Oracle PL/SQL proširenje za ažuriranje i dohvaćanje podataka iz tablica

Petrec, Domagoj

VELEUČILIŠTE U BJELOVARU
PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

Oracle PL/SQL proširenje za ažuriranje i dohvaćanje podataka iz tablica

Završni rad br. 13/RAČ/2022

Domagoj Petrec

Bjelovar, listopad 2022.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Domagoj Petrec** JMBAG: **0314019553**

Naslov rada (tema): **Oracle PL/SQL proširenje za ažuriranje i dohvaćanje podataka iz tablica**

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

Grana: **Obrada informacija**

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

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Krunoslav Husak, dipl. ing. rač., predsjednik**
2. **Tomislav Adamović, mag. ing. el., mentor**
3. **Ivan Sekovanić, mag. ing. inf. et comm. techn., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 13/RAČ/2022

U sklopu završnog rada potrebno je:

1. Izraditi optimiziranu verziju sistemske tablice all_tab_cols i tablicu za spremanje JSON shema
2. Napisati PL/SQL paket za dohvaćanje/unos/spremanje podataka u tablice
3. Napisati PL/SQL funkciju za dinamičko čitanje i filtriranje podataka iz tablica i view-ova
4. Izraditi PL/SQL funkciju za integraciju s ostalim proširenjima u Oracle bazi podataka
5. Izraditi tablicu za spremanje poruka i PL/SQL funkciju za upravljanje porukama u Oracle bazi podataka
6. Upravljati s iznimkama u komunikaciji između procedura i funkcija u PL/SQL-u

Datum: 31.08.2022. godine

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

REPUBLIKA HRVATSKA VELEUČILIŠTE U BJELOVARU BJELOVAR

Zahvala

Zahvaljujem mentoru mag. ing. el. Tomislavu Adamoviću, pri pomoći u odabiru teme i pisanju završnoga rada. Također zahvaljujem svim ostalim profesorima na stečenom znanju prilikom studiranja na Veleučilištu u Bjelovaru.

Sadržaj

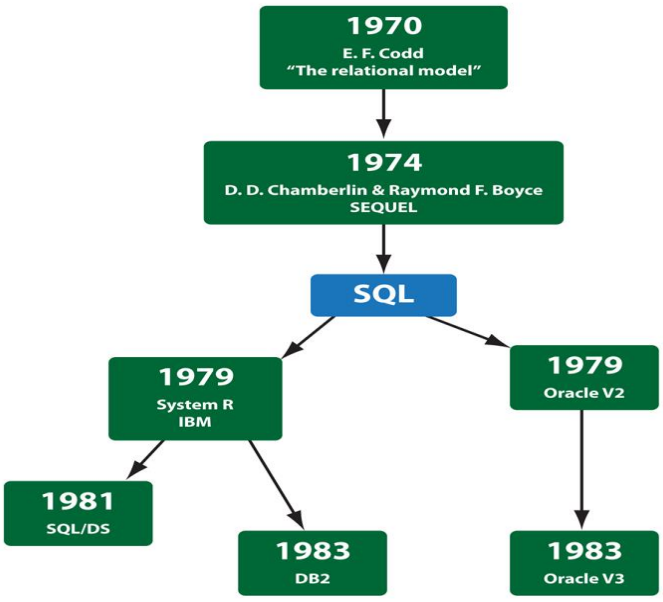
1. UVOD

U ovom završnom radu tema je proširenje CRUD (engl. *Create, Read, Update, Delete*) operacija nad tablicama u Oracle bazi podataka. Operacije CRUD-a osnova su za bilo koju bazu podataka. U ovom radu prikazano je dinamičko umetanje i dohvaćanje podataka iz baze. Dinamičko umetanje podataka pruža fleksibilnost i sigurnost podataka. Za realizaciju rada korišten je JSON (engl. *JavaScript Object Notation*) format zbog sve češće upotrebe. JSON format ne treba se transformirati i može biti tekstualno velik te čitljiv i razumljiv ljudima.
Bit ovoga rada jest pokazati kako dinamičkim putem umetnuti jedan veliki JSON na što sigurniji način uz što manje pogrešaka zbog dodanih poruka i kontrola koji su razumljivi korisnicima. Ovaj rad podijeljen je na sedam poglavlja. U prvim četirima poglavljima nalaze se općeniti podaci koji su korišteni za realizaciju rada i njihovo značenje. U drugom poglavlju ukratko je objašnjena povijest SQL-a. U trećem poglavlju objašnjeni su svi osnovni pojmovi SQL-a koji su korišteni u radu. U četvrtom poglavlju objašnjen je JSON format te njegovo korištenje u Oracle bazi. U posljednjim trima poglavljima objašnjeno je kako program radi i od čega se sastoji. Od petog poglavlja počinje realizacija rada s porukama. Poruke su napravljene zbog lakšeg upravljanja pogreškama da se budućim korisnicima programa olakša dohvat i unos poruka te shvaćanje same pogreške. U šestom poglavlju radi se o tablici „vub_tabs_cols“ koja je napravljena po uzoru na sistemsku tablicu „all_tabs_cols“ koja sadrži samo bitne informacije, poput imena tablica i imena kolona. Indeksirana je, što omogućava brži pronalazak informacija te je s tablicom moguće manipulirati i spremati u nju, dok sa sistemskom tablicom to nije moguće. U sedmom poglavlju pokazano je kako završni rad funkcionira. Radi se o paketu json_razrada koji je ogledni primjer kako bi se trebalo pristupati JSON-u i razlamati na objekte i spremati u postojeće strukture. Zaključak sadrži konačne misli o cjelokupnom radu.

.

2. POVIJEST SQL-A

SQL su 1970-ih godina razvili IBM-ovi istraživači Raymond Boyce i Donald Chamberlin [1]. Tada se zvao 'Sequel Structured English Query Language' i nastao je od relacijskog modela koji je napravio Edgar Frank Codd, a koji se zvao 'Relacijski model podataka za velike zajedničke banke podataka'. U relacijskom modelu Edgar Frank Codd predložio je da se svi podaci u bazi predstavljaju u relacijama. Na temelju tih informacija Raymond Boyce i Donald Chamberlin došli su do SQL-a. Izvorna verzija SQL-a dizajnirana je za manipuliranje i dohvaćanje podataka pohranjenih u IBM-ovu sustavu upravljanja relacijskom bazom podataka, poznatijem kao 'System R'. Nekoliko godina kasnije SQL jezik postao je javno dostupan. Godine 1979. tvrtka Relation Software, koja je kasnije postala Oracle, komercijalno je izdala vlastitu verziju programskog jezika SQL nazvanu 'Oracle V2'. Američki nacionalni institut za standarde (ANSI) i Međunarodna organizacija za standardizaciju (ISO) smatraju SQL standardnim jezikom u komunikaciji relacijskih baza podataka, dok glavni dobavljači SQL-a modificiraju jezik prema svojim željama i potrebama. Danas je SQL prihvaćen kao standardni jezik.



Slika 2.1: Struktura povijesti Baza podataka [1]

3. SQL

SQL je strukturni jezik i koristi se najčešće za pohranjivanje, manipuliranje, dohvaćanje podataka te komunikaciju s bazama podataka koje se sastoje od tablica sastavljenih od redaka i stupaca [2]. SQL koristi skup naredbi pomoću kojih svi programi i korisnici pristupaju podacima u bazi podataka. SQL koristi naredbe za obavljanje zadataka kao što su unos, brisanje, ažuriranje podataka ili dohvaćanje podataka iz baze podataka. Postoje tri skupine naredbi, a to su naredbe za definiranje (DDL), manipuliranje (DML) te kontrolne naredbe (DCL). SQL se smatra standardnim jezikom za sustave upravljanja relacijskim bazama podataka. Aplikacijski programi često dopuštaju korisnicima pristup bazi bez izravnog korištenja SQL-a, ali zauzvrat aplikacije moraju koristiti SQL kada izvršavaju korisnički zahtjev. Primjer SQL naredbe može se vidjeti u programskom kodu 3.1. Neki uobičajeni sustavi upravljanja relacijskim bazama podataka koji koriste SQL su Oracle, Sysbase, Microsoft SQL server, Access, Ingres itd. SQL je standardiziran preko standarda ANSI & ISO.

ANSI je glavna organizacija koja podržava razvoj tehnoloških standarda u SAD-u, bavi se razvojem i održavanjem standarada za produkte, servise, sisteme i osoblje. ANSI nadzire rad na nekoliko dugotrajnih standarda, uključujući američki standardni kod za razmjenu informacija ASCII.

ISO je međunarodna organizacija za normizaciju, sastavljena od predstavnika raznih nacionalnih normizacijskih tijela. ISOizdaje industrijske i komercijalne norme.

Programski kod 3.1: Primjer SQL naredbe

```
SELECT * FROM STUDENTI WHEREIME = 'DOMAGOJ' AND GODINE = 23;
```

3.1. PL/SQL

PL/SQL je blokovski strukturirani jezik koji razvojnim programerima omogućuje kombiniranje SQL-a s proceduralnim strukturama. Ujedno je i proceduralna ekstenzija tvrtke Oracle Corporation za SQL i Oracle relacijsku bazu podataka [3]. PL/SQL kombinira snagu SQL-a za manipulaciju podacima s procesorskom snagom proceduralnog jezika za stvaranje

SQL upita te također osigurava besprijekornu obradu naredbi poboljšavanjem sigurnosti, prenosivosti i robusnosti baze podataka. PL/SQL daje više kontrole programerima korištenjem petlji, uvjeta i objektno orijentiranih koncepata. Blokovi se mogu jednom izvršiti i pohraniti u izvršnom obliku kako bi se poboljšalo vrijeme odaziva. PL/SQL je dizajniran za izračunavanje i vraćanje jedne skalarne vrijednosti. Korisnici mogu kreirati vlastite funkcije koje se nadopunjuju. Funkcije se mogu koristiti u SQL upitu, a procedure ne mogu. Bez PL/SQL-a Oracle mora odrađivati SQL naredbe jednu po jednu. U mrežnom okruženju to može utjecati na protok prometa i usporiti vrijeme odaziva.

PL/SQL	SQL
Blok kodova koji se koristi za pisanje cijelih blokova programa	Jedan upit koji se koristi za izvođenje DML i DDL operacija
Proceduralni koji definira kako se stvari trebaju učiniti	Deklarativno i definira što treba učiniti, a ne kako stvari treba učiniti
Izvršava se kao cijeli blok	Izvršava se kao jedna izjava
Koristi se za izradu aplikacija	Koristi se za održavanje podataka
Nema interakcije s poslužiteljima baze podataka	Interakcija s poslužiteljima baze podataka
Proširenje SQL-a može sadržavati SQL kod unutar sebe	Ne može sadržavati PL/SQL kod

Tablica 3.1: Usporedba SQL-a i PL/SQL-a

3.2. Anonimni blok

PL/SQL anonimni blok jest izvršeni upit koji može sadržavati SQL i PL/SQL kontrolne izjave, a koristi se za implementaciju proceduralne logike [4]. Osnovna jedinica u PL/SQL je blok, a svi programi sastoje se od blokova koji se mogu ugnijezditi jedan u drugi. Blok bez naziva je anonimni blok i on se ne sprema na poslužitelj. Anonimni blokovi su samo za

jednokratnu upotrebu, ali mogu biti korisni u svrhu testiranja programa. Najčešće služe za pozivanje procedura ili funkcija te unos podataka u bazu. S obzirom na to da nemaju ime, ne može ih pozvati nijedan drugi blok jer nema referencu. PL/SQL blok sastoji se od tri odjeljka: deklaracija, izvršni odjeljak te odjeljak za rukovanje iznimkama. U bloku izvršni odjeljak je obavezan, dok su deklaracija i odjeljci za rukovanje iznimkama izborni.

1. **DECLARE** – PL/SQL ima odjeljak za deklaraciju u kojemu se deklariraju varijable, dodjeljuje memoriju za kursore, tipove podataka te procedure. Svaka deklaracija mora biti završena točkom sa zarezom.
2. **BEGIN** – PL/SQL blok ima izvršnu sekciju, on počinje ključnom riječi BEGIN i završava ključnom riječi END. Izvršni odjeljak mora imati najmanje jednu izvršnu naredbu, čak i ako je to NULL naredba koja ne radi ništa, i također može sadržavati ugniježđene blokove.
3. **EXCEPTIONS** – PL/SQL blok ima odjeljak za rukovanje iznimkama koji počinje ključnom riječi EXCEPTION. Odjeljak za rukovanje iznimkama mjesto je gdje se hvataju i obrađuju iznimke koje pokreću kod u odjeljku za izvršavanje.

Programski kod 3.2: Primjer anonimnog bloka

```
SET SERVEROUTPUT ON
DECLARE
   l_danas date  := sysdate;
BEGIN
   DBMS_OUTPUT.PUT_LINE(
        'Danas je '||to_char(l_danas,'day'));
EXCEPTION WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE( 'Greška:' || sqlerrm);
END;
```

Programski kod 3.2 prikazuje jednostavan primjer koji deklarira varijablu 'l_danas' te joj dodaje vrijednost današnjega dana i ispisuje poruku na ekran. Ako se slučajno dogodi pogreška, ispisat će se na ekran.

3.3. Iznimke

Iznimka se događa kada PL/SQL mehanizam naiđe na instrukciju koju ne može izvršiti zbog pogreške koja se javlja tijekom izvođenja koda [5]. PL/SQL tretira sve pogreške koje se pojave u anonimnom bloku, proceduri, funkciji kao iznimke. Iznimke mogu imati različite uzroke, kao što su pogreške kodiranja, bugovi, pa čak i hardverski kvarovi. Iznimke će zaustaviti daljnje izvršavanje programa, a da bi se izbjeglo takvo stanje, potrebno ih je uhvatiti i rukovati njima zasebno. Takav proces naziva se „rukovanje iznimkama“, u kojemu programer obrađuje iznimke koje se mogu pojaviti tijekom izvođenja koda. Nije moguće predvidjeti sve potencijalne iznimke. U Oracle bazi postoje tri vrste iznimki:

1. **Unaprijed definirane iznimke** su pogreške koje definira PL/SQL. Jedna od otprilike 20 pogrešaka koje se najčešće pojavljuju u PL/SQL kodu. Takve iznimke imaju jedinstveni naziv i broj pogreške. U programu se mogu izravno upotrijebiti.
2. **Nepredefinirane iznimke** uključuju sve standardne pogreške, moraju se navesti u deklarativnom odjeljku, implicitno se javlja pogreška i može se koristiti rukovatelj iznimkama da se uhvati pogreška.
3. **Korisnički definirane iznimke** su iznimke specifične za određene aplikacije, pogreška je pokrenuta od strane aplikacije. Pogreške se moraju navesti u deklarativnom odjeljku. Razvojni programer izričito navodi iznimku te su vidljive samo u tom potprogramu. Iznimka koja je definirana u specifikaciji paketa javna je iznimka i vidljiva je gdje god je paket dostupan.

3.4. Procedure

Procedure su potprogrami koji se mogu kreirati i spremiti u bazu podataka kao objekti baze podataka. To je jedinica za višekratnu upotrebu koja enkapsulira specifičnu poslovnu logiku aplikacije. Tehnički govoreći, PL/SQL procedura je imenovani blok pohranjen kao objekt sheme u Oracle bazi podataka. Koriste se za poboljšanje performansi aplikacije, modularizaciju aplikacija, provjeru valjanosti podataka, kontrolu pristupa ili za smanjenje mrežnog prometa između klijenata i DBMS poslužitelja. Mogu se pozvati ili uputiti i unutar drugih blokova. Pozivaju se imenom. Učestali kod koji se piše iznova sprema se kao pohranjena procedura koja

se kasnije samo poziva i izvršava. Procedura se može pozivati na temelju vrijednosti parametara koji joj se prosljeđuju. Kada se kreira procedura, mogu se definirati parametri. Postoje tri vrste parametara koji se mogu deklarirati:

1. **IN** – parametar se može referencirati procedurom, vrijednost parametra ne može se prebrisati procedurom, služi samo za čitanje unutar programa i koristi se za unos podataka u potprograme.
2. **OUT** – procedura ne može referencirati parametar, ali procedura može prebrisati vrijednost parametra, koristi se za dobivanje izlaza iz potprograma. To je varijabla za čitanje i pisanje unutar potprograma; njihove vrijednosti mogu se mijenjati unutar potprograma.
3. **IN OUT** – parametar može biti referenciran od strane procedure i vrijednost parametra može biti prebrisana od strane procedure; ovaj parametar koristi se i za unos i za dobivanje izlaza iz potprograma. To je varijabla za čitanje i pisanje unutar potprograma; njihove vrijednosti mogu se mijenjati unutar potprograma.

Programski kod 3.4: Primjer procedure

```
CREATE OR REPLACE PROCEDURE ispisi_studente(
   in_studenti_id NUMBER )
IS
 r_kontakt studenti%ROWTYPE;
BEGIN
 -- dobiti kontakt studenta na temelju ID-a
 SELECT *
 INTO r_kontakt
 FROM studenti
 WHERE studenti_id = p_studenti_id;

 -- Ispis podataka o studentu ( ime, prezime i email)
 dbms_output.put_line( r_kontakt.ime || ' ' ||  r_kontakt.prezime || '<' ||
r_kontakt.email ||'>' );

EXCEPTION
  WHEN OTHERS THEN
```

```
        dbms_output.put_line( 'Greska: ' || SQLERRM );
    END ispisi_studente;
```

Programski kod 3.4 prikazuje primjer jednostavne procedure koja koristi ID studenta kako bi se dobili kontaktni podaci te se ispisali na ekran. Ako se dogodi pogreška, ispisat će se na ekran.

3.5. Funkcije

Funkcije su potprogrami koji se koriste za vraćanje jedne vrijednosti. Nazivaju se još i skup PL/SQL naredbi koje se mogu pozvati imenom. Pohranjene funkcije vrlo su slične procedurama, osim što funkcija vraća vrijednost okolini u kojoj je pozvana. Funkcija će uvijek vratiti neku vrijednost ili pogrešku. Korisničke funkcije mogu se koristiti kao dio SQL izraza. Funkcija se mora deklarirati i definirati prije nego što se pozove. Može se deklarirati i definirati u isto vrijeme ili se može prvo deklarirati i kasnije definirati u istom bloku. Ako se pozove SQL funkcija s argumentom tipa podataka koji nije tip podataka koji očekuje SQL funkcija, tada Oracle pokušava pretvoriti argument u očekivani tip podataka prije izvođenja SQL funkcije. Ako se pozove funkcija s null argumentom, SQL funkcija automatski vraća null vrijednost. Zaglavlje funkcije ima obaveznu klauzulu RETURN, koja specificira tip podataka vraćene vrijednosti. Za razliku od procedure, funkcija obavezno mora imati barem jednu naredbu RETURN u izvršnoj naredbi. Kada se kreira funkcija, mogu se definirati parametri kao i kod procedure. Postoje tri vrste parametara: in, out i in out, sukladno procedurama.

Programski kod 3.5: Primjer funkcije

```
CREATE OR REPLACE FUNCTION oduzimac
  (a in number, b in number)
RETURN number
IS
c number(8);
BEGIN
c := a-b;
return c;
END;
```

Programski kod 3.5 prikazuje primjer jednostavne funkcije nazvane „oduzimac“. Funkcija prima dvije brojčane vrijednosti „a“ i „b“ te vraća brojčanu vrijednost „c“ u kojoj se prethodna dva broja oduzmu.

Programski kod 3.6: Primjer poziva funkcije

```
DECLARE
  c number(2);
BEGIN
  c := oduzimac (10, 5);
  dbms_output.put_line('Oduzimanje je: ' || c);
END;
```

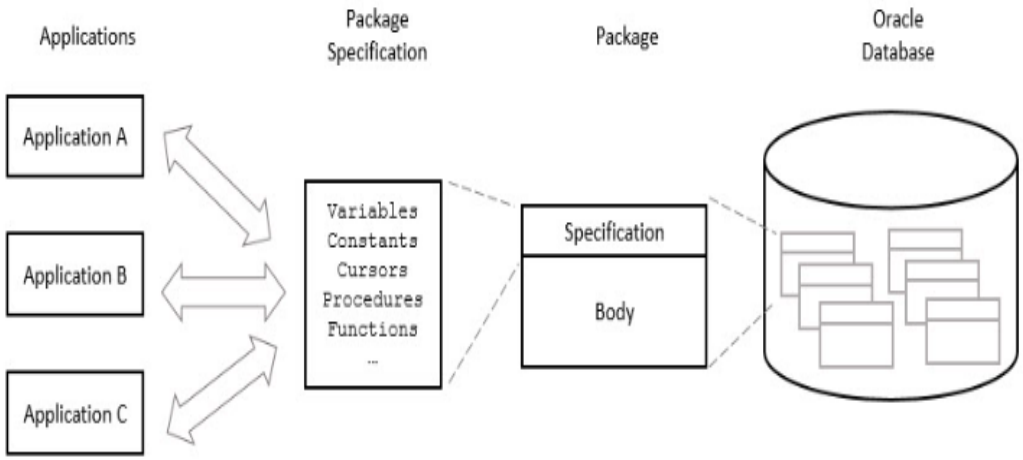
Programski kod 3.6 prikazuje kako se poziva funkcija koja se prethodno kreirala. Funkciji se prosljeđuju vrijednosti varijabli. Dobiveni rezultat je vrijednost „c“ kao rezultat oduzimanja varijabli „a“ i „b“.

3.6. Paketi

Paket je u PL/SQL objekt sheme koji sadrži definicije za grupu povezanih funkcionalnosti. Grupa logički povezanih PL/SQL varijabli, konstanti, kursora, iznimaka, procedura, funkcija i potprograma sastavlja se i pohranjuje u Oracle bazi kao objekt baze koji se može kasnije koristiti [6]. Paket sadrži specifikaciju i tijelo paketa od kojega je sastavljen. Specifikacija paketa je obaveza, dok tijelo paketa može biti obavezno ili izborno, ovisno o specifikaciji paketa. Specifikaciju se zamišlja kao sučelje, a tijelo kao crna kutija. Mogu se otkloniti pogreške, poboljšati ili zamijeniti tijelo bez promjene specifikacije paketa.

1. **Specifikacija** deklarira sve tipove, varijable, konstante, iznimke, kursore i potprograme koji se mogu referencirati izvan paketa. Sadrži javne deklaracije koje su vidljive u pohranjenim procedurama i drugom kodu izvan paketa. Potprogrami se moraju deklarirati na kraju specifikacije nakon svih ostalih stavki. Specifikacija paketa je samostalan element, što znači da može postojati sam bez tijela paketa. Kad god je paket upućen, instanca paketa kreira određenu sesiju, a svi elementi paketa koji su pokrenuti u toj instanci vrijede do kraja sesije.

2. **Tijelo** se sastoji od definicija svih elemenata koji su prisutni u specifikaciji paketa. Tijelo sadrži detalje implementacije i privatne deklaracije, koje su skrivene od koda izvan paketa. Nakon deklarativnog dijela tijela paketa slijedi opcijski dio za inicijalizaciju, koji sadrži izjave koje inicijaliziraju varijable paketa i izvršavaju bilo koje druge jednokratne korake postavljanja. To je pouzdan objekt i ovisi o specifikaciji paketa. Stanje tijela postaje 'Nevažeće' kad god nisu sukladni specifikacija i tijelo paketa, onda se ponovo treba kompajlirati paket.



Slika 3.6: Shematski prikaz paketa [6]

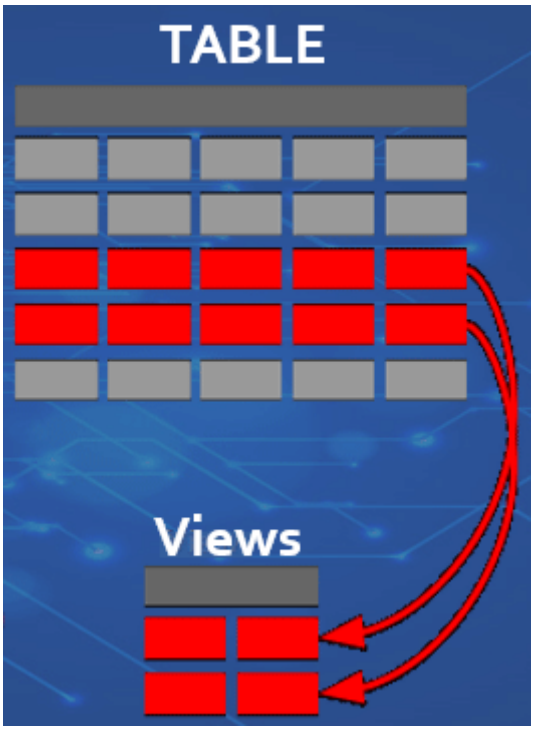
3.7. Pogledi

Pogledi u Oracle bazi potvrđeni su i imenovani SQL upiti pohranjeni u rječniku baze podataka [7]. Pogledi su jednostavno pohranjeni upiti koji se mogu izvršiti kada je potrebno, ali oni ne pohranjuju podatke. Korisno je zamisliti prikaz kao virtualnu tablicu ili kao proces mapiranja podataka iz jedne ili više tablica. Pogled također ima retke i stupce kakvi su u stvarnoj tablici u bazi podataka. Pogled se može stvoriti odabirom polja iz jedne ili više tablica prisutnih u bazi podataka. Pogled može sadržavati sve retke tablice ili određene retke na temelju određenog uvjeta. Pogledima se može postavljati upit, ažurirati, umetati i brisati iz njih uz određena ograničenja. Sve operacije koje se izvode na pogledu zapravo utječu na osnovne tablice pogleda. Pogledi mogu pružiti dodatnu razinu sigurnosti ograničavanjem pristupa unaprijed određenom skupu redaka i stupaca tablice. Također mogu sakriti složenost podataka

i pohraniti složene upite. Pogledi omogućuju korisnicima odabir informacija iz više tablica bez potrebe da korisnici zapravo znaju kako izvesti spajanje. Upravljanje sadržaja pogleda postiže se pomoću naredbi INSERT, UPDATE, DELETE i MERGE.

Postoje dvije vrste pogleda:

1. **Statički prikaz rječnika podataka –** Pogledi rječnika podataka nazivaju se statičnima jer se rijetko mijenjaju, samo kada se napravi promjena u rječniku podataka. Primjeri promjena rječnika podataka uključuju stvaranje nove tablice ili dodjeljivanje privilegija korisniku. Tablice rječnika podataka nisu izravno dostupne, ali informacijama u njima može se pristupiti putem prikaza rječnika podataka. Za popis prikaza rječnika podataka koji su dostupni mora se postaviti upit prema rječniku. Pogled „All_“ prikazuje sve informacije dostupne trenutnom korisniku, uključujući informacije iz sheme trenutnog korisnika, kao i informacije iz objekta u drugim shemama, ako trenutni korisnik ima pristup tim objektima putem uloga. Pogled „User_“ prikazuje sve informacije iz sheme trenutnog korisnika. Za njega nisu potrebne posebne uloge za postavljanje upita.
2. **Dinamički prikazi izvedbe** – Dinamički prikazi prate tekuću aktivnost baze podataka, uglavnom pružaju informacije o Oracle instanci, kao informacije koje instanca održava o bazi podataka. Pogledi u ovoj kategoriji smatraju se dinamičkima jer se općenito njihov sadržaj mijenja ovisno o izvedbi instance. Ovi pogledi pružaju podatke o unutarnjim strukturama diska i memorijskim strukturama. Sadržaji ovih pogleda reprezentativni su za ukupno radno opterećenje instance ili klastera. Dostupni su samo administratorima. Nazivi dinamičkih prikaza izvedbe počinju znakovima 'V$'.



Slika 3.7: Prikaz pogleda [7]

Programski kod 3.7: Primjer kreiranja pogleda

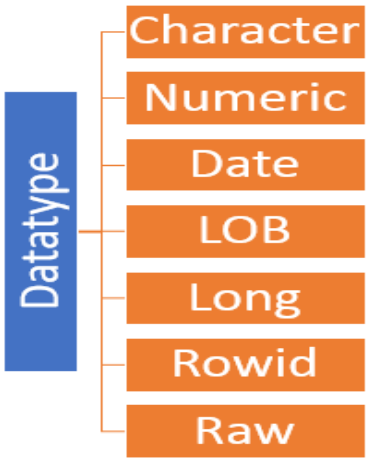
```
CREATE VIEW studenti_bjelovar AS
SELECT ime, prezime
FROM studenti
WHERE grad = 'BJELOVAR';
```

Programski kod 3.7 prikazuje primjer kreiranja pogleda koji će prikazivati samo studente iz Bjelovara.

3.8. Tipovi

Svaka vrijednost kojom manipulira Oracle baza podataka ima tip podatka. Tip podataka vrijednosti pridružuje fiksni skup svojstava s vrijednošću. Korištenjem ovih svojstava Oracle tretira vrijednost jedne vrste podataka drugačije od vrijednosti druge [8]. Na primjer, mogu se dodati vrijednosti tipa podataka NUMBER, ali ne i vrijednosti tipa podataka CHAR. Oracle nudi niz ugrađenih tipova podataka, kao i nekoliko kategorija za korisnički definirane tipove koji se mogu koristiti kao tipovi podataka. Tip objekta je vrsta tipa podatka. Može se koristiti na isti način na koji se koriste standardni tipovi podataka kao što su NUMBER ili VARCHAR2.

Mogu se navesti, na primjer, tip objekta kao tip podatka stupca u relacijskoj tablici i mogu se deklarirati varijable tipa objekta. Oracle implementira objekte korištenjem TYPE-ova, definiranih na sličan način kao i paketi. Za razliku od paketa, gdje je instanca paketa ograničena na trenutnu sesiju, instanca tipa objekta može pohraniti u bazu podataka za kasniju upotrebu. Definicija tipa sadrži popis svojstava odvojenih zarezima, definiranih na isti način kao varijable paketa i funkcija ili procedura članova. Ako tip sadrži članske funkcije ili procedure, proceduralni rad za te elemente definiran je u tijelu tipa (engl. *Type body*). Tip objekta može predstavljati bilo koji entitet iz stvarnoga svijeta.



Slika 3.8: Prikaz vrsta tipova [8]

Programski kod 3.8: Primjer kreiranja tipa (engl. type).

```
CREATE OR REPLACE TYPE student_informacije AS OBJECT (
  ime              VARCHAR2(100),
  prezime          VARCHAR2(100),
  godine           NUMBER,
  smjer            VARCHAR2(100),
  datum            DATE,
  status           CHAR(2),
  drzava           VARCHAR2(100),
  grad             VARCHAR2(100),
  ulica            VARCHAR2(100),
  postanski_broj   VARCHAR2(10)  );
```

Programski kod 3.8 prikazuje kreiranje jednostavnog tipa (engl. *type*) koji ima u sebi više različitih tipova podataka.

3.9. Dinamički SQL

Dinamički SQL metodologija je programiranja koja omogućuje dinamičku izgradnju i izvođenje SQL naredbi tijekom izvođenja. Uglavnom se koristi za pisanje programa opće namjene i fleksibilnih programa gdje će se SQL naredbe kreirati i izvršavati tijekom izvođenja na temelju zahtjeva. Dinamički SQL sporiji je od statičkog pa ga ne bi trebalo koristiti osim ako je to apsolutno neophodno. Dinamika je suprotna od statičnosti. Statički SQL odnosi se na SQL naredbe koje su potpuno specificirane ili fiksne u trenutku kompajliranja koda koji sadrži tu naredbu. Dinamički PL/SQL odnosi se na cijele PL/SQL blokove koda koji se dinamički konstruiraju, zatim kompajliraju i izvode. PL/SQL nudi dva načina pisanja dinamičkog SQL-a:

1. **NDS** – izvorni dinamički SQL lakši je način za pisanje dinamičkog SQL-a, obrađuje većinu dinamičkih SQL naredbi s naredbom EXECUTE IMMEDIATE. Naredbe služe za stvaranje i izvršavanje SQL-a tijekom izvođenja. Klauzula INTO nije obavezna i koristi se samo ako dinamički SQL sadrži naredbu koja dohvaća vrijednosti. Tip varijable treba odgovarati tipu varijable iz SELECT izjave. Klauzula USING je izborna i koristi se samo ako dinamički SQL sadrži bilo koju varijablu vezanja. Ako se koristi ovaj način, tip podataka i broj varijabli koji će se koristiti u vremenu izvođenja moraju biti poznati prije. Ima bolje performanse i manju složenost u usporedbi s DBMS_SQL-om. Redovi koji su rezultat upita mogu se izravno dohvatiti u PL/SQL zapise pomoći izvornog dinamičkog SQL-a, što nije moguće s DBMS_SQL paketom.
2. **DBMS_SQL** je tradicionalni oblik dinamičkog SQL-a u Oracleu. Koristi ugrađeni paket s velikim API-jem za analizu i izvršavanje dinamičkih SQL potreba koji omogućuje rad s dinamičkim SQL-om. Pruža sučelje za korištenje dinamičkog SQL-a za izvršavanje izraza za manipulaciju podacima (DML) i jezika za definiranje podataka (DDL), izvršavanje PL/SQL anonimnih blokova i povezivanje pohranjenih procedura i funkcija. Postoje stvari koje DBMS_SQL može učiniti, a koje se ne mogu učiniti ni na koji drugi način. Ako se koristi ovaj način za pisanje dinamičkog SQL-a, moramo se pridržavati procesa kreiranja i izvođenja koji sadrži sljedeće:

OPEN CURSOR – Dinamički SQL izvršit će se na isti način kao kursor; PARSE SQL – Raščlanjivanje dinamičkog SQL-a; BIND VARIJABLE – Dodjeljivanje vrijednosti za varijable ako postoje; DEFINE COLUMN – Definiranje stupca korištenjem njihovih relativnih položaja u naredbi odabira; FETCH – Dohvaćanje izvršenih vrijednosti; CLOSE – Nakon što se dohvate rezultati, kursor treba zatvoriti.

Programski kod 3.9: Primjer dinamičkog SQL-a.

```
DECLARE
lv_sql            VARCHAR2(500);
lv_emp_ime      VARCHAR2(50);
ln_emp_broj     NUMBER;
ln_student        NUMBER;
BEGIN

lv_sql:= 'SELECT emp_ime,emp_broj,student FROM emp WHERE emp_no
=:empmo';
EXECUTE IMMEDIATE lv_sql INTO lv_emp_ime,ln_emp_broj,ln_student
USING 1001;
Dbms_output.put_line('Ime studenta:' ||lv_emp_ime);
Dbms_output.put_line('Broj studenta: '||ln_emp_broj);
Dbms_output.put_line('Student ID:' ||ln_student);
END;
```

4. JSON

JSON je popularan format tekstualnih podataka koji se koristi za razmjenu podataka u modernim internetskim i mobilnim aplikacijama. JSON funkcije prvi su put predstavljene u SQL serveru 2016. godine. Format JSON identičan je kodu za stvaranje JavaScript objekata. Sintaksa je izvedena iz JavaScript sintakse notacije objekta, ali format je samo tekst. Kod za čitanje i generiranje JSON podataka može se napisati u bilo kojem programskom jeziku i podržava sve razvojne okvire i biblioteke. JSON se također koristi za pohranjivanje nestrukturiranih podataka u log datotekama ili NoSQL bazama podataka kao što je Microsoft Azure Cosmos DB. Može se kombinirati klasične relacijske stupce sa stupcima koji sadrže dokumente formatirane kao JSON tekst u istoj tablici, analizirati i uvesti JSON dokumente u relacijskim strukturama ili formatirati relacijske podatke u JSON tekst. Mnoge REST web-usluge vraćaju rezultate koji su formatirani kao JSON. Također je glavni format za razmjenu podataka između web-stranica i web-poslužitelja pomoću AJAX poziva. Današnje relacijske baze podataka isporučuju se s podrškom za pohranjivanje i postavljanje upita JSON podacima. JSON je generički format podataka s minimalnim brojem tipova vrijednosti poput nizova, brojeva, booleova, popisa, objekata. U JSON-u podaci su predstavljeni u parovima ključ-vrijednost, a vitičaste zagrade drže objekte. Iza svakog imena slijedi dvotočka. Zarez se koristi za odvajanje parova ključ-vrijednost. Uglate zagrade koriste se za držanje nizova gdje je svaka vrijednost odvojena zarezom. Naziv JSON datoteke ima ekstenziju '.json'. Primjer jednog jednostavnog JSON-a:

Programski kod 4.0: Primjer JSON sintakse

```
{
   "id": "298",
   "ime": "Domagoj",
   "godine": 23,
   "grad:": "Bjelovar"
   "spol": "muško",

}
```

Programski kod 4.0 prikazuje primjer jednostavnog oblika JSON objekta koji u sebi sadrži različite vrijednosti popust string-a i number-a.

4.1. JSON u Oracle bazi podataka

Oracle baza podataka u potpunosti podržava razvoj pomoću JSON podatkovnog modela. Time se omogućuje hibridni razvojni pristup. JSON format osmišljen je za pružanje najboljeg uklapanja između svijeta relacijske pohrane i postavljanja upita s JSON podacima. JSON podaci često su pohranjeni u NoSQL bazama podataka oni omogućuju pohranu i dohvaćanje podataka koji se ne temelje ni na kakvoj shemi, ali ni ne nude rigorozne modele dosljednosti relacijskih baza podataka. JSON podaci mogu se pridružiti relacijskim podacima čineći ih dostupnima za relacijske procese i alate. Oracle pohranjuje, upravlja i indeksira JSON dokumente. Također pruža sofisticirano SQL postavljanje upita i izvješća preko JSON dokumenata. JSON podaci u bazi su tekstualni, ali tekst se može pohraniti pomoći tipa podataka CLOB, VARCHAR2 ili BLOB. Preporučeno je koristiti BLOB kada je moguće. Konkretno, time se izbjegava potreba za pretvorbom skupa znakova. Oracle ne postavlja ograničenja na tablice koje se mogu koristiti za pohranu JSON dokumenata. Stupac koji sadrži JSON dokumente ne treba se koegzistirati s bilo kojom drugom vrstom podataka, a tablica također može imati više stupaca koji sadrže JSON. Kada se Oracle baza koristi kao pohrana JSON podataka, tablice koje sadrže JSON stupce obično imaju i nekoliko stupaca koji ne sadrže JSON za održavanje, oni obično prate metapodatke o dokumentima. Također, ako se JSON koristi za dodavanje fleksibilnosti primarno relacijskoj aplikaciji, tada neke tablice vjerojatno imaju stupac za JSON dokumente koje koriste za upravljanje podacima aplikacije koje se ne preslikavaju izravno na relacijski model. JSON podacima pohranjenima u bazi podataka može se pristupiti na isti način na koji se pristupa drugim podacima baze podataka, uključujući korištenje OCI, Microsoft .NET framework i JDBC.

4.2. JSON i XML u Oracle bazi

JSON i XML podaci mogu se koristiti u Oracle bazi podataka na slične načine. Za razliku od relacijskih podataka, oni se mogu pohraniti, indeksirati i postavljati upite bez potrebe za shemom koja definira podatke. Oracle baza podataka izvorno podržava JSON sa značajkama relacijske baze podataka, uključujući transakcije, indeksiranje, deklarativno postavljanje upita i prikaze. XML je jezik za određivanje prilagođenih označnih jezika. XML definira skup pravila za kodiranje dokumenata u format koji je čitljiv i za ljude i za strojeve. Dizajn XML-a usmjeren

je na jednostavnost, općenitost i upotrebljivost na internetu. Dizajniran je za prijenos podataka, a ne za prikaz podataka. Sa svojom strogom semantikom, XML je definirao standard za potvrđivanje integriteta podataka XML dokumenata, bilo kojeg XML pod jezika. Jezik se široko koristi za predstavljanje proizvoljnih struktura podataka poput onih koje se koriste u web-uslugama. XML razlikuje velika i mala slova, također omogućava definiranje elemenata označavanja i generiranje prilagođenog jezika označavanja. Element je osnovna jedinica u XML jeziku. Naziv XML datoteke ima ekstenziju '.xml'. I JSON i XML mogu se koristiti za primanje podataka s web-poslužitelja, a također se mogu dohvatiti HTTP zahtjevom. Primjer istog koda u JSON i XML formatu:

Programski kod 4.2: Primjer JSON-a

```
{"studenti":[
  { "Ime":"Domagoj", "Prezime":"Petrec" },
  { "Ime":"Pero", "Prezime":"Perić" },
]}
```

Programski kod 4.3: Primjer XML-a

```
<studenti>
   <student>
     <Ime>Domagoj</Ime> <Prezime>Petrec</Prezime>
   </student>
   <student>
     <Ime>Pero</Ime> <Prezime>Perić</Prezime>
   </student>
</studenti>
```

U programskim kodovima 4.2 i 4.3 prikazani su isti primjeri zapisani u drugačijim formatima JSON-a i XML-a.

JSON	XML
Način za predstavljanje objekata.	Koristi strukturu za predstavljanje podatkovnih stavki.
Ne pruža nikakvu podršku za prostore imena.	Podržava podršku za prostore imena.
Podržava niz.	Ne podržava niz.
Datoteke su vrlo lake za čitanje.	Datoteke je razmjerno teško čitati i tumačiti.
Ne koristi završnu oznaku.	Ima početnu i završnu oznaku.
Manje je osiguran.	Sigurniji je od JSON.
Ne podržava komentare.	Podržava komentare.
Podržava samo UTF-8 kodiranje.	Podržava različita kodiranja.
JSON objekti imaju vrstu.	XML podaci su bez tipa.
JSON tipovi: riječ, broj, niz, boolean.	Svi XML podaci moraju biti nizovi.
Podaci su lako dostupni kao JSON objekti.	XML podatke treba analizirati.
Podržava većinu preglednika.	Raščlanjivanje XML-a u različitim preglednicima može biti nezgodno.
Nema mogućnosti prikaza.	Nudi mogućnost prikaza podataka jer je označni jezik.
Izvorna podrška za objekt.	Objekt mora biti izražen u konvencijama.
Podržavaju ga mnogi AJAX alati.	Ne podržavaju ga AJAX alati u potpunosti.
Potpuno automatizirani način deserijalizacije / serijalizacije JavaScripta.	Mora se napisati JavaScript kod za serijalizaciju / deserijalizaciju iz XML-a.
Dohvaćanje vrijednosti je lako.	Dohvaćanje vrijednosti je teško.
JSON podržava samo tekstualne i brojčane vrste podataka.	XLM podržava različite tipove podataka kao što su brojevi, tekst, slike, dijagrami, grafikoni itd.

Tablica 4.2. Usporedba JSON-a i XML-a

4.3. JSON Schema

JSON Schema je vrsta JSON medija, tj. specifikacija za format za definiranje strukture JSON podataka [9] . Napisan je u okviru IETF nacrta koji je istekao 2011. godine. JSON Schema pruža ugovor o tome koji su JSON podaci potrebni za određenu aplikaciju i kako s njima komunicirati [10]. Ova specifikacija definira temeljnu terminologiju i mehanizme JSON Scheme, uključujući ukazivanje na drugu JSON Schemu referencom. Određivanje dijalekta koji se koristi, određivanje zahtjeva rječnika dijalekta i definiranje očekivanog izlaza. JSON Schema namijenjena je definiranju provjere valjanosti, dokumentacije, navigacije hipervezom i kontrole interakcije JSON podataka. Primjena takvih standarda za JSON dokument omogućuje provođenje dosljednosti na sličnim JSON podacima. JSON Schema opisuje postojeći format podataka. Jasan je ljudima i strojno je čitljiva dokumentacija. Potpuno strukturalna validacija korisna je za automatizirano testiranje. Trenutačno najusklađeniji validator za JSON Scheme koji je dostupan jest JSV. API Gateway može provjeriti jesu li JSON poruke usklađene s formatom koji očekuje web-usluga provjerom valjanosti zahtjeva prema određenoj JSON Schemi. JSON Schema precizno definira stavke koje čine instancu dolazne JSON poruke, također navodi njihove vrstepodataka kako bi se osiguralo da je odgovarajućim podacima dopušten pristup web-usluzi. Svojstva na objektu definirana su pomoću ključne riječi svojstva. Vrijednost je objekt gdje je svaki ključ naziv svojstva, a svaka vrijednost je shema koja se koristi za provjeru valjanosti toga svojstva.

Programski kod 4.3: Primjer JSON Scheme

```
(q'{
        "call": {
                "procedura": "p_save",
                "table": "PROIZVODI"
        },
        "kolone": [{
                        "NAZIV": "Smeđa jaja",
                        "TIP": "Mliječni proizvodi",
                        "OPIS": "Sirova organska smeđa jaja u košari",
                        "VISINA": 600,
```

```
            "SIRINA": 400,
            "CIJENA": 28.1,
            "OCJENA": 4
        }, {
            "NAZIV": "Slatke svježe jagode",
            "TIP": "Voće",
            "OPIS": "Slatke svježe jagode na drvenom stolu",
            "VISINA": 450,
            "SIRINA": 299,
            "CIJENA": 29.45,
            "OCJENA": 4
        }, {
            "NAZIV": "Šparoge",
            "TIP": "Povrće",
            "OPIS": "Šparoge sa šunkom na drvenom stolu",
            "VISINA": 450,
            "SIRINA": 299,
            "CIJENA": 18.95,
            "OCJENA": 3
        }
    ]

}');
```

4.4. JSON i PL/SQL objekti

PL/SQL objekti omogućuju detaljnu programsku konstrukciju i manipulaciju JSON podacima u memoriji. Podaci se mogu pregledati, modificirati i serijalizirati natrag u tekstualne JSON podatke. Instance ovih vrsta nisu postojane. Umjesto toga, prvo se pročitaju podaci iz tablice u odgovarajuće instance tipova objekata ili se konstruira instanca kroz analizu i druge operacije izravno u PL/SQL kodu. Po potrebi se manipulira instancama mijenjanjem, uklanjanjem ili dodavanjem vrijednosti. Konačno ih se pretvara ili serijalizira u VARCHAR2 ili podatke velikog objekta CLOB. Nakon toga serijalizirani podaci pohranjuju se natrag u

tablicu baze podataka ili se prosljeđuju korisniku tih podataka, kao što je web-aplikacija temeljena na JavaScriptu. Ne mora se brinuti o oslobađanju memorije povezane s tim instancama. Baza podataka automatski će obavljati zadatke skupljanja smeća. Tipovi objekata nazivaju se i „apstraktni tipovi podataka“. Ovi tipovi JSON objekata pružaju hijerarhijski programski prikaz JSON podataka koji su pohranjeni u bazi podataka u memoriji. Glavni tipovi PL/SQL JSON objekata su:

JSON_ELEMENT_T – Supertip koji proširuju neki drugi tipovi objekata, ova vrsta se ne koristi izravno. Za izradu instance koristi se funkcija parse. Ne može se stvoriti prazna instanca. Za pretvaranje treba se izvršiti eksplicitno pretvaranje koristeći TREAT AS. Funkcija kao ulaz uzima podatke VARCHAR2, BLOB ili CLOB i vraća instancu.

JSON_OBJECT_T – Objekt koji predstavlja JSON niz. Ima funkciju konstruktora istog imena kao i tip, koja se može koristiti za konstruiranje instance tipa: prazan prikaz JSON objekta ili polja. Ispuniti niz prema potrebi dodajući članove objekta ili elemente niza koje su predstavljene instancama tipa PL/SQL objekta.

JSON_ARRAY_T – Skalarna vrijednost povezana s ključem kao što je niz, broj, boolean ili null. Tip objekta nudi unaprijed definiranu funkciju konstruktora za instanciranje novih instanci toga tipa, statičkih metoda i metoda članova. Indeksiranje JSON nizova počinje od 0.

JSON_SCALAR_T – Niz naziva ključeva, obično vraćen metodom GET_KEYS. Hvata jednu skalarnu vrijednost. Npr. Niz ili broj 1. Ova vrsta nema nikakve funkcije ili procedure osim onih naslijeđenih od JSON_ELEMENT_T. Ne može se izravno stvoriti instanca ove vrste.

Programski kod 4.4: Primjer JSON objekata

```
DECLARE
  jot              JSON_OBJECT_T;
  jat              JSON_ARRAY_T;
  keys             JSON_KEY_LIST;
  keys_string      VARCHAR2(100);
BEGIN
  jat := new     JSON_ARRAY_T;
  jot := JSON_OBJECT_T.parse('{ "Ime firme": "Codex",
                                "Vrsta posla": "Programiranje",
                                "Broj zaposlenih": "15" }');
  keys := jot.get_keys;
```

```
  FOR i IN 1..keys.COUNT LOOP
    jat.append(keys(i));
  END LOOP;
  keys_string := jat.to_string;
  DBMS_OUTPUT.put_line(keys_string);
END;
```

Programski kod 4.4. prikazuje JSON objekte za dohvaćanje podataka, spremanje u varijablu i ispisivanje na ekran.

5. PORUKE PROGRAMA

Ovaj program koristi tri različita paketa za svoju realizaciju. Jedan od njih jest paket poruke koji služi kako bi krajnji korisnik shvatio na što lakši način u čemu je problem i što je pogrešno u programu.

5.1. Tablica poruke

Primjer kreiranja tablice poruke

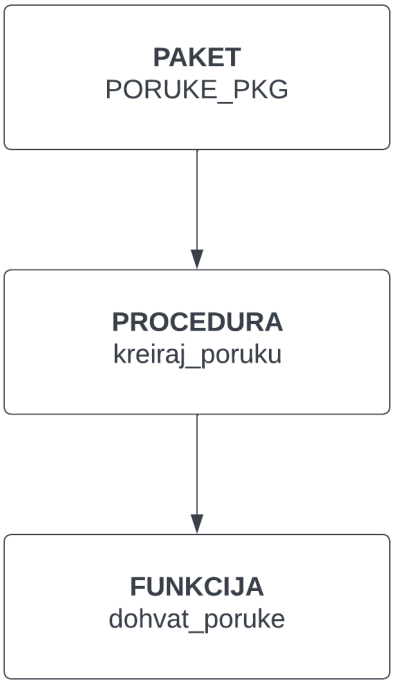
Programski kod 5.1: Kreiranje tablice poruke

```
CREATE TABLE PORUKE (
JEZIK                NUMBER(1)          NOT NULL,
BROJ_PORUKE          NUMBER(6)          NOT NULL,
TEKST_PORUKE         VARCHAR2(1000)     NOT NULL,
DATUM_AZURIRANJA     TIMESTAMP(6)       NOT NULL,
AZURIRAO             VARCHAR2(3)        NOT NULL,
DATUM_KREIRANJA      TIMESTAMP(6)       NOT NULL,
KREIRAO              VARCHAR2(3)        NOT NULL);
```

Programski kod 5.1 prikazuje kreiranje tablice poruke. Kreirana je tablica s automatskom sekvencom koju kasnije nije potrebno navoditi kod unosa (engl. *insert*). Također su dodani komentari na tablicu i kolone kako bi korisniku bilo lakše shvatiti što znače kolone i kako program radi. Ograničenje (engl. *constraint*) je stavljeno na primarni ključ (engl. *primary key*), kao i jedinstveno ograničenje (engl. *unique constraint*) na broj_poruke i jezik. Check constraint je na jeziku kako bi se koristio samo hrvatski i engleski jezik. Paket još sadrži i okidače (engl. *trigger*) za automatsko popunjavanje određenih kolona.

5.2. Paket poruke

Paket poruke se sastoji od procedure i funkcije za kreiranje te pozivanje poruka.



Slika 5.2: Grafički slijed paketa

Slika 5.2 prikazuje paket koji se sastoji od tipa (engl. *type*) te procedure koja služi za kreiranje poruke i funkcije za dohvat teksta poruke. Procedura kreiraj_poruku služi samo za kreiranje poruka. Za pozivanje procedure potrebno je navesti jezik, broj i tekst u tipu (engl. *type*). Pri kreiranju procedure provjeravaju se ulazni parametri. Ako su pogrešni, aplikacija prikaže pogrešku na ekran, ako je sve u redu program nakon provjere napravi unos (engl. *insert*) poruke. Funkcija dohvat_poruke služi za dohvat teksta poruke, a za pozivanje funkcije potrebno je proslijediti broj i jezik poruke. U slučaju pogreške funkcija će vratiti null vrijednost.

5.3. Primjer poziva poruka

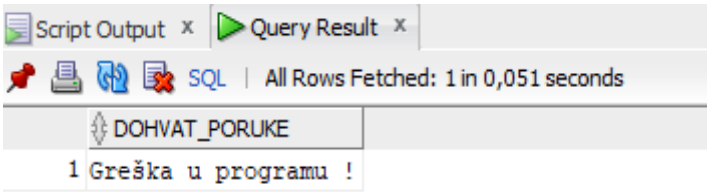
Primjer poziva poruka u anonimnom bloku.

Programski kod 5.4: Primjer poziva poruka

```
DECLARE
        l_typ_poruke poruke_pkg.t_poruka;
BEGIN

   l_typ_poruke.jezik  := 1;                    -- 1 - HR, 2 - ENG
   l_typ_poruke.broj   := 30;
   l_typ_poruke.tekst  := 'Greška u programu !';

   poruke_pkg.kreiraj_poruku(l_typ_poruke);        -- pozivanje procedure
   dbms_output.put_line (l_typ_poruke.err);         --ispis greške
END;
SELECT poruke_pkg.dohvat_poruke(30,1)  AS DOHVAT_PORUKE FROM DUAL;
```



Slika 5.5: Prikaz rezultata

Programski kod 5.4 prikazuje primjer poziva poruke. U declare dijelu napravljena je deklaracija tipa (engl. *type*). U begin odjeljku potrebno je navesti polja za kreiranje poput jezika, broja poruke i teksta. Nakon toga poziva se procedura, a u slučaju pogreške ispisat će se na ekran. Ako je prazno, znači da je prošlo. Na kraju se dohvati poruka po broju i jeziku, što prikazuje slika 5.5.

6. PAKET I TABLICA VUB_TABS_COLS

Tablica VUB_TABS_COLS napravljena je po uzoru na sistemsku tablicu ALL_TAB_COLS koja se puni kada se kreira neka nova tablica. Opis svih polja nalazit će se u tablici.

6.1. Tablica VUB_TAB_COLS

Primjer kreiranja tablice VUB_TAB_COLS.

Programski kod 6.1: Primjer kreiranja tablice

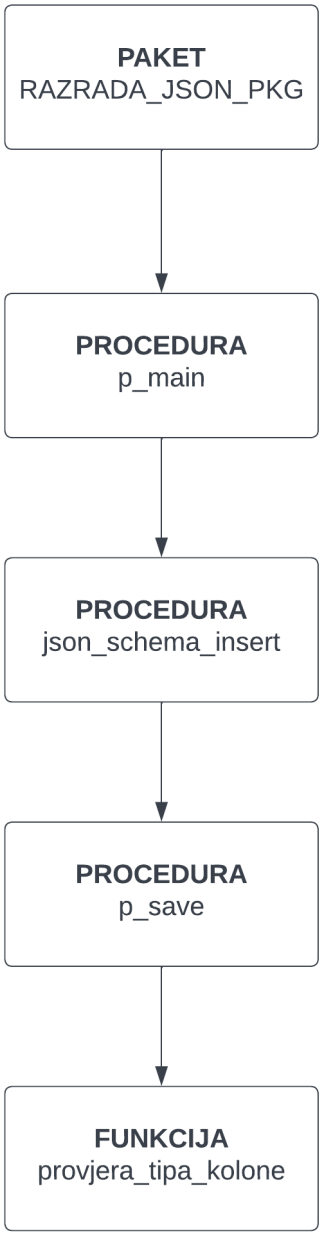
```
CREATE TABLE VUB_TAB_COLS (

TABLE_NAME          VARCHAR2(128)    NOT NULL,
COLUMN_NAME         VARCHAR2(128)    NOT NULL,
DATA_TYPE           VARCHAR2(128)    NOT NULL,
DATA_LENGTH         NUMBER           NOT NULL,
NULLABLE            VARCHAR2(1)      NOT NULL,
DATUM_AZURIRANJA    TIMESTAMP(6)     NOT NULL,
AZURIRAO            VARCHAR2(3)      NOT NULL,
DATUM_KREIRANJA     TIMESTAMP(6)     NOT NULL,
KREIRAO             VARCHAR2(3)      NOT NULL);
```

Programski kod 6.1 prikazuje primjer kreiranja tablice vub_tabs_cols po uzoru na sistemsku tablicu all_tabs_cols. Tablica je indeksirana i brža za pretragu. Sadrži automatsku sekvencu te prilikom unosa (engl .*insert*) nije potrebno navoditi istu. Dodani su komentari kako bi korisniku bilo lakše shvatiti poantu tablice. Ograničenje (engl. *constraint*) je nadodano na primarni ključ (engl. *Primary key*), a i jedinstveno ograničenje (engl. *unique constraint)* dodan je na table_name i column_name. Indeks je stavljen na datum_azuriranja da se brže i lakše može tražiti po datumu ažuriranja te okidači (engl. *triggeri*) za automatsko popunjavanje ostalih kolona.

6.2. Paket VUB_TABS_COLS_BATCH

Paket VUB_TABS_COLS_BATCH služi za izjednačavanje sadržaja između dviju tablica ako je bilo izmjena nad njima da budu kompatibilne i jednake.



Slika 6.2: Grafički slijed paketa

Slika 6.2 prikazuje grafički slijed paketa VUB_TABS_COLS_BATCH koji sadrži tip (engl. *type*) te proceduru koja služi za izjednačavanje vub_tab_cols i all_tab_cols. Body paketa vub_tab_cols_batch koji se sastoji od procedure inicijalna_kontrola koja služi ako postoji različita kolona između tablica vub_tab_cols i all_tabs_cols ako je rađen rename kolone i da je izbriše ako postoji, te procedure main koja puni record, a zatim se radi kontrola. Ako je u redu, napravi se umetanje (engl. *insert*), a ako se dogodi pogreška, prikaže se na ekranu i izađe iz programa. Ako program naiđe na razlike u slogovima, napravit će ažuriranje (engl. *update*).

6.3. Primjer poziva

Primjer poziva tipa (engl. *type*).

Programski kod 6.3: Primjer poziva

```
DECLARE
        tajp vub_tab_cols_batch.ERR ;
BEGIN

      vub_tab_cols_batch.main(tajp);

     dbms_output.put_line (tajp.O_ERRMSG);

END;
```

U programskom kodu 6.3 prikazan je primjer poziva u kojem je u declare odjeljku napravljeno deklariranje tipa (engl. *type*). U begin odjeljku poziva se main procedura. Ako se dogodi pogreška, pri pozivu procedure prikazat će se na ekranu.

7. RAZRADA JSON-A I JSON SCHEMA

7.1. Tablica JSON_SCHEMA

Kreiranje tablice JSON_SCHEMA.

Programski kod 7.1: Primjer kreiranja tablice JSON_SCHEMA

```
CREATE TABLE JSON_SCHEMA (
ULAZ                CLOB          NOT NULL,
DATUM_AZURIRANJA    TIMESTAMP(6)  NOT NULL,
AZURIRAO            VARCHAR2(3)   NOT NULL,
DATUM_KREIRANJA     TIMESTAMP(6)  NOT NULL,
KREIRAO             VARCHAR2(3)   NOT NULL);
```

U programskom kodu 7.1 prikazan je kod za kreiranje tablice JSON_SCHEMA koja sadrži automatsku sekvencu, komentare na tablici kako bi korisnik shvatio bolje tablicu. Ograničenje (engl. *constraint*) je dodano na primarni ključ (engl. *primary key*) te okidači (engl. *trigger*) za automatsko popunjavanje ostalih polja u tablici. Tablica služi za spremanje ulaznog podatka JSON-a.

7.2. Tablica PROIZVODI

Kreiranje tablice PROIZVODI.

Programski kod 7.2: Primjer kreiranja tablice PROIZVODI

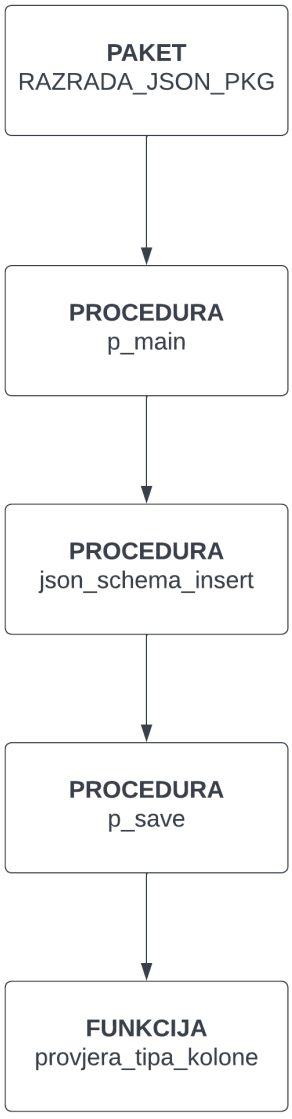
```
CREATE TABLE PROIZVODI (
NAZIV       VARCHAR2(1000)  NOT NULL,
TIP         VARCHAR2(1000)  NOT NULL,
OPIS        VARCHAR2(1000)  NOT NULL,
VISINA      NUMBER(8,2)     NOT NULL,
SIRINA      NUMBER(8,2)     NOT NULL,
```

```
CIJENA          NUMBER(8,2)        NOT NULL,
OCJENA          NUMBER(6)          NOT NULL);
```

U programskom kodu 7.2 prikazan je kod za kreiranje tablice PROIZVODI koja sadrži automatski ID i ograničenje (engl. *constraint*) je dodano na ID u kojoj će se spremati završni JSON.

7.3. Paket RAZADA_JSON_PKG

Glavni dio rada u kojem se nalazi sva logika programa.



Slika 7.3: Grafički slijed paketa

Slika 7.3 prikazuje grafički slijed paketa koji sadrži proceduru p_main te ima ulazni i izlazni parametar CLOB. U body paketa prikazano je kako program radi. Poziva se MAIN procedura koja je prethodno prošla KONTROLE i umetnula je JSON u tablicu JSON_SCHEMA. Nakon toga ulazni parametar p_in sprema se kao objekt radi lakšeg manipuliranja podacima. Zatim se dohvaća JSON_VALUE i dohvaća se podatak o kojoj se akciji radi. U ovom slučaju postoji case za dinamičko spremanje procedure p_save. Ako se ne nađe nijedan parametar, ispisat će se pogreška na ekran.

Procedura p_save na početku premjesti cijeli objekt u varijable da ne radi s izvornim objektom. Zatim se pokupi naziv tablice iz JSON-a i počne graditi dinamički unos (engl. *insert*). Nakon toga uhvati se array podataka iz JSON-a i počne graditi l_kolone od prvog dijela te se dobiva nazive kolona, tj. (engl. *keys)*. Nakon što se dohvate svi nazivi kolona za unos. (engl. *insert*) uzima se svaki tip od naziva i gleda se koje je vrste, npr. number ili string. S tim vrijednostima napuni se „virtualna tablica“ g_kolone koja sadrži imena kolona i tip kolone.

Nakon toga kreće se na dinamički unos (engl. *insert*) vrijednosti (engl. *values*), koji prolazi kroz ugniježđeni loop i dohvaća vrijednosti. Nakon što se dohvate sve vrijednosti stavlja se automatski u anonimni blok i izvršava se te komita. U slučaju pogreške dogodit će se rollback i ispisat će se pogreška na ekran.

7.4. Unos PROIZVODA

Unos proizvoda u tablicu JSON_SCHEMA.

Programski kod 7.4: Primjer unosa proizvoda

```
INSERT INTO JSON_SCHEMA (ulaz)
VALUES
(q'{
        "call": {
                "procedura": "p_save",
                "table": "PROIZVODI"
        },
"kolone": [{
        "NAZIV": "Svježa rajčica",
        "TIP": "Povrće",
```

```
            "OPIS": "Svježi sok od rajčice s bosiljkom",
            "VISINA": 600,
            "SIRINA": 903,
            "CIJENA": 16.3,
            "OCJENA": 2
        }, {
            ""NAZIV": "Zeleni smoothie",
            "TIP": "Mliječni proizvodi",
            "OPIS": "Čaša zelenog smoothieja sa žumanjkom od prepeličjih jaja, poslužena s tubom za koktel zelenom jabukom i listovima mladog špinata na limenoj površini..",
            "VISINA": 600,
            "SIRINA": 399,
            "CIJENA": 17.68,
            "OCJENA": 4
        }, {
            "NAZIV": "Zeleni grah",
            "TIP": "Povrće",
            "OPIS": "Sirovi organski zeleni grah spreman za jelo",
            "VISINA": 450,
            "SIRINA": 300,
            "CIJENA": 28.79,
            "OCJENA": 1
        }, {
            "NAZIV": "Svježe kruške",
            "TIP": "Voće",
            "OPIS": "Slatke svježe kruške na drvenom stolu",
            "VISINA": 600,
            "SIRINA": 398,
            "CIJENA": 15.12,
            "OCJENA": 5
        }
    ]
```

```
}');
COMMIT;

SELECT * FROM JSON_SCHEMA;
```

Programski kod 7.4 prikazuje umetanje proizvoda u tablicu JSON_SCHEMA. Tablica sadrži 'literal string' (q) koji gleda sve kao jedan string, u protivnom neće gledati kao jednu cjelinu.

8. ZAKLJUČAK

Tema ovog završnog rada bila je Oracle PL/SQL proširenje za CRUD operacije nad tablicama koja je uspješno realizirana. Izrađen je program koji prolazi kroz kontrole i dinamički manipulira velikim JSON-om, razlama ga na manje dijelove te ga sprema u tablicu. Spremanje JSON-a u Oracle bazu podataka sve češće se koristi u praksi. Pri obradi i spremanju JSON podataka nije potrebno raditi transformaciju podataka što ubrzava vrijeme dohvaćanja podataka. JSON format je razumljiv i čitljiv ljudima zbog svoje sintakse. JSON datoteke mogu biti tekstualno velike što je prednost JSON formata. Također poruke su napravljene kako bi korisnici programa znali u čemu je problem. U završnom radu još postoji tablica koja je izrađena po uzoru na sistemsku tablicu te služi za spremanje svih tablica koje su kreirane, kako bi se u svakom trenutku vidjele promjene.

9. LITERATURA

History of SQL, [1], 2022, Dostupno na: History of SQL, 05.07.2022.

Structured Query Language (SQL), [2] ,2005., Dostupno na: What is Structured Query Language (SQL)? (techtarget.com), 15.07.2022.

PL/SQL (procedural language extension to Structured Query Language), [3], 2018, Dostupno na: PL/SQL, 17.07.2022.

PL/SQL Block: STRUCTURE, Syntax, ANONYMOUS Example, [4], 2022, Dostupno na: PL/SQL Block, 18.07.2022.

Oracle PL/SQL Dynamic SQL, 2022, Dostupno na: Oracle PL/SQL Dynamic SQL, 20.07.2022.

PL/SQL – Handling Excpetions, [5], 2020, Dostupno na: PL/SQL – Handling Excpetions, 24.07.2022.

PL/SQL Package, 2021, [6], Dostupno na : SQL Packages, 25.07.2022.

Oracle Views, 2019, [7], Dostupno na: Oracle Views, 26.07.2022.

Oracle Dana Types, 2019 , [8] , Dostupno na: Oracle Dana Types, 26.07.2022.

Object Types, 2005, Dostupno na: Object Types, 01.08.2022.

JSON Schema, 2015, [9] ,Dostupno na: JSON Schema , 03.08.2022.

JSON Schema Excamples, 2018, [10], Dostupno na: JSON Schema Examples Tutorial, 04.08.2022.

10. OZNAKE I KRATICE

ANSI – American National Standards Institute

ISO – International Organization for Standardization

RDBMS – Relation database managment system

SQL – Structured Query Language

DDL – Data Definition Language

DML – Data Manipulation Language

DCL – Data Control Language

PL/SQL – Procedural Language for SQL

ASCII – American Standard Code for Information Interchange

JSON – Java Script Object Notation

AJAX – Asynchronous JavaScript And XML

REST – Representational state transfer

OCI – Oracle Call Interface

JDBC – Java Database Connectivity

XML – Extensible Markup Language

11. SAŽETAK

U ovom završnom radu opisan je princip rada kako bi se trebalo ispravno postupati s velikim JSON formatom. Za realizaciju projekta korištena je Oracle baza podataka te SQL developer kao alat. Za JSON dinamički unos (engl. *insert*) u tablicu bilo je potrebno razlomiti JSON na dijelove te pomoću kontrola provjeriti njegovu ispravnost prije umetanja u tablicu. Poruke su dodane da bi korisnik odmah znao u čemu je problem umjesto sistemskih i nejasnih poruka od same baze. U radu je još dodana i tablica koja je napravljena po uzoru na sistemsku tablicu u koju se spremaju sve nove tablice i promjene kako bi bilo jasnije što se događa s tablicama i njihovim izmjenama u svakom trenutku.

Ključne riječi: SQL, JSON, dinamički SQL, poruke

12. ABSTRACT

In this final paper, the working principle is described in order to properly handle the large JSON format. Oracle database and SQL developer were used as tools for project implementation. In order to dynamically insert the JSON into the table, it was necessary to break the JSON into pieces and use a control to check its correctness before inserting it into the table. Messages have been added so that the user knows immediately what the problem is instead of system and vague messages from the database. The work also added a table modeled after the system table in which all new tables and changes are saved in order to make it clearer what is happening with the tables and their changes at any time.

Key words: SQL, JSON, Dynamic SQL, messages

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, 14. 10. 2022.	DOMAGOJ PETREC	Petrec D.

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

DOMAGOJ PETREC

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, 14.10.2022

Petrec D.

potpis studenta/ice