Практичен проект по Напредни бази на податоци

База: Регистар на билки и производи од билки

Димитриј Мијоски, 111132

ФИНКИ, УКИМ, Скопје

[Mijoski.dimitrij@students.finki.ukim.mk](mailto:Mijoski.dimitrij@students.finki.ukim.mk)

**Резиме.** Целата на проектот е да се научи практично користење на релациона база преку креирање на модел и табели, креирање прашалници, програмирање процедури во самата база, полнење на базата со случајни податоци и оптимизација на прашалници. За овој проект обработуваме база со податоци од растенија и производи што се прават од тие растенија.

**Клучни зборови:** Oracle, логички модел, релациски модел, табели, прашалници, оптимизација, случајно генерирани податоци, PL/SQL, Explain plan

1. Вовед

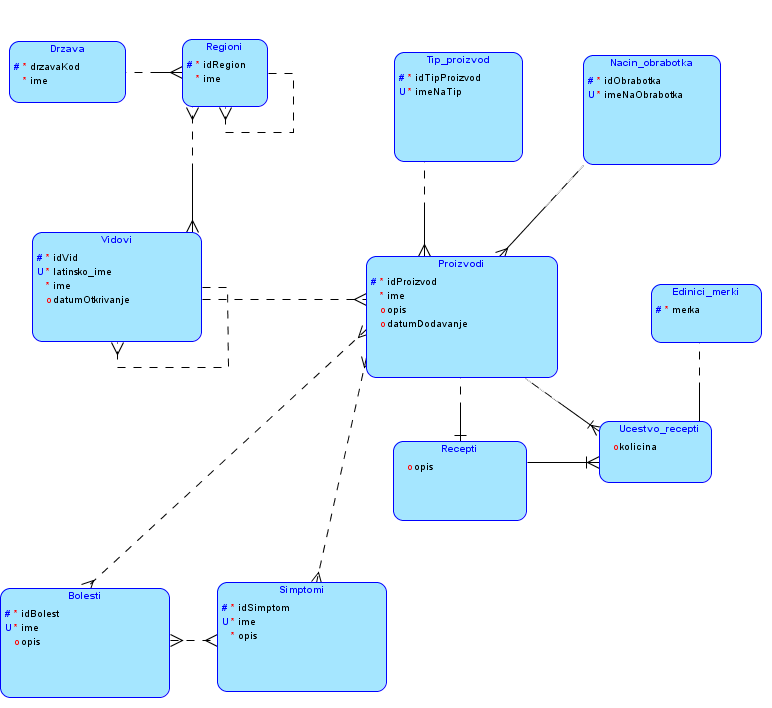
Овој проект е направен под барања поставени за предметот Напредни бази на податоци. Треба да се направат следниве чекори.

1. Да се дизајнира модел на база со ЕР-дијаграми Користена е алатката Oracle Data Modeler.
2. Да се претвори логичкиот модел во релациоски модел и јазик за опис на податоци, односно да се креираат табели. Искирстена е истата алатка.
3. Да се инсталира база Оракл.
4. Да се внеси моделот во базата и пробни податоци.
5. Да се напишат 3 прашалници.
6. Да се напишат 2 пакета, секој со најмалку 3 процедури, за внес, за ажурирање и за бришење, и со една функција што прави пресемтки според бизнис логиката.
7. Да се наполни базата со голема количина на случајно генерирани податоци. Во овој проект беше напишан пакет со процедури што ја праваат таа работа.
8. Напишаните прашалници да се тестираат врс многуте податоци, да се видат плановите на извршување и времето на извршување и евентуално да се оптимизираат тие прашалници со нивна измена или со додавање индекси.
9. Да се направи мала апликација со форма за преглед и уредување на некоја од табелите.

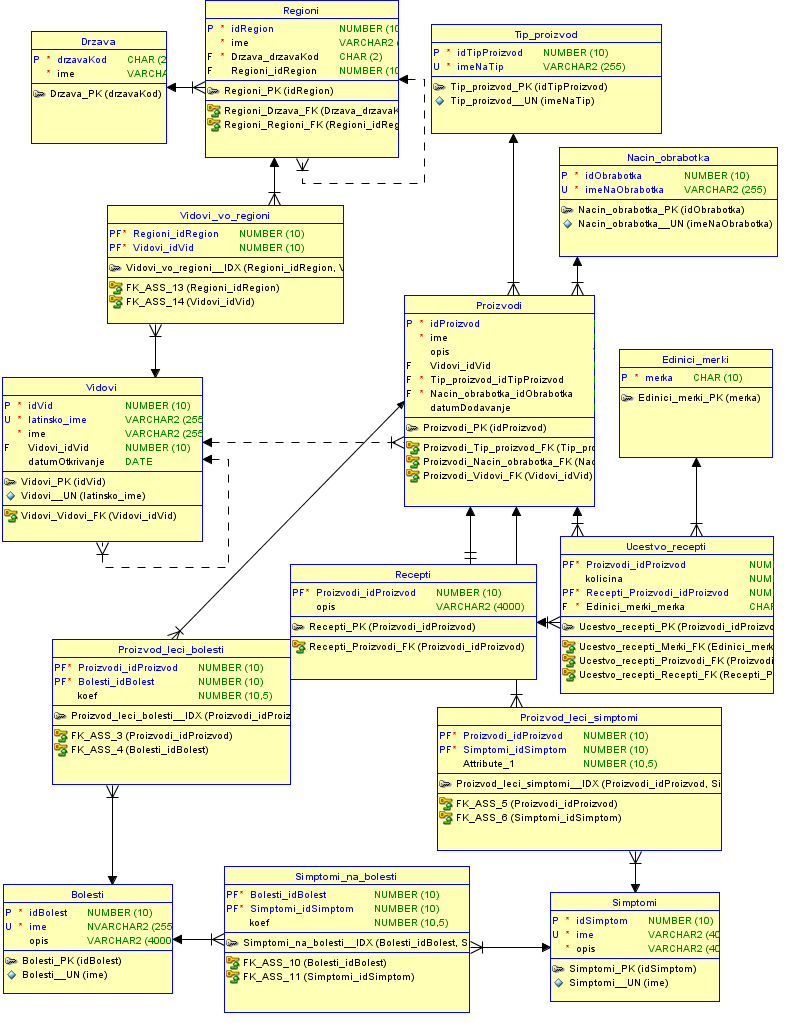
Користени алатки се база Oracle 11.2g инсталирана во виртуелна машина со оперативен систем Windows Server 2008, Oracle Data Modeler за моделирање на ЕР дијаграмот и релациските шеми, Oracle SQL Developer за пишувањето на прашалниците и процедурите и на крајот апликацијата беше напишана во јазик Јава во интегрираната околина Eclipse. За контрола на верзии на изворниот код беше користен Git.

1. База регистар на Билки

Базата што ја правиме е регистар на билки (растенија), производи што се прават од билките и од други производи (опишани со рецепти) и дадени се табели што поврзуваат одредени производи со разни болести кои ги лечат или помагаат при нивно третирање. Сега ќе ги прикажеле логичкиот и релациониот модел.



**Fig. 1.** Логички модел на база Билки.



**Fig. 2.** Релациски модел на база Билки

Понатамошната работа е концентрирана околу табелата Видови што е хиерархиска табела – дрво, и табелите Производи, Рецепти и Учество во рецепти што формираат граф каде табелата Производи ги содржи јазлите, а во табелата Учество во рецепти се наоѓаат ребрата.

1. Прашалници

#### 1. Прашалник пто ги бара сите видови на прво ниво според име или латинско име и потоа ги наоѓа сите нивни подвидови.

select \*

from vidovi v

start with V.VIDOVI\_IDVID is null AND

(lower(V.IME) like lower('%' || :search || '%') OR

lower(V.LATINSKO\_IME) like lower('%' || :search || '%'))

connect by prior V.IDVID = V.VIDOVI\_IDVID;

#### 2. Го користиме првиот прашалник за да пребараме видови и потоа ги наоѓаме сите производи што ги содржат тие видови директно и сите подпроизводи што ги содржат производите директно поврзани со видовите. Значи ги наоѓаме сите производи што некако ги содржат пребараните видови.

select unique P.IDPROIZVOD, P.IME

from PROIZVODI p left outer join UCESTVO\_RECEPTI ur on P.IDPROIZVOD = UR.RECEPTI\_PROIZVODI\_IDPROIZVOD

start with VIDOVI\_IDVID in (select V.IDVID

from vidovi v

start with V.VIDOVI\_IDVID is null AND

(lower(V.IME) like lower('%' || :search || '%') OR

lower(V.LATINSKO\_IME) like lower('%' || :search || '%'))

connect by prior V.IDVID = V.VIDOVI\_IDVID)

connect by PROIZVODI\_IDPROIZVOD = prior IDPROIZVOD;

#### 3. Наоѓаме најчесто користен вид

select V.IME, V.LATINSKO\_IME

from vidovi v

where V.IDVID in

(select P.VIDOVI\_IDVID

from PROIZVODI p left outer join UCESTVO\_RECEPTI ur on P.IDPROIZVOD = UR.PROIZVODI\_IDPROIZVOD

connect by prior P.IDPROIZVOD = UR.RECEPTI\_PROIZVODI\_IDPROIZVOD

group by P.VIDOVI\_IDVID

having count(\*) = (select max(COUNT(\*))

from PROIZVODI p left outer join UCESTVO\_RECEPTI ur on P.IDPROIZVOD = UR.PROIZVODI\_IDPROIZVOD

connect by prior P.IDPROIZVOD = UR.RECEPTI\_PROIZVODI\_IDPROIZVOD

group by P.VIDOVI\_IDVID));

Понатаму овие прашалници ќе ги оптимизираме.

1. Пакети со процедури и функции

За табелите Производи и Видови имаме два „тригери“ што се повикуваат пред внес на нов запис. Ако за внес немаме зададено вредност за клучот т.е. ако за клучот има НУЛ или број нула, земаме вредност од соодветните секвенци.

Потоа за двете табели ги имаме следниве пакети (во документов ќе ги прикажеме само декларациите на функциите, а не целиот код).

create or replace package pak\_vidovi as

Бара редица од табелата видови по латинското име и ја враќа целата редица.

function find\_vid\_po\_latinsko\_ime(platinsko\_ime varchar2) return vidovi%rowtype;

Бара редица од Видови по информативното име.

function find\_vid\_po\_ime(pime varchar2) return vidovi%rowtype;

Внесува нов вид. Клучот се генерира во тригерот и не се задава како параметар. Функцијата прави валидација со регуларен израз така што гледа латинското име да се состои само од букви и да има барем два збора. Гледа и датумот да не е поголем од денешниот. Ако датумот е НУЛ, го става денешниот.

procedure insert\_vid(platinsko\_ime IN varchar2, pime IN varchar2, pnadvid IN number, pdatum IN date);

Слична логика како погорната само што за надвид го задаваме латинското име. Неговиот клуч се наоѓа со погорната функција.

procedure insert\_vid(platinsko\_ime IN varchar2, pime IN varchar2, pnadvid\_latinsko\_ime IN varchar2, pdatum IN date);

Ажурираме вид. Ги ажурираме атрибутите за кои даваме вредности што не се НУЛ, додека за тие што има нул, вредностите остануваат непроменети.

procedure update\_vid(pid in number, platinsko\_ime IN varchar2, pime IN varchar2, pnadvid IN number, pdatum IN date);

Бришиме еден ред по ИД.

procedure delete\_vid(pid in number);

end pak\_vidovi;

Кај вториов пакет логиката за додавање, ажурирање и бришење записи е слична како погоре специфична е само функцијата „Најди број на производи“.

create or replace package pak\_proizvodi as

procedure insert\_proizvod(p\_ime varchar2, p\_opis varchar2, p\_idVid number, p\_idTip number, p\_idNacinObrabotka number, p\_datumDodavanje date);

procedure update\_proizvod(p\_id number, p\_ime varchar2, p\_opis varchar2, p\_idVid number, p\_idTip number, p\_idNacinObrabotka number, p\_datumDodavanje date);

procedure delete\_proizvod(p\_id number);

Оваа функција чесно може да се реализира само со прашалник, ама тука беше направена со PL/SQL за вежба. Таа ги селектира сите видови што директно содржат еден вид и потоа го изминува курсорот и брои елементи.

function najdiBrojNaProizvodiSodrzatVid(p\_vidId number) return number;

end pak\_proizvodi;

1. Полнење со случајни податоци

На овој дел беше посветено посебно внимание и доста време. Беа напишани неколку функции кои ги полнат дрвото претставено со табелата Видови и графот за производи претставен со неколку табели. Избран е метод за полнење со програмирање во PL/SQL за да се зачува некој логички редослед на внес на записи и некои статистички дистрибуции.

create or replace package slucajno\_polnenje\_tabeli as

Прво генерираме број по нормална распределба со очекување 0 и дисперзија 5, негативните предности ги правиме позитивни (ја превиткуваме распределбата околу вертикалната права во 0) и ја закружуваме кон долу за да добиеме цел број.

function vrati\_slucaen\_broj\_podvidovi return number;

Враќаме стринг со случајно генерирани букви со должина од 5 до 20 знака.

function vrati\_slucaen\_string return varchar2;

Враќаме речениза со случајно генерирани зоборв каде бројрот на зборови е од 5 до 100.

function vrati\_slucaen\_opis return varchar2;

Ја полниме табелата така што додаваме вис со случајно име и случајно латиснко име, датум кој не го надминува денешниот и за секој вид додаваме подвидови каде бројот на подвидовите е со изменета нормална распределба опишана погоре. Датумот на секој подвид е ограничен од долу да биде поголем од датумот на надвидот и помал од денешниот.

procedure napolni\_vidovi;

Ја полниме табелата тип производ со последователни ИД-ња и случајни имиња за типови.

procedure napolni\_tip\_proizvod;

Слично како претходната.

procedure napolni\_nacin\_obrabotka;

Со оваа функција ги полниме т.н. основни производи односно производи директно поврзани со некаков вид. За еден вид бројот на производи е според нормална распределба слична како погоре, само со дисперзија 2.

procedure napolni\_proizvodi;

Тука ги полниме прозиводите изградени од други производи, значи тука додаваме прозивод, негов рецепт и врски за тоа кои производи учествуваат во рецептот на еден прозивод (го полниме графот). Бројот на прозиводи во рецепт е спред нормална распределба со очекување 5 и дисперзија 5. Изборот на прозиводи во рецепти е според рамномерна распределба од досега вметнатите производи исклучувајќи ги последните стотина вметнати прозиводи. Значи процесот оди вака. Вметни 1000 производи. Потоа вметни нови 1000 и во нивните рецепти вклучувај ги првите 1000 (на крајот имаме 2000). Потоа вметнувај нови 1000 и во нивните рецепти вклучувај ги првите 2000. Значи симулираме сценарио каде поновите производи користат прозиводи кои „остареле“ малку и се познати.

procedure napolni\_proizvodi2;

end slucajno\_polnenje\_tabeli;

На крајот завршивме со следниве броеви на записи.

**Table 1.** Број на записи по табела.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| VIDOVI | PROIZVODI | RECEPTI | UCESVO\_RECEPTI | TIPOVI | NACINI |
| 233132 | 364737 | 82324 | 545461 | 100 | 100 |

1. Планови на извршување и оптимизации
   1. Прашалник 1

Ќе ги оптимизираме првите 2 прашалника.

За прашалник прој 1 од погоре го добиваме следниов план.



**Fig. 3.**

Прва работа кон оптимизација на прашалникот е да го отстраниме CONNECT BY зашто имаме хиерархија (дрво) со длабочина 1. Доволно е само едно спојување на табелата со самата себе (CONNECT BY прави онолку спојувања колку што е длабочината) Па го добиваме следниов прашалник кој е скоро еквивалентен ако немаме повеќе од две нивоа во податоците. Викаме скоро зашто редоследот може да биде различен.

-- 1.1 otstranet connect by

select \*

from vidovi v

where

V.VIDOVI\_IDVID is null AND

(lower(V.IME) like lower('%' || :search || '%') OR

lower(V.LATINSKO\_IME) like lower('%' || :search || '%'))

union

select \*

from vidovi v

where

--operatorot in e vsusnot spojuvanje

V.VIDOVI\_IDVID in (

select v.idvid

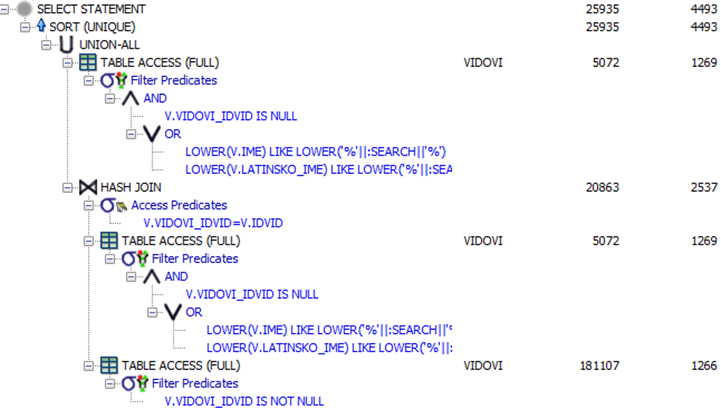
from vidovi v

where

V.VIDOVI\_IDVID is null AND

(lower(V.IME) like lower('%' || :search || '%') OR

lower(V.LATINSKO\_IME) like lower('%' || :search || '%')));



**Fig. 4.**

Сепак и со новиов прашалник не добиваме никаква предност. Ако пробаме да ставиме индекс на колоната со надворешен клуч (кон самата себе) vidovi\_idvid ќе добиеме целосно ист план. Па го исфрламе индексот.

create index vidovi\_nadvid on vidovi(vidovi\_idvid);

drop index vidovi\_nadvid;

Главната оптимизација треба да се направи всушност при пребарувањето со LIKE. За тоа не можеме да ставиме обичен индекс туку мора да користиме индекс за пребарување низ текст од пакет. <http://www.dba-oracle.com/oracle_tips_like_sql_index.htm> <http://docs.oracle.com/cd/B28359_01/text.111/b28303/ind.htm#g1020588>

Во Оракл освен стандардните индекси со Б+ дрво и битмап индекси може да се креираат т.н. доменски индески преку синтаксата Create index on … INDEXTYPE is ... Оракл дозволува врз табела програмер да креира разни дополнителни индексни структруи (пример КД-дрво, Квад-дрво, Р+ дрво за географски податоци). Треба само да се приспособи индексот кон интерфејс од Оракл за да може тие да комуницираат и што е приспособен со погорната синтака креираме индекси применливи во специфичен домен. Тука ќе го искорситиме CTXCAT индексот за пребарување низ текст.

--1.2 staven text index

create index vidovi\_ime on vidovi(ime) indextype is CTXSYS.CTXCAT;

create index vidovi\_lat\_ime on vidovi(latinsko\_ime) indextype is CTXSYS.CTXCAT;

select \*

from vidovi v

where

V.VIDOVI\_IDVID is null AND

(CATSEARCH(ime, :search, null) > 0 OR

CATSEARCH(latinsko\_ime, :search, null) > 0)

union

select \*

from vidovi v

where

V.VIDOVI\_IDVID in (

select v.idvid

from vidovi v

where

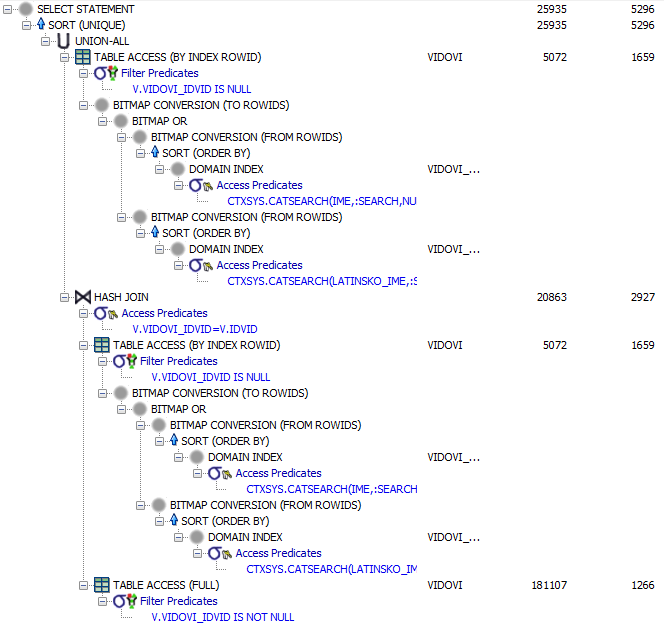
V.VIDOVI\_IDVID is null AND

(CATSEARCH(ime, :search, null) > 0 OR

CATSEARCH(latinsko\_ime, :search, null) > 0)

);

Добиваме забрзување од околу 3 пати односно од 1,5 секунда на половина секунда. Негативна страна на CATSEARCH е сто пребаруваме само по цели зборови или дел од почетокот на зборот ако ставиме ѕвездичка во стрингот десно, пример ‘felix\*’. Не пребарува по дел од крајот на зборот или од средината, па резултатите не се еквивалентни како со LIKE каде пребаруваме било каде во зборот. Исто така CATSEARCH има проблем ако пребаруваме само по една буква на почетокот на зборот пример ‘e\*’ или ‘x\*’, се закoчува. Тука подобро е со LIKE. Следува и планот.

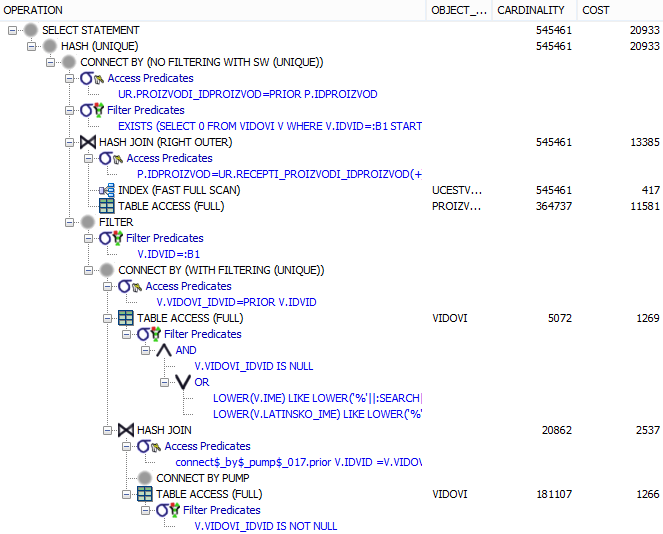


**Fig. 5.**

Забележиваме како доменскиот индекс се претвора во битмап-индекс и потоа зашто користиме индекс на две колони со услов или, таа операција елегантно се претвора во битово или на двата индекси.

* 1. Прашалник 2

Вториот прашалник прикажан погоре иако е точен, кога го пуштаме на голем граф трае долго и не завршува. Ако го видиме планот го добиваме ова.



**Fig. 6.** План на прашалник 2

Забележуваме дека иако во CONNECY BY имаме ставено услов за почетни јазли, изршувачот на прашалници тоа го преуредува и прави CONNECY BY без Start with (филтира некаде подоцна). CONNECT BY е всушност изминување со ДФС низ граф и без почеток значи дека ние ќе пуштиме ДФС почнувајќи од секој јазол што е премногу. Ако го препишеме прашалникот така што спојувањето од операторот ИН го направиме горе со LEFT OUTER JOIN го добиваме следниов прашалник.

select unique P.IDPROIZVOD, P.IME, P.VIDOVI\_IDVID

from PROIZVODI p left outer join UCESTVO\_RECEPTI ur on P.IDPROIZVOD = UR.RECEPTI\_PROIZVODI\_IDPROIZVOD

left outer join (

select V.IDVID

from vidovi v

start with V.VIDOVI\_IDVID is null AND

(lower(V.IME) like lower('%' || :search || '%') OR

lower(V.LATINSKO\_IME) like lower('%' || :search || '%'))

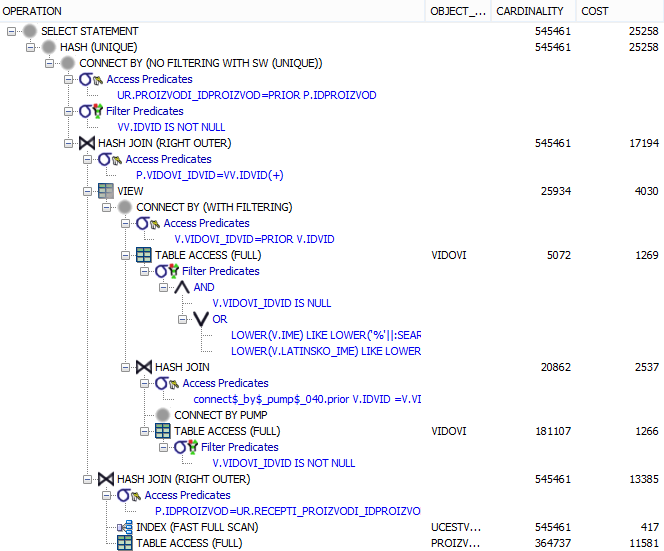
connect by prior V.IDVID = V.VIDOVI\_IDVID

) vv on P.VIDOVI\_IDVID = VV.IDVID

start with VV.IDVID is not null

connect by PROIZVODI\_IDPROIZVOD = prior IDPROIZVOD;

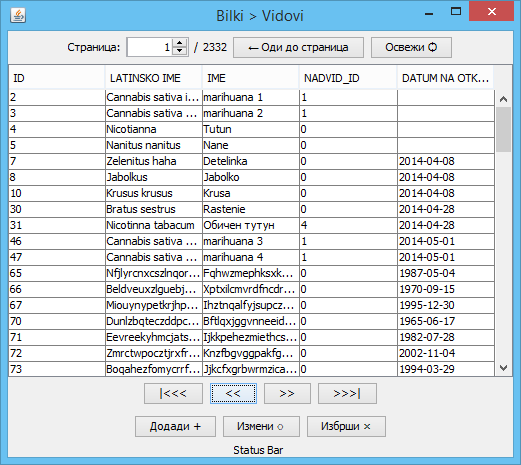
Прашалникот се извршува за околу 15-тина секунди. Може и дополнително да се оптимизира но ова засега е доволно. Ова е неговиот план.



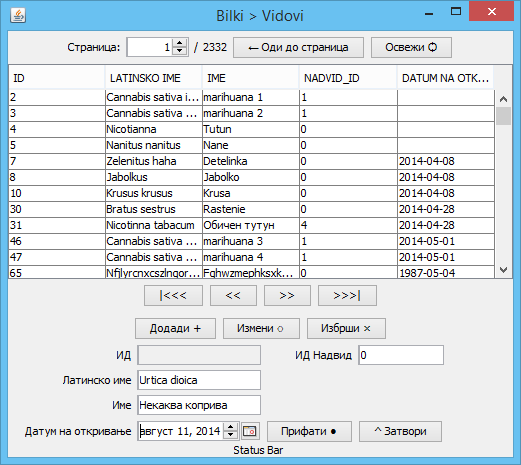
**Fig. 7.** Плн на оптимизиран прашалник 2.

1. Апликација

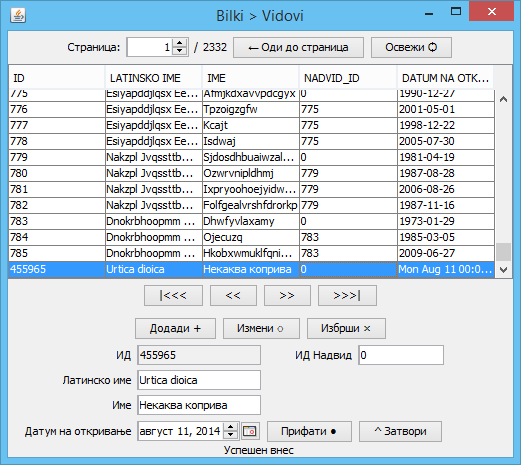
Апликацијата е изработена во Јава со рамката Свинг. За поврзување со базата се корсита стандардното АПИ ЈДБЦ кое пак внатрешно го користи Oacle thin driver. Направена класта Vid што претставува еден ентетиет вид и класта VidDao што го следи шаблонот „Data access object“ во кој се наоѓаат прашалниците со SQL поврзани со видови (читање, внес, ажурурање и бришење). Надвор од оваа класа нема SQL, односно таа е апстракција над релацискиот модел, премин од релациски во објектен модел. Потоа кај формите имаме две централни класи, JTable и TableModel. Идејата е да се следи шаблонот MVC (самата библиотека Свинг е градена со таа идеја) па тука TableModel е единствената класа што комуницира со VidDao, а пак формата комуницира со TableModel за било какви барања, пример вчитај следна страница, внеси запис, избриши запис. TableModel го известува погледот (JTable) за било какви промени, и објектот JTable ги прикажува на екран. Следуваат две слики од апликацијата.



**Fig. 8.**



**Fig. 9.**



**Fig. 10.**