Indeksiranje JSON podataka u Oracle bazi

Sitek, Antonio

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Bjelovar University of Applied Sciences / Veleučilište u Bjelovaru**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:144:469262

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-12-21**

Repository / Repozitorij:

Repository of Bjelovar University of Applied Sciences - Institutional Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

VELEUČILIŠTE U BJELOVARU

PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

INDEKSIRANJE JSON PODATAKA U ORACLE BAZI

Završni rad br. 07/RAČ/2021

Antonio Sitek

Bjelovar, listopad 2021.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Kandidat: **Sitek Antonio** Datum: 27.08.2021. Matični broj: 001990

JMBAG: 0149223360

Kolegij: **BAZE PODATAKA**

Naslov rada (tema): **Indeksiranje JSON podataka u Oracle bazi**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Programsko inženjerstvo**

Mentor: **Tomislav Adamović, mag.ing.el.** zvanje: **predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Ivan Sekovanić, mag.ing.inf.et comm.techn., predsjednik**
2. **Tomislav Adamović, mag.ing.el., mentor**
3. **Ante Javor, struč.spec.ing.comp., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 07/RAČ/2021

U radu je potrebno izraditi tablice s podacima koji sadrže kompleksnu JSON strukturu u Oracle bazi podataka. Definirati indekse nad JSON podacima koristeći ugrađene JSON funkcije. Analizirati plan izvršenja pretrage s indeksima i bez indeksa. Koristiti hintove za uključivanje pojedinih indeksa prilikom pretrage podataka u tablici s JSON podacima.

Zadatak uručen: 27.08.2021. Mentor: **Tomislav Adamović, mag.ing.el.**

SADRŽAJ

1. UVOD

2. KORIŠTENI SOFTVERI **1**
2.1. Baza podataka 1
2.2 Oracle SQL Developer 1
2.3. PL/SQL 2

3. JSON PODACI **4**
3.1. JSON podaci 4
3.2. Spremanje i popunjavanje JSON podataka u Oracle bazu 5
3.2.1 Izrada tablice za JSON podatke 5
3.2.2 Primjer JSON podataka korištenih za analizu 6
3.3. Rad s JSON podacima 8

4. INDEKSIRANJE PODATAKA **10**
4.1. Indeks 10
4.2 Vrste indeksa 11

5. ANALIZA BRZINE PRETRAŽIVANJA JSON PODATAKA U ORACLE BAZI **15**
5.1 Izrada tablice i popunjavanje podataka 15
5.2 Explain plan 16
5.3 Pretraga s JSON_TEXTCONTAINS 17
5.4 Analiza s explain planom 18
5.5 Analiza brzine pomoću paketa 22
5.5.1 Opis paketa GET_DATA 22
5.5.2 Rezultati testiranja s paketom GET_DATA 24
5.6.1 Izrada podređene tablice za testiranje vanjskog ključa 28
5.6.2 Opis funkcije FKEY_CHECK i triggera FK_CHECK 28
5.6.3 Analiza brzine pri umetanju podataka koristeći indeks i bez njega 30

6. ZAKLJUČAK **32**

7. LITERATURA

8. OZNAKE I KRATICE

9. SAŽETAK

10. ABSTRACT

1. UVOD

Motivacija za odabir teme završnog rada je rad s JSON (engl. *JavaScript Object Notation*) podacima u Oracle bazama podataka, zbog sve veće potrebe za spremanjem podataka u JSON formatu. S obzirom na to da postoji potreba spremanja podataka u JSON formatu, bez da se radi transformacija pri dohvaćanju i slanju podataka, koji mogu biti tekstualno veliki, testira se brzina dohvaćanja podataka uz pomoć indeksa i bez njega. JSON format sve više zamjenjuje XML (engl. *Extensible Markup Language*) zato što XML koristi oznake, dok JSON ne koristi, što ga čini lakšim za pisanje i razumljivim za čitanje čovjeku. Funkcije za rad s JSON podacima unutar Oracle baze omogućuju poboljšan rad s JSON podacima, što u ranijim verzijama nije bilo moguće. Prilikom pretraživanja velike količine JSON podataka može doći do usporenja sustava te postoji velika potreba za indeksiranjem takvih podataka. Ovim radom bit će prikazani principi rada s JSON podacima u Oracle okruženju, s alatom SQL Developer te korištenjem indeksa nad JSON podacima.

Rad je podijeljen na četiri poglavlja. Prvo poglavlje je uvodno i u njemu su spomenute osnovne informacije o Oracle SQL Developer alatu i SQL jeziku korištenog za rad s bazom podataka. Na početku drugog poglavlja opisano je što su JSON podaci i XML, kako se radi s JSON podacima u Oracle SQL Developer alatu i koja je razlika između JSON i XML formata. Treće poglavlje je posvećeno indeksiranju podataka u Oracle SQL Developer alatu. Opisuje se što su to indeksi i kako rade, za što ih se koristi te kako ih se može koristiti za rad s JSON podacima da bi pretrage bile brže. Četvrto poglavlje sadrži analizu koja pokazuje brzinu pretrage JSON podataka koristeći i ne koristeći indekse**.** Kako bi se usporedila brzina dohvaćanja podataka napravljena je skripta koja mjeri vrijeme od slanja upita do odgovora na upit, obavlja se više testova i uzima se prosječna vrijednost trajanja odgovora na upit. Testovi su izrađeni za ugrađene JSON funkcije unutra Oracle SQL Developer alata, a korištenje indeksa manipulira se uz pomoć naputaka (engl. *hint)*. Na kraju rada slijedi zaključak kojim se rezimira cjelokupni rad.

2. KORIŠTENI SOFTVERI

2.1. Baza podataka

Baza podataka [1] je skup podataka koji su pohranjeni i organizirani tako da mogu zadovoljiti zahtjeve korisnika. Također se može definirati kao skup međusobno povezanih podataka, pohranjenih zajedno, uz isključenje bespotrebne redundancije koja će biti objašnjena. Prilikom dodavanja novih podataka, ažuriranje i pretraživanja postojećih podataka primjenjuje se zajednički i kontrolirani pristup.

Baze podataka se koriste u svim vrstama poslovanja u kojima je nužna evidencija i obrada podatka kao što su trgovina, financije, društvene mreže, državne institucije, telekomi, gaming industrija, edukacijske ustanove i drugim primjenama.

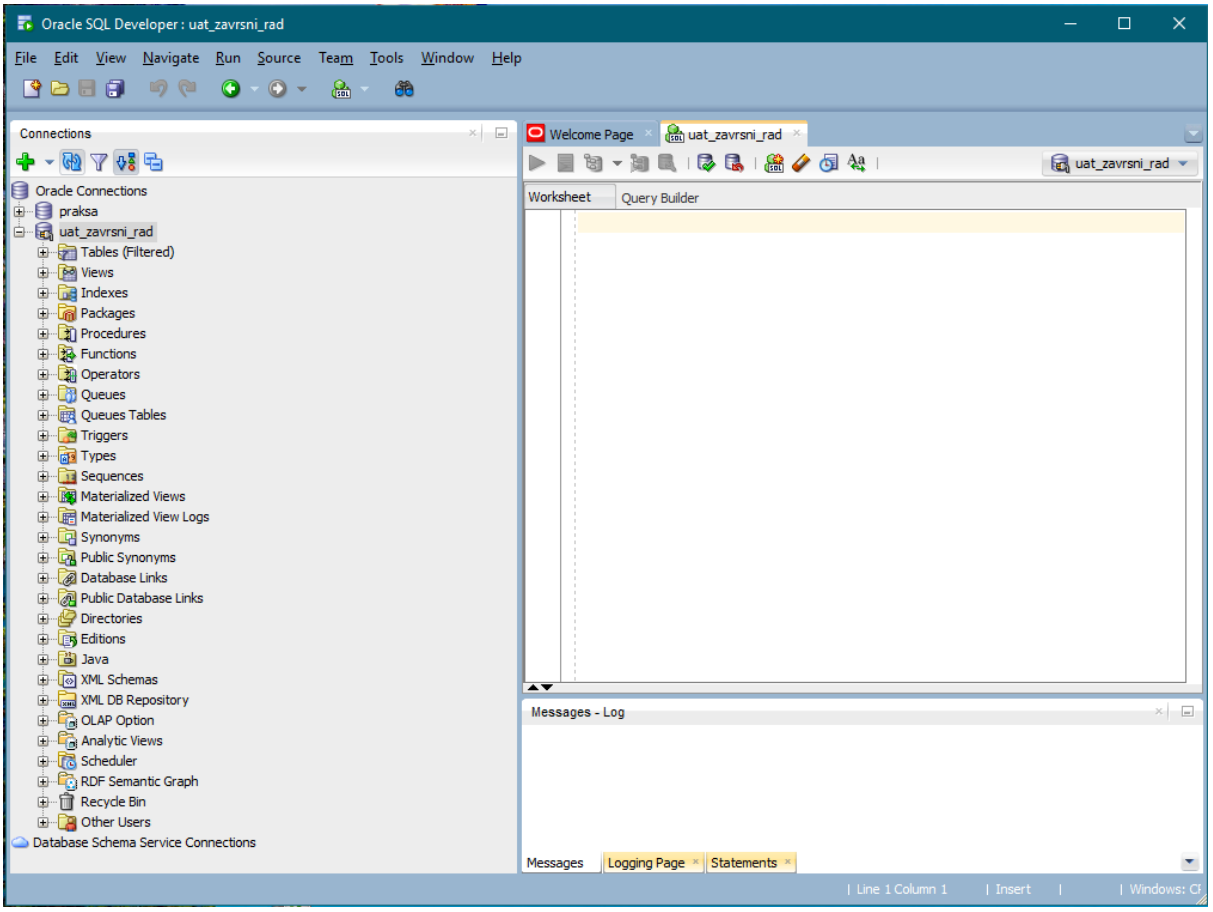
Neke prednosti korištenja baze podataka su manja redundancija podataka (nepotrebno ponavljanje), izbjegava se nedosljednost npr. ako se neki podatak treba izmijeniti, mijenja ga se samo na jednom mjestu. Integritet podataka je svojstvo koje garantira da su podaci pouzdani od trenutka nastanka. Sigurnost i povjerljivost podataka koje je lako omogućiti u centraliziranom sustavu za razliku od čuvanja podatka u običnim tekstualnim datotekama. Sam rad s bazom podataka omogućuje sustav za upravljanje bazom podataka (engl. *Database Management System* - DBMS). To je poslužitelj baze podataka koji oblikuje fizički prikaz baze s obzirom na logičku strukturu. Sustavi za upravljanje bazom podataka su:

- Oracle
- MySQL
- DB2
- MS SQL Server
- PostgreSQL

2.2 Oracle SQL Developer

Oracle SQL Developer [2] je besplatan alat koji omogućuje kreiranje, uređivanje, pregledavanje i brisanje podataka unutar baze podataka. Pomoću njega se izvršavaju SQL upiti i skripte te se izrađuju procedure i funkcije. Unutar alata omogućen je izvoz podataka, migracije baza podataka te pregled podataka i kreiranje izvješća. Kako bi se koristio taj alat mora se imati pristup bazi podataka. Na slici 2.1 vidljivo je korisničko sučelje unutar Oracle

SQL Developer alata. Posljednja verzija Oracle SQL Developer alata 21.2.1 može se skinuti sa stranice Oracle.com



Slika 2.1: Korisničko sučelje Oracle SQL Developer alata

2.3. PL/SQL

U ovom radu koristit će se PL/SQL jezik [3]. Takav jezik je napravljen da bi se mogla odvijati komunikacija između baze i servisa koji koristi bazu razumljiva čovjeku kako bi čovjek mogao zadavati upite, umetati podatke i brisati podatke. Postoji više vrsta jezika, ali danas se uglavnom koristi SQL koji je i standardiziran. Organizacije koje se bave standardima su ANSI (engl. *American National Standards Institute*) i ISO (engl. *International Organization for Standardization*).

SQL sadrži naredbe za potpuni rad s relacijskom bazom podataka kao što su kreiranje objekata, manipuliranje podacima i izvršavanje upita. *CREATE, ALTER i DROP* su naredbe koje služe za kreiranje objekata. Uz pomoć njih mogu se kreirati i ažurirati objekti. Objekti su tablice, pogledi i dr. *INSERT, UPDATE i DELETE* služe za manipulaciju podataka. Uz pomoć

tih naredbi podaci se unose, mijenjaju i brišu. Za dohvaćanje podataka unutar objekata koristi se naredbu *SELECT* koja se koristi kod upita.

3. JSON PODACI

3.1. JSON podaci

JSON (engl. *JavaScript Object Notation*) [4] je format teksta koji je dizajniran za čitljiv i razumljiv prijenos podatka ljudima i strojevima. Datoteka s podacima u JSON formatu ima ekstenziju *.json*, dok je meta oznaka (MIME format) *application/json*. Primjer JSON formata može se vidjeti u programskom kodu 3.1. Osnovni tipovi podataka JSON formata [4,5] su:

Logički tip podatka - istinite ili lažne vrijednosti (engl. *true*/*false*)

Polje - Sortirani niz vrijednosti koje mogu biti bilo kojeg tipa podataka uključujući i objekte. Polje može sadržavati i druga polja. Zatvoreni su unutar uglatih zagrada *[...]*, vrijednosti se odvajaju zarezom, a indeksiranje polja za dohvaćanje podataka može početi s nulom ili jedinicom.

Null vrijednost - *Null* vrijednosti predstavljaju praznu vrijednost.

Objekt - Nesortirani niz oblika ključ/vrijednost (engl. *key/value*) parova. Nalaze se unutar vitičastih zagrada *{...}*, nakon svakog ključa slijedi dvotočka *:*, a ključ/vrijednost parova su odvojeni zarezom ,. Ključevi također moraju biti nizovi znakova (engl. *string*) i moraju imati jedinstven naziv unutar polja.

Tekst - Niz od nula ili više Unicode znakova. Stringovi su odvojeni s dvostrukim navodnicima i koristi backslash u slučaju izlaznog znaka (engl. *escape character*).

Broj - JSON podržava cijele i decimalne brojeve.

Programski kod. 3.1: Primjer JSON formata

```
[
{    "ime":"Antonio",
      "prezime":"Sitek",
      "email":"asitek@vub.hr",
      "redovni":true},
{    "img":"uploads/vub-logo.png"}
]
```

JSON format sve više zamjenjuje XML format. XML koristi oznake dok JSON ne koristi što ga čini lakšim za pisanje i razumljivim za čitanje čovjeku. Primjer XML formata može se vidjeti u programskom kodu 3.2. Glavna prednost JSON formata je zbog toga što se

JSON predvodi kroz standardnu *JavaScript* funkciju, dok XML kroz XML prevoditelj. XML je kratica za Extensible Markup Language i tehnologija je predviđena za kodiranje podataka u čitljivom obliku. Takav format razvijen je za korištenje na internetskim stranicama. JSON za razliku od XML nema početne i završne oznake i sadrži manje redundancije nego unutar XML formata. XML format zauzima više znakova za prijenos podatka. Oba su dobro dokumentirani tekstualni formati čitljivi ljudima. Oba su jednostavni za učenje, a jezici su neovisni. Svaki format odgovara bolje različitim zadacima. XML je jezik za dodavanje dodatnih informacija običnom tekstu dok je JSON učinkovit način predstavljanja strukturiranih podataka u čitljivom obliku.

Programski kod. 3.2: Primjer XML zapisa

```
<?xml version="1.0"?>
<OSOBA>
  <IME>Antonio</IME>
  <SPOL>Musko</SPOL>
  <GODINE>25</GODINE>
</OSOBA>
```

3.2. Spremanje i popunjavanje JSON podataka u Oracle bazu

3.2.1 Izrada tablice za JSON podatke

JSON podaci se nalaze u tekstualnom formatu te ih se može vrlo jednostavno spremiti u bazu unutar polja tablice. Pošto je JSON tekst, prvi izbor je spremati ga kao *VARCHAR2* za manje dokumente, ali ako se koriste veći JSON podaci može se koristiti *CLOB* ili *BLOB*. Ako se podaci spremaju u *BLOB* zauzimaju manje mjesta i ne budu čitljivi bez konverzije podataka. Ako se koristi *CLOB* podaci su čitljivi odmah i nije potrebno raditi konverziju podataka te zauzimaju više mjesta. Za spremanje JSON podataka [6] potrebno je izraditi tablice u koje se on sprema. Za potrebe završnog rada koristi se programski kod 3.3. za izradu primarne tablice (engl. *parent table*).

Programski kod. 3.3: Izrada tablice osoba_json

```
create table osoba_json (
   osoba_id number(6) not null,
   osoba_podatak clob not null,
   constraint OSOBA_JSON_PK PRIMARY KEY (OSOBA_ID)
);
CREATE sequence OSOBA_JSON_ID_SEQ;
CREATE trigger BI_OSOBA_JSON_ID
  before insert on OSOBA_JSON
  for each row
begin
  select OSOBA_JSON_ID_SEQ.nextval into :NEW.OSOBA_ID from dual;
end;
```

Oracle baza omogućava da se za vrijeme spremanja podataka može provjeriti je li su podaci spremljeni unutar polja *osoba_podatak* JSON formata. Da bi se to omogućilo potrebno je dodati *IS JSON* ograničenje (engl. *constraint*). U programskom kodu 3.4. može se vidjeti primjer dodavanja ograničenja.

Programski kod. 3.4: Dodavanje ograničenja

```
alter table osoba_json
add constraint osoba_podatak_json
check (osoba_podatak is json);
```

Nakon kreiranja tablica, one su prazne te ih se može popunjavati s JSON podacima. Ako podaci nisu JSON tipa baza će odbiti umetanje (engl. *insert*) i vratiti poruku gdje ograničenje nije zadovoljeno.

3.2.2 Primjer JSON podataka korištenih za analizu

JSON podaci koji su uneseni unutar baze izgledaju kao u programskom kodu 3.5. Unutar JSON podatka nalaze se podaci tipa tekst (engl. *string*), broj (engl. *number*), istina/neistina (engl. *true/false*), JSON polja i JSON objekti.

Programski kod. 3.5: Primjer JSON podataka za tablicu OSOBA_JSON

```
{
   "_id":"611617389ef0832d88a72e91",
   "index":5,
   "guid":"625e7732-8f92-4367-a43a-d942a9c70686",
   "isActive":true,
   "balance":"$3,024.23",
   "picture":"http://placehold.it/32x32",
   "age":33,
   "eyeColor":"blue",
   "name":"Aurelia Kidd",
   "gender":"female",
   "company":"RUBADUB",
   "email":"aureliakidd@rubadub.com",
   "phone":"+1 (864) 565-2429",
   "address":"962 Sapphire Street, Kanauga, California, 5744",
   "about":"Laboris ea sint in magna ut consequat minim inci...",
   "registered":"2017-04-05T07:10:21 -02:00",
   "latitude":21.273277,
   "longitude":125.930193,
   "tags":[
      "aliquip",
      "consectetur"
   ],
   "friends":[
      {
         "id":0,
         "name":"Porter Bishop"
      },
      {
         "id":1,
         "name":"Elisa Schroeder"
      }
   ],
   "greeting":"Hello, Aurelia Kidd! You have 3 unread messages.",
   "favoriteFruit":"apple"
}
```

3.3. Rad s JSON podacima

Oracle za rad s JSON podacima koristi točku za dohvaćanje vrijednosti po ključu. Rad s točkom kao oznakom je jednostavan za koristiti. Vraćena vrijednost je uvijek string (*VARCHAR2*) a vraćeni string ovisi o podacima koji se dohvaćaju. Ako je podatak koji se dohvaća singularna vrijednost bit će string bez obzira je li je podatak objekt ili polje. Ako postoji više vrijednosti onda će vraćeni tekst (engl. *string*) biti JSON polje.

Za kompleksnije pretrage se koriste SQL/JSON ugrađene funkcije ***json_value*** i ***json_query*** koje vraćaju *NULL* vrijednost ili pogrešku (engl. *error*) ovisno o tome kako je funkcija parametrizirana.

Json_value [7] je ugrađena funkcija koja vraća skalarnu vrijednost iz JSON podatka i vraća ju kao SQL vrijednost. ***Json_value*** se može koristiti s indeksima za rad s JSON podacima. Prvi parametar unutra ***json_value*** je naziv JSON dokumenta iz kojeg se dohvaćaju podaci. Drugi parametar prima putanju do vrijednosti unutar JSON podatka/dokumenta. Funkcija ***json_value*** će vratiti *null* ako se dogodi pogreška (engl. *error*), ako se želi znati da se dogodila pogreška koristi se parametar *ON ERROR* te parametrizira rezultat koji se želi vratiti. Primjer upita s funkcijom ***json_value*** i rezultat može se vidjeti u programskom kodu 3.6.

Programski kod. 3.6: Primjer dohvaćanja vrijednosti s funkcijom json_value

```
SELECT
   json_value('{a:100}', '$.a')
FROM
   dual;

REZULTAT:
100
```

Funkcija ***json_query*** [8] odabire više vrijednost po ključu (engl. *key*) iz JSON podatka i vraća tekst (engl. *string*) tip *VARCHAR2*. Za razliku od ***json_value*****,** podatak koji vraća ne može biti broj. Funkcija se koristi za dohvaćanje JSON podataka koji nisu objekti. Mogu biti *VARCHAR2*, *BLOB*, *CLOB*. Kada se dogodi pogreška ***json_query*** vraća prazno polje *[ ]* osim ako se ne definira drukčiji rezultat unutar *ON ERROR* parametra koji nije obavezno definirati.

Prvi parametar u funkciji je dokument nad kojim se traži podatak, drugi je putanja gdje se može definirati vraća li se podatak s *WRAPPER* parametrom ili ne. Ako se koristi parametar *WRAPPER* vraćeni podatak će se nalaziti unutar *[]*. U programskom kodu 3.7. može se vidjeti primjer upita s funkcijom ***json_query*** i rezultat istoga.

Programski kod. 3.7: Primjer dohvaćanja vrijednosti s funkcijom json_query

```
SELECT
   json_query('{a:100, b:200, c:300}','$.a' WITH WRAPPER)
FROM
   dual;

REZULTAT:
[100]
```

Funkcija ***json_exist*** [9] može se koristi unutar *WHERE* uvjeta kao filter podataka, vraća redove u bazi koji unutar JSON dokumenta zadovoljavaju uvjete koji su zadani. Funkcija vraća istinu (engl. *true)* ili neistinu (engl. *false*) *s* obzirom na to da li traženi podatak postoji unutar JSON dokumenta. Ako se želi imati uređena pogreške potrebno je koristiti *ON ERROR* parametar. Drugi argument unutar funkcije sadrži uvjet koji treba biti zadovoljen da bi funkcija vratila istinu (engl. *true*). Funkcija ***json_exist*** može raditi s podacima kao što su : *VARCHAR2*, *NUMBER*, *BINARY_DOUBLE*, *DATE*, *TIMESTAMP* i *TIMESTAMP WITH TIMEZONE*. Primjer upita s funkcijom ***json_exist*** može se vidjeti u programskom kodu 3.8.

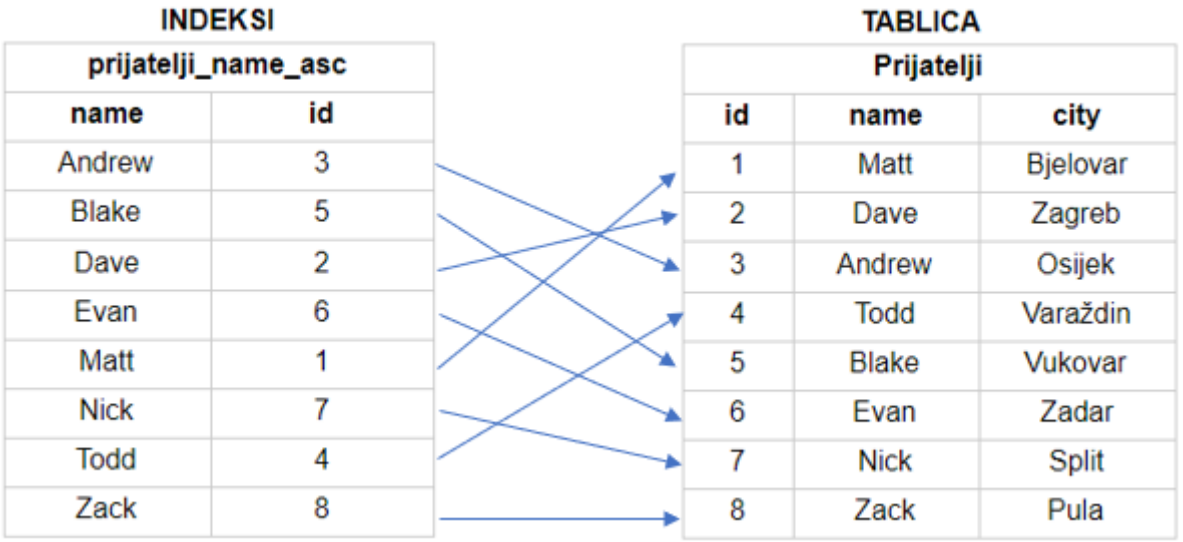
Programski kod. 3.8: Primjer dohvaćanja vrijednosti s funkcijom json_exist

```
SELECT
   t.json_dokument
FROM
   tablica.t
WHERE
   json_exists(t.json_dokument, '$.Putanja.put == neki_broj)';
```

4. INDEKSIRANJE PODATAKA

4.1. Indeks

Uz pomoć indeksa Oracle baza može brže pronalaziti i sortirati zapise. Indeks [10] pohranjuje lokaciju zapisa na temelju polja koja su odabrana za indeksiranje. Kada Oracle baza dohvati lokaciju iz indeksa tada ona može dohvatiti podatke izravnim premještanjem na točnu lokaciju. Zbog toga korištenja indeksa može biti znatno brže od pregledavanja svih zapisa radi pronalaženja podatka. Ako se ne koriste indeksi, baza mora pretražiti cijelu tablicu kako bi vratila podatke.



Slika 4.1: Primjer indeksa [10]

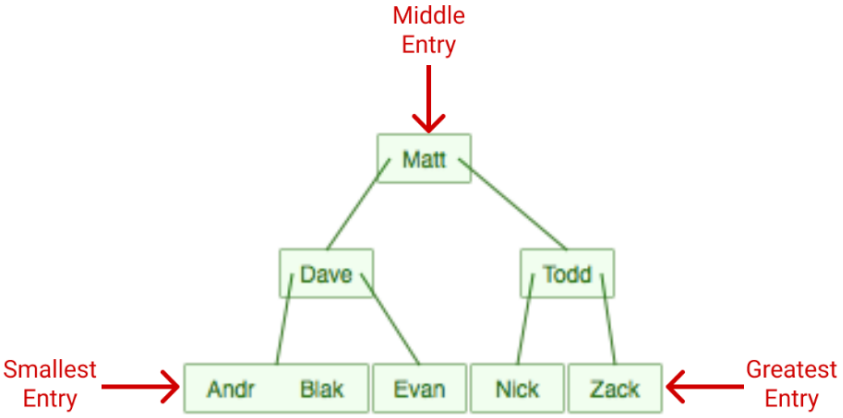
Na slici 4.1. može se vidjeti kako indeks ima pohranjene podatke po abecednom redu dok u tablici nisu tako pohranjeni. To omogućuje znatno bržu pretragu podatka, kada je traženi podatak *Zack* mogu se preskočiti sve riječi koje ne počinju na *Z* što uzrokuje brzo vraćanje podatka na traženi upit.

Ključ u indeksu sadrži jedno ili više polja i/ili operatora. Iako se indeks i ključ koriste zajedno oni su različiti. Indeksi su strukture spremljene u bazi s kojima se upravlja sa SQL izrazima dok je ključ logički dio.

Pri izvršavanju upita prema bazi bez korištenja indeksa baza mora pretražiti red po red da bi pronašla odgovor na upit. Ako se podatak nalazi na kraju tablice ovisno o veličini tablice traženje može trajati duži vremenski period.

4.2 Vrste indeksa

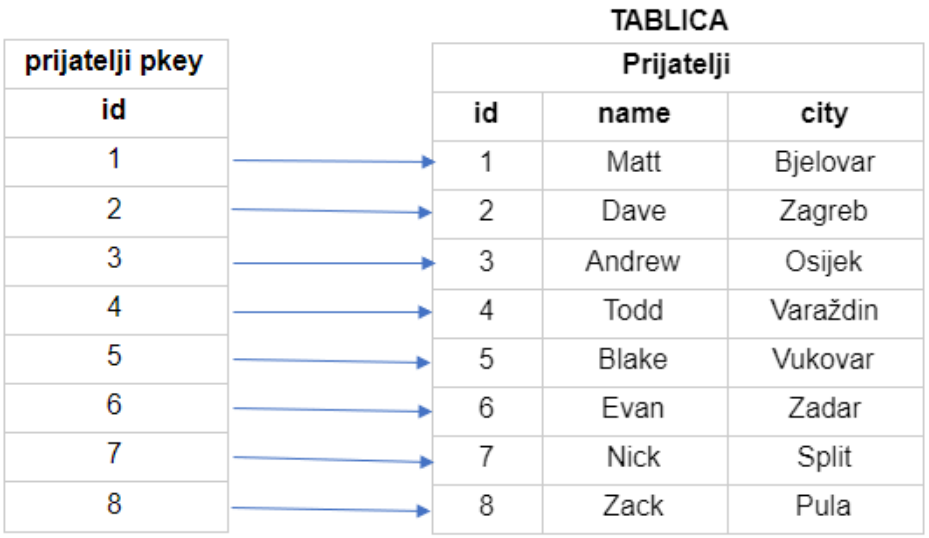
Vrste indeksa koje postoje su grupirani i negrupirani indeksi. Grupirani i negrupirani indeksi koriste B-stablo za pretragu podataka, B-stabla su strukture podataka. B-stablo je slično binarnom stablu. Karakteristike su mu potpuna balansiranost, sortiranje podataka po vrijednosti ključa i čuvanje određenog broja elemenata u jednom čvoru stabla.



Slika 4.2: Primjer B-stabla [10]

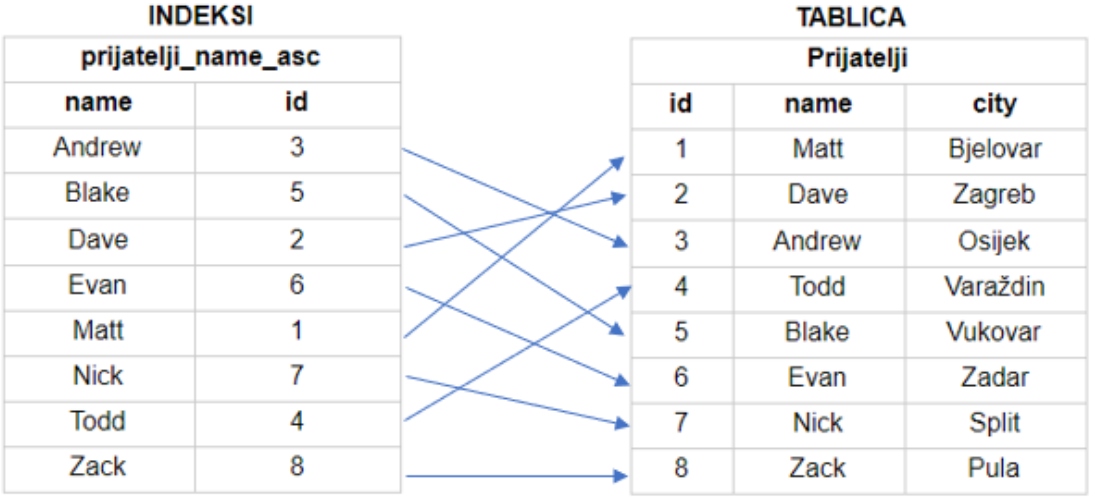
Na slici 4.2. može se vidjeti primjer B-stabla s indeksima. Najmanji podatak je lijevo, a najveći desno. Svako pretraživanje kreće od gore te se grana prema traženom podatku. Npr. ako je traženi podatak *Zack* prvo se uspoređuje s *Matt* zatim se odlazi usporediti s *Todd* nakon toga dolazi se do traženog podatka *Zack*. Kako bi se ubrzala pretraga, broj slova unutar podatka se obično limitira na manji broj slova koji je u ovom slučaju 4.

Grupirani indeksi su posebni indeksi po tablici koji koriste *primarni ključ* da bi organizirali podatke unutar tablice. Grupirani indeksi osiguravaju da se *primarni ključ* sprema po nekom redoslijedu. Grupirani indeksi se stvaraju sa stvaranjem tablice te ih se ne mora posebno deklarirati.



Slika 4.3: Primjer grupiranih indeksa po primarnom ključi [10]

Na slici 4.3. može se vidjeti kako će baza napraviti indekse za *primarni ključ id* nazvan *prijatelji_pkey*. Kada se podaci traže unutar tablice po *id* polju baza odgovara na upit brže, ali kada se podatak traži po polju *name* ili *city* baza ide redak po redak da pronađe traženi podatak. Zbog toga se koriste negrupirani indeksi. Negrupirani indeksi su reference na određena polja, oni sadrže pokazivače na zapise u tablici. Primjer negrupiranih indeksa može se vidjeti 4.4.



Slika 4.4: Negrupirani indeksi [10]

Negrupirani indeksi se koriste kako bi ubrzali pretragu podataka na tablicama s velikim brojem podataka. Takvi indeksi se kreiraju od strane developera. Indeksi nisu nova

tablica nego spremljene reference na polja u tablicama. Ne postoji limit na broj negrupiranih indeksa.

Binarna složenost pretraživanja	
Broj zapisa	**Najmanji broj prolaza za pronalazak traženog zapisa**
8	3
100	7
1 000	10
10 000	14
100 000	17
1 000 000	20

Tablica 4.1: Broj prolazaka za pretragu s indeksima [10]

U tablici 4.1. može se vidjeti najmanji broj skokova koje baza treba napraviti da bi pronašla zapis koristeći binarnu pretragu i indekse. Na primjer ne koristeći indekse baza za broj od 8 zapisa mora proći 8 puta dok koristeći binarnu pretragu mora samo 3 puta. Povećavajući broj zapisa vidljivo je da je bazi potrebno puno manje da pronađe traženi zapisi. Zbog toga se dolazi do zaključka, da se indeksi koriste kako bi ubrzali performance baze kada je to potrebno. Povećavajući bazu imajući sve više podataka unutar nje pronalazi se korist koristeći indekse da se brže dobivaju odgovori na SQL upite. Negrupirane indekse ne treba koristiti kada baza konstantno dobiva nove zapise unutar nje zbog potrebe za ažuriranjem indeksa nakon umetanja podatka. Zbog toga se obično indeksi koriste gdje se ažuriranja izvršavaju periodično.

4.3 Kreiranje indeksa nad JSON podacima

Za rad nad JSON podacima može se kreirati više vrsta indeksa. Ovisno kako se želi izvršavati pretraga baze i s kojim funkcijama se izvršava. Za kreiranje indeksa [11,12] koji se koristi za pretragu podataka uz pomoć ugrađenih JSON funkcija koristi se programskim kod 4.1. Indeks se kreira za podatke koji se mogu vidjeti u programskom kodu 3.5.

Programski kod. 4.1: Primjer kreiranja indeksa za pretragu podataka

```
CREATE SEARCH INDEX osoba_json_idx_search
ON osoba_json (osoba_podatak) FOR JSON;
```

Nakon kreiranja indeksa koristeći programski kod 4.1. baza stvori indekse za polje unutar kojeg se nalaze JSON podaci. Kako tablicu s indeks podacima izgleda vidljivo je na slici 4.5. Može se vidjeti da je baza stvorila za svaki ključ (engl. *key*) iz JSON podatka putanju po kojoj se pretražuje.

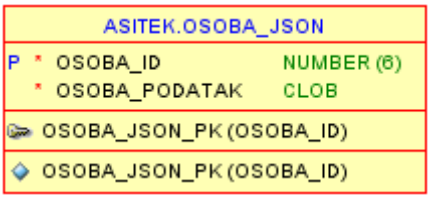
	NAME	PATH	TYPE	TLENGTH
1	_id	$._id	4	32
2	age	$.age	3	2
3	guid	$.guid	4	64
4	name	$.name	4	32
5	tags	$.tags	6	128
6	(null)	$.tags[*]	1004	16
7	about	$.about	4	1024
8	email	$.email	4	64
9	index	$.index	3	2
10	phone	$.phone	4	32
11	gender	$.gender	4	8
12	address	$.address	4	128
13	balance	$.balance	4	16
14	company	$.company	4	16
15	friends	$.friends	6	128
16	id	$.friends.id	1003	1
17	name	$.friends.name	1004	32
18	picture	$.picture	4	32
19	eyeColor	$.eyeColor	4	8
20	greeting	$.greeting	4	64
21	isActive	$.isActive	2	8
22	latitude	$.latitude	3	16
23	longitude	$.longitude	3	16
24	registered	$.registered	4	32
25	favoriteFruit	$.favoriteFruit	4	16

Slika 4.5: Primjer kreiranih indeksa za JSON podatke

5. ANALIZA BRZINE PRETRAŽIVANJA JSON PODATAKA U ORACLE BAZI

5.1 Izrada tablice i popunjavanje podataka

U ovom poglavlju usporedit će se pretrage JSON podataka s indeksima i bez njih. Za izradu testiranja potrebno je kreirati tablicu koja će sadržavati JSON podatke s indeksima i tablicu bez indeksa. Model tablice može se vidjeti na slici 5.1.

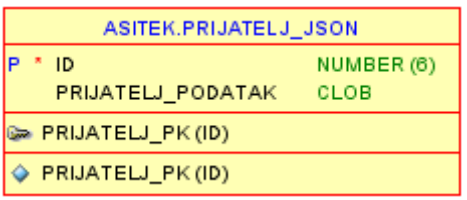


Slika 5.1: Model tablice OSOBA_JSON

Za primjer će se izraditi tablica naziva *osoba_json* koja će sadržavati stupce *osoba_id* i *osoba_podatak* koji će sadržavati JSON podatke o osobi.

Nakon izrade tablica potrebno je popuniti tablice s generiranim JSON podacima koji se mogu kreirati uz pomoć json-generatora[13] zatim umetnuti pomoću skripte u programskom kodu 5.1.

Za testiranje vanjskog ključa potrebna je druga tablica. Tablica se zove *prijatelj_json* i sadržava polja *id* i *prijatelj podatak*. Tablica će sadržavati okidač (engl. *trigger*) koji se može vidjeti u programskom kodu 5.15. te će pri umetanju JSON podataka provjeravati je li postoji vanjski ključ u primarnoj tablici (engl. *parent table*) *osoba_json*. Model tablice može se vidjeti na slici 5.2.



Slika 5.2: Model tablice PRIJATELJ_JSON

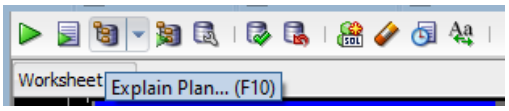
U programskom kodu 5.1. može se vidjeti skripta za unos podataka u bazu. Skripta prolazi po JSON polju generiranim json-generatorom[13], uzima svaki JSON objekata zatim unosi JSON podatak u bazu.

Programski kod. 5.1: Skripta za unos JSON podataka unutar Oracle baze

```
SET SERVEROUTPUT ON
DECLARE
   l_array json_array_t := json_array_t('[]');
   l_naredba clob;
   l_tmparray json_array_t := json_array_t('[]');
   l_tmp JSON_OBJECT_T :=  new JSON_OBJECT_T;
BEGIN
   l_array := json_array_t('[
   / *JSON podaci koji se nalaze unutar polja */
   ]');
   FOR indx IN 0 .. l_array.get_size - 1
   LOOP
      l_tmp := json_object_t(l_array.get(indx));
      --DBMS_OUTPUT.PUT_LINE(l_tmp.TO_STRING);
      l_naredba := 'insert into osoba_json (osoba_podatak)
                    values ( :1 )';
      EXECUTE IMMEDIATE l_naredba using l_tmp.TO_STRING;
   END LOOP;
END;
```

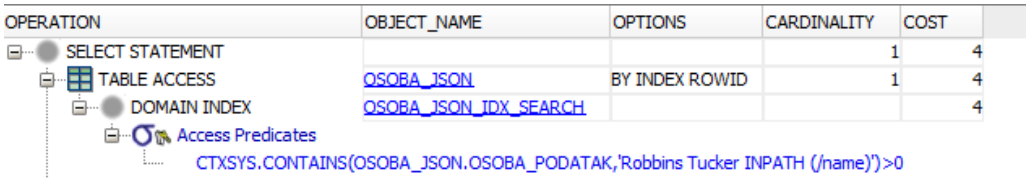
5.2 Explain plan

Za analizu pretrage podataka koristiti se ugrađena opcija *Explain plan* [14] unutar Oracle SQL Developer alata. Lokacija na korisničkom sučelju može se vidjeti na slici 5.3.



Slika 5.3: Lokacija Explain Plana na alatnoj traci

Nakon pokretanja *Explain plana* unutar Oracle SQL Developer alata dobiva se rezultat vidljiv na slici 5.4.



Slika 5.4: Explain Plan rezultat

Explain plan sadrži:

OPERATION

OBJECT_NAME

OPTION

CARDINALITY

COST

OPERATION govori o kojoj operaciji se radi. Operacije koje mogu biti su odabir, umetanje, brisanje i ažuriranje.

OBJECT_NAME sadrži ime objekta nad kojim se operacije izvršavaju.

OPTIONS govori o kakvom se pretraživanju radi.

CARDINALITY je omjer koji govori koliko će redaka i operacija baza proći da bi nam vratila podatke

COST troškovi nisu određeni za operacije pristupa tablicama. Vrijednost ovog stupca nema neku posebnu mjernu jedinicu; to je samo neka vrijednost koja se koristi za usporedbu troškova planova izvršenja. Vrijednost ovog stupca je dobivena iz *CPU_COST* i *IO_COST*.

Kada je *CARDINALITY* veliki obično se povećava i *COST*. Da bi se utvrdili i analizirali brzinu obrade podataka u bazi gledat će se *COST*. Usporedbom *COST* može se utvrditi koji upit vraća brže pretragu podataka.

5.3 Pretraga s JSON_TEXTCONTAINS

Kada je potrebno napraviti pretragu s uvjetima može se koristiti funkcija ***json_textcontains*** [15]. Sintaksa funkcije se ponaša kao SQL funkcija *contains*. Ako riječ sadrži neku od rezerviranih riječi Oracla programskog jezika mora se koristiti izlazni znak (engl. *escape character*). Da bi se funkcija mogla koristiti a da funkcija ne vraća pogrešku (engl. *error*) potrebno je definirati JSON *search indeks*. Primjer kreiranja indeksa može vidjeti u programskom kodu 4.1. Nakon što postoji indeks može se izvršiti pretraga kao u programskom kodu 5.2.

Programski kod. 5.2.: Primjera pretrage pomoću json_textcointains

```
SELECT
   *
FROM
   osoba_json
WHERE
   json_textcontains(osoba_podatak, '$.name','Robbins Tucker');
```

Zbog toga što je potrebno definirati indeks prije upotrebe funkcije nije moguće usporediti brzinu pretrage bez indeksa.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	4
TABLE ACCESS	OSOBA_JSON	BY INDEX ROWID	1	4
DOMAIN INDEX	OSOBA_JSON_IDX_SEARCH			4
Access Predicates				
CTXSYS.CONTAINS(OSOBA_JSON.OSOBA_PODATAK,'Robbins Tucker INPATH (/name)')>0				

Slika 5.5: Explain plan za json_textcontains s indeksom

U *explain planu* na slici 5.5. može se vidjeti da je COST mali i da bazi ne treba puno resursa da vrati odgovore na upit.

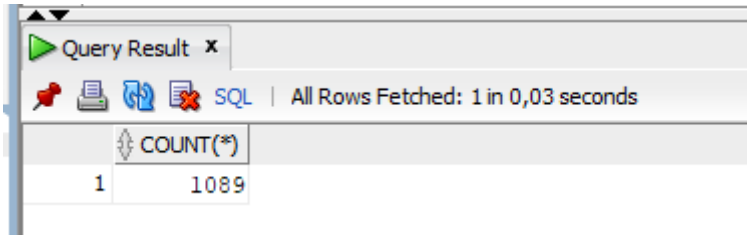
5.4 Analiza s explain planom

Za potrebe analize koristi se tablica popunjena s 1089 podataka. Za provjeru broja podataka izvršava se upit naveden u programskom kodu 5.3.

Programski kod. 5.3: Broj redaka u bazi

```
SELECT
   COUNT(*)
FROM
   OSOBA_JSON;
```

Nakon toga dobiven je rezultat na upit vidljiv na slici 5.6.



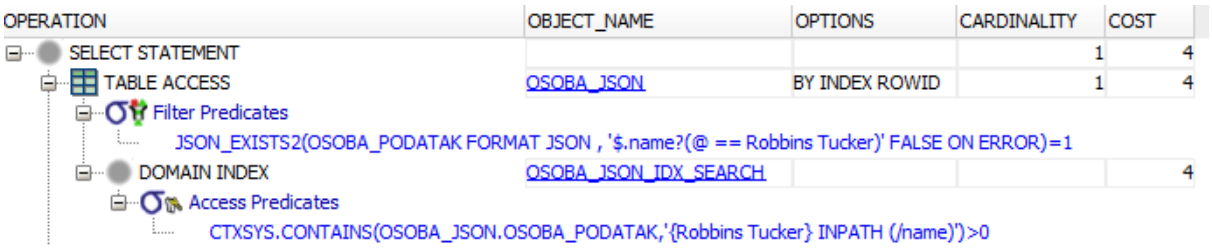
Slika 5.6: Broj redaka u tablici

Za prvu usporedbu brzine pretrage podataka koristit će se programski kod 5.4. Nakon što se provjeri da programski kod radi i vraća podatke može se provjeriti *COST* pritiskom na tipku F10 ili ikonice koja se nalazi na alatnoj traci.

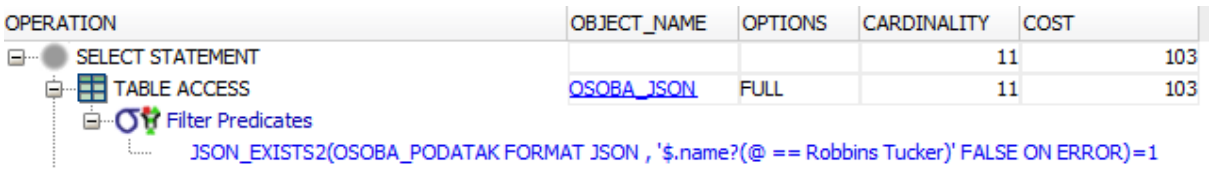
Programski kod. 5.4: Pretraga s json_exists

```
select
   *
from
   osoba_json
where
   json_exists(osoba_podatak,'$.name?(@=="Robbins Tucker")');
```

Na dolje navedenim slikama 5.7 i 5.8 može se vidjeti *explain plan* kada se traže podaci unutar JSON dokumenta s indeksom i bez indeksa koristeći programski kod 5.4. Dok se koristi indeks vidljivo je da *COST* iznosi 4, a dok se ne koristi indeks *COST* iznosi 103. Što dovodi do zaključka da se s indeksom upit brže izvršava.



Slika 5.7: Explain plan za json_exist s indeksom



Slika 5.8: Explain plan za json_exist bez indeksa

Programski kod. 5.5: Pretraga s ROWID

```
select
   JSON_VALUE(osoba_podatak,'$.name')
from
   osoba_json
where
   osoba_id = 854;
```

Kad se izvršava pretraga s *ROWID* baza vraća najbrže podatke. Primjer upita vidi se u programskom kodu 5.5. Ali za takvu pretragu je potrebno znati vrijednost *primarnog ključa* u ovom slučaju *osoba_id (ROWID)*. Rezultat upita s programskog koda 5.5. može se vidjeti na slici 5.9.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	2
TABLE ACCESS	OSOBA_JSON	BY INDEX ROWID	1	2
INDEX	OSOBA_JSON_PK	UNIQUE SCAN	1	1
Access Predicates				
OSOBA_ID=854				

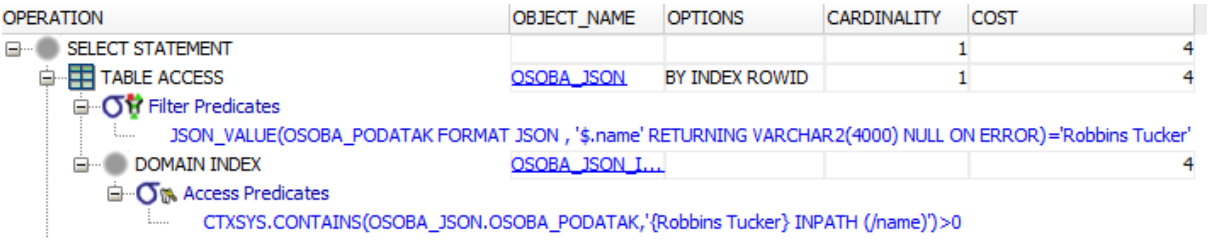
Slika 5.9: Explain plan za rowid

Programski kod. 5.6: Pretraga s JSON_VALUE u where uvjetu

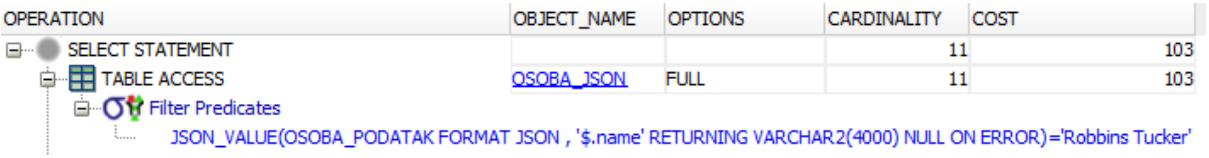
```
select
   *
from
   osoba_json
where
   JSON_VALUE(osoba_podatak,'$.name')='Robbins Tucker';
```

Na dolje navedenim slikama 5.10. i 5.11. može se vidjeti *explain plan* kada se traže podaci unutar JSON dokumenta s funkcijom ***json_value*** unutar *WHERE* uvjeta s indeksom i bez indeksa. Upit je vidljiv u programskom kodu 5.6. Dok se koristi index može se primjetiti

da je *COST* 4 a dok se ne koristi indeks *COST* je 103. Što znači da se pretraga s indeksom izvršava brže.



Slika 5.10: Explain plan za json_value s indeksom

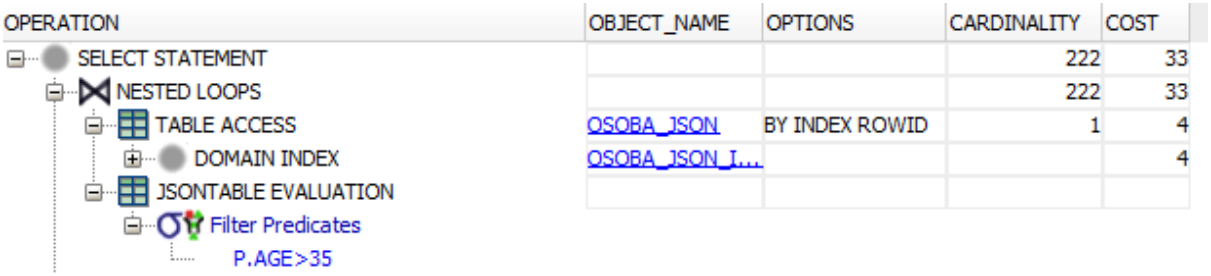


Slika 5.11: Explain plan za json_value bez indeksom

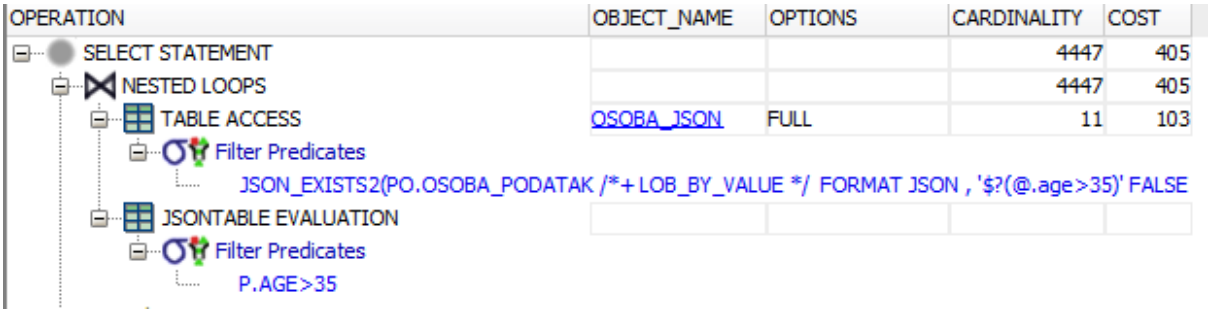
Programski kod. 5.7: Pretraga s JSON_TABLE

```
SELECT
   jt.*
FROM
   osoba_json po,
   json_table(
              po.osoba_podatak,
              '$' COLUMNS po_name  VARCHAR2(50 CHAR) PATH '$.name',
              age     NUMBER(3) PATH '$.age'
              ) jt
WHERE
   age > 35;
```

Nakon izvršavanja upita u programskom kodu 5.7. dobiveni rezultati se mogu vidjeti na slikama 5.12. i 5.13. Indeks ima utjecaj na pretragu korištenjem funkcije ***json_table***. Uspoređujući cost iz upita s indeksom koji iznosi ukupno 37 i bez indeksa 508 vidljivo je da baza vraća podatke brže koristeći indeks.



Slika 5.12: Explain plan za json_value s indeksom



Slika 5.13: Explain plan za json_table bez indeksom

5.5 Analiza brzine pomoću paketa

5.5.1 Opis paketa GET_DATA

Za simulaciju u svrhu mjerenja vremena s i bez indeksa napravljena je skripta i paket unutar Oracle SQL Developer alata. Paket će se pozvati unutar skripte kao što bi se napravilo da se šalje poziv izvan baze s nekog servisa. Skripta vraća vrijeme trajanja koje je trebalo paketu da odgovori. Programski kod skripte može se vidjeti u programskom kodu 5.8. Za mjerenje vremena koristi se funkcija koja se nalazi unutar SQL Developer alata *systimestamp*. Paketu se predaju 3 parametra strukturirana kao JSON podatak. Prvi parametar je *key* po kojemu se traži unutar JSON objekta, drugi je *value* koji se uspoređuje za vraćanje podatka i treći *case* koji označuje o kojoj pretrazi se radi. U paketu se koriste *napuci* (engl. *hint*) kako bi se manipuliralo pretragom s indeksom i bez njega.

Programski kod.5.8: Skripta za mjerenje izvršavanja vremena paketa GET_DATA

```
SET SERVEROUTPUT ON
declare
   t1 timestamp;
   t2 timestamp;
   l_string clob;
   l_out JSON_OBJECT_T;
   l_sout clob;
begin
   t1 := systimestamp;
   l_string := '{"key":"friends.name","value":"Ava
               Stewart",”case”:”JV”}';
   get_data.p_get_data(l_string,l_sout);
   t2 := systimestamp;
   dbms_output.put_line(l_sout);
   dbms_output.put_line('Start: ' || t1);
   dbms_output.put_line('  End: ' || t2);
   dbms_output.put_line('Elapsed Seconds: '||TO_CHAR(t2-t1, 'SSSS.FF'));
end;
```

U programskom kodu skripte deklarirane su varijable za vrijeme *t1* i *t2* i varijabla naziva *l_string* koji sadrži parametre za pretragu. Mjeri se vrijeme prije izvršavanja poziva prema *GET_DATA* paketu te nakon što se izvršio poziv. Oduzima se konačno vrijeme od početnog i dobije se brzina izvršavanja paketa. U programskom kodu 5.9. u prvom dijelu programskog paketa *GET_DATA* definiraju se varijable koje paket koristi. Poslije toga se izvršava *select* koji iz ulaznih podataka predanih paketu pronalazi vrijednosti za *l_key, l_value i l_case*. *L_key* varijabla služi za predaju podatka o ključu prema kojem se traži podatak. Koriste ga funkcije ***json_value***, ***json_query***, ***json_exist***. *L_value* služi kao parametar s kojim se uspoređuju JSON podaci. *L_case* je vrsta pretrage koju će paket raditi. Generira se varijabla *l_naredba* s kojom se izvršava upit prema bazi i vraćaju podaci koji se spremaju u *l_ostring*. Primjer *l_naredbe* može se vidjeti u programskom kodu 5.10.

Programski kod.5.9: PAKETA GET_DATA

```
procedure p_get_data(l_string in clob, l_ostring out clob) as
   l_naredba varchar2(300);
   l_value varchar2(120);
   l_key varchar2(120);
   l_case varchar2(120);
   l_izlaz clob;
begin
   select
      JSON_VALUE(l_string, '$.key' RETURNING VARCHAR2),
      JSON_VALUE(l_string, '$.value' RETURNING VARCHAR2),
      JSON_VALUE(l_case, '$.case' RETURNING VARCHAR2)
   into
      l_key,
      l_value,
      l_case
   from
      DUAL;
   CASE l_case
      /*različiti slučajevi za koje se kreira l naredba s obzirom na
       poslane podatke*/
   END CASE;
   EXECUTE IMMEDIATE l_naredba into l_izlaz;
   l_ostring := l_izlaz;
end p_get_data;
```

Programski kod. 5.10: Primjer l_naredbe

```
l_naredba := 'select OSOBA_PODATAK from osoba_json where ' ||
             ' JSON_VALUE(OSOBA_PODATAK, ''$.''|| l_key|| ''||''''''|| ')='''
             || l_value || '''';
```

5.5.2 Rezultati testiranja s paketom GET_DATA

Prvo testiranje dohvaćanja podataka se izvršava s pomoću funkcije ***json_exist*** unutar *WHERE* uvjeta gdje nam poziv vraća podatke. Podaci koji se šalju su vidljivi u programskom kodu 5.11.

Programski kod. 5.11: JSON podaci za pretragu

```
{"key":"name","value":"Singleton Osborn","case":"JE"}
```

Nakon obavljenih 10 upita mijenja se case i rade testovi s korištenjem indeksa.

Tablica 5.1: Rezultati upita prema bazi s funkcijom json_exist

JSON_EXIST		JSON_EXIST S INDEKSOM	
14,224	ms	4,622	ms
17,728	ms	4,381	ms
15,935	ms	3,953	ms
17,942	ms	3,821	ms
15,850	ms	3,523	ms
15,254	ms	3,275	ms
15,809	ms	6,198	ms
15,911	ms	4,298	ms
16,697	ms	3,547	ms
14,855	ms	3,111	ms
17,453	ms	3,444	ms
24,180	ms	10,619	ms
15,777	ms	3,528	ms
13,161	ms	4,543	ms
14,464	ms	4,643	ms
17,019	ms	3,034	ms
16,208	ms	3,031	ms
16,427	ms	3,791	ms
15,273	ms	3,713	ms
14,729	ms	3,216	ms
16,245	ms	**4,215**	ms

Usporedbom rezultata upita prema bazi bez indeksa i s indeksom vidljivih na tablica 5.1. može se zaključiti da indeks u pretrazi korištenjem ***json_exist*** funkcije se izvršava puno brže. Žutim poljem označena je prosječna vrijednost izvršavanja upita prema bazi. Vidljivo je

da prosječno vrijeme izvršavanja s indeksom znatno manje uspoređujući prosječno vrijeme izvršavanja bez indeksa.

Drugo testiranje dohvaćanja podataka se izvršava s pomoću funkcije ***json_value*** unutar *WHERE* uvjeta gdje poziv vraća podatke. Podaci koji se šalju su kao programski kod. 5.8. JSON podaci za pretragu ali je izmijenjen case.
Nakon obavljenih 10 upita mijenja se case i rade testovi s korištenjem indeksa.

Tablica 5.2: Rezultati upita prema bazi s funkcijom json_value

JSON_VALUE		**JSON_VALUE S INDEKSOM**	
7,912	ms	3,755	ms
7,543	ms	4,003	ms
7,944	ms	3,502	ms
6,825	ms	2,397	ms
5,342	ms	5,131	ms
11,556	ms	3,035	ms
5,491	ms	4,523	ms
5,784	ms	3,415	ms
7,615	ms	4,730	ms
5,766	ms	3,591	ms
11,103	ms	3,014	ms
11,109	ms	3,971	ms
7,160	ms	3,897	ms
6,246	ms	3,261	ms
8,490	ms	6,011	ms
5,354	ms	3,494	ms
10,212	ms	3,864	ms
5,575	ms	5,044	ms
8,360	ms	4,040	ms
5,530	ms	3,214	ms
7,546	ms	**3,895**	ms

Usporedbom rezultata upita prema bazi na tablici 5.2. bez indeksa i s indeksom može se zaključiti da indeks u pretrazi korištenjem funkcije ***json_value*** izvršava puno brže. Označenim žutim poljem označena je prosječna vrijednost izvršavanja upita prema bazi. Vidljivo je da prosječno vrijeme izvršavanja s indeksom znatno manje uspoređujući prosječno vrijeme izvršavanja bez indeksa. Također uzimajući u obzir da se radi o istim

podacima može se primijetiti da se upit izvršava brže s funkcijom ***json_value*** nego funkcijom ***json_exist***.

Treće testiranje dohvaćanja podataka se izvršava s pomoću funkcije ***json_query*** unutar *WHERE* uvjeta gdje upit vraća podatke. Podaci koji se šalju su kao programski kod. 5.11. JSON podaci za pretragu ali je izmijenjen case. Nakon obavljenih 10 upita mijenja se case i rade testovi s korištenjem indeksa.

Tablica 5.3: Rezultati upita prema bazi s funkcijom json_query

JSON_QUERY		**JSON_QUERY S INDEKSOM**	
26,941	ms	2,629	ms
17,991	ms	3,214	ms
14,403	ms	7,674	ms
15,074	ms	3,260	ms
14,127	ms	3,439	ms
14,434	ms	3,079	ms
17,559	ms	3,165	ms
11,679	ms	3,879	ms
16,063	ms	4,360	ms
14,270	ms	3,653	ms
15,397	ms	3,665	ms
13,929	ms	19,151	ms
16,264	ms	3,044	ms
17,229	ms	4,333	ms
22,750	ms	4,468	ms
15,207	ms	4,206	ms
19,730	ms	3,463	ms
12,862	ms	4,895	ms
14,557	ms	3,808	ms
12,229	ms	5,092	ms
16,135	ms	**4,724**	ms

Usporedbom podataka dobivenih testiranjem iz priloženih tablica vidljivo je da su pretrage s indeksima brže.

5.6 Analiza brzine umetanja JSON podatka koji sadrži vanjski ključ

5.6.1 Izrada podređene tablice za testiranje vanjskog ključa

Da bi se mogla testirati brzina usporedbe primarnog i vanjskog ključa pri umetanju JSON podataka unutar podređene tablice potrebno je prvo izraditi podređene tablicu (engl. *child table*). U programskom kodu 5.12. može se vidjeti kod za izradu podređene tablice *prijatelj_json*.

Programski kod. 5.12: Izrada tablice za prijatelj_json

```
create table prijatelj_json (
   prijatelj_id number(6) not null,
   prijatelj_podatak clob not null,
   constraint PRIJATELJ_JSON_PK PRIMARY KEY (PRIJATELJ_ID)
);
CREATE sequence PRIJATELJ_JSON_ID_SEQ;
CREATE trigger BI_PRIJATELJ_JSON_ID
  before insert on PRIJATELJ_JSON
  for each row
begin
  select PRIJATELJ_JSON_ID_SEQ.nextval into :NEW.PRIJATELJ_ID from dual;
end;
```

Nakon izrade tablice potrebno je postaviti *IS JSON* ograničenje (engl. *constraint*) kako bi se spriječio unos podataka koji nisu JSON formata. Koristi se programski kod 5.13.

Programski kod.5.13: Dodavanje ograničenja

```
alter table prijatelj_json
add constraint osoba_prijatelj_json
check (prijatelj_podatak is json);
```

5.6.2 Opis funkcije FKEY_CHECK i triggera FK_CHECK

Zbog toga što Oracle nema provjeru *vanjskog ključa* s *primarnim* za JSON podatke napravljena je funkcija koja uspoređuje ulazne podatke unutar okidača (engl. *trigger)* pri umetanju podataka u bazu. Funkcija vraća istinu (engl. *true*) ako polje i njegova vrijednost

koja je odabrana da bude vanjski ključ (engl. *foreign key*) postoji unutar primarne (engl. *parent*) tablice.

Programski kod. 5.14: Funkcija FKEY_CHECK

```
CREATE OR REPLACE FUNCTION FKEY_CHECK (p_string IN VARCHAR2)
   RETURN boolean
IS
   l_value varchar2(120);
   l_key varchar2(120);
   l_tablica varchar2(120);
   l_naredba varchar2(300);
   l_izlaz clob;
BEGIN
   l_key := 'friends.name';
   l_tablica := 'osoba_json';
   l_value := p_string;
   BEGIN
      l_naredba := 'select OSOBA_PODATAK from '|| l_tablica ||'
                   osoba_json where ' || 'JSON_EXISTS(OSOBA_PODATAK,
                   ''$.'|| l_key || '?(@ == "' || l_value ||   '")'
                   || '''' || ')';
                   EXECUTE IMMEDIATE l_naredba into l_izlaz;
   exception when no_data_found THEN
      return false;
   END;
RETURN true;
END FKEY_CHECK;
```

Funkcija vidljiva u programskom kodu 5.14. se poziva unutar okidača (engl. *trigger)* prikazanom u programskom kodu 5.15. pri umetanju i ažuriranju podataka unutar podređene tablice (engl. *child table*) *prijatelj_json*. Pri umetanju u bazu uzima podatke koji su poslani za umetanje te provjerava postoji li *primarni ključ* (engl. *primary key*). Ako *primarni ključ* ne postoji vraća poruku da ključ ne postoji te ne izvršava umetanje unutar baze. Ako *primarni ključ* postoji umetanje se izvršava.

Programski kod. 5.15: Trigger FK_CHECK

```
CREATE OR REPLACE TRIGGER FK_CHECK
BEFORE INSERT OR UPDATE ON PRIJATELJ_JSON
FOR EACH ROW
DECLARE
   l_value VARCHAR2(120);
BEGIN
   select
      JSON_VALUE(:NEW.PRIJATELJ_PODATAK, '$.name' RETURNING VARCHAR2)
   into
      l_value
   from
      DUAL;


   IF (FKEY_CHECK(l_value)) THEN
      null;
   ELSE
      RAISE_APPLICATION_ERROR(-20001, 'PK ne postoji')
   END IF;
END;
```

5.6.3 Analiza brzine pri umetanju podataka koristeći indeks i bez njega

Da bi se mjerilo vrijeme izvršavanja iskorištava se skripta s programskog koda 5.8. gdje je potrebno izmijeniti poziv paketa prema bazi s naredbom za umetanje koja je popunjena s podacima koji sadrže valjani *vanjski ključ* i ostale podatke. Da bi se izračunalo vrijeme trajanja izvršeno je 10 umetanja koristeći indeks na primarnu tablicu (engl. *parent table*) i 10 bez korištenja indeksa.

Tablica 5.4: Rezultati umetanja u bazu

UMETANJE S INDEKSOM		**UMETANJE BEZ INDEKSA**	
5,376	ms	21,088	ms
39,020	ms	13,514	ms
3,683	ms	16,602	ms
3,013	ms	15,444	ms
4,883	ms	15,674	ms
3,679	ms	18,136	ms
5,775	ms	15,808	ms
4,037	ms	17,226	ms
5,752	ms	16,234	ms
3,837	ms	14,250	ms
4,310	ms	26,701	ms
4,655	ms	23,185	ms
5,250	ms	18,001	ms
5,064	ms	17,380	ms
5,629	ms	17,164	ms
3,433	ms	16,310	ms
3,642	ms	18,494	ms
3,219	ms	15,774	ms
4,394	ms	15,713	ms
14,925	ms	16,556	ms
6,679	ms	**17,463**	ms

U tablici 5.4. vidljivo je da indeks ima značajan utjecaj na umetanje podataka. Indeks značajno smanjuje usporedbu *vanjskog ključa* s *primarnim ključem* zbog čega se podaci unose u bazu brže.

6. ZAKLJUČAK

Spremanje JSON podataka u bazu podataka postaje sve češća praksa. Pri spremanju JSON podatka nije potrebno raditi transformaciju podataka pri dohvaćanju i spremanju u bazu što značajno ubrzava proces dohvaćanja podataka. Oracle kontinuirano radi na poboljšanju rada s JSON podacima te sa svakom novom verzijom omogućuju korisniku bolje performanse. U radu je istraženo i dokazano da se pri korištenju indeksa s ugrađenim funkcijama za rad s JSON podacima unutar Oracle okruženja značajno poboljšavaju performanse pretrage i pohrane podataka. U slučajevima kada se koriste funkcije ***json_value***, ***json_query***, ***json_table***, ***json exist*** i ***json_textcontains*** s korištenjem indeksa dokazano je da baza radi s podacima efikasnije.

Pod uvjetom da se podaci kontinuirano ne ažuriraju preporučuje se korištenje indeksa zbog ubrzanja vraćanja podataka.

7. LITERATURA

[1] What is Database [Online] Dostupno na: https://www.oracle.com/database/what-is-database/ (28.09.2021.)

[2] Oracle SQL Developer [Online] Dostupno na: https://www.oracle.com/database/technologies/appdev/sqldeveloper-landing.html (04.10.2021.)

[3] PL/SQL [Online] Dostupno na: https://www.oracletutorial.com/plsql-tutorial/what-is-plsql/ (28.09.2021.)

[4] JSON [Online] Dostupno na: https://www.webprogramiranje.org/json/ (20.09.2021.)

[5] JSON Data Types [Online] Dostupno na: https://www.w3schools.com/js/js_json_datatypes.asp (28.09.2021.)

[6] Chris Saxon How to Store, Query, and Create JSON Documents in Oracle Database 12.01.2021. [Online] Dostupno na: https://blogs.oracle.com/sql/post/how-to-store-query-and-create-json-documents-in-oracle-database (28.09.2021.)

[7] JSON_VALUE [Online] Dostupno na: https://docs.oracle.com/database/121/SQLRF/functions093.htm#SQLRF56668 (28.09.2021.)

[8] JSON_QUERY [Online] Dostupno na: https://docs.oracle.com/database/121/SQLRF/functions091.htm#SQLRF56718 (28.09.2021.)

[9] SQL/JSON Condition JSON_EXISTS JSON_VALUE [Online] Dostupno na: https://docs.oracle.com/en/database/oracle/oracle-database/18/adjsn/condition-JSON_EXISTS.html#GUID-8A0043D5-95F8-4918-9126-F86FB0E203F0 (28.09.2021.)

[10] Blake Barnhill Indexing [Online] Dostupno na: https://dataschool.com/sql-optimization/how-indexing-works/ (28.09.2021.)

[11]Indexes for json data [Online] Dostupno na: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/adjsn/indexes-for-json-data.html#GUID-8A1B098E-D4FE-436E-A715-D8B465655C0D (28.09.2021.)

[12] Stvaranje i korištenje indeksa radi poboljšanja performansi [Online] Dostupno na: https://support.microsoft.com/hr-hr/office/stvaranje-i-kori%C5%A1tenje-indeksa-radi-poboljšanja-performansi-0a8e2aa6-735c-4c3a-9dda-38c6c4f1a0ce (28.09.2021.)

[13] JSON Generator [Online] Dostupno na: https://www.json-generator.com/ (28.08.2021.)

[14] Using EXPLAIN PLAN [Online] Dostupno na: https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm (28.09.2021.)

[15] Full Text Search Queries [Online] Dostupno na: https://docs.oracle.com/en/database/oracle/oracle-database/21/adjsn/full-text-search-queries.html#GUID-58ADCDE5-7564-4DA0-BED7-B0DBFD5AE6FB (28.09.2021.)

8. OZNAKE I KRATICE

JSON - JavaScript Object Notation

XML - EXtensible Markup Language

BLOB - binarno veliki objekt (engl. *binary large object*)

CLOB - karakterno veliki objekt (engl. *character large object*)

DBMS - Database Management System

ANSI - American National Standards Institute

ISO - Međunarodna organizacija za standardizaciju

9. SAŽETAK

U radu je opisan princip rada s JSON podacima u Oracle okruženju koristeći Oracle SQL Developer alata. Zbog sve veće primjene JSON podataka istražena je brzina pretrage podataka unutar Oracle okruženja koristeći indekse i bez njih. Koristeći ugrađene funkcije za rad s JSON podacima unutar Oracle SQL Developer alata izmjereno je vrijeme trajanja od slanja upita prema bazi do vraćanja podataka. Poslije upita rezultati su zabilježeni i uspoređeni da bi se vidio utjecaj indeksa na performanse baze. Na kraju rada dokazano je da indeksi značajno ubrzavaju pretragu podataka te bi se trebali koristiti u situacijama gdje je to moguće.

ključne riječi: oracle, sql, json, index

10. ABSTRACT

The final thesis describes the principle of working with JSON data in an Oracle environment using Oracle SQL Developer. Due to the increasing use of JSON data, the speed of data retrieval within the Oracle environment using indexes and without them has been investigated using the built-in functions for working with JSON data within Oracle SQL Developer, the time from sending the query to the database to restoring the data was measured. After the query, the results were recorded and compared to see the impact of the index on base performance. At the end of the paper, it was proven that indexes significantly speed up data retrieval and should be used in situations where possible.

key words: oracle, sql, json, index

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 6.10.2021.	ANTONIO SITEK	Antonio Sitek

Prema Odluci Veleučilišta u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

ANTONIO SITEK

ime i prezime studenta/ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 06.10.2021.

Antonio Sitek

potpis studenta/ice