optimization false, Expressions true, Deforming true
25    Timing: Generation 4.940 ms (Deform 2.571 ms), Inlining 0.000 ms, Optimization 2.800 ms, Emission 54.470 ms, Tota
26  Execution Time: 1019.278 ms
```

```
    QUERY PLAN
1   Finalize GroupAggregate  (cost=224405.13..224406.19 rows=2 width=26) (actual time=976.363..984.770 rows=2 loops=1)
2     Group Key: patient.gender
3     ->  Gather Merge  (cost=224405.13..224406.09 rows=8 width=26) (actual time=976.316..984.724 rows=10 loops=1)
4           Workers Planned: 4
5           Workers Launched: 4
6           ->  Sort  (cost=223405.07..223405.07 rows=2 width=26) (actual time=943.739..943.741 rows=2 loops=5)
7                 Sort Key: patient.gender
8                 Sort Method: quicksort  Memory: 25kB
9                 Worker 0:  Sort Method: quicksort  Memory: 25kB
10                Worker 1:  Sort Method: quicksort  Memory: 25kB
11                Worker 2:  Sort Method: quicksort  Memory: 25kB
12                Worker 3:  Sort Method: quicksort  Memory: 25kB
13                ->  Partial HashAggregate  (cost=223405.04..223405.06 rows=2 width=26) (actual time=943.649..943.651 row…
14                      Group Key: patient.gender
15                      Batches: 1  Memory Usage: 24kB
16                      Worker 0:  Batches: 1  Memory Usage: 24kB
17                      Worker 1:  Batches: 1  Memory Usage: 24kB
18                      Worker 2:  Batches: 1  Memory Usage: 24kB
19                      Worker 3:  Batches: 1  Memory Usage: 24kB
20                      ->  Parallel Seq Scan on patient  (cost=0.00..198403.52 rows=2500152 width=3) (actual time=0.060..…
21  Planning Time: 0.972 ms
```

View 6: v_ordered_test_details

1. Примарен филтер

   request_id

2. Намена

   Погледот обезбедува информации за нарачаните тестови, нивниот тип и цена. Се користи при преглед на лабораториски барања и анализа на извршените тестови.

3. Тестирање

   EXPLAIN ANALYZE
   SELECT *
   FROM v_ordered_test_details
   WHERE request_id = 100;

4. Време на извршување

   180.527 ms

   Анализа на планот на извршување

   PostgreSQL користи:

   Index Scan using ordered_test_request_id_test_id_key
   Index Scan using test_pkey
   Index Scan using test_type_pkey

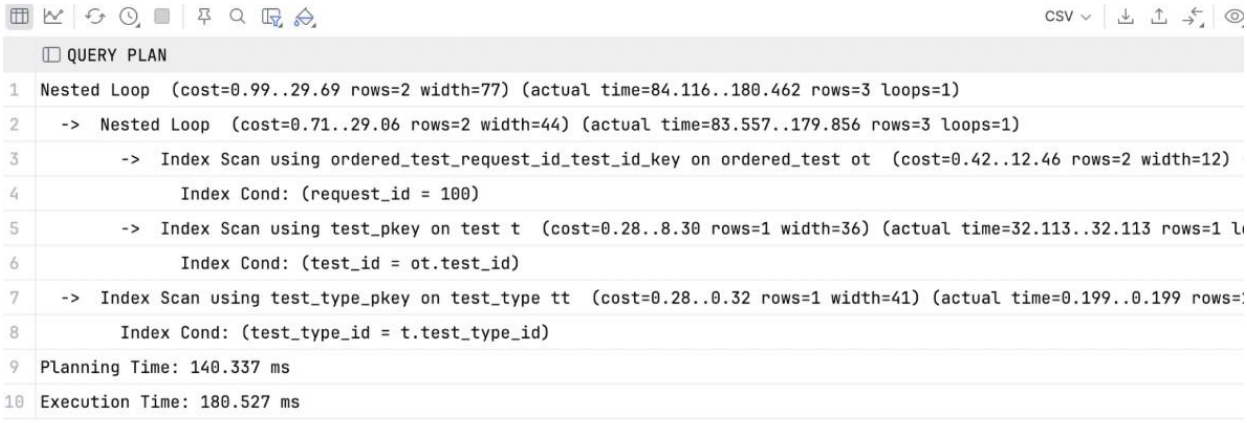
   Сите JOIN операции користат индекси и не се забележани Sequential Scan операции.

5. Оптимизација

   Не беше потребна дополнителна оптимизација бидејќи PostgreSQL веќе користи соодветни индекси за пребарување и поврзување на табелите.

6. Заклучок

   Погледот е добро оптимизиран и овозможува брз пристап до податоците дури и при работа со табела која содржи приближно 1 милион записи.



```
QUERY PLAN
Nested Loop  (cost=0.99..29.69 rows=2 width=77) (actual time=84.116..180.462 rows=3 loops=1)
  ->  Nested Loop  (cost=0.71..29.06 rows=2 width=44) (actual time=83.557..179.856 rows=3 loops=1)
        ->  Index Scan using ordered_test_request_id_test_id_key on ordered_test ot  (cost=0.42..12.46 rows=2 width=12)
              Index Cond: (request_id = 100)
        ->  Index Scan using test_pkey on test t  (cost=0.28..8.30 rows=1 width=36) (actual time=32.113..32.113 rows=1 l
              Index Cond: (test_id = ot.test_id)
  ->  Index Scan using test_type_pkey on test_type tt  (cost=0.28..0.32 rows=1 width=41) (actual time=0.199..0.199 rows=
        Index Cond: (test_type_id = t.test_type_id)
Planning Time: 140.337 ms
Execution Time: 180.527 ms
```

7. Примарен филтер v_ordered_package_details

request_id

1. Намена

   Погледот обезбедува информации за нарачаните пакети анализи, нивната цена и бројот на тестови кои ги содржат. Се користи при преглед на лабораториски барања и анализа на пакетите кои се извршуваат.

2. Тестирање

   EXPLAIN ANALYZE
   SELECT *
   FROM v_ordered_package_details
   WHERE request_id = 100;

3. Време на извршување

   28.015 ms

   Анализа на планот на извршување

   PostgreSQL користи:

   Index Scan using ordered_package_request_id_package_id_key
   Index Scan using analysis_package_pkey
   Index Only Scan using test_in_package_package_id_test_id_key

Дополнително се користат:

GroupAggregate Sort за пресметка на

бројот на тестови во секој пакет.

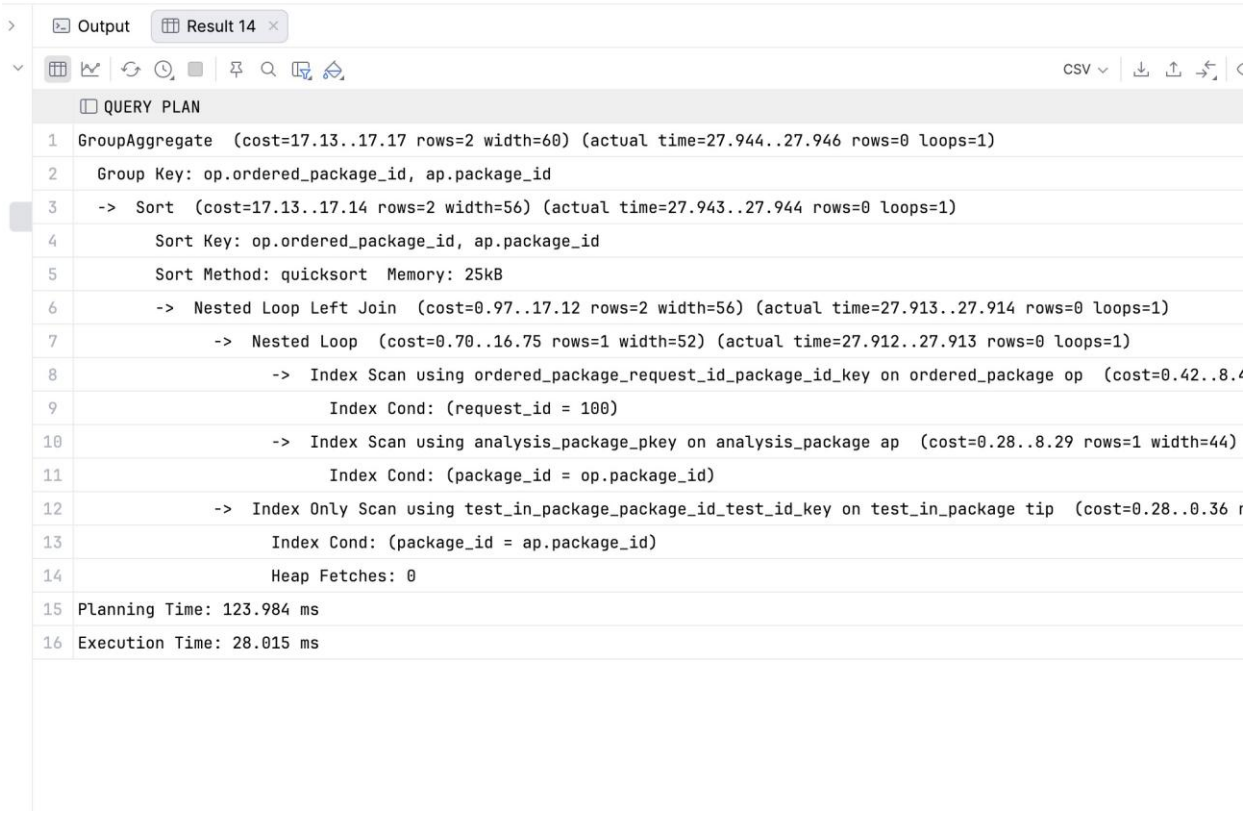
Не се забележани Sequential Scan операции.

4. Оптимизација

   Не беше потребна дополнителна оптимизација бидејќи PostgreSQL веќе ги користи соодветните индекси.

5. Заклучок

   Погледот е добро оптимизиран. И покрај употребата на агрегатни функции и GROUP BY операции, времето на извршување останува многу ниско и изнесува само 28.015 ms.



8.v_employee_schedule_details

View 8: v_employee_schedule_details

1. Примарен филтер:
   employee_id

2. Намена:
   Погледот се користи за приказ на распоредот на вработените, заедно со лабораторијата во која работат и типот на смена.

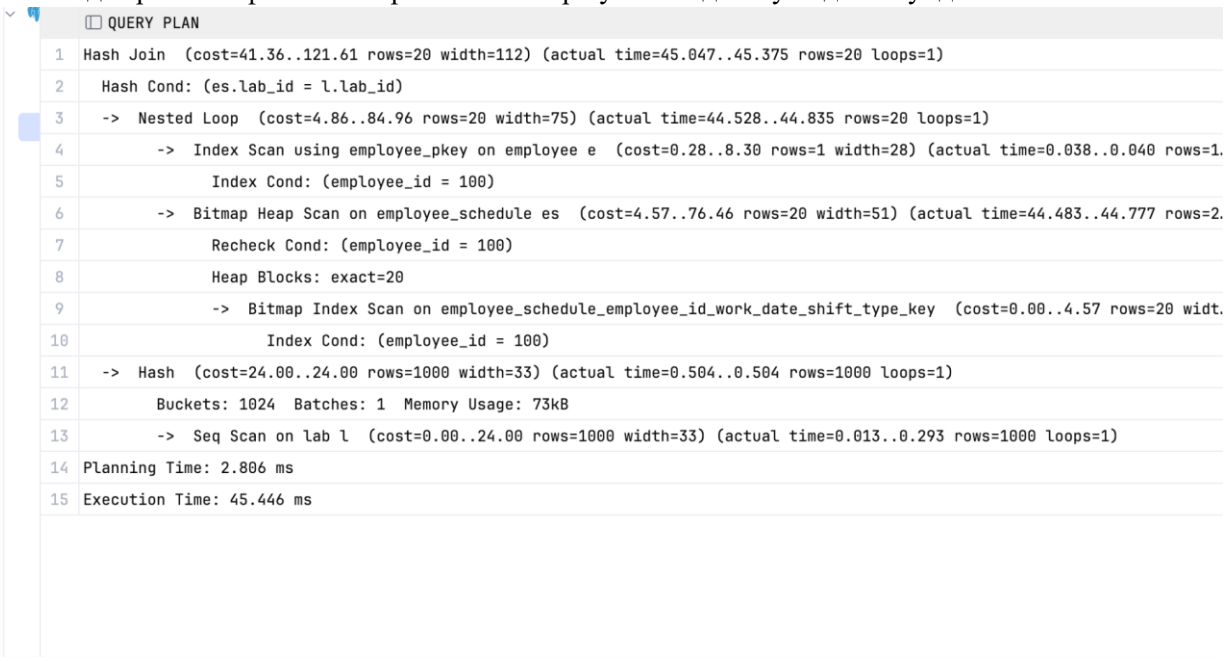
3. Тестирање:
   EXPLAIN ANALYZE
   SELECT *
   FROM v_employee_schedule_details
   WHERE employee_id = 100;

4. Време на извршување:
   45.446 ms

   План на извршување:
   PostgreSQL користи Index Scan врз employee и Bitmap Index Scan врз employee_schedule. За табелата lab се користи Seq Scan, но ова е прифатливо бидејќи табелата lab содржи само 1000 записи.

5. Оптимизација:
   Не беше потребна дополнителна оптимизација.

6. Заклучок:
   Погледот работи ефикасно и времето на извршување е далеку под 2 секунди.



```
QUERY PLAN
Hash Join  (cost=41.36..121.61 rows=20 width=112) (actual time=45.047..45.375 rows=20 loops=1)
  Hash Cond: (es.lab_id = l.lab_id)
  ->  Nested Loop  (cost=4.86..84.96 rows=20 width=75) (actual time=44.528..44.835 rows=20 loops=1)
        ->  Index Scan using employee_pkey on employee e  (cost=0.28..8.30 rows=1 width=28) (actual time=0.038..0.040 rows=1.
              Index Cond: (employee_id = 100)
        ->  Bitmap Heap Scan on employee_schedule es  (cost=4.57..76.46 rows=20 width=51) (actual time=44.483..44.777 rows=2.
              Recheck Cond: (employee_id = 100)
              Heap Blocks: exact=20
              ->  Bitmap Index Scan on employee_schedule_employee_id_work_date_shift_type_key  (cost=0.00..4.57 rows=20 widt.
                    Index Cond: (employee_id = 100)
  ->  Hash  (cost=24.00..24.00 rows=1000 width=33) (actual time=0.504..0.504 rows=1000 loops=1)
        Buckets: 1024  Batches: 1  Memory Usage: 73kB
        ->  Seq Scan on lab l  (cost=0.00..24.00 rows=1000 width=33) (actual time=0.013..0.293 rows=1000 loops=1)
Planning Time: 2.806 ms
Execution Time: 45.446 ms
```

View 9: v_lab_equipment_skills

1. Примарен филтер:

   employee_id

2. Намена:
   Погледот обезбедува информации за вештините и сертификациите на вработените за работа со лабораториска опрема. Ги поврзува податоците од employee, employee_equipment_skill, lab_equipment и lab.

3. Тестирање:

   ```
   EXPLAIN ANALYZE
   SELECT *
   FROM v_lab_equipment_skills
   WHERE employee_id = 100;
   ```

   Време на извршување:

1.045 ms

Анализа на планот на извршување:

PostgreSQL користи:

Index Scan using employee_pkey
Bitmap Index Scan using employee_equipment_skill_employee_id_equipment_id_key
Bitmap Heap Scan on employee_equipment_skill
Index Scan using lab_pkey

За табелата lab_equipment се користи:

Seq Scan on lab_equipment

Ова е прифатливо бидејќи табелата содржи само околу 2000 записи и PostgreSQL проценил дека Sequential Scan е побрз од користење индекс.

Дополнително се користи:

Hash Join Nested Loop за

поврзување на табелите.

4. Оптимизација:

   Не беше потребна дополнителна оптимизација бидејќи сите главни филтри користат постоечки индекси.

5. Заклучок:

   Погледот е високо оптимизиран. Времето на извршување изнесува само 1.045 ms, што е значително под поставениот праг од 2 секунди.