Optimizacija upita

Indeksne strukture s B-stablima, implementacije spojeva

1

Sadržaj

- Indeksne strukture
- Optimizacija upita
- Naredba *explain plan*

2

Indeksne strukture

3

Performanse BP

- Vremenom može doći do pogoršanja performansi
- Neki razlozi:
  - rast BP,
  - nove putanje pristupa podacima,
  - dodatni korisnici,
  - promene u poslovanju,
  - **neoptimizovani upit**

4

4

Performanse BP

- Optimizacijom upita često mogu postići veći pozitivni efekti na performanse aplikacije
  - nego optimizacijom hardvera, podešavanjem parametara operativnog sistema ili konfiguracije SQL servera
- Neoptimizovani upit nad bazom podataka može da zauzme veliki broj raspoloživih resursa
  - i na taj način da smanji uticaj uloženih sredstava na performanse čitave aplikacije

5

5

Uzroci loših performansi upita

- Neki od razloga:
  - loše projektovana BP
    - engl. *Poor Database Design*
  - skeniranje tabela
  - nedostatak indeksa i neodgovarajuci izbori indeksiranja
    - engl. *Poor indexing*
  - ako se ne upotrebljavaju raspoloživi indeksi
  - zastarele statisticke informacije BP
    - engl. *Inaccurate Statistics*
  - neoptimalno spojene tabele i loša specifikacija upita
    - engl. *Non-Set-Based Operations, Poor Query Design*
  - previše međusobnog blokiranja transakcija i zastoja
    - engl. *Excessive blocking and deadlocks*
  - plan izvršenja upita koji se ne kešira
    - engl. *Nonreusable Execution Plan*

6

6

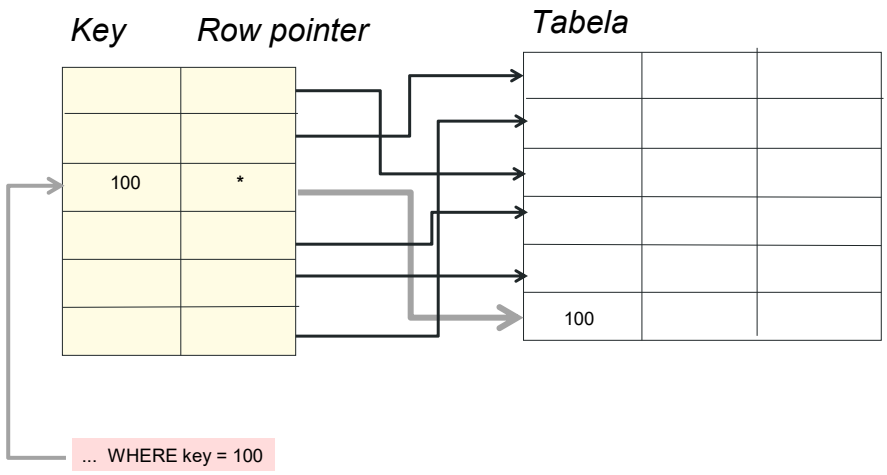
Tehnike indeksiranja

- **Indeks je objekat baze podataka koji obezbeđuje direktan, brz pristup torkama u tabeli**
- Indeks ne menja podatke on samo predstavlja brzu pristupnu putanju do podatka
- U praksi, pravljenje indeksa je iterativan proces
- DBMS automatski održava i koristi indekse
- Kolona na kojoj se kreira indeks se zove ***key* kolona**

7

7

Indeksi



8

8

Indeksi - Row pointer (Rowid)

```
select rowid, ime from radnik;
```

AAAGKcAAEAAAAGNAAA	Pera
AAAGKcAAEAAAAGNAAB	Milan
AAAGKcAAEAAAAGNAAC	Eva
AAAGKcAAEAAAAGNAAD	Moma
AAAGKcAAEAAAAGNAAE	Rastko

Broj File-a **Adresa bloka** **Adresa torke**

9

9

Indeksi

```
select rowid, nap from projekat;
```

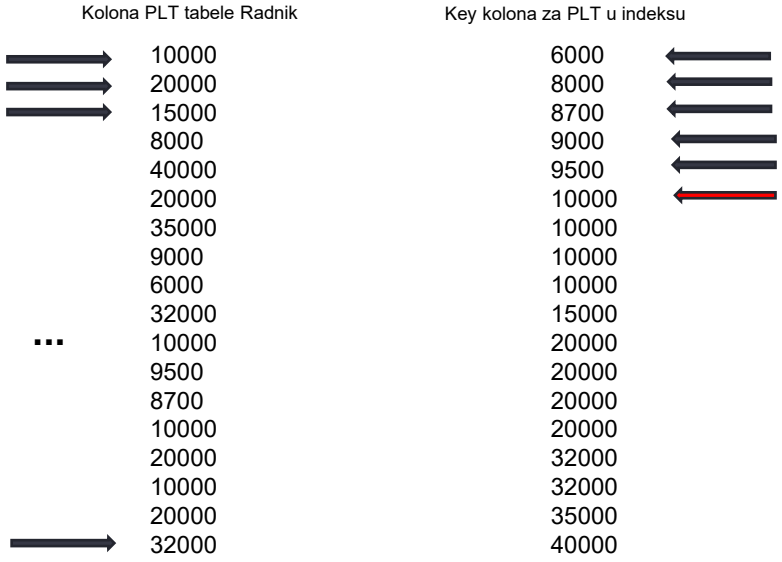
AAAGKeAAEAAAAGdAAA	Projekat jacanja sistema osiguranja dep.
AAAGKeAAEAAAAGdAAB	Mali Biznis kredit
AAAGKeAAEAAAAGdAAC	Internet bankarstvo
AAAGKeAAEAAAAGdAAD	Osiguranje
AAAGKeAAEAAAAGdAAE	Gotovinski krediti
AAAGKeAAEAAAAGdAAF	Ziro racuni
AAAGKeAAEAAAAGdAAG	Refinansiranje kredita
AAAGKeAAEAAAAGdAAH	Data Warehouse OB

10

10

Indeksi

```
select plt
from radnik
where plt < 10000;
```



11

11

Tehnike indeksiranja

- Indeks
  - Ako se često radi verifikacija na osnovu neke kolone onda se kreira indeks
    - smanjuje broj verifikacija vrednosti
  - Indeks održava redosled podataka
    - podrazumevana vrednost za indeks je ascending order
  - Ako ne postoji potreba za drugim podacima iz tabele onda se i ne pristupa tabeli već samo indeksu
  - Unos, izmena i brisanje torki iz tabele se automatski propagira na relevantne indekse
  - Tabela može imate više indeksa
    - **ne preveliki broj indeksa po tabeli** (maksimalno do 3-4)
      - **zbog propagacije operacija ažuriranja** nad podacima

12

12

Indeksi

- Indeks se koristi:
  - kada se kolona često koristi u traženju ili u izrazima upita
  - kada se koristi u spojevima tabela
  - kada je velika selektivnost kolone
    - selektivnost = d / n
      - (broj različitih vrednosti / broj torki tabele)
    - selektivnost se smatra niskom, ako je manja od 20%.
  - za strani ključ

13

13

Indeksi

- Indeks se ne koristi:
  - kada kolona ili izraz imaju samo nekoliko različitih vrednosti
    - izuzev bitmap indeksa u DW
  - kada se često radi *update* date kolone
  - kada se kolone koriste samo u okviru funkcija ili izraza

14

14

Indeksi

- Kreiranje indeksa

```
CREATE [UNIQUE] INDEX index_name
ON table_name (column1, column2, ...) [REVERSE];
```

- Brisanje indeksa

```
DROP INDEX index_name;
```

15

15

Tipovi indeksa

- *Simple index*
  - Kreira se nad jednom kolonom
- *Composite index*
  - Kreira se nad više kolona
- *Unique index*
  - Kreira se nad kolonom/kolonama koja treba da bude unique
- *Reverse index*
  - Ako najčešće tražimo najveću vrednost
  - Ne opterećuje insert naredbu
- *Function based index*
  - Kreira se nad izrazima ili funkcijama

16

16

Simple index – primer

- Kreirati prosti (*simple*) indeks nad kolonom PLT u tabeli RADNIK:

```
CREATE INDEX idx_plt ON radnik (plt);
```

- Napomene i korisni upiti:

`select * from tab;`
  - sve tabele i poglede koje imamo u šemi

```
select constraint_name from user_constraints
where table_name='RADNIK';
```
  - sva ograničenja u okviru tabele

`desc user_indexes;`
  - koje kolone ima tabela koja čuva podatke o indeksima

```
select index_name from user_indexes
where table_name = 'RADNIK';
```
  - provera koji sve indeksi postoje

17

17

Composite index

- Kompozitni indeks sadrži više od jedne kolone
- Treba razmotriti kreiranje kompozitnog indeksa na kolonama koje se često koriste zajedno u uslovima `WHERE` klauzule kombinovanim sa `AND` operatorom
  - Posebno ako je kombinovana selektivnost bolja od selektivnosti svake kolone pojedinačno
- Ako nekoliko upita selektuje isti skup kolona treba razmotriti o kreiranju kompozitnog indeksa na svim tim kolonama

18

18

Composite index

- Redosled kolona
  - Stavlja se na prvo mesto kolona sa najviše unique vrednosti jer će to smanjiti skup torki rezultata
- Unapređuje selektivnost
  - Dve ili više kolona sa malom selektivnošću mogu se kombinovati tako da formiraju indeks sa višom selektivnošću
- Redukuje I/O
  - Ako se sve kolone koje se selektuju u okviru upita nalaze u kompozitnom indeksu onda Oracle baza podataka vraća ove vrednosti bez pristupa tabeli

19

19

Composite index – primer

- Kreirati kompozitni indeks nad kolonama IME i PRZ u tabeli Radnik:

```
CREATE INDEX ind_cp ON radnik(ime, prz);
```

20

20

Unique i Reverse index – primeri

- Kreirati jedinstveni (unique) indeks nad kolonom NAP u tabeli Projekat:

```
CREATE UNIQUE INDEX ind_uq ON projekat (nap);
```

- Kreirati reverzni indeks nad kolonom GOD u tabeli Radnik:

```
CREATE INDEX ind_rv ON radnik (god) REVERSE;
```

21

21

Function based index

- Izračunava vrednost izraza u kojem učestvuju jedna ili više kolona i smešta ih u indeks
  - upiti mogu dobiti vrednost izraza iz indeksa umesto da je izračunavaju
- Izraz može biti:
  - aritmetički izraz ili
  - izraz koji sadrži funkcije
    - SQL funkcije
    - PL/SQL funkcije
    - package funkcije
- Indeks ne sme da sadrži null vrednosti
  - koristiti `NVL` funkciju ukoliko postoji mogućnost za `null` vrednost
- Pogledati:
  - https://oracle-base.com/articles/8i/function-based-indexes

22

22

Function based index – primer

- Kreirati indeks koji je pogodan za dati upit:

```
SELECT * FROM radnik
WHERE plt+pre > 50000
```

- Funkcijski indeks nad izrazom (PLT+PRE) u tabeli Radnik:

```
CREATE INDEX ind_fb
ON radnik (plt+pre);
```

23

23

Default Indeksi

- Oracle Server implicitno kreira i koristi indekse kada se definišu sledeća ograničenja:
  - ograničenje primarnog ključa i
  - ograničenje jedinstvenosti vrednosti obeležja

```
select index_name
from user_indexes
where table_name = 'PROJEKAT';
```

24

24

Indeksi i strani ključ

- Indeks se ne kreira automatski na stranom ključu
- Ako se kolone stranog ključa često koriste u uslovima spajanja onda treba kreirati ključ na tim kolonama
- DML operacije se takođe unapređuju ovom vrstom indeksa
  - Brisanje ili izmena torke iz tabele roditelja zahteva proveru da ne postoji ni jedna torka u tabeli potomka koja je referencira
  - Bez indeksa mora se proći kroz čitavu tabelu potomka

25

25

Indeksi i strani ključ

- Kreirati indeks na koloni RUK u tabeli projekat.

```
CREATE INDEX ind_fk ON projekat(ruk);
```

26

26

Optimizacija upita

27

Optimizator

- Faktori koje razmatra optimizator:
  - Kardinalitet (engl. *cardinality*)
  - Pristupni put (engl. *access path*)
  - Metod spajanja (engl. *join method*)
  - Redosled spajanja (engl. *join order*)
  - Paralelizacija izvršavanja (engl. *parallel execution*)

28

28

Optimizator

- Biranje pristupnog puta
  - Dostupni pristupni putevi za naredbu (engl. *statement*)
  - Procena koštanja izvršavanja naredbe korišćenjem svakog od pristupnih puteva ili kombinacije
    - generiše skup mogućih planova izvršavanja
    - procenjuje koštanje svakog plana
      - koristi staru statistiku (`LAST_ANALYZED, BLOCKS` u `ALL_TABLES`)
    - bira plan sa najnižom cenom
      - *cost-based optimizer*
      - I/O, CPU, korišćenje mrežnih resursa
  - Konstruiše **matricu redosleda spajanja i metoda spajanja** i pridružuje cenu za svaki par
    - tabela prikazuje moguće varijante metoda i redosleda spajanja sa cenom
    - na osnovu toga odlučuje i bira opciju sa najmanjom cenom

29

29

Optimizator

- Mogući pristupni putevi (engl. *Access path*)
  - *Full table scans*
  - *Row ID scans*
  - *Index scans*

30

30

Access path - Full table scans

- Čita sve torke iz tabele i onda filtrira one koji zadovoljavaju uslov selekcije
- Full table scans se koristi kada imamo:
  - nedostatak indeksa
  - veliku količinu podataka
  - malu tabelu
- Ako optimizator misli da mu treba većina blokova onda će birati *full table scan* bez obzira na indeks

31

31

Access path - Full table scans

- Skenira sve blokove iz tabele koji su ispod *high-water mark* (HWM)
  - HWM označava količinu korišćenog prostora ili prostora koji je formatiran da prihvati podatke
- Blokovi se čitaju sekvencijalno
- Pošto su blokovi susedni može se nekada koristiti povećanje bloka u I/O pozivima
  - Inicijalizacijom parametra `DB_FILE_MULTIBLOCK_READ_COUNT`
  - Specificira maksimalni broj blokova u jednoj I/O operaciji
  - Platformski je zavisan

32

32

Full table scans

```
select last_analyzed, blocks
from all_tables
where table_name='RADNIK';
```

- Ako tabela nije analizirana od kada je kreirana i mala je optimizator će birati *full table scans*
- Ručno pokretanje saklupljanja statistike

```
ANALYZE TABLE Radnik COMPUTE STATISTICS;
```

33

33

Full table scans – primer

- *Full table scans*

```
SELECT plt FROM radnik
WHERE plt > 25000;
```

34

34

Full table scans – primer

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |        |     8 |   104 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| RADNIK |     8 |   104 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PLT">25000)
```

35

35

Access path

- *Row ID scans*
  - Row ID specificira file i blok u kojem se nalazi tražena torka kao i lokaciju torke u tom bloku
  - Korišćenje Row ID je najbrži način za dobijanje jedne torke
  - Svako skeniranje indeksa ne podrazumeva pristup tabeli pomoću Row ID
    - ukoliko su sve tražene kolone deo indeksa

36

36

Row ID scans – primer

```
SELECT * FROM radnik
WHERE mbr > 190;
```

37

Row ID scans – primer

```
Plan hash value: 243654634
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |     4 |   124 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| RADNIK    |     4 |   124 |     2   (0)| 00
|*  2 |   INDEX RANGE SCAN           | RADNIK_PK |     4 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("MBR">190)
```

38

Access path

- *Index scans*
  - *Index Range scan*
  - *Index Full scan*
  - *Index Fast-full scan*
  - *Index Join scan*
  - *Index Skip scan*

39

39

Index Range Scan

- *Index Range Scan*
  - Skeniranje opsega vrednosti u redosledu sortiranih vrednosti indeksa

```
SELECT plt FROM radnik
WHERE plt > 25000;
```

- Napomena:
  - prethodno je kreiran indeks
  - `create index idx_plt on radnik (plt);`

40

40

Index Range Scan

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |         |     8 |   104 |     1   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN | IDX_PLT |     8 |   104 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("PLT">25000)
```

41

41

Index Full scan

- *Index full scan* čita ceo indeks redom
  - zapravo čita samo listove stabla i prvu granu stabla potrebnu radi pronalaženja prvog lista
- Može da eliminiše operacije sortiranja (ORDER BY) zato što je indeks sortiran po *Key* koloni
  - ukoliko su sve kolone iz ove klauzule u indeksu i poredane su na isti način
- Koristi se u sledećim situacijama:
  - Predikat referencira kolonu u indeksu
    - Ne mora biti vodeća kolona.
  - Nije specificiran predikat ali postoje sledeći uslovi:
    - Sve kolone u tabeli koje su u upitu su i u indeksu
    - Najmanje jedna indeksirana kolona nije null

42

42

Index Full scan – primer

- Index Full scan

```
SELECT spr, nar
FROM projekat
ORDER BY spr;
```

43

Index Full scan – primer

```
Plan hash value: 2118842401

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |     9 |   198 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| PROJEKAT    |     9 |   198 |
|   2 |   INDEX FULL SCAN            | PROJEKAT_PK |     9 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

44

Index fast full scan

- *Index fast full scan*
  - Čita blokove indeksa u nesortiranom redosledu, kako su smešteni na disku.
  - Korsiti se kada se čita indeks umesto tabele, korisi indeks kao da je tabela
  - Optimizator razmatra ovo skeniranje kada se u upitu pristupa samo atributima koji su u indeksu
  - Ne može da eliminiše operacije sortiranja zato što ne čita indeks u sortiranom redosledu

45

45

Index Join Scans

- *Index join scan* spaja više indeksa zajedno i vraća sve kolone zahtevane upitom
  - Nije potrebno pristupanje tabelama zato što se svi podaci dobijaju iz indeksa

46

46

Index Skip scan

- *Index Skip scan* se dečava kada inicijalna kolona kompozitnog indeksa nije specificirana u upitu.
- Primer:

```
SELECT plt
FROM radnik WHERE PRZ='Peric';
```

- **Napomena**: Mora postojati kreiran indeks nad ime, prz

```
create index idx_imeprz on radnik (ime, prz);
```

47

47

Index Skip scan – primer

```
Plan hash value: 2538798176

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |     1 |    11 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| RADNIK  |     1 |    11 |     2   (0)| 00:0
|*  2 |   INDEX SKIP SCAN            | IND_CP  |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("PRZ"='Peric')
       filter("PRZ"='Peric')
```

48

48

Spajanje više tabela – Join

- Mogu se spojiti samo dva izvora torki u isto vreme. Spajanje više od dve tabele vrši se na sledeći način:
  - spajanjem dve tabele dobija se izvor torki
  - sledeća tabela se spaja sa izvorom torki koji je rezultat iz koraka jedan
  - korak 2 se ponavlja dok se ne spoje sve torke

49

49

Join terminologija

- Join naredba
- Join predikat i nonjoin predikat
- Single-row predikat

```
SELECT p.spr, p.nap,  r.ime, r.prz
FROM projekat p, radnik r
WHERE p.ruk=r.mbr
AND (r.mbr=100
OR p.spr=10);
```

Join predikat

Nonjoin predikat

Single-row predikat

50

50

Optimizator

- Redosled spajanja
  - Pravilo 1. Single-row predikat forsira da njegov izvor torki bude prvi u redosledu spajanja
  - Pravilo 2. Tabela sa outer join mora da bude poslednja u redosledu spajanja

- Primeri (pravilo 1):

```
select * from projekat, radproj
where projekat.spr = radproj.spr

select * from projekat, radproj
where projekat.spr = radproj.spr and projekat.nap = 'IIS'
```

51

51

Optimizator

- Vrste implementacije spojeva
  - *Nested loop join*
  - *Hash join*
  - *Sort-merge join*
  - *Cartesian join*

52

52

Join method

- *Nested loop join*
  - Jedna od dve tabele se proglašava spoljnom (outer table) ili vodećom tabelom (driving table)
  - Druga tabela je nazvana unutrašnjom (inner table)
  - Za svaku torku spoljašnje tabele vraćaju se sve odgovarajuće torke unutrašnje tabele
  - Pogodne kada se spajaju mali podskupovi podataka i kada postoji efikasan način za pristupanje drugoj tabeli
    - npr. indeks nad kolonom unutrašnje tabele, nad kojom se vrši spajanje
- hintovi optimizatoru
  - `/*+ ..... */`
  - `USE_NL(table)`
  - `USE_NL_WITH_INDEX(table index)`

53

53

Nested loop join

- Konceptualno nested loops je ekvivalentan sa dve ugnježdene iteracije

```
FOR erow IN (select * from employees where X=Y)
LOOP
    FOR drow IN (select * from departments where erow is matched)
    LOOP
            output values from erow and drow
    END LOOP
END LOOP
```

54

54

Join method

- *Hash join*
  - Koristi se za spajanje velikih skupova podataka
  - Optimizator koristi manji skup podataka od dva za pravljenje hash tabele na koloni za spajanje
    - u memoriji, koristeći deterministička hash funkciju za specificiranje lokacije u hash tabeli u kojoj je smeštena svaka torka
  - Zatim se skenira veći skup podataka da nađe torke za spajanje

- hintovi optimizatoru
  - `USE_HASH(table)`

55

55

Hash join

```
FOR small_table_row IN (SELECT * FROM small_table) LOOP
     slot_number := HASH(small_table_row.join_key);
     INSERT_HASH_TABLE(slot_number, small_table_row);
END LOOP;
```

56

56

Hash join

```
FOR large_table_row IN (SELECT * FROM large_table) LOOP
    slot_number := HASH(large_table_row.join_key);
    small_table_row =
          LOOKUP_HASH_TABLE(slot_number, large_table_row.join_key);
    IF small_table_row FOUND THEN
           output small_table_row + large_table_row;
    END IF;
END LOOP;
```

57

57

Join method

- *Sort-merge join*
  - Torke iz oba izvora torki su sortirane na kolonama join predikata
  - Dva sortirana izvora torki se spajaju (merge) u rezultujući izvor torki
- hintovi optimizatoru
  - `NO_INDEX and USE_MERGE`

58

58

Sort-merge join

- Optimizator može izabrati sort merge join umesto hash join za spajanje velike količine podataka kada postoji neki od uslova:
  - uslov spajanja između dve tabele nije equijoin, već se koristi uslovi kao što su <, <=, >, or >=
    - za razliku od sort merge, **hash join zahteva uslov jednakosti**
  - zbog zahteva za sortiranjem drugih operacija, optimizator nalazi da je jeftinije da koristi sort merge
  - ako postoji indeks tada se ne mora sortirati prvi skup podataka jer je već sortiran, dok se uvek sortira drugi skup podataka bez obzira na indeks

59

59

Naredba *explain plan*

60

Execution plan

- Plan izvršavanja je skup koraka koje obavlja optimizator u izvršavanju SQL naredbi
  - uključuje metodu pristupa svakoj tabeli kojoj se pristupa iz naredbe i redosled tabela (*join order*)
  - koristi različite metode za kombinovanje torki iz različitih tabela (*join method*)
  - koraci u planu izvršavanja se ne obavljaju u redosledu u kojem su numerisani
- Metod za gledanje plana izvršavanja
  - `EXPLAIN PLAN`

61

61

EXPLAIN PLAN

- Naredba `EXPLAIN PLAN`
  - Generiše plan izvršavanja optimizatora
  - Čuva plan u tabeli `PLAN_TABLE`
  - Ne izvršava naredbu

62

62

EXPLAIN PLAN

- Sintaksa

```
EXPLAIN PLAN
[SET STATEMENT_ID = ‘text’]
[INTO korisnička tabela za plan]
FOR sql naredba
```

63

63

EXPLAIN PLAN

```
[SET STATEMENT_ID = ‘text’]
```

- Zadaje se identifikator naredbe radi kasnijeg korišćenja
- Posebno je pogodno:
  - kada se deli tabela planova sa drugim korisnicima
  - kada se čuva više planova izvršavanja u istoj tabeli planova

```
[INTO korisnička tabela za plan]
```

- Korisnik može zadati sopstvenu tabelu planova
  - Default je `PLAN_TABLE`

64

64

Primer EXPLAIN PLAN

```
EXPLAIN PLAN
SET STATEMENT_ID = 'primer1'
FOR
SELECT p.spr, p.nap, r.ime, r.prz
FROM projekat p, radnik r
WHERE p.ruk = r.mbr;
```

- Rezultat izvršenja naredbe:
  `Explained.` ili `Plan SET succeeded.`

65

65

Primer EXPLAIN PLAN - output

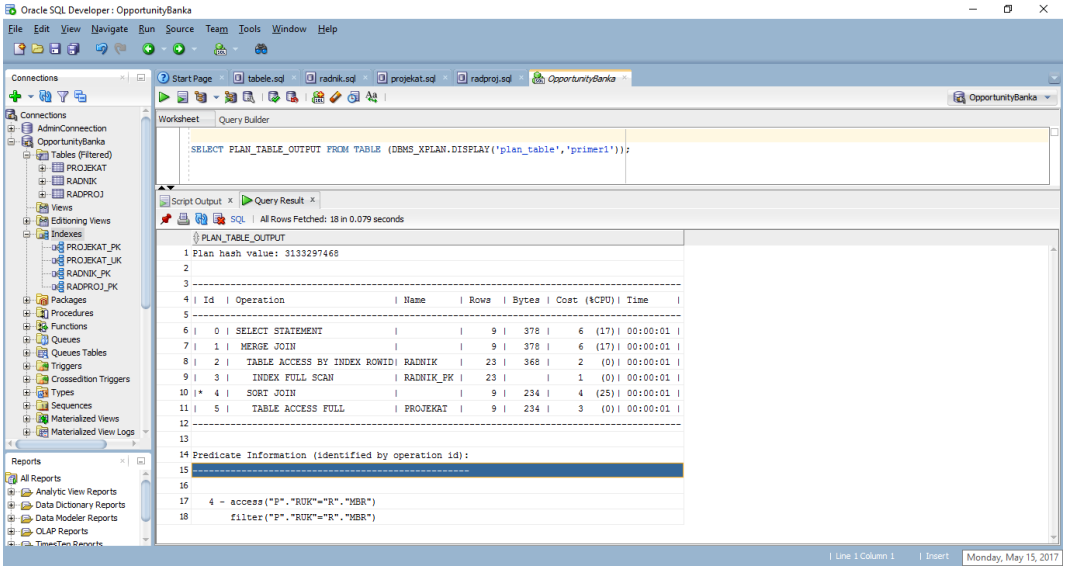
```
SELECT PLAN_TABLE_OUTPUT
FROM TABLE (DBMS_XPLAN.DISPLAY());

SELECT PLAN_TABLE_OUTPUT
FROM TABLE (DBMS_XPLAN.DISPLAY('plan_table','primer1'));
```

- `DBMS_XPLAN` paket
  - funkcija `DISPLAY` koristi se za formatiranje i prikazivanje poslednje naredbe smeštene u tabeli planova

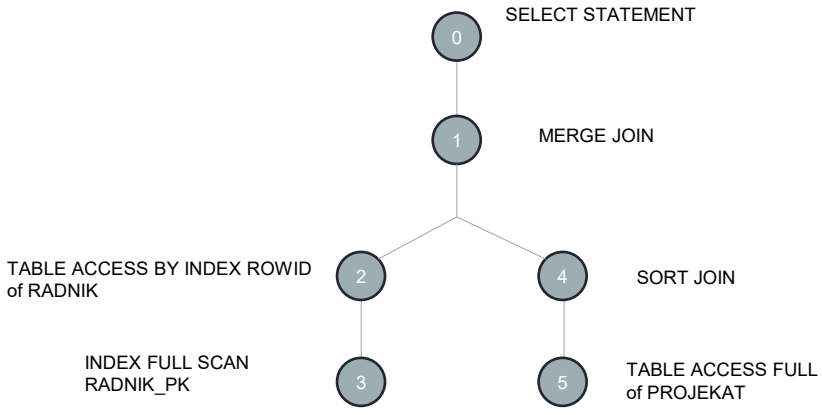
66

66

Primer EXPLAIN PLAN - output



Primer – stablo izvršavanja (Parse tree)



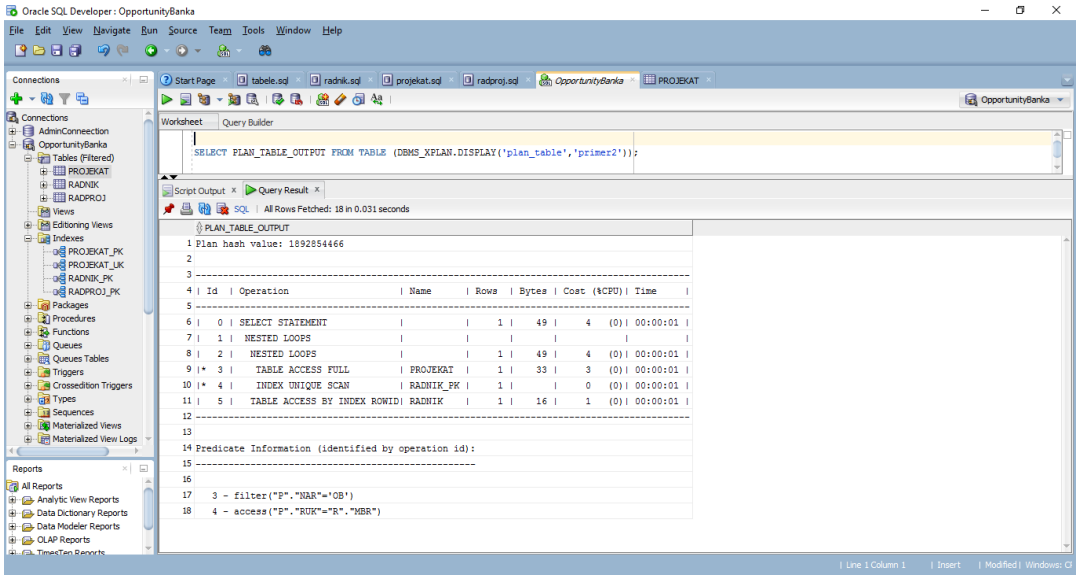
Primer 2 EXPLAIN PLAN

```
EXPLAIN PLAN
SET STATEMENT_ID = 'primer2'
FOR
SELECT p.spr, p.nap, r.ime, r.prz
FROM projekat p, radnik r
WHERE p.ruk = r.mbr AND p.nar = 'KBC';


SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY('plan_table','primer2'));
```

69

69

Primer 2 EXPLAIN PLAN



70

70

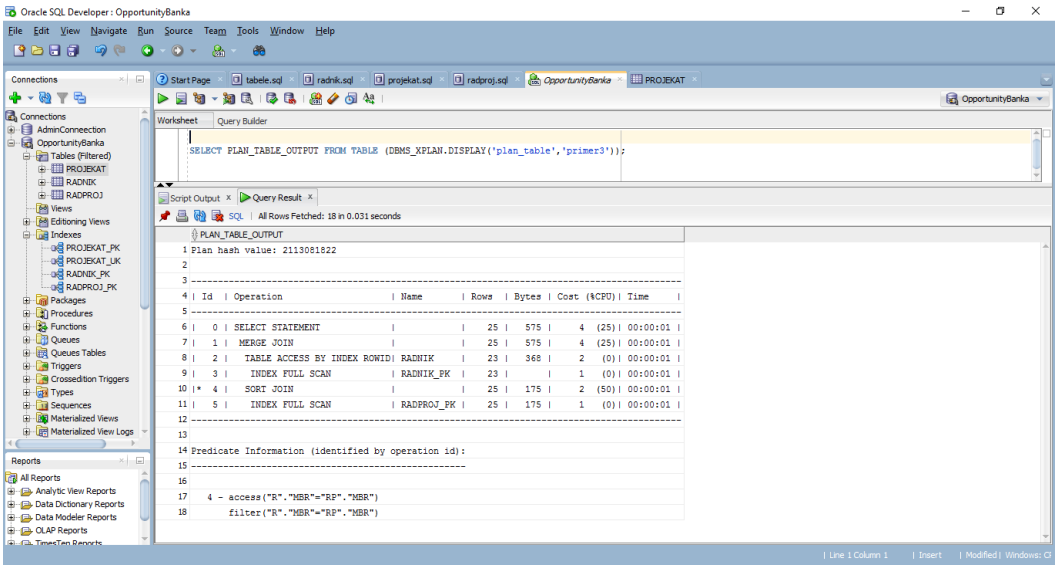
Primer 3 EXPLAIN PLAN

```
EXPLAIN PLAN
SET STATEMENT_ID = 'primer3'
FOR
SELECT r.ime, r.prz, rp.spr
FROM radnik r, radproj rp
WHERE r.mbr=rp.mbr;

SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY('plan_table','primer3'));
```

71

71

Primer 3 EXPLAIN PLAN



72

72

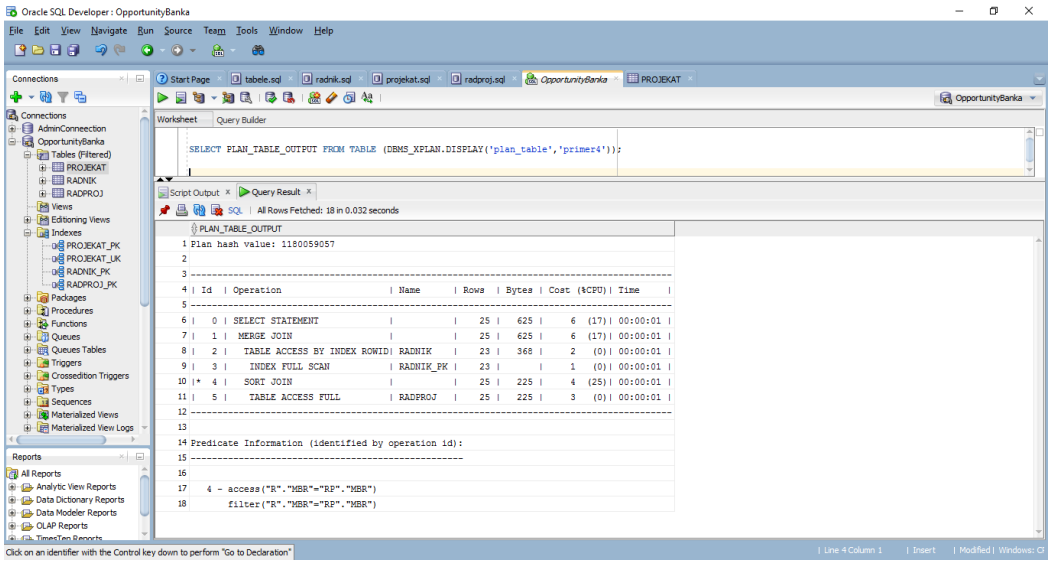
Primer 4 EXPLAIN PLAN

```
EXPLAIN PLAN
SET STATEMENT_ID = 'primer4'
FOR
SELECT r.ime, r.prz, rp.spr, rp.brc
FROM radnik r, radproj rp
WHERE r.mbr=rp.mbr;

SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY('plan_table','primer4'));
```

73

73

Primer 4 EXPLAIN PLAN



74

74

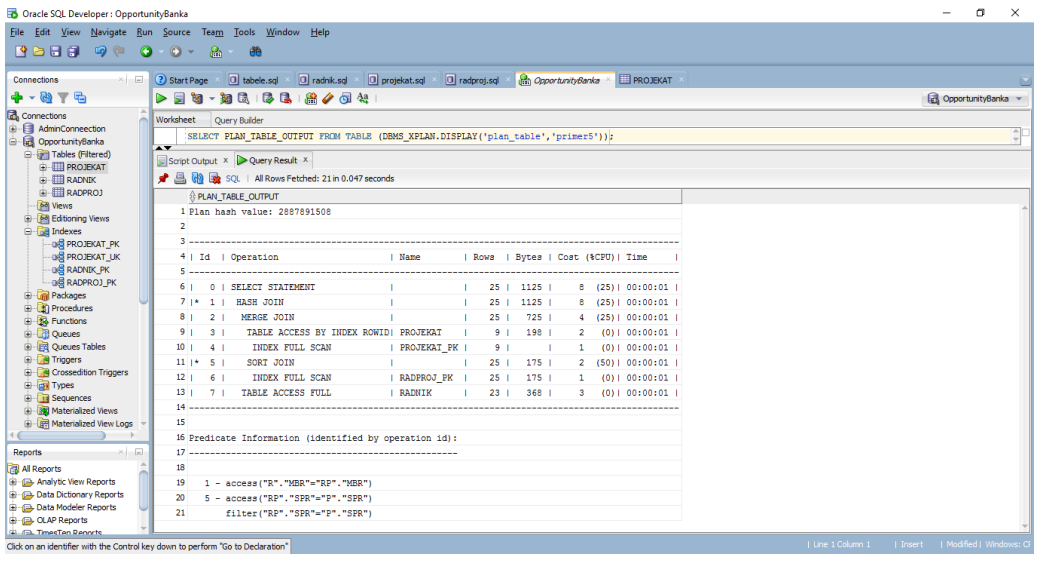
Primer 5 EXPLAIN PLAN

```
EXPLAIN PLAN
SET STATEMENT_ID = 'primer5'
FOR
SELECT r.ime, r.prz, p.spr, p.nap
FROM radnik r, radproj rp, projekat p
WHERE r.mbr=rp.mbr AND rp.spr=p.spr;

SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY('plan_table','primer5'));
```

75

75

Primer 5 EXPLAIN PLAN



76

76

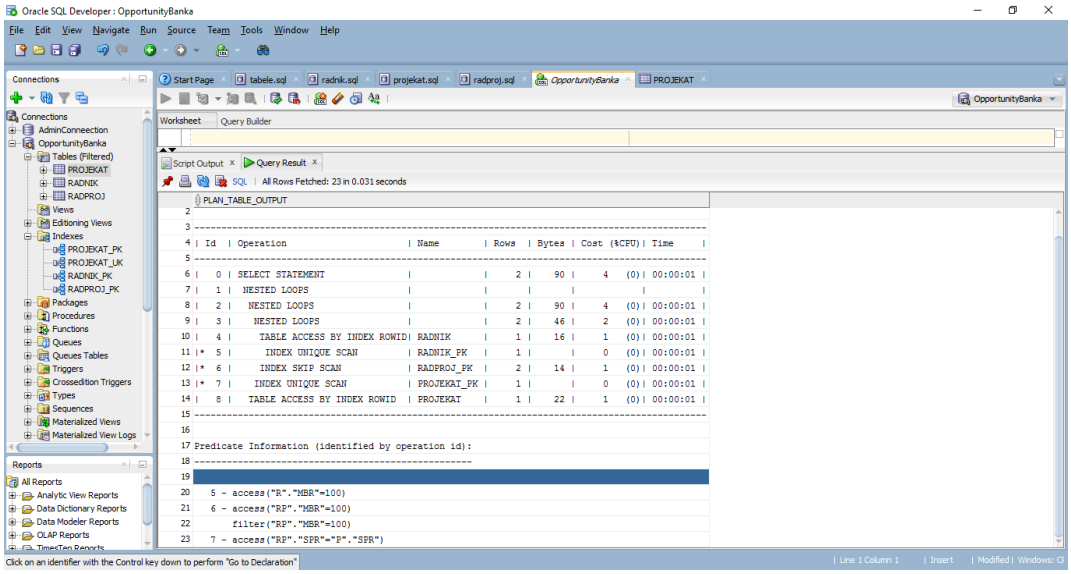
Primer 6 EXPLAIN PLAN

```
EXPLAIN PLAN
SET STATEMENT_ID = 'primer6'
FOR
SELECT r.ime, r.prz, p.spr, p.nap
FROM radnik r, radproj rp, projekat p
WHERE r.mbr=rp.mbr AND rp.spr=p.spr and r.mbr=100;

SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY('plan_table','primer6'));
```

77

77

Primer 6 EXPLAIN PLAN



78

78

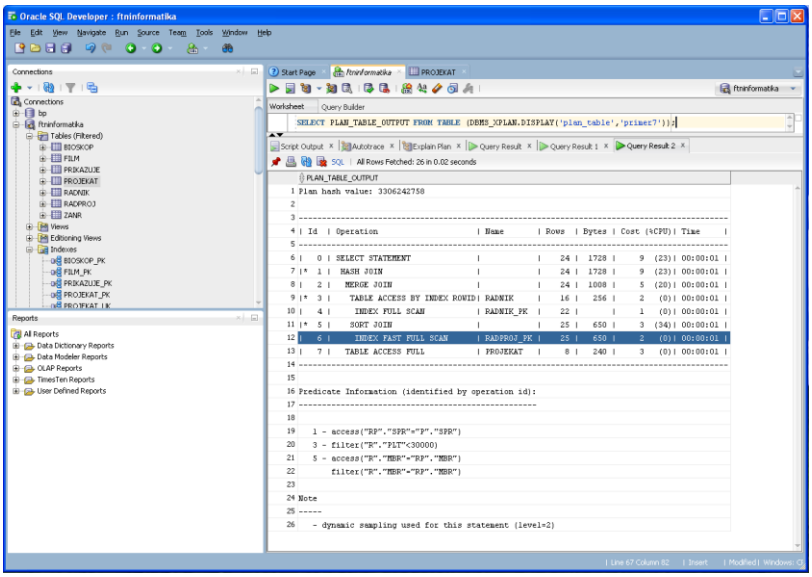
Primer 7 EXPLAIN PLAN

```
EXPLAIN PLAN
SET STATEMENT_ID = 'primer7'
FOR
SELECT r.ime, r.prz, p.spr, p.nap
FROM radnik r, radproj rp, projekat p
WHERE r.mbr=rp.mbr AND rp.spr=p.spr and r.plt<30000;

SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY('plan_table','primer7'));
```

79

79

Primer 7 EXPLAIN PLAN



80

80

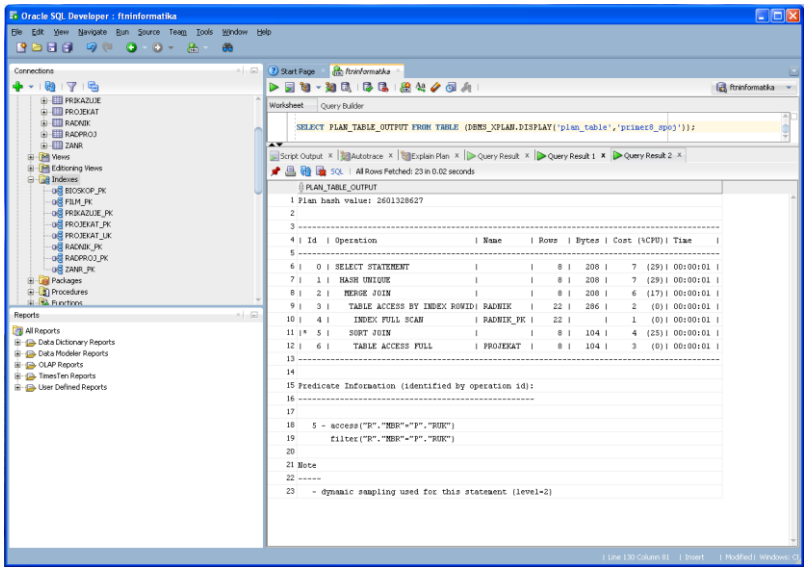
Primer 8 - razlike između ugneždenog upita i spoja

```
EXPLAIN PLAN
SET STATEMENT_ID = 'primer8_spoj'
FOR
SELECT distinct r.ime, r.prz
FROM radnik r, projekat p
WHERE r.mbr = p.ruk;

SELECT PLAN_TABLE_OUTPUT FROM TABLE
(DBMS_XPLAN.DISPLAY('plan_table','primer8_spoj'));
```

81

81

Primer 8 - razlike između ugneždenog upita i spoja



82

82

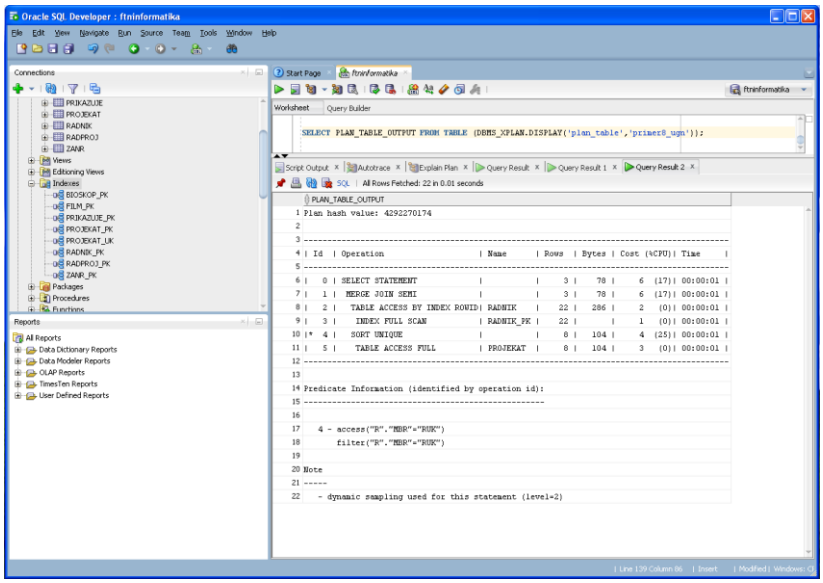
Primer 8 - razlike između ugneždenog upita i spoja

```
EXPLAIN PLAN
SET STATEMENT_ID = 'primer8_ugn'
FOR
SELECT r.ime, r.prz
FROM radnik r
WHERE r.mbr in (SELECT ruk FROM projekat);

SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY('plan_table','primer8_ugn'));
```

83

83

Primer 8 - razlike između ugneždenog upita i spoja



84

84

Primer 8 - Indeksi i strani ključ

- Kreirati indeks na ruk u tabeli projekat.

```
CREATE INDEX ind_fk ON projekat(ruk);
```

85

85

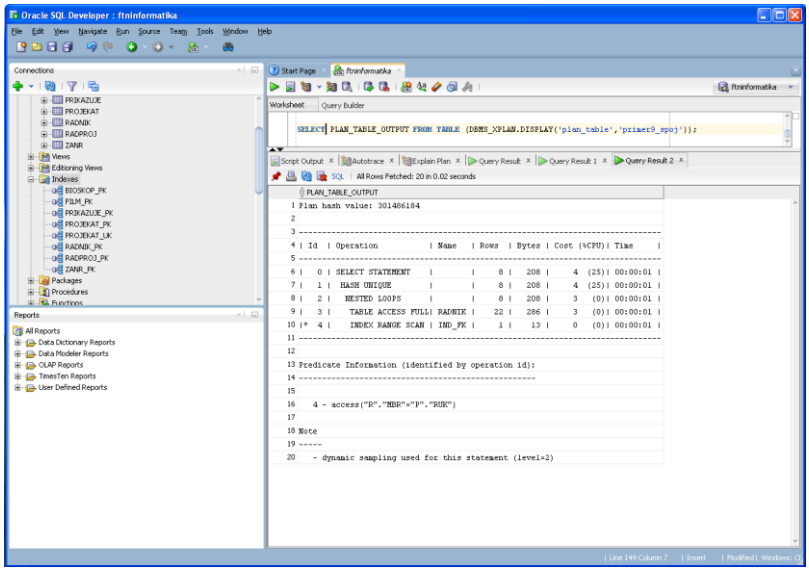
Indeksi i strani ključ

```
EXPLAIN PLAN
SET STATEMENT_ID = 'primer9_spoj'
FOR
SELECT distinct r.ime, r.prz
FROM radnik r, projekat p
WHERE r.mbr = p.ruk;

SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY('plan_table','primer9_spoj'));
```

86

86

Indeksi i strani ključ



87

87

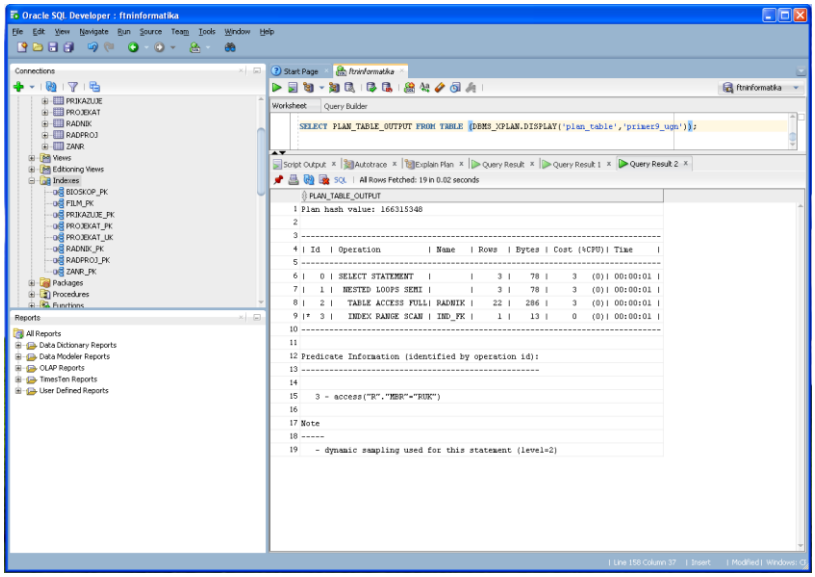
Indeksi i strani ključ

```
EXPLAIN PLAN
SET STATEMENT_ID = 'primer9_ugn'
FOR
SELECT r.ime, r.prz
FROM radnik r
WHERE r.mbr in (SELECT ruk FROM projekat);

SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY('plan_table','primer9_ugn'));
```

88

88

Indeksi i strani ključ



89

89

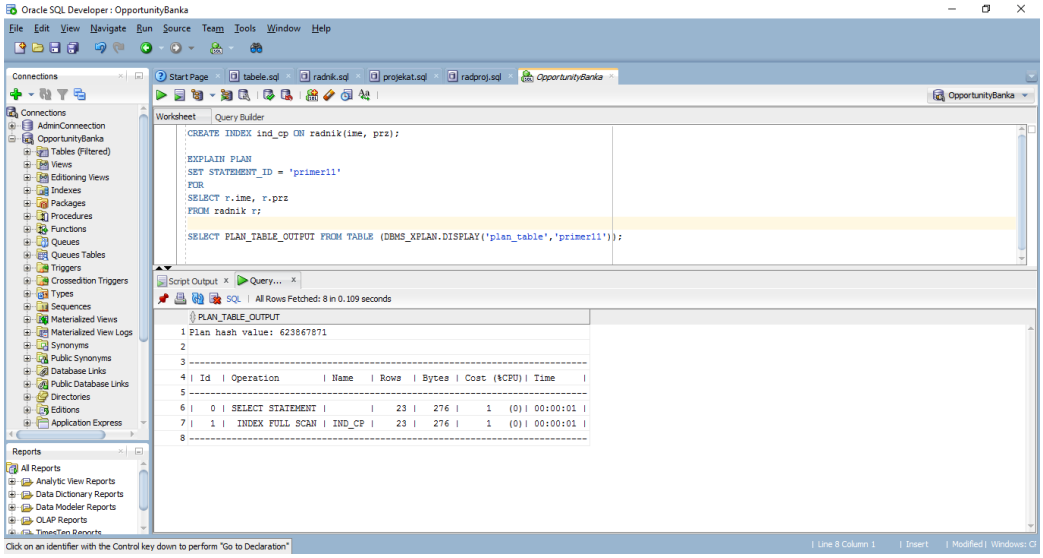
Primer kompozitnog indeksa

```
CREATE INDEX ind_imeprz ON radnik(ime,prz);

EXPLAIN PLAN
SET STATEMENT_ID = 'primer11'
FOR
SELECT r.ime, r.prz
FROM radnik r;

SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY('plan_table','primer11'));
```

90

90

Primer kompozitnog indeksa



91