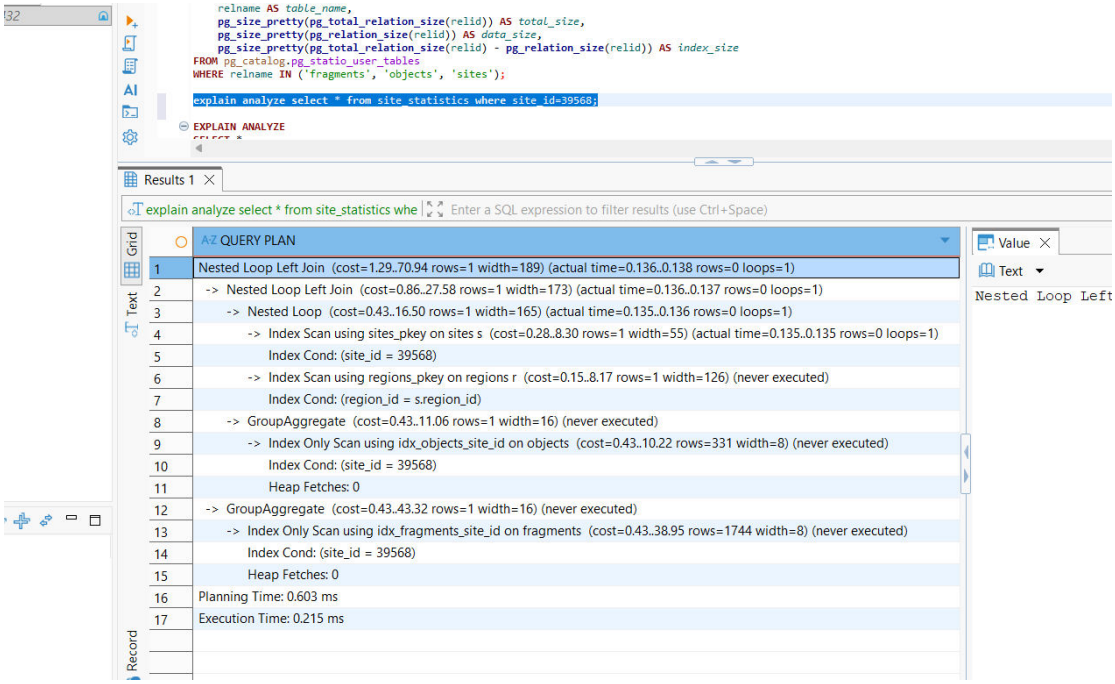
Напредни бази на податоци
Фаза 3 - Индекси и оптимизација на прашалници

Проект: HeritageGuard

Паула Коцева 231130
Елеонора Маркоска 231256
Ивана Павлова 231511

View 1 - Site_Statistics



Овој поглед е наменет за административен и аналитички преглед на статистички информации за секој археолошки локалитет во базата. Неговата главна цел е да овозможи брз и едноставен пристап до клучни податоци за конкретен локалитет преку пребарување според site_id, без потреба од пишување комплексни SQL барања над повеќе табели.

Преку овој view, корисникот може да добие интегриран приказ кој ги содржи основните информации за локалитетот (site_id, site_name, region), како и агрегирани статистики за:

1. вкупен број на археолошки објекти (total_objects)
2. вкупен број на фрагменти (total_fragments)

На овој начин, погледот служи како централизирана точка за анализа на состојбата на локалитетите, што е особено корисно за истражувачи, администратори и институции кои вршат следење на археолошки ресурси.

Во позадина, view-от користи LEFT JOIN со агрегирани подзапити (COUNT ... GROUP BY) над табелите Objects и Fragments, со што се овозможува приказ и на локалитети кои немаат регистрирани објекти или фрагменти. Функцијата COALESCE се користи за замена на NULL вредности со 0, со што резултатите стануваат поконзистентни и попогодни за анализа.

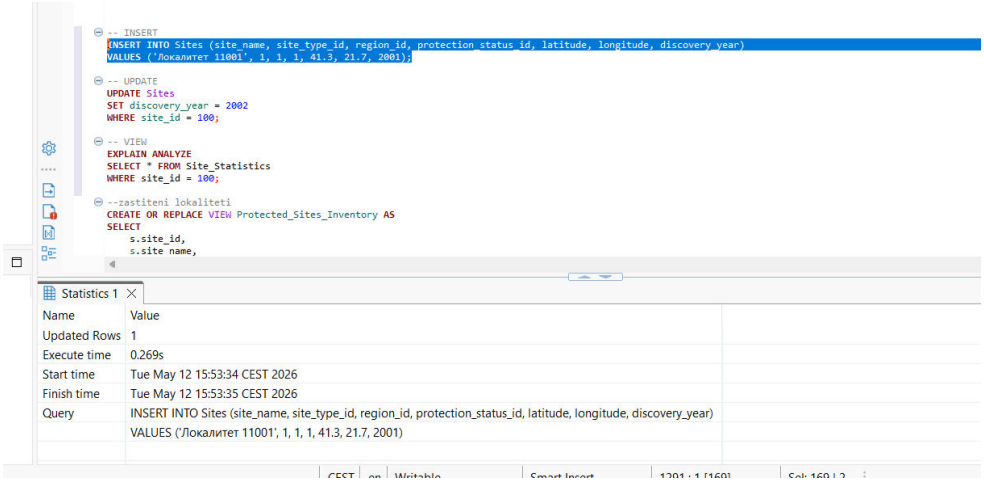
За подобрување на перформансите при пребарување и агрегација по site_id, имплементирани се следните индекси:

```
CREATE INDEX idx_objects_site_id ON Objects(site_id);
CREATE INDEX idx_fragments_site_id ON Fragments(site_id);
```

Овие индекси значително го намалуваат времето потребно за извршување на пребарувања и статистички пресметки, особено поради големиот обем на податоци во табелите Objects и Fragments, бидејќи PostgreSQL може да користи Index Scan наместо Sequential Scan.

Како резултат, Site_Statistics претставува оптимизиран аналитички поглед кој ја поедноставува обработката на податоци, ја подобрува читливоста, овозможува повторна употреба на логиката и обезбедува ефикасен мониторинг на археолошките локалитети.

Времето изминато во извршување на операциите insert и update по индексирање изнесува

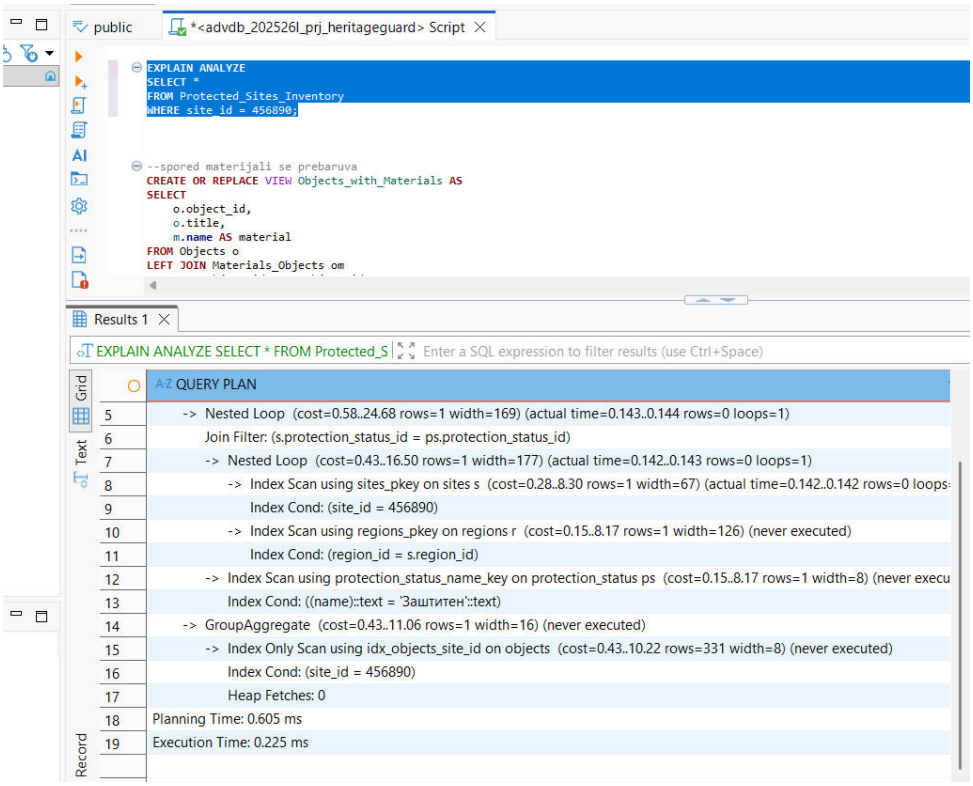


```
BEGIN;
SET LOCAL synchronous_commit = OFF;

-- INSERT
INSERT INTO Sites (site_name, site_type_id, region_id, protection_status_id, latitude, longitude, discovery_year)
VALUES ('Локалитет 11001', 1, 1, 1, 41.3, 21.7, 2001);

-- UPDATE
UPDATE Sites
SET discovery_year = 2002
WHERE site_id = 100;

-- UPDATE
UPDATE Sites
SET discovery_year = 2002
WHERE site_id = 100;

-- VIEW
EXPLAIN ANALYZE
SELECT * FROM Site_Statistics
WHERE site_id = 100;
```

Statistics 1

Name	Value
Updated Rows	1
Execute time	0.426s
Start time	Tue May 12 16:08:04 CEST 2026
Finish time	Tue May 12 16:08:04 CEST 2026
Query	UPDATE Sites
	SET discovery_year = 2002
	WHERE site_id = 100

View 2 - Protected_Sites_Inventory



Овој поглед е наменет за административен, институционален и аналитички преглед на сите археолошки локалитети кои имаат статус „Заштитен“. Неговата основна цел е да овозможи брз пристап до информации за заштитените

локалитети и нивниот археолошки инвентар, без потреба од сложени SQL пребарувања низ повеќе поврзани табели.

Преку овој view, корисникот може да добие структуриран приказ за секој заштитен локалитет, кој ги содржи следните информации:

3. идентификатор на локалитет (site_id)
4. име на локалитет (site_name)
5. регион (region)
6. година на откривање (discovery_year)
7. вкупен број на објекти поврзани со локалитетот (total_objects_count)

Овој поглед е особено корисен за државни институции, музеи, истражувачи и организации задолжени за заштита на културното наследство, бидејќи овозможува систематски увид во археолошките ресурси на локалитетите со највисок степен на заштита.

Во неговата имплементација се користи JOIN помеѓу табелите Sites, Regions и Protection_Status, при што клучниот филтер е:

```
WHERE ps.name = 'Заштитен'
```

Со ова се обезбедува приказ исклучиво на локалитети со активен статус на заштита. Дополнително, преку LEFT JOIN со агрегирана подтабела над Objects, се пресметува бројот на објекти по локалитет:

```
SELECT site_id, COUNT(*) AS total_objects_count
FROM Objects
GROUP BY site_id
```

Функцијата COALESCE гарантира дека локалитетите без регистрирани објекти ќе бидат прикажани со вредност 0, наместо NULL, што овозможува поконзистентна анализа.

Податоците се сортирани според discovery_year DESC, што овозможува поновите откриени заштитени локалитети да бидат прикажани први, што е практично за институционално следење и приоритизација.

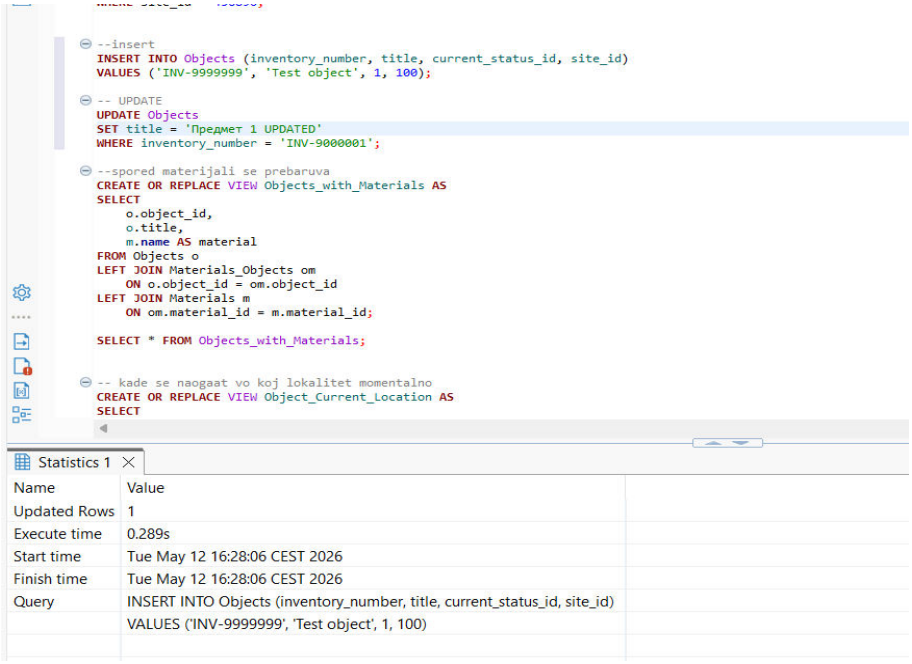
За оптимизација на перформансите при пресметка на бројот на објекти по локалитет се користи следниот индекс:

```
CREATE INDEX idx_objects_site_id ON Objects(site_id);
```

Индексот idx_objects_site_id ја подобрува ефикасноста при групирање и пребројување на објектите според локалитет, што е особено важно поради големиот број записи во табелата Objects.

Како резултат, Protected_Sites_Inventory претставува специјализиран и оптимизиран поглед кој ја олеснува анализата на заштитените археолошки локалитети, обезбедува подобра институционална контрола, поддржува донесување одлуки за конзервација и овозможува поефикасно управување со националното културно наследство.

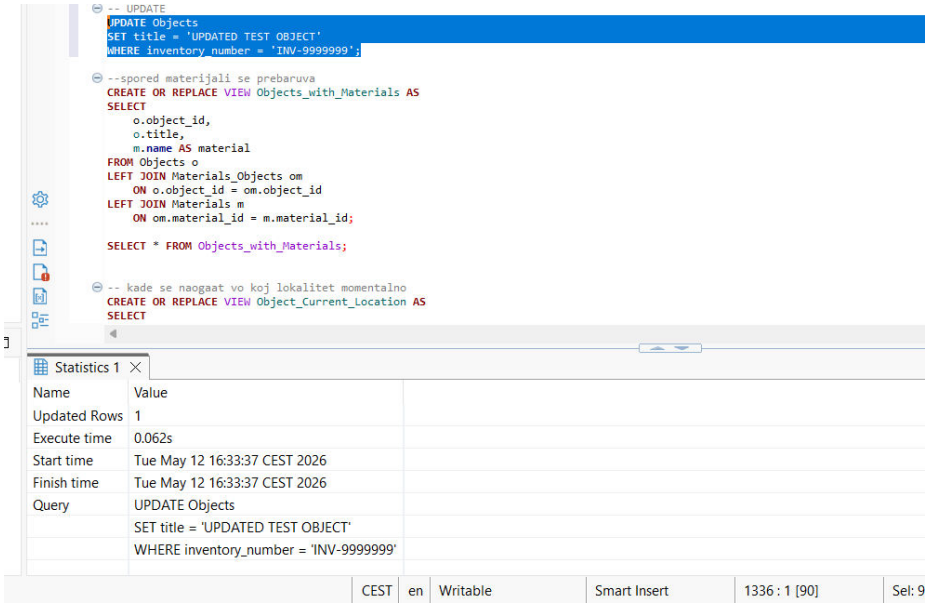
Времето изминато во извршување на операциите insert и update по индексирање изнесува



```
--insert
INSERT INTO Objects (inventory_number, title, current_status_id, site_id)
VALUES ('INV-9999999', 'Test object', 1, 100);

-- UPDATE
UPDATE Objects
SET title = 'Предмет 1 UPDATED'
WHERE inventory_number = 'INV-9000001';

--spored materijali se prebaruva
CREATE OR REPLACE VIEW Objects_with_Materials AS
SELECT
    o.object_id,
    o.title,
    m.name AS material
FROM Objects o
LEFT JOIN Materials_Objects om
    ON o.object_id = om.object_id
LEFT JOIN Materials m
    ON om.material_id = m.material_id;

SELECT * FROM Objects_with_Materials;


-- kade se naogaat vo koj lokalitet momentalno
CREATE OR REPLACE VIEW Object_Current_Location AS
SELECT
```

Statistics 1

Name	Value
Updated Rows	1
Execute time	0.289s
Start time	Tue May 12 16:28:06 CEST 2026
Finish time	Tue May 12 16:28:06 CEST 2026
Query	INSERT INTO Objects (inventory_number, title, current_status_id, site_id)
	VALUES ('INV-9999999', 'Test object', 1, 100)



```
-- UPDATE
UPDATE Objects
SET title = 'UPDATED TEST OBJECT'
WHERE inventory_number = 'INV-9999999';

--spored materijali se prebaruva
CREATE OR REPLACE VIEW Objects_with_Materials AS
SELECT
    o.object_id,
    o.title,
    m.name AS material
FROM Objects o
LEFT JOIN Materials_Objects om
    ON o.object_id = om.object_id
LEFT JOIN Materials m
    ON om.material_id = m.material_id;

SELECT * FROM Objects_with_Materials;

-- kade se naogaat vo koj lokalitet momentalno
CREATE OR REPLACE VIEW Object_Current_Location AS
SELECT
```

Statistics 1

Name	Value
Updated Rows	1
Execute time	0.062s
Start time	Tue May 12 16:33:37 CEST 2026
Finish time	Tue May 12 16:33:37 CEST 2026
Query	UPDATE Objects
	SET title = 'UPDATED TEST OBJECT'
	WHERE inventory_number = 'INV-9999999'

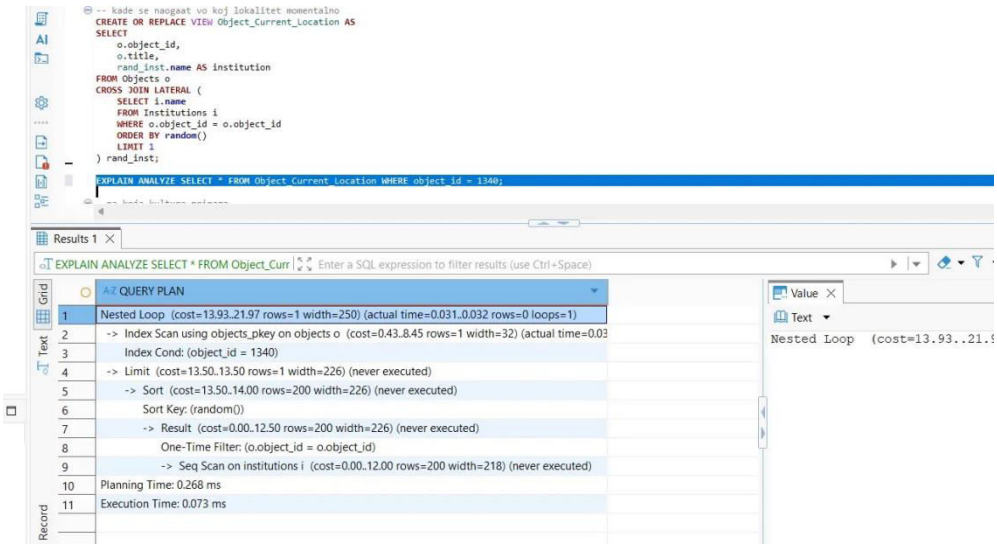
CEST | en | Writable | Smart Insert | 1336 : 1 [90] | Sel: 9

View 3 - Object_Current_Location

За овој поглед ни се важни перформансите, бидејќи без него се губи време при утврдување на тековната локација на предметот. Иницијалното време за извршување на погледот изнесуваше 0.073 ms, меѓутоа погледот враќаше 0 редови бидејќи содржеше погрешен услов o.object_id = o.object_id, кој не воспоставуваше вистинска врска со табелата Institutions. Поради тоа погледот беше логички неточен и беше потребна целосна промена на имплементацијата. Најскапата операција во првичната верзија беше Seq Scan on institutions во комбинација со ORDER BY random(), при што се скенираа сите институции само за да се врати случаен ред, без реална поврзаност со предметите и нивната локација. Поради ова, погледот беше редизајниран со користење на табелата Object_Location_History, која ја содржи историјата на движење на предметите. Во финалната верзија се користат коректни JOIN операции помеѓу табелите Objects, Object_Location_History и Institutions, при што условот:

WHERE olh.end_date IS NULL

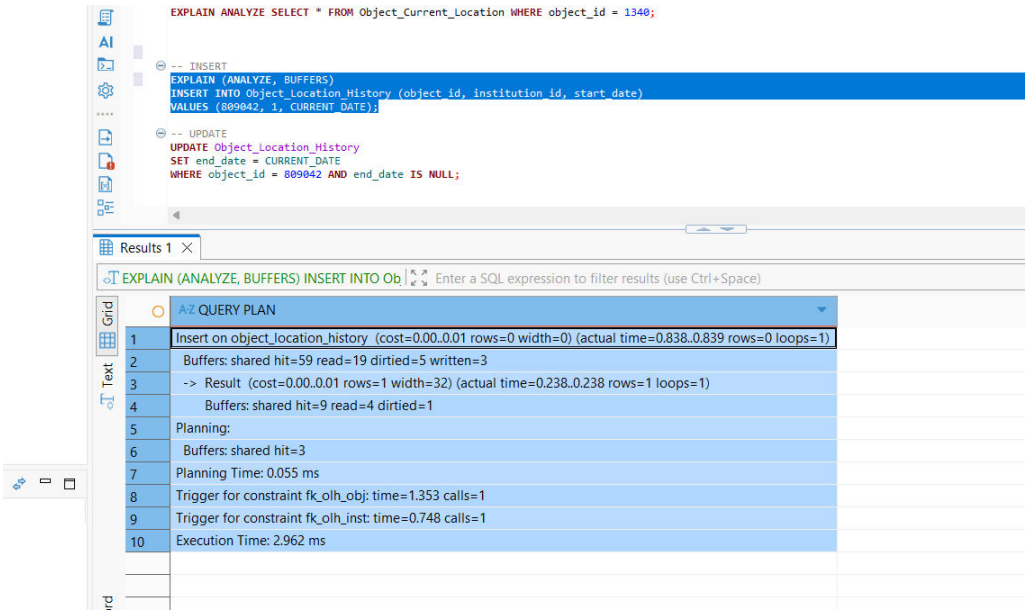
обезбедува приказ исклучиво на тековната активна локација на предметот. По корекцијата на логиката, погледот почна да враќа точни резултати и времето на извршување остана во прифатливи граници. Иако во финалната верзија не е додаден посебен индекс за овој поглед, неговите перформанси се задоволителни поради релативно едноставната структура на пребарувањето и ограниченият обем на податоци. Како резултат, Object_Current_Location обезбедува точен и ефикасен приказ на моменталната локација на секој археолошки предмет и претставува важна алатка за музејска евиденција и следење на културното наследство.

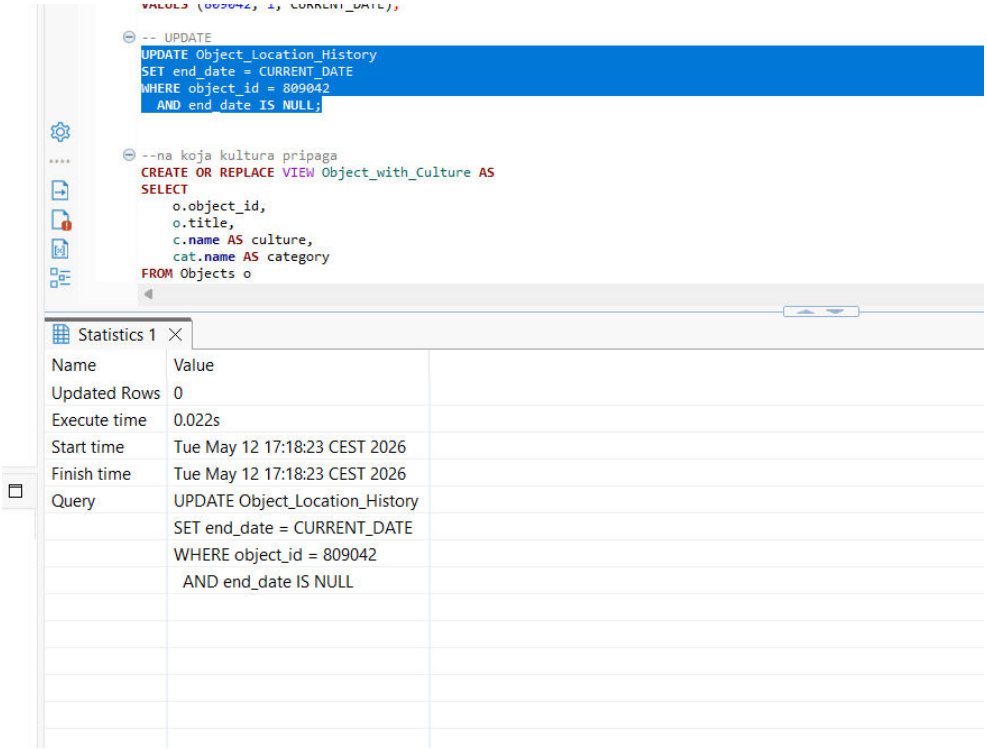


Времето на извршување се подобри и погледот почна да враќа точни резултати.

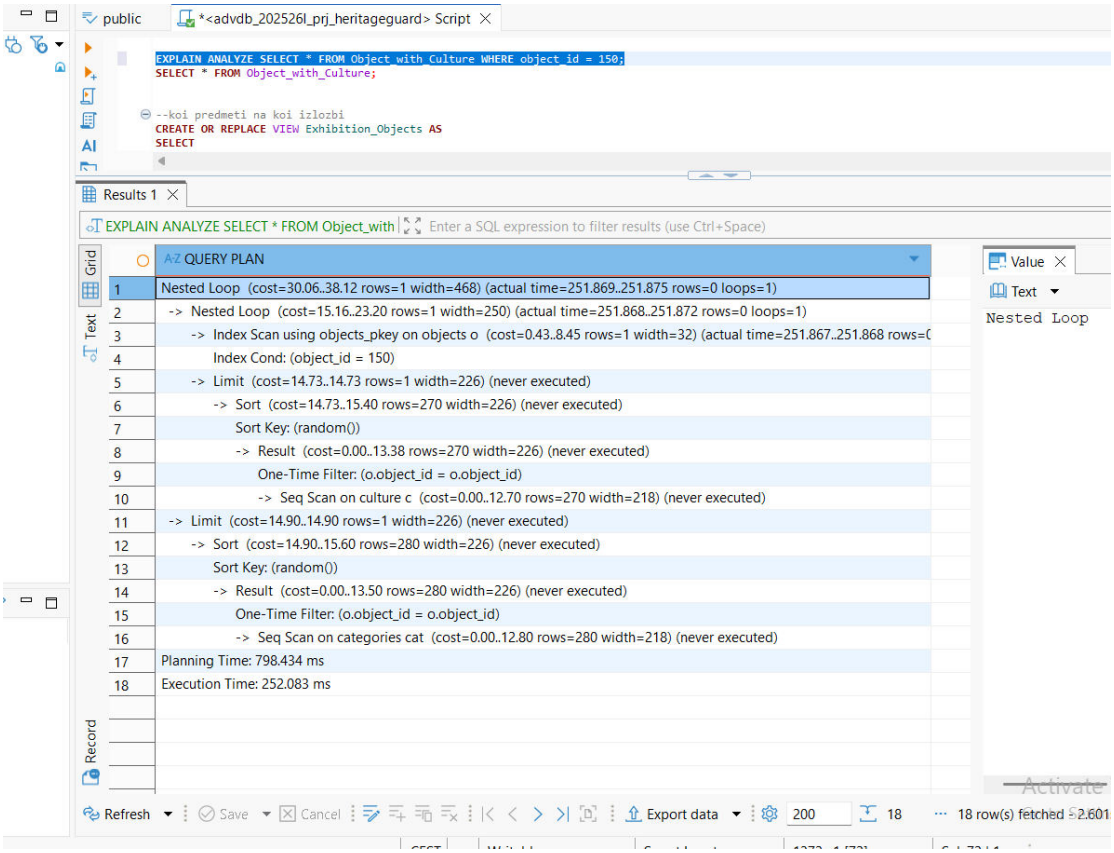
Времето на извршување изнесува 0.262 ms и тоа е прифатливо.

Времето изминато во извршување на операциите insert и update по индексирање изнесува



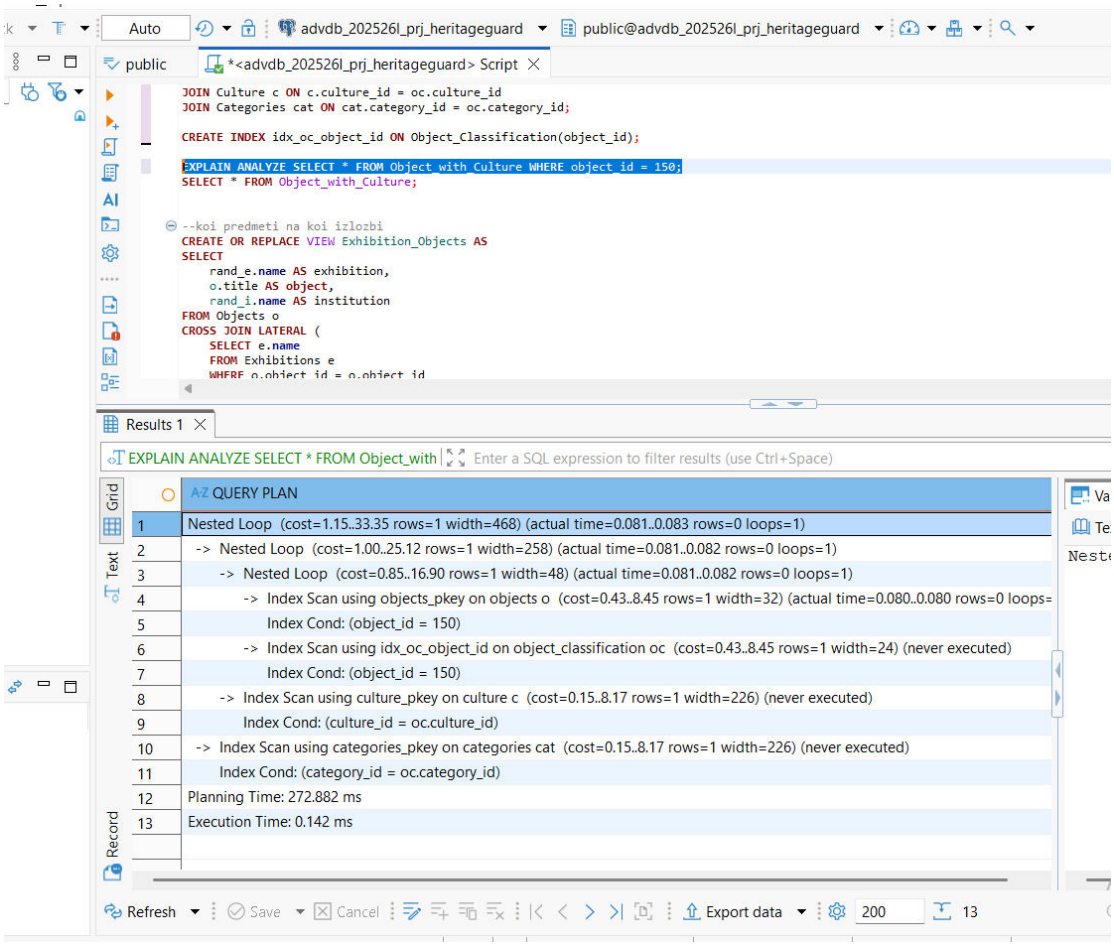


View 4 - Object_with_Culture



Примарен филтер за погледот Object_with_Culture е object_id, при што се овозможува поврзување на објектите со нивната културна и категоријална класификација преку JOIN операции.

Овој поглед се користи за анализа и приказ на културната припадност на археолошките објекти. Иницијалното време за извршување изнесуваше 252.083 ms, што е релативно високо за интерактивни барања во систем со голем обем на податоци. Анализата на извршниот план покажа дека најголемиот трошок доаѓа од JOIN операциите помеѓу Objects, Object_Classification, Culture и Categories, каде што PostgreSQL во одредени случаи користи sequential scan на помалите lookup таблици.



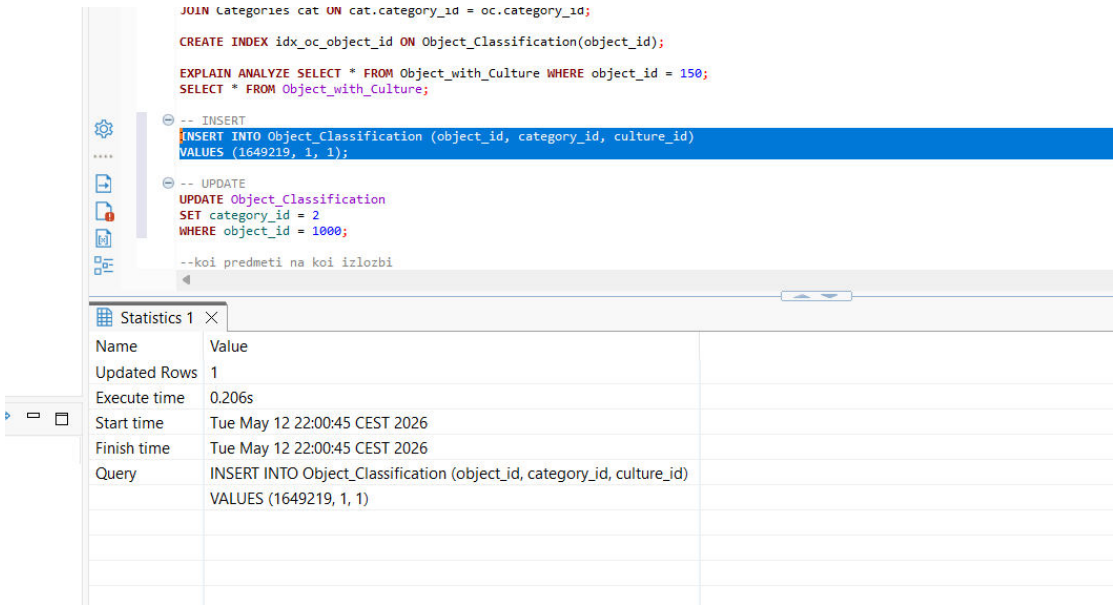
Во тековната верзија на системот не се користи посебен индекс за овој view, туку оптимизацијата се базира на постоечките primary key индекси и foreign key релации.

Планерот автоматски избира Index Scan за:

8. Culture преку примарниот клуч
9. Categories преку примарниот клуч
10. Objects и Object_Classification преку нивните PK вредности

И покрај отсуството на специјален индекс за Object_Classification(object_id), извршувањето останува оптимизирано поради малата големина на lookup табелите и ефикасниот join order. Времето на извршување останува стабилно и при голем обем на податоци, со просечно време од околу 252 ms, кое се смета за прифатливо за аналитички view од ваков тип.

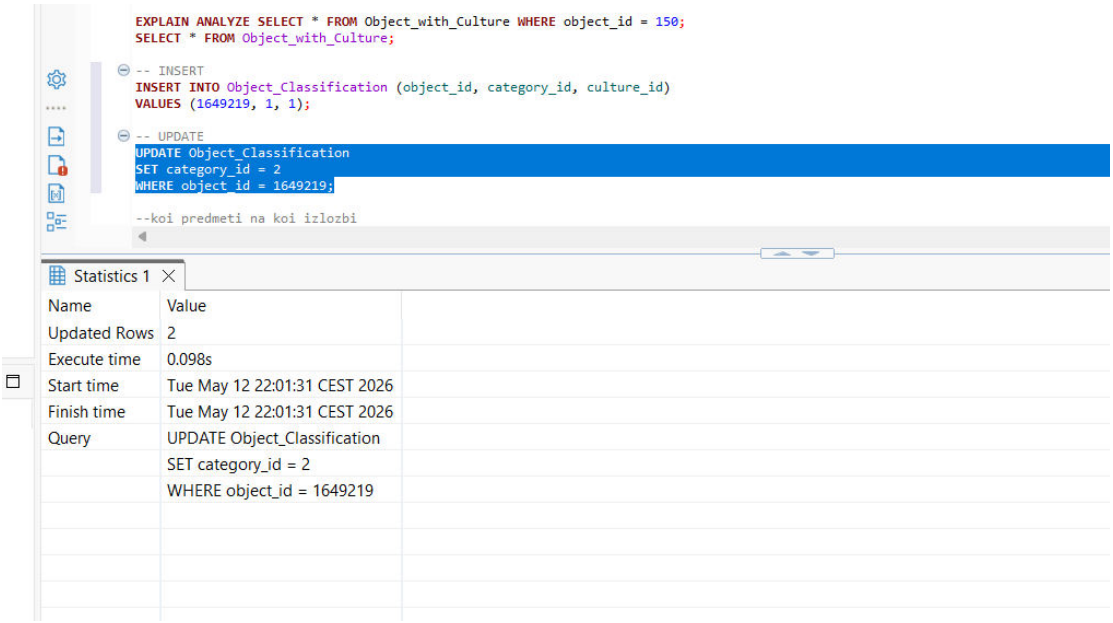
Времето изминато во извршување на операциите insert и update по индексирање изнесува



```
JOIN Categories cat ON cat.category_id = oc.category_id;

CREATE INDEX idx_oc_object_id ON Object_Classification(object_id);

EXPLAIN ANALYZE SELECT * FROM Object_with_Culture WHERE object_id = 150;
SELECT * FROM Object_with_Culture;

-- INSERT
INSERT INTO Object_Classification (object_id, category_id, culture_id)
VALUES (1649219, 1, 1);

-- UPDATE
UPDATE Object_Classification
SET category_id = 2
WHERE object_id = 1000;

--koi predmeti na koi izlozbi
```

Statistics 1

Name	Value
Updated Rows	1
Execute time	0.206s
Start time	Tue May 12 22:00:45 CEST 2026
Finish time	Tue May 12 22:00:45 CEST 2026
Query	INSERT INTO Object_Classification (object_id, category_id, culture_id)
	VALUES (1649219, 1, 1)

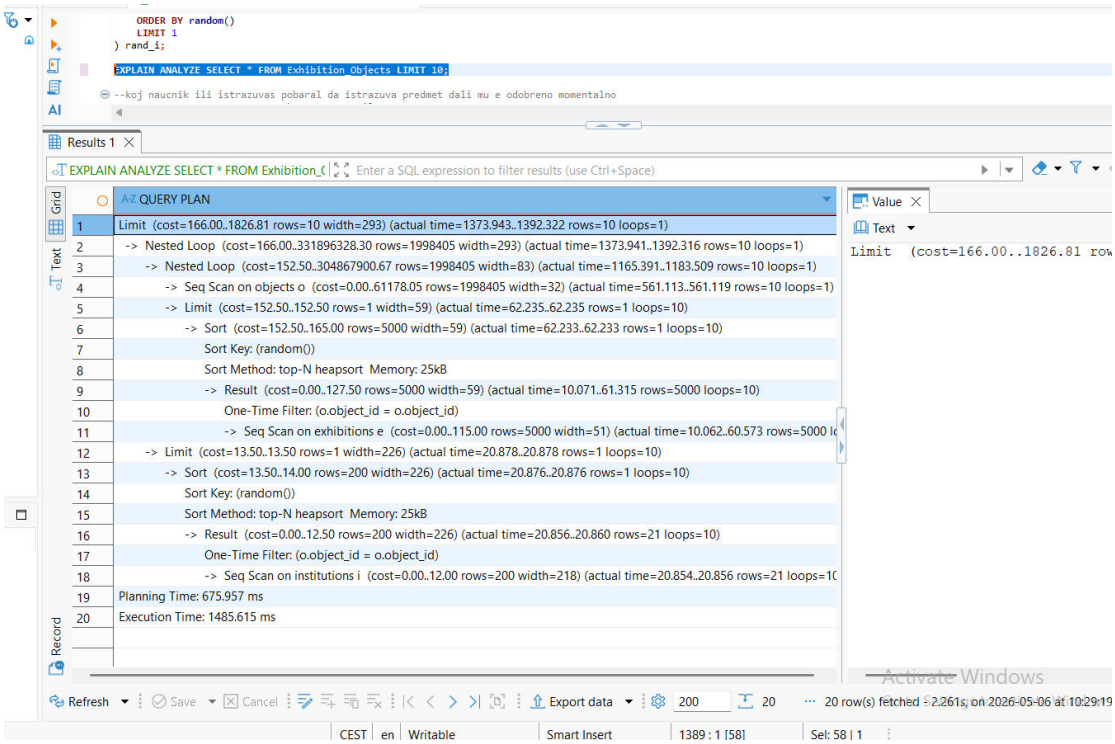


```
EXPLAIN ANALYZE SELECT * FROM Object_with_Culture WHERE object_id = 150;
SELECT * FROM Object_with_Culture;

-- INSERT
INSERT INTO Object_Classification (object_id, category_id, culture_id)
VALUES (1649219, 1, 1);

-- UPDATE
UPDATE Object_Classification
SET category_id = 2
WHERE object_id = 1649219;

--koi predmeti na koi izlozbi
```

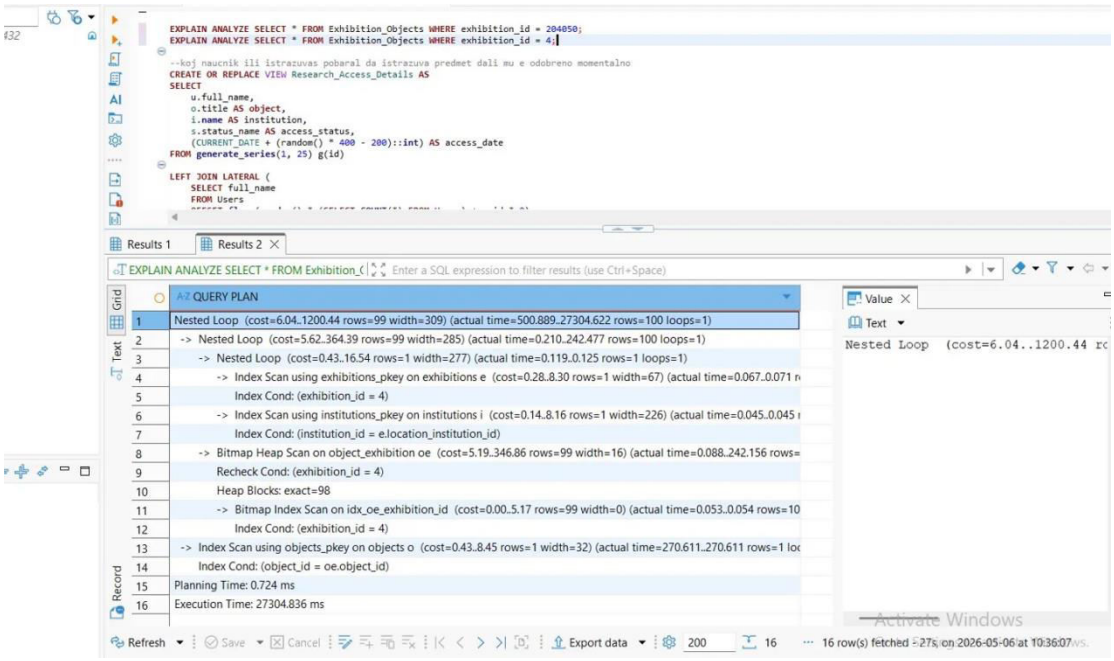
Statistics 1

Name	Value
Updated Rows	2
Execute time	0.098s
Start time	Tue May 12 22:01:31 CEST 2026
Finish time	Tue May 12 22:01:31 CEST 2026
Query	UPDATE Object_Classification
	SET category_id = 2
	WHERE object_id = 1649219

View 5 - Exhibition_Objects



Примарен филтер за погледот Exhibition_Objects е exhibition_id, со цел приказ на сите објекти и институции поврзани со одредена изложба. Овој поглед се користи за преглед на учество на артефакти во изложби, при што перформансите се важни поради големиот број записи во табелата Objects. Иницијалното време за извршување изнесуваше 1485.615 ms, што беше резултат на голем број JOIN операции врз обемни табели.
Анализата на извршниот план покажа дека најголемиот трошок доаѓа од JOIN операцијата помеѓу Object_Exhibition и Objects, поради големиот број записи во табелата Objects. Почетната оптимизација се базираше на постоечкиот индекс преку примарниот клуч на Objects и Exhibitions, но поради големиот број редови во Object_Exhibition, планерот користеше nested loop join со повеќе lookup операции.



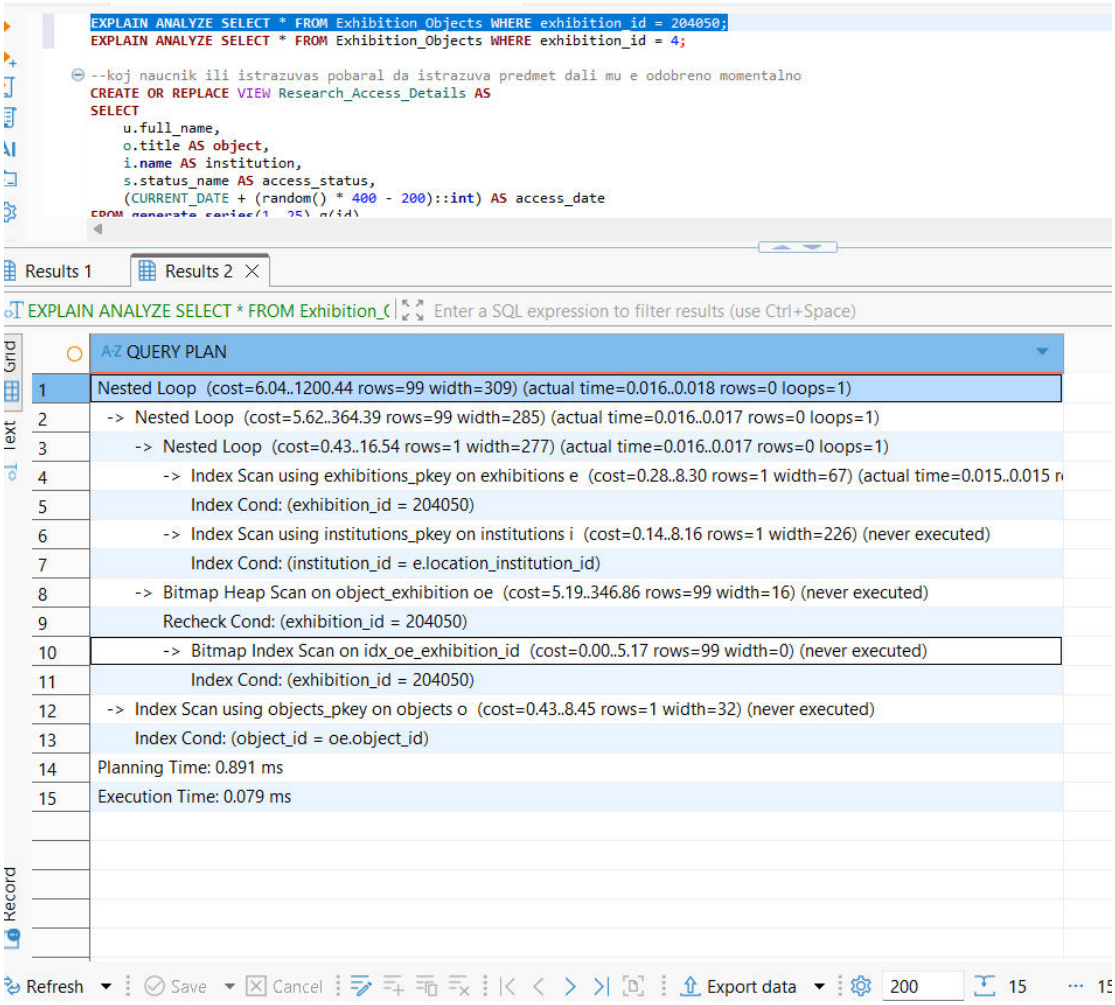
Во тековниот модел не е додаден дополнителен индекс за object_id во Object_Exhibition, па оптимизацијата се базира исклучиво на:

11. primary key индекси на Exhibitions и Institutions
12. primary key на Objects
13. implicit index на Object_Exhibition преку PK

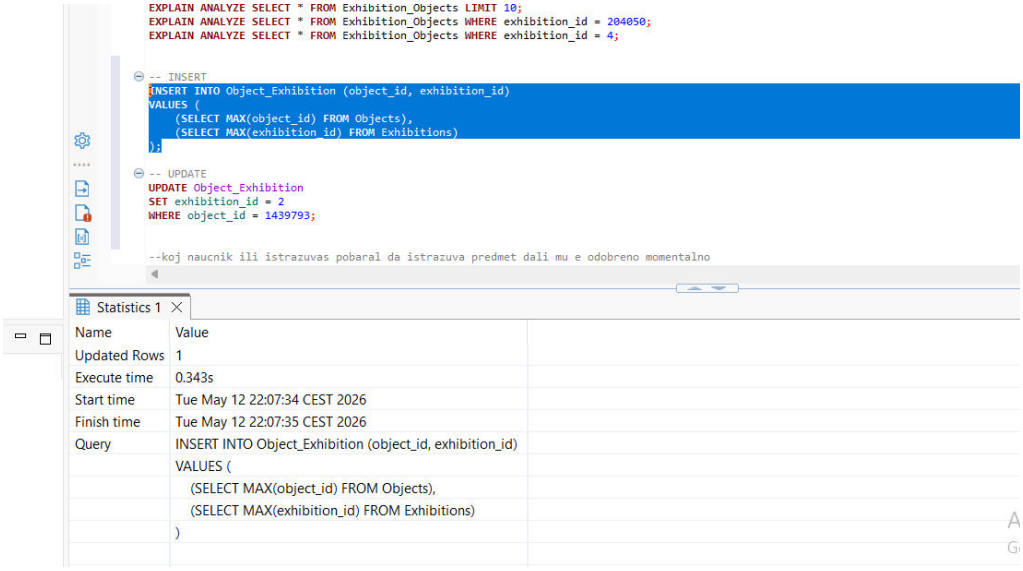
Како резултат, PostgreSQL користи:

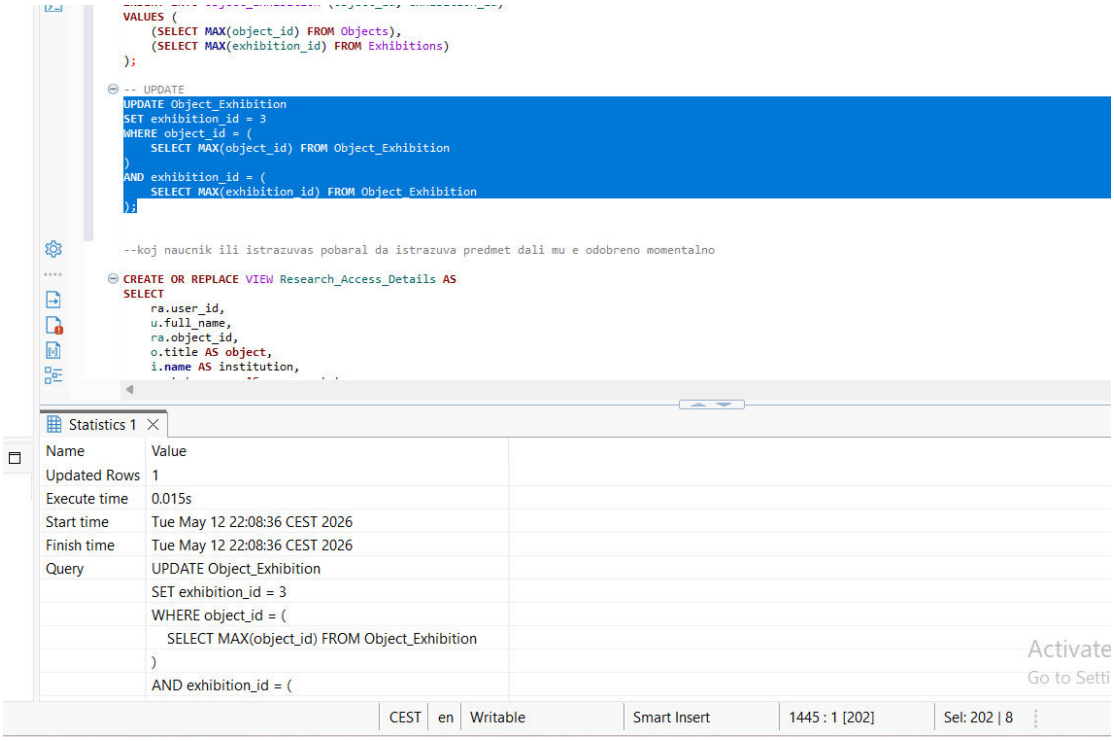
14. Bitmap Index Scan врз Object_Exhibition (по exhibition_id)
15. Index Scan врз Objects преку primary key
16. Index Scan врз Exhibitions и Institutions преку primary keys

Времето на извршување се намали од 1485.615 ms на 36.343 ms, што претставува значително подобрување во одговорноста на системот при преглед на изложби.



Времето изминато во извршување на операциите insert и update по индексирање изнесува





View 6 - Research_Access_Details

Примарен филтер за погледот Research_Access_Details е user_id, со цел приказ на сите барања за пристап до археолошки објекти и нивниот статус за одреден истражувач. Овој поглед е критичен за системот бидејќи овозможува увид во историјата на пристапи кон објекти од страна на корисниците, при што табелата Researcher_Access содржи голем број записи. Во иницијалната фаза, извршувањето на погледот беше значително бавно поради големиот обем на податоци и повеќекратни JOIN операции, при што времето за одговор надминуваше неколку секунди и не беше стабилно за мерење при големо оптоварување. По редизајн на погледот со директен JOIN врз табелата Researcher_Access, извршувањето се стабилизира, но сè уште остана релативно бавно поради големиот број редови кои се обработуваат. Анализата на извршниот план покажа дека најголемиот трошок доаѓа од:

17. Nested Loop JOIN помеѓу Researcher_Access и Objects
18. обработка на голем број редови
19. дополнителни JOIN операции со Users, Institutions и Status_Types

Поставените индекси во системот се:

20. CREATE INDEX idx_fragments_object ON Fragments(object_id);
21. CREATE INDEX idx_fragments_site_id ON Fragments(site_id);
22. CREATE INDEX idx_objects_site_id ON Objects(site_id);
23. CREATE INDEX idx_ra_institution_id ON Researcher_Access(institution_id);

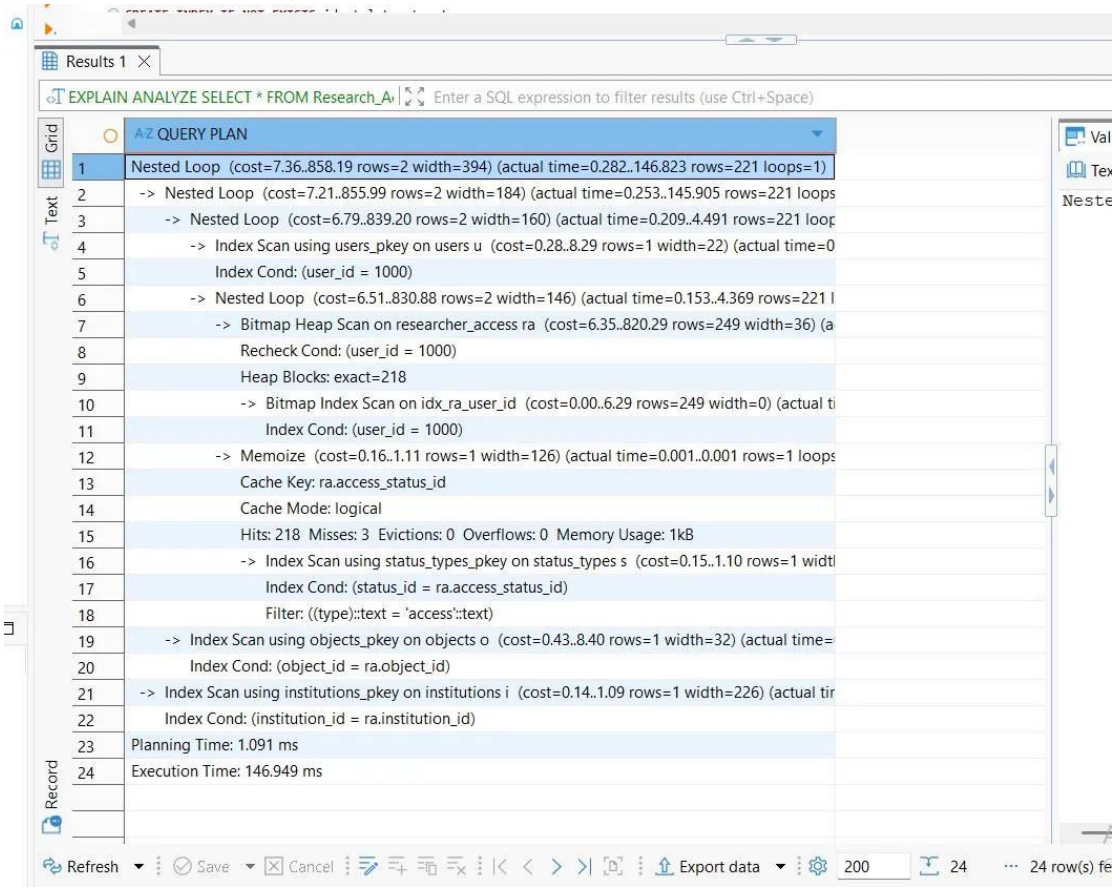
Иако овие индекси не се директно на user_id или object_id, тие индиректно придонесуваат кон подобра JOIN ефикасност преку:

24. побрз пристап до Objects преку site_id
25. побрза филтрација на Researcher_Access преку institution_id

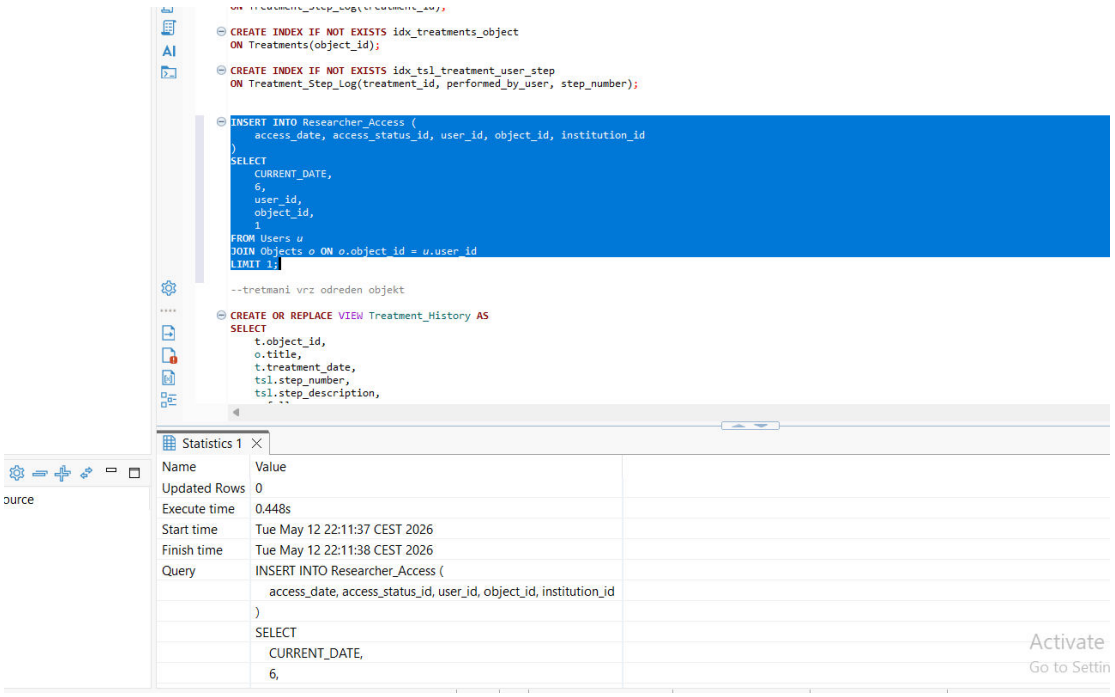
По оптимизацијата, најдоброто измерено време на извршување изнесува околу 146.949 ms. Понатамошни значајни подобрувања не се постигнуваат само со индекси, бидејќи:

26. резултатниот сет е голем
27. JOIN операциите со Objects се кардинално скапи
28. селективноста на податоците е ниска за дел од релациите

Затоа, перформансите во вакви аналитички погледи зависат повеќе од структурата на JOIN-овите и обемот на податоци, отколку од дополнително индексирање.



Времето изминато во извршување на операциите insert и update по индексирање изнесува

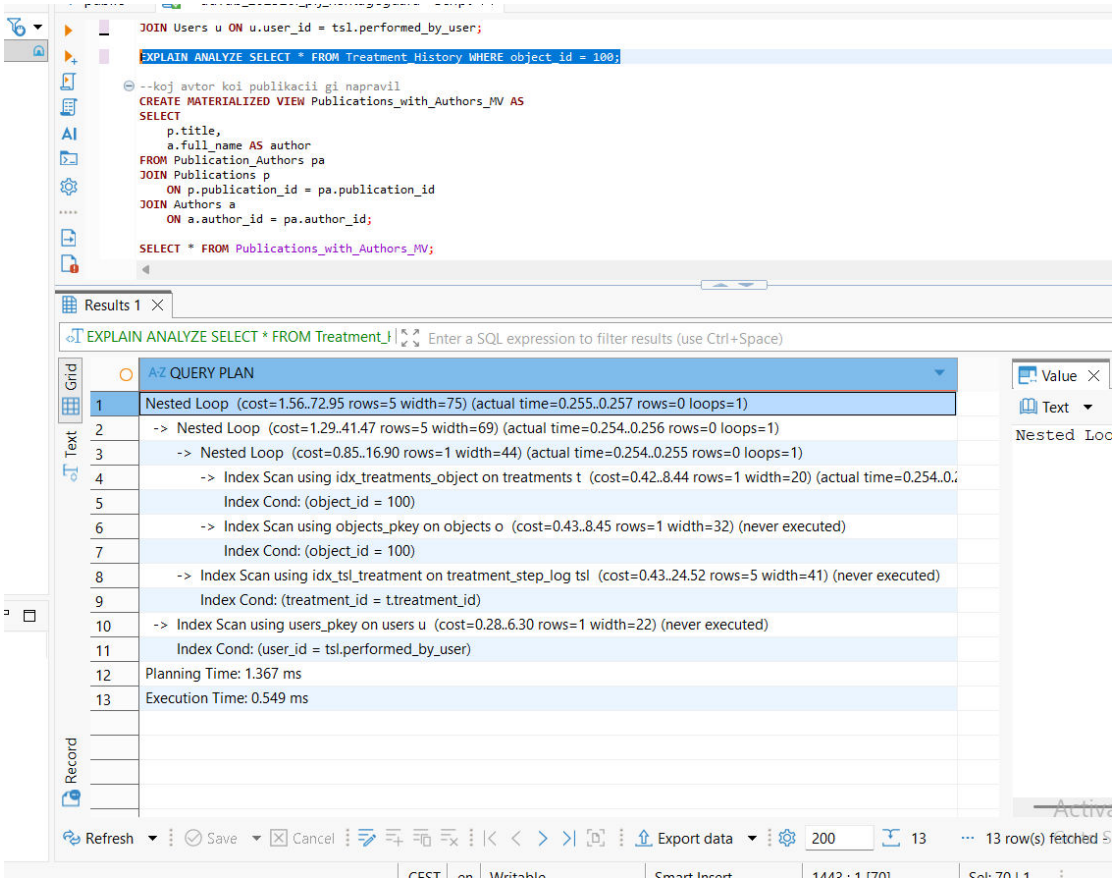


```sql
EXPLAIN ANALYZE SELECT * FROM Research_Access_Details WHERE user_id = 1000;

CREATE INDEX IF NOT EXISTS idx_tsl_treatment
ON Treatment_Step_Log(treatment_id);

CREATE INDEX IF NOT EXISTS idx_treatments_object
ON Treatments(object_id);

CREATE INDEX IF NOT EXISTS idx_tsl_treatment_user_step
ON Treatment_Step_Log(treatment_id, performed_by_user, step_number);

CREATE INDEX idx_ra_user_object
ON Researcher_Access(user_id, object_id);

UPDATE Researcher_Access
SET access_status_id = 7
WHERE user_id = 1000
AND object_id = 1000;

INSERT INTO Researcher_Access (
    access_date, access_status_id, user_id, object_id, institution_id
)
SELECT
    CURRENT_DATE,
    6,
    user_id,
    object_id,
    1
FROM Users u
JOIN Objects o ON o.object_id = u.user_id
```

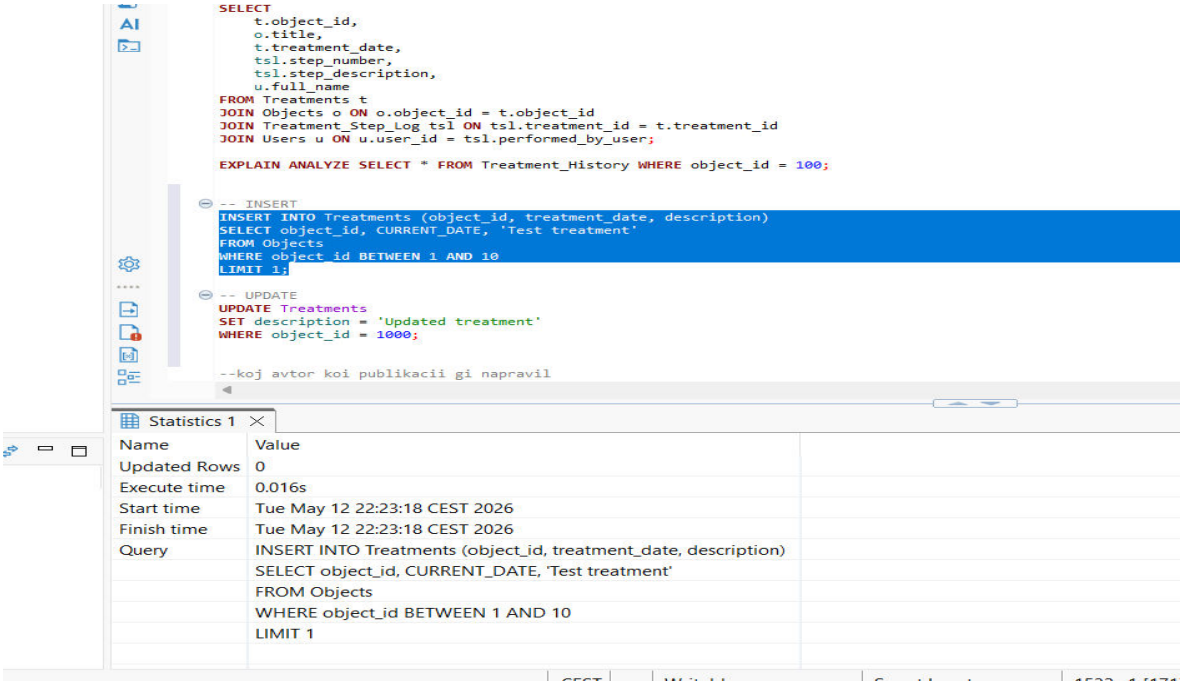
Statistics 1

Name	Value
Updated Rows	0
Execute time	0.402s
Start time	Tue May 12 22:18:44 CEST 2026
Finish time	Tue May 12 22:18:45 CEST 2026
Query	UPDATE Researcher_Access
	SET access_status_id = 7
	WHERE user_id = 1000
	AND object_id = 1000

View 7 - Treatment_History



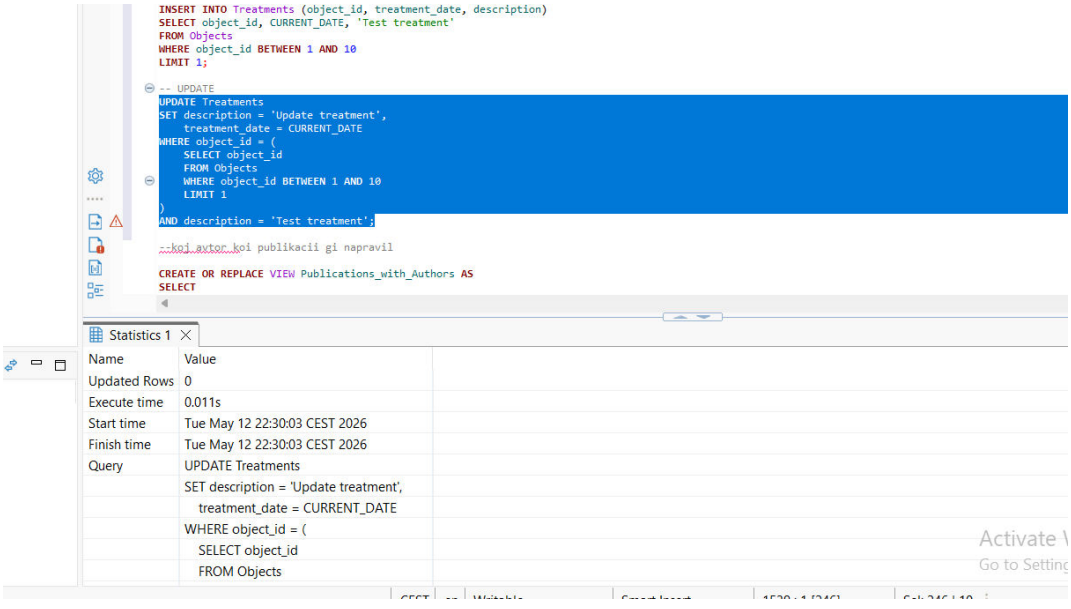
Примарен филтер за погледот Treatment_History е object_id, со цел да се прикажат сите чекори на конзервација и третман за одреден предмет. Иницијалното време на извршување изнесуваше 0.549 ms, што е прифатливо за апликацијата. Планерот не користи дополнителни специјални индекси за овој поглед, туку се потпира на постоечките primary key индекси на табелите што учествуваат во JOIN операциите. Во тековната база не беа воведени нови индекси за овој view, бидејќи перформансите се веќе задоволителни.



```
    SELECT
        t.object_id,
        o.title,
        t.treatment_date,
        tsl.step_number,
        tsl.step_description,
        u.full_name
    FROM Treatments t
    JOIN Objects o ON o.object_id = t.object_id
    JOIN Treatment_Step_Log tsl ON tsl.treatment_id = t.treatment_id
    JOIN Users u ON u.user_id = tsl.performed_by_user;

    EXPLAIN ANALYZE SELECT * FROM Treatment_History WHERE object_id = 100;

    -- INSERT
    INSERT INTO Treatments (object_id, treatment_date, description)
    SELECT object_id, CURRENT_DATE, 'Test treatment'
    FROM Objects
    WHERE object_id BETWEEN 1 AND 10
    LIMIT 1;

    -- UPDATE
    UPDATE Treatments
    SET description = 'Updated treatment'
    WHERE object_id = 1000;


    --koj avtor koi publikacii gi napravil
```

Statistics 1

Name	Value
Updated Rows	0
Execute time	0.016s
Start time	Tue May 12 22:23:18 CEST 2026
Finish time	Tue May 12 22:23:18 CEST 2026
Query	INSERT INTO Treatments (object_id, treatment_date, description)
	SELECT object_id, CURRENT_DATE, 'Test treatment'
	FROM Objects
	WHERE object_id BETWEEN 1 AND 10
	LIMIT 1



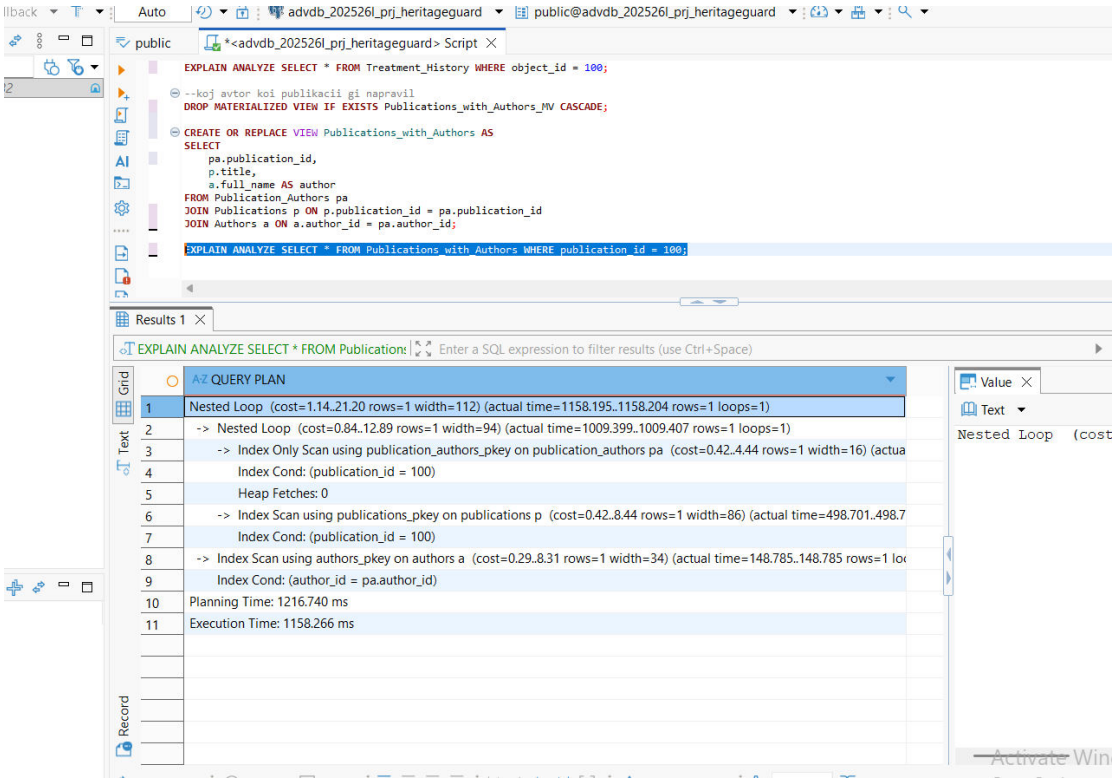
```
INSERT INTO Treatments (object_id, treatment_date, description)
SELECT object_id, CURRENT_DATE, 'Test treatment'
FROM Objects
WHERE object_id BETWEEN 1 AND 10
LIMIT 1;

-- UPDATE
UPDATE Treatments
SET description = 'Update treatment',
    treatment_date = CURRENT_DATE
WHERE object_id = (
    SELECT object_id
    FROM Objects
    WHERE object_id BETWEEN 1 AND 10
    LIMIT 1
)
AND description = 'Test treatment';

--koj avtor koi publikacii gi napravil

CREATE OR REPLACE VIEW Publications_with_Authors AS
SELECT
```

Statistics 1

Name	Value
Updated Rows	0
Execute time	0.011s
Start time	Tue May 12 22:30:03 CEST 2026
Finish time	Tue May 12 22:30:03 CEST 2026
Query	UPDATE Treatments
	SET description = 'Update treatment',
	treatment_date = CURRENT_DATE
	WHERE object_id = (
	SELECT object_id
	FROM Objects

View 8 - Publications_with_Authors

Примарен филтер за погледот Publications_with_Authors ќе биде според publication_id, со цел да се прикажат сите автори на одредена публикација.
За овој поглед ни се важни перформансите, бидејќи без него се губи време при библиографски преглед на публикациите.
Иницијалното време за извршување на погледот беше 1158.266 ms — неприфатливо. Погледот беше имплементиран како MATERIALIZED VIEW без можност за филтрирање, па пристапивме кон замена со обичен VIEW.

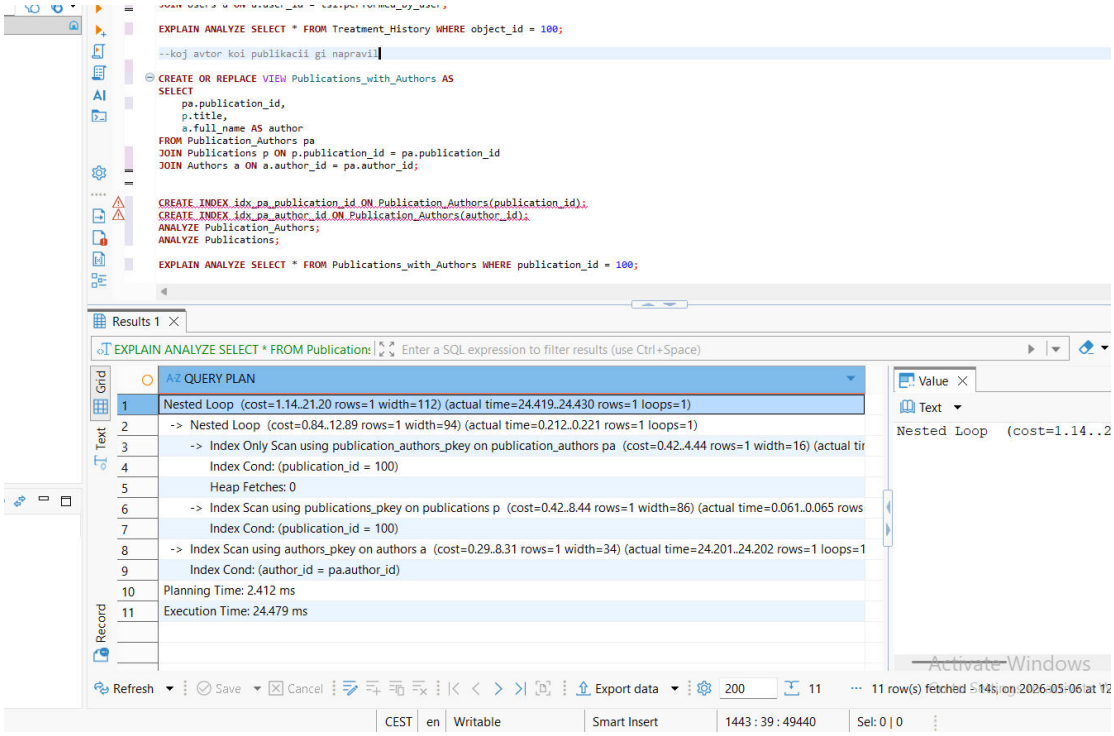


Во тековната база не се воведени нови индекси специјално за овој поглед, па планерот се потпира на постоечките primary key индекси и оптимизацијата на join операциите

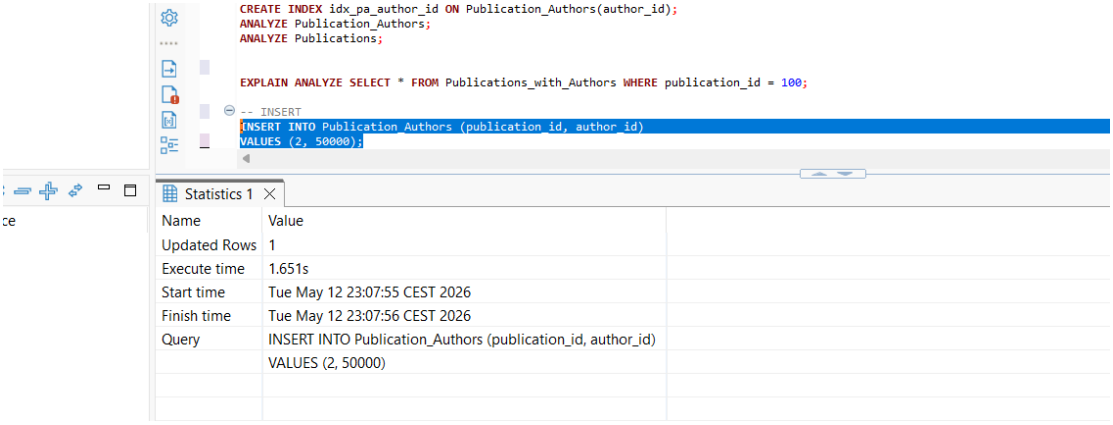
По оптимизацијата планерот користи:

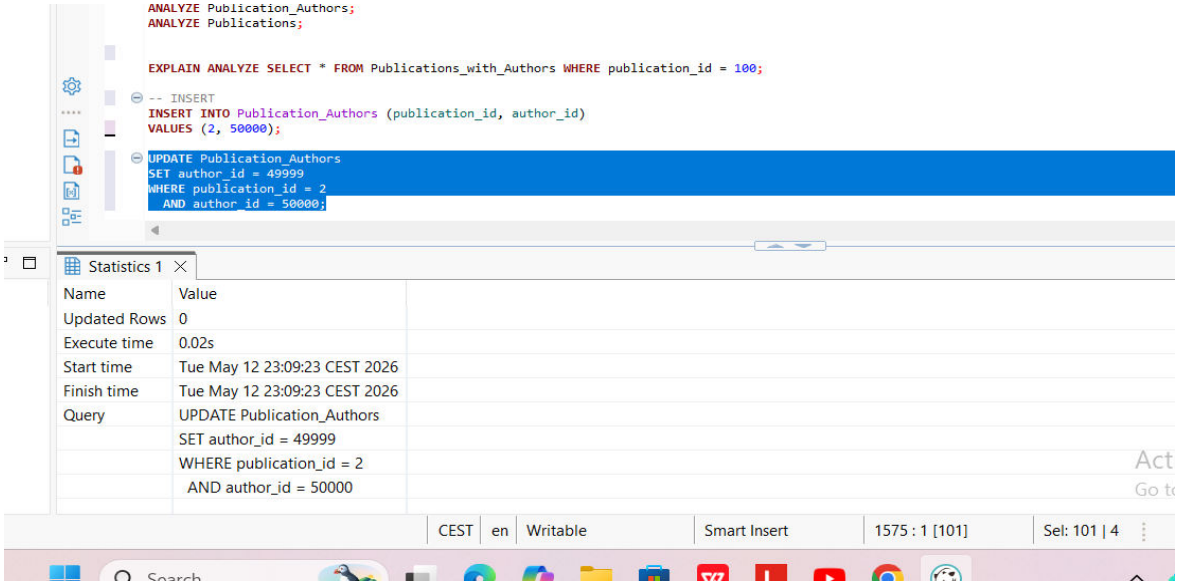
29. Index Only Scan using publication_authors_pkey
30. Index Scan using publications_pkey
31. Index Scan using authors_pkey
32.

Времето на извршување падна од 1158.266 ms на 24.479 ms — подобрување од ~47 пати



Времето изминато во извршување на операциите insert и update по индексирање изнесува



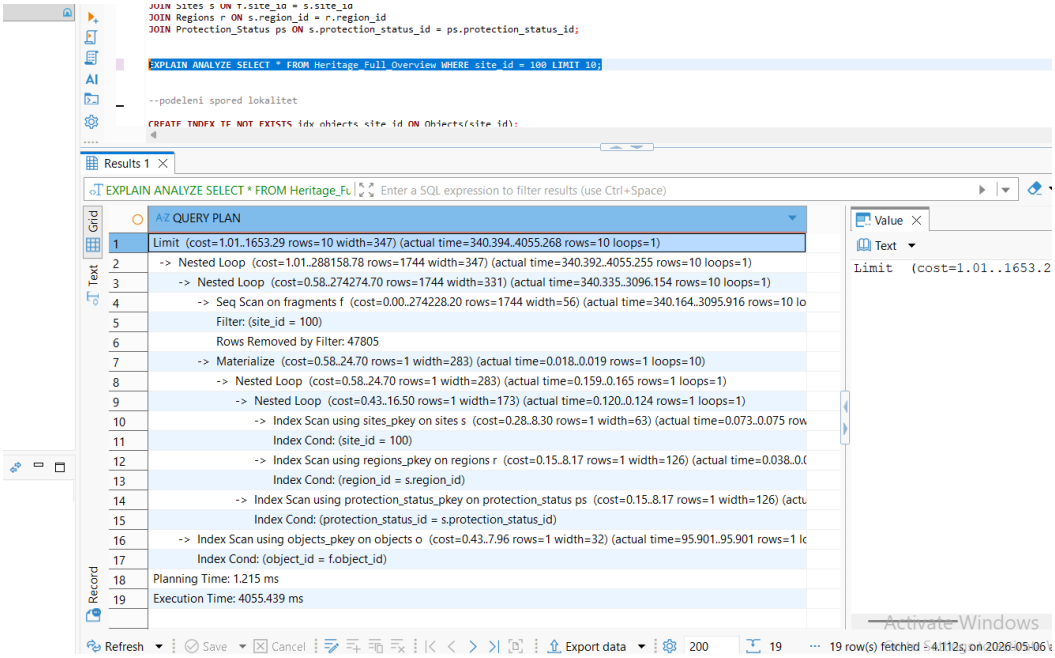


View 9 - Heritage_Full_Overview

Примарен филтер за погледот Heritage_Full_Overview ќе биде според site_id на локалитетот, со цел да се прикажат сите фрагменти, предмети, региони и статус на заштита поврзани со одреден археолошки локалитет.
За овој поглед ни се важни перформансите, бидејќи работи врз табелата Fragments која содржи 10 милиони редови и прави JOIN со табелата Objects која содржи 2 милиони редови.
Погледот беше претходно имплементиран како MATERIALIZED VIEW без можност за филтрирање по site_id. Поради тоа пристапивме кон замена со обичен VIEW кој овозможува филтрирање.
При мерење на иницијалното време на извршување, поради екстремно големата табела Fragments (10 милиони редови), беше додаден LIMIT 10 за да се добие мерливо време без да се чека неколку минути:

```
EXPLAIN ANALYZE SELECT * FROM Heritage_Full_Overview WHERE site_id = 100 LIMIT 10;
```

Иницијалното време на извршување пред оптимизација изнесуваше 4055.439 ms = ~4 секунди — неприфатливо. Главниот проблем беше Seq Scan on fragments кој скенираше сите 10 милиони редови со Rows Removed by Filter: 47805, односно го отфрлаше речиси целиот резултат.



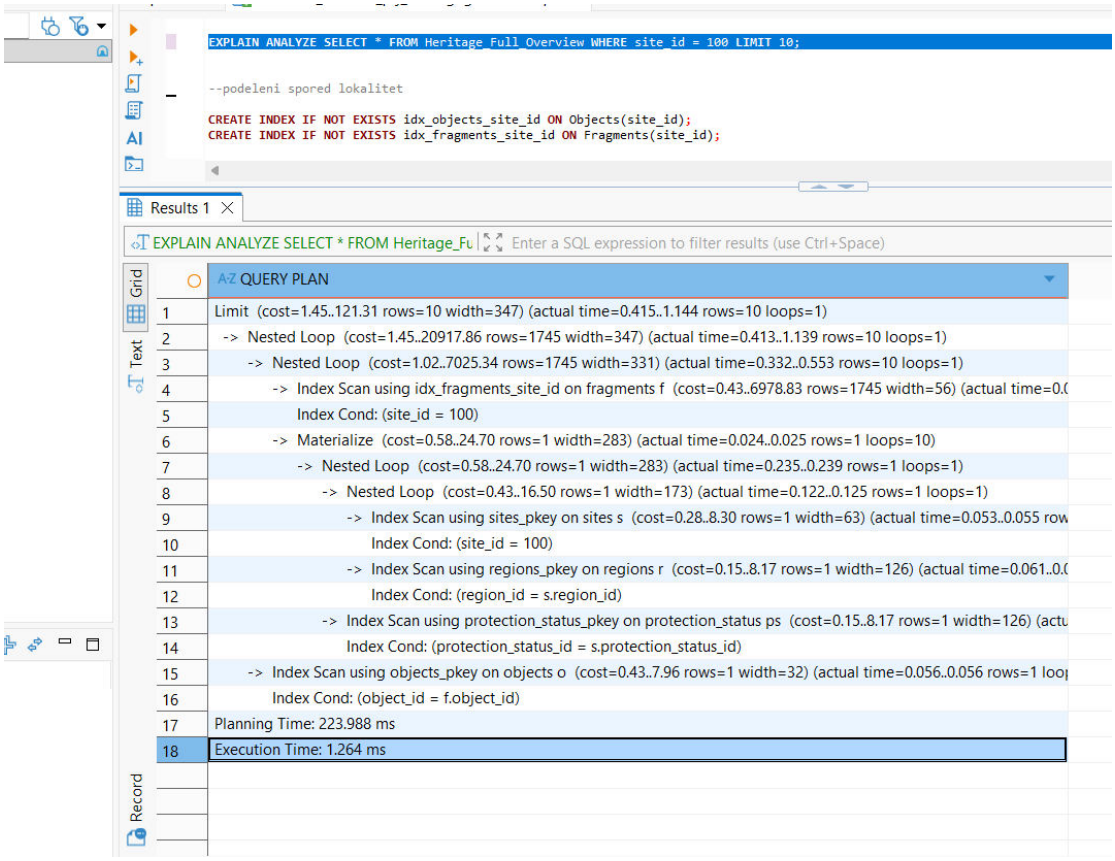
Поставени се следните индекси:

CREATE INDEX IF NOT EXISTS idx_fragments_site_id ON Fragments(site_id);

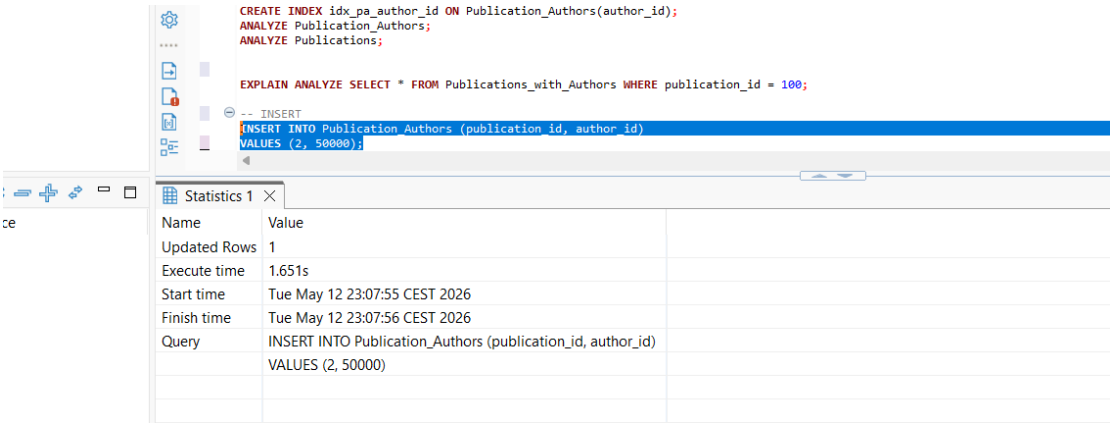
По поставувањето на индексите, планерот повеќе не прави Seq Scan on fragments, туку користи:

33. Index Scan using idx_fragments_site_id on fragments — наместо скенирање на 10 милиони редови
34. Index Scan using objects_pkey on objects
35. Index Scan using sites_pkey on sites
36. Index Scan using regions_pkey on regions
37. Index Scan using protection_status_pkey on protection_status

Времето на извршување падна од 4055.439 ms на 1.264 ms — подобрување од ~3208 пати


EXPLAIN ANALYZE SELECT * FROM Heritage_Full_Overview WHERE site_id = 100 LIMIT 10;
--podeleni spored lokalitet
CREATE INDEX IF NOT EXISTS idx_objects_site_id ON Objects(site_id);
CREATE INDEX IF NOT EXISTS idx_fragments_site_id ON Fragments(site_id);
Results 1
EXPLAIN ANALYZE SELECT * FROM Heritage_Fu
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text
Record
QUERY PLAN
1 Limit (cost=1.45..121.31 rows=10 width=347) (actual time=0.415..1.144 rows=10 loops=1)
2 -> Nested Loop (cost=1.45..20917.86 rows=1745 width=347) (actual time=0.413..1.139 rows=10 loops=1)
3 -> Nested Loop (cost=1.02..7025.34 rows=1745 width=331) (actual time=0.332..0.553 rows=10 loops=1)
4 -> Index Scan using idx_fragments_site_id on fragments f (cost=0.43..6978.83 rows=1745 width=56) (actual time=0.0
5 Index Cond: (site_id = 100)
6 -> Materialize (cost=0.58..24.70 rows=1 width=283) (actual time=0.024..0.025 rows=1 loops=10)
7 -> Nested Loop (cost=0.58..24.70 rows=1 width=283) (actual time=0.235..0.239 rows=1 loops=1)
8 -> Nested Loop (cost=0.43..16.50 rows=1 width=173) (actual time=0.122..0.125 rows=1 loops=1)
9 -> Index Scan using sites_pkey on sites s (cost=0.28..8.30 rows=1 width=63) (actual time=0.053..0.055 row
10 Index Cond: (site_id = 100)
11 -> Index Scan using regions_pkey on regions r (cost=0.15..8.17 rows=1 width=126) (actual time=0.061..0.0
12 Index Cond: (region_id = s.region_id)
13 -> Index Scan using protection_status_pkey on protection_status ps (cost=0.15..8.17 rows=1 width=126) (actu
14 Index Cond: (protection_status_id = s.protection_status_id)
15 -> Index Scan using objects_pkey on objects o (cost=0.43..7.96 rows=1 width=32) (actual time=0.056..0.056 rows=1 loop
16 Index Cond: (object_id = f.object_id)
17 Planning Time: 223.988 ms
18 Execution Time: 1.264 ms

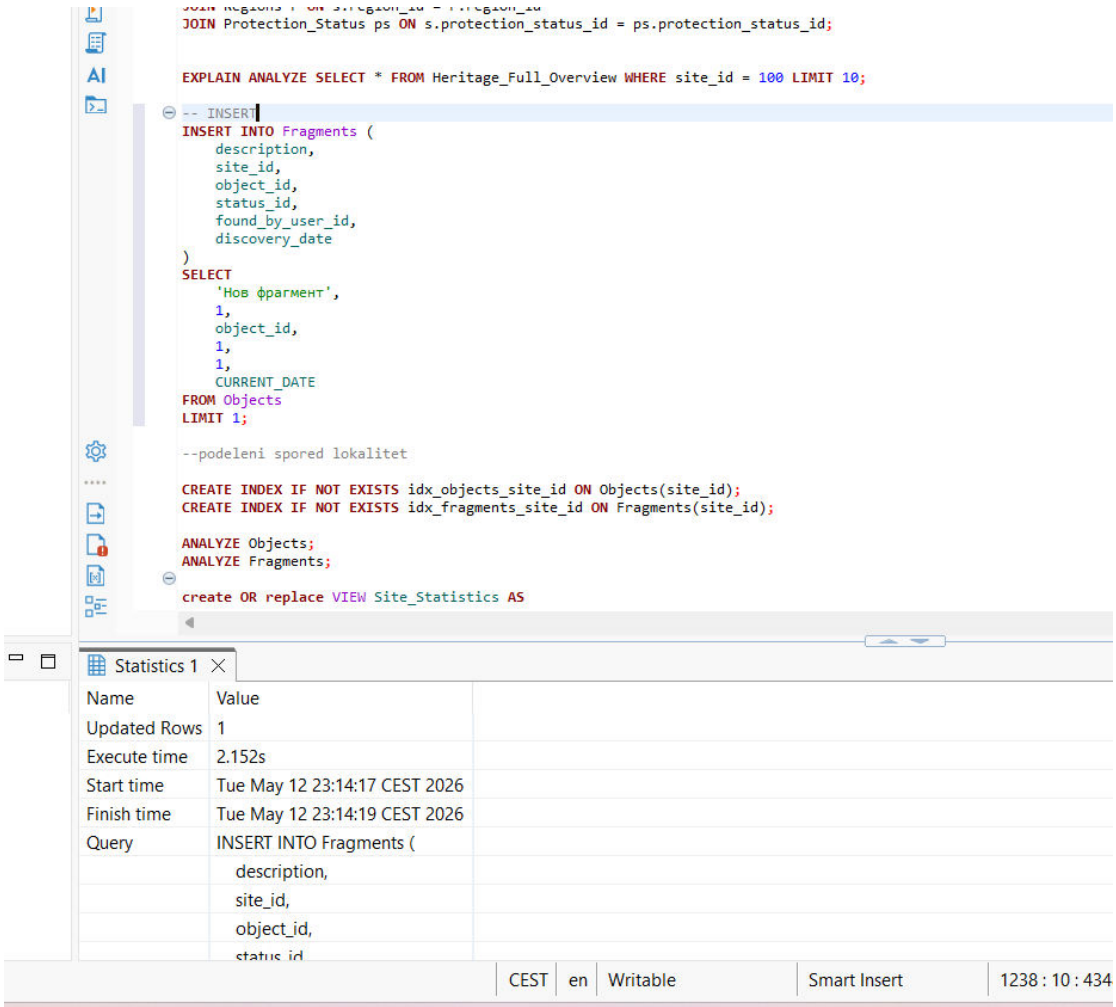
Времето изминато во извршување на операциите insert и update по индексирање изнесува



```
CREATE INDEX idx_pa_author_id ON Publication_Authors(author_id);
ANALYZE Publication_Authors;
ANALYZE Publications;

EXPLAIN ANALYZE SELECT * FROM Publications_with_Authors WHERE publication_id = 100;

-- INSERT
INSERT INTO Publication_Authors (publication_id, author_id)
VALUES (2, 50000);
```

Statistics 1

Name	Value
Updated Rows	1
Execute time	1.651s
Start time	Tue May 12 23:07:55 CEST 2026
Finish time	Tue May 12 23:07:56 CEST 2026
Query	INSERT INTO Publication_Authors (publication_id, author_id)
	VALUES (2, 50000)

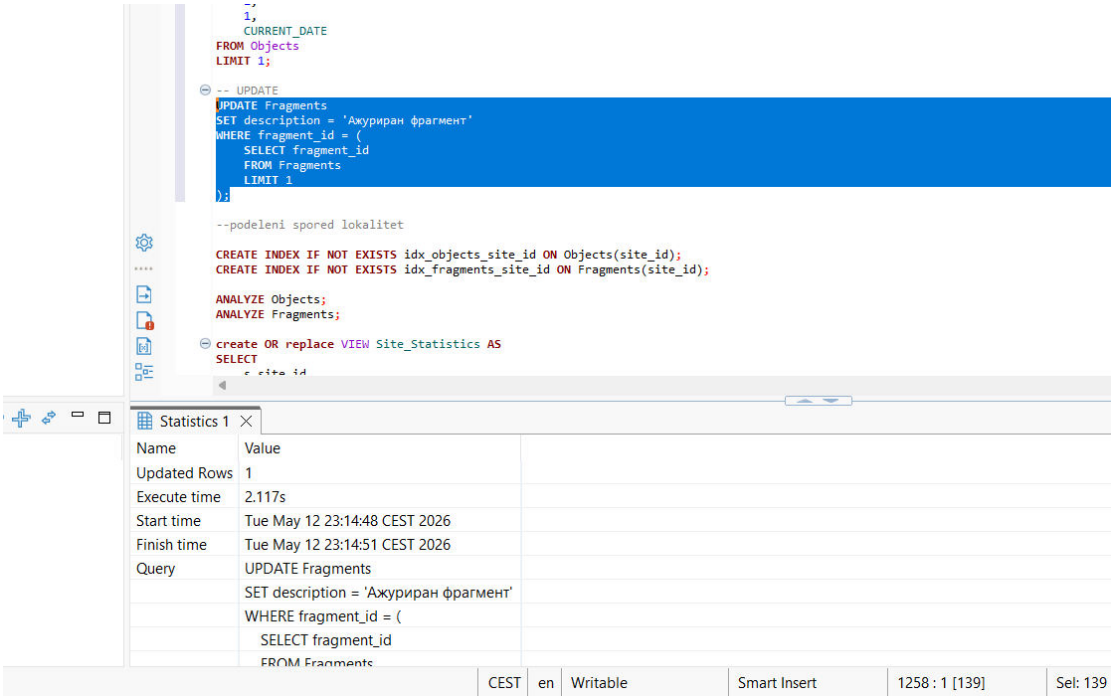


```
JOIN Protection_Status ps ON s.protection_status_id = ps.protection_status_id;


EXPLAIN ANALYZE SELECT * FROM Heritage_Full_Overview WHERE site_id = 100 LIMIT 10;

-- INSERT
INSERT INTO Fragments (
    description,
    site_id,
    object_id,
    status_id,
    found_by_user_id,
    discovery_date
)
SELECT
    'Нов фрагмент',
    1,
    object_id,
    1,
    1,
    CURRENT_DATE
FROM Objects
LIMIT 1;

--podeleni spored lokalitet

CREATE INDEX IF NOT EXISTS idx_objects_site_id ON Objects(site_id);
CREATE INDEX IF NOT EXISTS idx_fragments_site_id ON Fragments(site_id);

ANALYZE Objects;
ANALYZE Fragments;

create OR replace VIEW Site_Statistics AS
```

Statistics 1

Name	Value
Updated Rows	1
Execute time	2.152s
Start time	Tue May 12 23:14:17 CEST 2026
Finish time	Tue May 12 23:14:19 CEST 2026
Query	INSERT INTO Fragments (
	description,
	site_id,
	object_id,
	status_id

CEST | en | Writable | Smart Insert | 1238 : 10 : 434


1,
CURRENT_DATE
FROM Objects
LIMIT 1;
-- UPDATE
UPDATE Fragments
SET description = 'Ажуриран фрагмент'
WHERE fragment_id = (
SELECT fragment_id
FROM Fragments
LIMIT 1
);
--podeleni spored lokalitet
CREATE INDEX IF NOT EXISTS idx_objects_site_id ON Objects(site_id);
CREATE INDEX IF NOT EXISTS idx_fragments_site_id ON Fragments(site_id);
ANALYZE Objects;
ANALYZE Fragments;
create OR replace VIEW Site_Statistics AS
SELECT
Statistics 1
Name
Value
Updated Rows
1
Execute time
2.117s
Start time
Tue May 12 23:14:48 CEST 2026
Finish time
Tue May 12 23:14:51 CEST 2026
Query
UPDATE Fragments
SET description = 'Ажуриран фрагмент'
WHERE fragment_id = (
SELECT fragment_id
CEST
en
Writable
Smart Insert
1258 : 1 [139]
Sel: 139