Oracle PL/SQL proširenje za kontrolu ulaznih podataka

Somljačan, Matej

VELEUČILIŠTE U BJELOVARU
PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

Oracle PL/SQL proširenje za kontrolu ulaznih podataka

Završni rad br. 03/RAČ/2022

Matej Somljačan

Bjelovar, rujan 2022.

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Matej Somljačan** JMBAG: **0314019777**

Naslov rada (tema): **Oracle PL/SQL proširenje za kontrolu ulaznih podataka**

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

2. ZADATAK ZAVRŠNOG RADA BROJ: 03/RAČ/2022

U sklopu završnog rada potrebno je:

1. Izraditi JSON objekt za kontrolu ulaznih podataka po uzoru na parametre jQuery metode attr
2. Napisati PL/SQL funkciju za dohvaćanje pripadajućih JSON Schema i povezati s ROUTER PL/SQL paketom
3. Napisati PL/SQL funkciju za čitanje podataka iz JSON Scheme
4. U okviru PL/SQL procedure dohvaćati podatke iz JSON formata
5. Izraditi member funkciju za pretvorbu Oracle tipa podatka type u JSON
6. Izraditi dinamičke kontrole nad ulaznim podacima na temelju JSON Schema unutar PL/SQL procedure
7. Upravljati s iznimkama u komunikaciji između procedura i funkcija u PL/SQL-u

Datum: 30.06.2022. godine

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

Zahvala

Zahvaljujem se mentoru mag. ing. el. Tomislavu Adamoviću na pomoći tijekom pisanja ovog završnog rada. Također se zahvaljujem svim ostalim profesorima na znanju stečenom tijekom pohađanja Veleučilišta u Bjelovaru te svim kolegama studentima. Konačno zahvaljujem se svojoj obitelji na podršci tijekom pisanja rada.

Sadržaj

1. UVOD

Prilikom unosa podataka u bazu podataka u većini slučajeva se ulazni podatci ne kontroliraju dovoljno i ovaj rad pokazuje kako riješiti taj problem. Ukoliko se ne obave kontrole u bazi podataka može doći do narušavanja integriteta i konzistentnosti podataka. Svrha rada je provjeriti sve podatke na razini baze podataka prije no što uopće dođe do unosa podataka u bazu podataka te javiti pozitivnu odnosno negativnu poruku ovisno o tome jesu li podatci prošli kontrolu ili ne. Ta kontrola odvija se na način da se programu proslijede dva JSON objekta, prvi s ulaznim podatcima i drugi s kontrolom za svaki podatak, te se provjeri prolaze li ulazni podatci sve kontrole. Također program može kao parametar primiti i tip (engl. *type*) koji prethodno mora biti pretvoren u oblik JSON objekta uz pomoć *member* funkcije. Kroz ovaj rad će biti opisani preduvjeti za uspješno izvođenje programa, izgled ulaznih JSON objekta, sam način rada programa te kontroliranje ispravnosti ulaznih JSON objekta, rada programa i prikazivanje grešaka korisniku uz pomoć iznimki. Za izvršavanja kontrole koristi se JSON zbog jednostavnosti dohvaćanja podataka i zbog lake čitljivosti podataka za korisnika. Program sve podatke dohvaća dinamički tako da pri dodavanju nove kontrole potrebne su minimalne dorade kako bi provjera funkcionirala za novu kontrolu. Ovaj program će raditi na svim mjestima gdje se za unos podataka u bazu podataka koriste JSON objekti te će na vrlo jednostavan način provjeriti podatke iz njega i time unaprijediti rad programa.

2. SQL

SQL je strukturni upitni programski jezik. Najrašireniji je jezik koji se koristi za rad s podatcima unutar relacijske baze podataka. Koristi se za strukturirane podatke to jest podatke koje međusobno imaju relacije. Unutar SQL-a postoje takozvani pod jezici DQL, DDL, DCL i DML. Načini na koji SQL upravlja podatcima u bazi su dohvaćanje podataka putem SELECT izraza, manipulacija i izmjena pomoću INSERT, UPDATE i DELETE izraza. SQL, iako je deklarativni jezik, ima neke proceduralne elemente. Prva verzija SQL-a se pojavila prije 48. godina to jest 1974. godine.

Programski kod 2.1: Primjer SQL naredbe

```
SELECT * FROM STUDENTI WHERE GODINE > 20 ORDER BY IME DESC
```

2.1. PL/SQL

PL/SQL je dodatak proceduralnih mogućnosti, kao što su uvjeti i petlje, na sami SQL. U okviru navedenog PL dodaje mogućnost kreiranja anonimnog bloka, deklaracije konstanti i varijabli, procedura, funkcija i tipova. Također posjeduje značajke objektno orijentiranog programskog jezika. Kreiran je od strane Oracle Corporation početkom devedesetih godina prošlog stoljeća.

2.2. Anonimni blok

PL/SQL daje mogućnost kreiranja anonimnog bloka koji predstavlja osnovnu jedinicu PL/SQL programa. Razlog iza imena anonimni blok je taj što blok nije pohranjen u bazi podataka te stoga nema određeni naziv. Anonimni blok je definiran ključnim riječima DECALRE, BEGIN, EXCEPTION i END. Navedene ključne riječi dijele anonimni blok u dijelove. DECLARE označava deklarativni dio, BEGIN i END izvršni dio i EXCEPTION dio za rukovanje iznimkama. Jedini dio anonimnog bloka koji je potreban da se program izvrši jest izvršni dio dok su deklarativni i dio za rukovanje iznimkama izborni. Blokovi također mogu biti ugniježđeni jedan u drugome.

Programski kod 2.2: Primjer anonimnog bloka

```
SET SERVEROUTPUT ON
DECLARE
  broj1 NUMBER(2);
BEGIN
  broj1 := 12;
  DBMS_OUTPUT.PUT_LINE(broj1);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error poruka: ' || sqlerrm);
END;
```

Programski kod 2.2. prikazuje primjer jednostavnog anonimnog bloka koji deklarira varijablu broj1, dodaje vrijednost toj varijabli i na kraju ju ispisuje korisniku na ekran. U slučaju neke greške korisniku će se na ekranu ispisati tekst greške zbog EXCEPTION djela unutar bloka.

2.3. Funkcije

Generalno, svrha funkcija u PL/SQL-u jest odraditi određeni posao i vrati rezultat. Rezultat koji funkcija vraća može biti skalarna vrijednost, ali također može biti i kolekcija (niz). Funkcije koje su definirane od strane korisnika dopunjavaju funkcije ugrađene od strane Oracle Corporationa. Funkcije kao parametar mogu primiti više vrijednosti koje su isključivo tipa IN. Jedini OUT tip vrijednosti funkcije je vrijednost koju funkcija vraća.

2.4. Procedure

Procedure nalikuju funkcijama po tome jer su imenovane programske jedinice koje se mogu pozivati više puta. Osnovna razlika je u tome što se funkcije mogu koristiti u SQL izrazima dok se procedure ne mogu. Još jedna velika razlika je u tome što funkcije vraćaju isključivo jedan rezultat dok procedura može vratiti više rezultata. Procedura može primiti tri tipa parametara. IN parametar se koristi kao *input only*. OUT parametar je u početku *null* i koristi se da se u njega spremi rezultat koji procedura vraća. Zadnja vrsta parametra je IN OUT koji se koristi i kao ulazni i kao izlazni parametar.

2.5. Tipovi

Tipovi u Oracle-u omogućavaju modeliranje objekata i spremanje objektno orijentiranih podataka u bazu. Tipovi su zapravo implementacija objekata definirani na sličan način kao i paketi. Za razliku od paketa tipovi se spremaju u bazu i mogu biti za kasniju upotrebu. Definicija tipa sadrži listu atributa odvojenu zarezima. Također u sebi može sadržavati *member* funkcije ili procedure. Ukoliko tip sadrži takve funkcije ili procedure njihov kod je definiran u tijelu tog tipa.

Programski kod 2.3: Primjer kreiranja tipa

```
CREATE OR REPLACE TYPE OSOBA AS OBJECT (
  IME          VARCHAR2(50),
  PREZIME   VARCHAR2(50),
  GODINE     NUMBER,
  MEMBER FUNCTION to_json RETURN clob
);
```

U programskom kodu 2.3. je prikazan kod potreban za stvaranje tipa koji se sastoji od više atributa različitih tipova podataka s *member* funkcijom koja pretvara taj tip u oblik JSON objekta.

3. JSON

JSON je podskup zapisa JavaScript objekta. U većini slučajeva JSON služi za razmjenu podataka. JSON se u JavaScript programskom jeziku može koristiti bez potrebe parsiranja ili serijaliziranja i na jednostavan način prikazivanja objekata, nizova i skalarnih podataka kao tekstualni zapis. Iako je JSON definiran kao notacija JavaScript objekta može se koristiti neovisno o programskom jeziku to jest većina programskih jezika može generirati i parsirati JSON podatke. Velika prednost JSON objekta jest njegova laka čitljivost, preglednost i pisanje za čovjeka. Također vrlo je lako za softver da ga parsira i generira. Često se koristi za serijalizaciju strukturiranih podataka i dijeljenje tih podataka između servera i web aplikacija.

3.1. JSON sintaksa

JSON se sastoji od para ključ i vrijednost. Ključevi uvijek moraju biti tipa *string* dok vrijednost može biti raznih tipova podataka kao što su *string, number, object,* polje i *bool*. Ključ i vrijednost unutar JSON objekta su razdvojeni pomoću dvotočke. Vrijednost također može biti prazan podatak to jest null. Svaki ključ unutar JSON objekta se mora pisati unutar jednostrukih (') ili dvostrukih navodnika (") dok vrijednost mora biti pod navodnicima samo ako se radi o tipu podatka *string*. Svaki par ključ - vrijednost unutar JSON objekta se razdvaja pomoću zareza. Svi parovi ključ – vrijednost se nalaze unutar vitičastih zagrada ({ }). Unutar JSON objekta se ne može koristiti datumski tip podatka kao takav nego se prikazuje uz pomoć *stringa*. Ako se unutar JSON objekta želi prikazati polje to jest niz koriste se uglate zagrade ([,]) oko elemenata koji su unutar zagrada odvojeni zarezom. JSON polje unutar sebe može imati nula ili više elemenata. Unutar JSON objekta se također kao vrijednost može zapisati još jedan JSON te se tako, u teoriji, mogu nizati beskonačno.

Programski kod 3.1: Jednostavan JSON

```
{
    "Ime": "Matej",
    "Prezime": "Somljačan",
    "Godine": 22,
    "Student": true
}
```

Programski kod 3.2: Polje unutar JSON objekta

```
{
    "Ime": "Matej",
    "Prezime": "Somljačan",
    "Godine": 22,
    "Student": true,
    "Položeno": ["Matematika", "Baze", "Web"]
}
```

Programski kod 3.3: JSON unutar JSON objekta

```
{
        "Ime": "Matej",
        "Prezime": "Somljačan",
        "Godine": 22,
        "Student": true,
        "Adresa": {
                "Ulica": "Pere Perića",
                "Broj": 5,
                "Grad": "Bjelovar"
        }
}
```

U programskom kodu 3.1. prikazan je najjednostavniji oblik JSON objekta u kojemu je vrijednost običan podatak tipa *string*, *number* i *bool*.

Programski kod 3.2. prikazuje malo kompleksniji JSON gdje se kao vrijednost jednog para koristi polje to jest niz podataka unutar kojeg se nalazi *string*.

Programski kod 3.3. prikazuje korištenje JSON objekta kao vrijednost unutar drugog JSON objekta.

3.2. JSON u usporedbi s XML-om

Uz JSON, za razmjenu podataka se još koristi i XML. Najveće razlike između JSON objekta i XML-a su te da se JSON koristi za jednostavne i strukturirane podatke, a XML je koristan i u slučaju kada su djelomično strukturirani podatci. Također JSON se koristi isključivo za prikaz

podataka dok se XML još koristi kao *Markup* jezik. Zbog svoje jednostavne definicije podatke iz JSON objekta je lakše generirati, parsirati i procesirati od podataka iz XML-a.

3.3. JSON u Oracle bazi podataka

Oracle baza podataka podržava JSON sa svim funkcionalnostima relacijske baze podataka kao što su view-ovi, indeksiranje i transakcije. Pri kreiranju tablice u bazi podataka JSON polje se može napraviti na dva načina. Prvi način je da se uz naziv polja kao tip podatka postavi JSON kao što se vidi u programskom kodu 3.4. dok je drugi način da se tip polja postavi na *varchar2* ili *clob* i doda *check constraint is json* koji provjerava je li uneseni *string* podataka validan JSON ili ne kao što se vidi u programskom kodu 3.5.

Programski kod 3.4: Kreiranje tablice – 1. način

```
CREATE TABLE PRIMJER_JEDAN
 (ID            NUMBER NOT NULL PRIMARY KEY,
  IME           VARCHAR2(100),
  ADRESA        JSON);
```

Programski kod 3.5: Kreiranje tablice – 2. način

```
CREATE TABLE PRIMJER_DVA
 (ID            NUMBER NOT NULL PRIMARY KEY,
  IME           VARCHAR2(100),
  ADRESA        CLOB,
  CONSTRAINT "ENSURE_JSON" CHECK (ADRESA IS JSON));
```

Pri unosu JSON objekta u bazu podataka izvršava se provjera je li taj JSON validan zbog constrainta koje je postavljen na taj podatak prilikom kreiranja tablice u bazi.

Programski kod 3.6: Insert podataka u bazu

```
INSERT INTO primjer_jedan VALUES (
        1,
        'Matej',
        '{"Ulica": "Pere Perića",
          "Broj": 5,
          "Grad": "Bjelovar"}' );
```

U programskom kodu 3.6. je prikazan insert izraz za unos podataka u bazu koji u sebi sadrže JSON. SQL *string* koji u sebi sadrži JSON je automatski pretvoren u tip podatka JSON prilikom unosa u bazu podataka. U Oracle bazi podataka podatci iz JSON objekta se lako mogu dohvatiti. Programski kod 3.7. prikazuje na koji način se dohvaćaju podatci unutar JSON objekta.

Programski kod 3.7: Dohvaćanje podataka unutar JSON objekta

```
SELECT J.ADRESA.ULICA FROM PRIMJER_JEDAN J
```

U SQL-u postoje razne ugrađene funkcije koje se koriste za rad s JSON objektima. Neki od primjera funkcija koje se češće koriste su *json_value* koja vraća vrijednost za određeni ključ, *json_table* koja vraća podatke iz JSON objekta u obliku tablice u bazi i *json_serialize* koja vraća tekstualni prikaz podataka iz JSON objekta. Također postoje funkcije *json_exists*, *is json* i *is not json* koje služe za provjeru postoji li JSON te je li varijabla odnosno polje tipa JSON.

3.4. JSON i PL/SQL

Tipovi objekata iz PL/SQL-a dozvoljavaju finu programsku konstrukciju i manipulaciju podataka iz JSON objekta. Lako se može pregledati, modificirati JSON i vratiti ga nazad u tekstualni oblik. Glavne PL/SQL JSON vrste objekata su JSON_ELEMENT_T, JSON_OBJECT_T, JSON_ARRAY_T i JSON_SCALAR_T. Takvi objekti se također nazivaju apstraktni podatkovni tipovi. Ti JSON objekti daju mogućnost hijerarhijskog programskog prikaza podataka iz JSON objekta koji su pohranjeni u bazi podataka. Za pretvorbu JSON teksta u tipove JSON_ELEMENT_T, JSON_OBJECT_T ili JSON_ARRAY_T koristi se statička funkcija *parse*.

Programski kod 3.8: Primjer JSON objekata

```
set serveroutput on
DECLARE
  jo             JSON_OBJECT_T;
  ja             JSON_ARRAY_T;
  keys           JSON_KEY_LIST;
  keys_string    VARCHAR2(100);
BEGIN
  ja := new JSON_ARRAY_T;
  jo := JSON_OBJECT_T.parse('{
     "Ime": "Matej",
     "Prezime": "Somljačan",
     "Godine": 22
}');
  keys := jo.get_keys;
  FOR i IN 1..keys.COUNT LOOP
    ja.append(keys(i));
  END LOOP;
  keys_string := ja.to_string;
  DBMS_OUTPUT.put_line(keys_string);
END;
```

Programski kod 3.8. prikazuje korištenje JSON objekata za dohvaćanje podataka, spremanje tih podataka u varijablu i ispisivanje podataka iz JSON objekta korisniku na ekran.

4. JSON SCHEMA

JSON Schema je JSON objekt koji omogućava komentiranje i validaciju drugog JSON objekta. Schema je korisna za automatizirano testiranje i osiguravanje kvalitete unesenih podataka od strane korisnika. Unutar JSON Schema-e se nalaze ključne informacije koje opisuju kakvi podatci moraju biti, primjerice, kojeg tipa podatak mora biti ili je li podatak obavezan. Schema također prikazuje kako izvući informacije iz JSON objekta te kako komunicirati s tim podatcima. Uz opisivanje podataka JSON Schema još pruža pregled osnovnih informacija o podatcima u formatu koji je prilagođen korisniku, ali i stroju to jest računalu. Schema može sadržavati i informacije o tome koja se procedura zove ili iz koje se tablice podatci koriste kao što je prikazanu u programskom kodu 4.1. JSON Schema može opisivati isključivo JSON objekte, što također znači da se mogu koristiti i tipovi ukoliko se prvo pretvore u oblik JSON objekta.

Programski kod 4.1: Primjer JSON Schema-e

```
{
    "call": {
            "procedura": "p_save",
            "table": "KORISNICI"
    },
    "ID": {
            "html": "<input>",
            "atrb": {
                    "type": "hidden",
                    "required": true
            }
    },
    "IME": {
            "label": "Ime:",
            "html": "<input>",
            "atrb": {
                    "type": "text",
                    "required": true,
                    "maxlength": 60,
                    "width": "20%"
```

```
            }
        },
        "EMAIL": {
            "label": "Email:",
            "html": "<input>",
            "atrb": {
                "type": "email",
                "required": true
            }
        },
        "action": {
            "cancel": {
                "html": "<button>",
                "atrb": {
                    "type": "submit",
                    "html": "Pošalji"
                }
            }
        }
}
```

5. DINAMIČKI SQL

Dinamički SQL je metodologija za generiranje i izvršavanje SQL izraza pri izvršavanju programa. Koristi se u slučajevima kada postoje vrijednosti koje su nepoznate u trenutku pokretanja programa ili kada treba izvršiti SQL koji nije podržan u statičkom SQL-u. Ukoliko je moguće koristiti statički SQL uvijek ga treba koristiti umjesto dinamičkog zbog svojih prednosti. Prednost je brzina izvođenja jer je potrebno manje resursa. Također, statički SQL se prethodno obrađuje što znači da se izrazi analiziraju, provjeravaju i optimiziraju samo jednom. PL/SQL pruža dva načina pisanja dinamičkog SQL-a. Prvi način je *Native*, a drugi DMB_SQL paket to jest API za gradnju, pokretanje i opis dinamičkih SQL izraza. *Native* dinamički SQL procesira dinamički SQL izrazi pomoću EXECUTE IMEDIATE izraza. Također on nam omogućuje da se za SELECT izraza koje vraćaju više redaka koriste OPEN, FOR, FETCH i CLOSE izrazi za kursore.

Programski kod 5.1: Primjer dinamičkog SQL-a

```
EXECUTE IMMEDIATE 'SELECT P.IME FROM PRIMJER P WHERE ID = 1' INTO PIME
```

6. PREDUVJETI ZA KONTROLU PODATAKA

Svaka slučajna pogreška koju korisnik može napraviti, primjerice invalidan JSON kao ulazni parametar programa, može se prepoznati jer program automatski javlja da nešto nije u redu. Problem nastaje dok dva JSON objekta nemaju jednak broj ključeva to jest podataka koji se koriste u programskoj logici. U broj podataka unutar JSON objekta se ne ubrajaju ključevi *call* i *action* koje u kodu preskačemo jer nisu potrebni za kontrolu ulaznih podataka. Da bi kontrola uspješno prošla redoslijed ključeva unutar dva JSON objekta mora biti jednak. Razlog tome je što program ključeve iz oba JSON objekta dohvaća redoslijedom kako su napisani unutar JSON objekta. Kako bi se izbjeglo dobivanje neispravnih izlaznih informacija korisnik mora provjeriti oba JSON objekta i utvrdi da redoslijed ključeva ide istim redoslijedom. Ukoliko to ne napravi, program će vratiti poruku da redoslijed nije dobar pa korisnik mora to ispraviti. Ako kao ulazni parametar za proceduru korisnik odluči koristiti tip, mora se pobrinuti da ga prethodno pretvori u oblik JSON objekta iz razloga što program radi isključivo s JSON objektima. Pretvorba tipa u JSON se u većini slučajeva radi pomoću *member* funkcije koja se dodaje u tijelo tipa i jedina joj je svrha pretvorba. Tip podatka koje procedura prima kao ulazne parametre je clob zato što taj tip podatka dozvoljava spremanje velike količine znakova u jednu varijablu. Iako će uslijed invalidnog JSON objekta korisnik dobiti poruku da nešto nije u redu, korisnik bi trebao provjeriti je li JSON objekt ispravno napisan. To će postići tako da prvobitno provjeri jesu li svi ključevi napisani pod dvostrukim navodnicima te jesu li vrijednosti koje su alfanumeričke također napisane pod dvostrukim navodnicima. Korisnik bi uvijek trebao osigurati provjeru ovih preduvjeta kako bi program radio na način koji je ispravan te kako bi program vratio ispravne povratne informacije o tome prolaze li podatci kontrole ili ne.

Programski kod 6.1: Primjer dva odgovarajuća JSON objekta

```
{
	"ID": 1,
	"IME": "Matej",
	"GODINE": 22,
}



{
	"call": {
		"procedura": "p_save",
		"table": "KORISNICI"
	},
	"ID": {
		"html": "<input>",
		"atrb": {
			"type": "number",
			"required": true,
			"maxlength": 60
		}
	},
	"IME": {
		"label": "Ime:",
		"html": "<input>",
		"atrb": {
			"type": "text",
			"required": true,
			"maxlength": 60,
			"width": "20%"
		}
	},
	"GODINE": {
		"label": "Godine:",
		"html": "<input>",
```

```
                "atrb": {
                    "type": "number",
                    "required": true,
                    "maxlength": 3
                }
            }
        }
```

Programski kod 6.1 prikazuje dva odgovarajuća JSON objekta za koje se može izvršiti program. Redoslijed svih podataka to jest ključeva unutar oba JSON objekta je isti tako da će sve proći uspješno. Prvi ključ unutar kontrolnog JSON objekta kada bi se gledao ne bi odgovarao drugom JSON objektu, no program radi na način da ukoliko dođe do polja unutar JSON objekta pod nazivom *call* ili *action* preskoči ih i ne ubraja kao dio kontrole.

7. PROCEDURE I FUNKCIJE PROGRAMA

Program se sastoji od jedne glavne procedure u kojoj se nalazi sva logika za obradu ulaznih podataka i vraćanje poruke korisniku programa. Program također koristi i tri funkcije od kojih se svaka koristi za određenu kontrolu nad ulaznim podatcima ovisno o kojoj se kontroli radi. Ukoliko je potrebno zbog korisničkih želja dodati neku novu kontrolu u program jednostavnost izrade dozvoljava da se to napravi na način da se doda nova funkcija za kontrolu i poziv funkcije unutar procedure. Sve funkcije rade na način da ukoliko je uvjet zadovoljen i sve je u redu funkcija vrati broj 0, a ukoliko nešto nije prošlo to jest uvjeti nisu zadovoljeni vrati broj 1.

Programski kod 7.1: Funckija za provjeru dužine

```
function f_provjeri_maxlength(l_test_input varchar2, l_maxlength varchar2) return number as
begin
   if (length(l_test_input) > l_maxlength) then
      return 1;
   else
      return 0;
   end if;
end f_provjeri_maxlength;
```

Programski kod 7.1. prikazuje funkciju koja provjera zadovoljava li duljina podatka iz prvog JSON objekta odgovara duljini definiranoj za taj podatak u kontrolnom JSON objektu. Ukoliko je duljina u redu funkcija vraća broj 0 zbog kojeg kasnije procedura zna da li je podatak prošao kontrolu ili nije.

Programski kod 7.2: Funkcija za provjeru obveznosti unosa

```
function f_provjeri_required(l_test_input in varchar2, l_required in varchar2) return number as
begin
    if (l_required = 'true') then
        if (l_test_input is not null) then
            return 0;
        else
            return 1;
        end if;
    else
        return 0;
    end if;
end f_provjeri_required;
```

Programski kod 7.2. pokazuje funkciju koja izvršava kontrolu za podatak je li unesen to jest postoji li ako je obavezan. Ukoliko je *required* za neki podatak *true* program izvrši kontrolu i ako postoji podatak prolazi kontrolu. Ako je *required* za podatak *false* funkcija će automatski vratiti 0 što znači da podatak prolazi kontrolu neovisno o tome postoji li ili ne.

Programski kod 7.3: Funkcija za provjeru tipa podatka

```
function f_provjeri_type(l_test_input in varchar2, l_type in varchar2, l_naziv in varchar2) return number as
begin
    if (length(l_test_input) > 0) then
        if (l_type = 'number') then
            if (REGEXP_LIKE(l_test_input, '^[[:digit:]]+$')) then
                return 0;
            else
                return 1;
            end if;

        elsif (l_type = 'text') then
```

```
            if (LENGTH(TRIM(TRANSLATE(l_test_input,
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZčćžšđČĆŽŠĐ', ' '))) is
null) then
                return 0;
            else
                return 1;
            end if;

        elsif (l_type = 'email') then
            if (REGEXP_LIKE (l_test_input, '^[A-Za-z]+[A-Za-z0-9.]+@[A-Za-z0-9.-]+\.[A-Za-
z]{2,4}$')) then
                return 0;
            else
                return 1;
            end if;

        elsif (l_type = 'enum') then
            if (upper(l_naziv) = 'SPOL') then
                if (l_test_input in (0, 1)) then
                    return 0;
                else
                    return 1;
                end if;
            end if;

        else
            return 2;

        end if;
    end if;
end f_provjeri_type;
```

Programski kod 7.3 prikazuje funkciju za provjeru tipa podatka to jest odgovara li korisnički uneseni podatak definiranom tipu podatka unutar kontrolnog JSON objekta. Funkcija provjerava,

ovisno o definiciji u kontrolnom JSON objektu, je li podatak tekst, broj, e-mail ili *enum*. Provjere za tekst, broj i e-mail se odvijaju pomoću regularnih izraza i ovisno o tome prolazili li kontrolu ili ne, vraća 0 odnosno 1. Provjera za *enum* prije svega provjerava o kojem podatku se radi pa tek onda ovisno o tome provjerava uvjet koji podatak mora zadovoljavati.

Glavna procedura poziva te funkcije unutar FOR petlje gdje također dinamički dobiva podatke iz oba JSON objekta i šalje te podatke u funkcije. Funkcije vraćaju 0 ili 1 ovisno o tome prolaze li podatci kontrolu ili ne i tada se generiraju poruke uspješnosti ili neuspješnosti kontrole. Prije no što podatke šalje funkcijama procedura te podatke dohvaća iz JSON objekta unutar dvije FOR petlje. Jedna FOR petlja dohvaća informacije iz kontrolnog JSON objekta koji sadrži opis podataka, a druga dohvaća informacije iz JSON objekta koji sadrži podatke nad kojima se kasnije bude izvršila kontrola.

Programski kod 7.4: Procedura

```
procedure p_provjera_inputa(l_input in clob, l_control in clob)
```

Programski kod 7.5: Obrada podataka iz JSON objekta

```
ja := new JSON_ARRAY_T;
jo := JSON_OBJECT_T.parse(l_control);
keys := jo.get_keys;

ja_input := new JSON_ARRAY_T;
jo_input := JSON_OBJECT_T.parse(l_input);
keys_input := jo_input.get_keys;
```

Programski kod 7.6: Petlje za dohvaćanje podataka

```
    FOR i IN 1..keys.COUNT LOOP
        IF (keys(i) not in ('call', 'action')) then
            ja.append(keys(i));
        end if;
    END LOOP;

    FOR i IN 1..keys_input.COUNT LOOP
        ja_input.append(keys_input(i));
    END LOOP;
```

8. PRIMJERI POZIVA PROCEDURE

8.1. Uspješna kontrola

Programski kod 8.1: Poziv procedure s uspješnim rezultatom

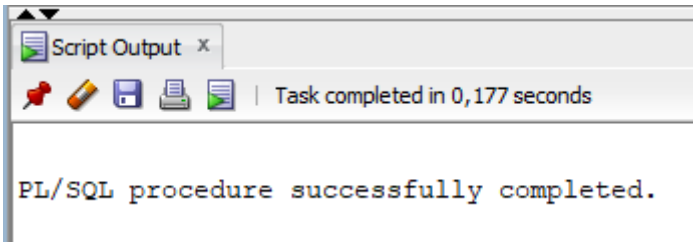
```
set serveroutput on
declare
   l_input clob := '{
   "ID": 1,
        "IME": "Matej",
        "PREZIME": "Somljačan",
        "EMAIL": "msomljacan@vub.hr",
   "GODINE": 2211,
   "SPOL": 1
   }';

   l_kontrola clob := '{
        "call": {
                "procedura": "p_save",
                "table": "KORISNICI"
        },
        "ID": {
                "html": "<input>",
                "atrb": {
                        "type": "number",
                        "required": true,
                        "maxlength": 60
                }
        },
        "IME": {
                "label": "Ime:",
                "html": "<input>",
                "atrb": {
                        "type": "text",
                        "required": true,
```

```
                "maxlength": 60,
                "width": "20%"
            }
        },
        "PREZIME": {
            "label": "Prezime:",
            "html": "<input>",
            "atrb": {
                "type": "text",
                "required": true,
                "width": "20%"
            }
        },
        "EMAIL": {
            "label": "Email:",
            "html": "<input>",
            "atrb": {
                "type": "email",
                "required": true,
                "maxlength": 60
            }
        },
        "GODINE": {
            "label": "Godine:",
            "html": "<input>",
            "atrb": {
                "type": "number",
                "required": true,
                "maxlength": 3
            }
        },
    "SPOL": {
            "label": "Spol:",
            "html": "<input>",
```

```
                "atrb": {
                    "type": "enum",
                    "required": true
                }
            },
            "action": {
                "cancel": {
                    "html": "<button>",
                    "atrb": {
                        "type": "submit",
                        "html": "Pošalji"
                    }
                }
            }
        }';
        begin
        kontrole.p_provjera_inputa(l_input, l_kontrola);
        end;
```

U programskom kodu 8.1 prikazana su dva JSON objekta koja odgovaraju jedan drugome i za koji će kontrola proći bez greške što znači da svaki podatak iz podatkovnog JSON objekta odgovara kontroli definiranoj u kontrolnom JSON objektu. Na slici 8.1. vidi se rezultat pokretanja proceduru u anonimnom bloku.



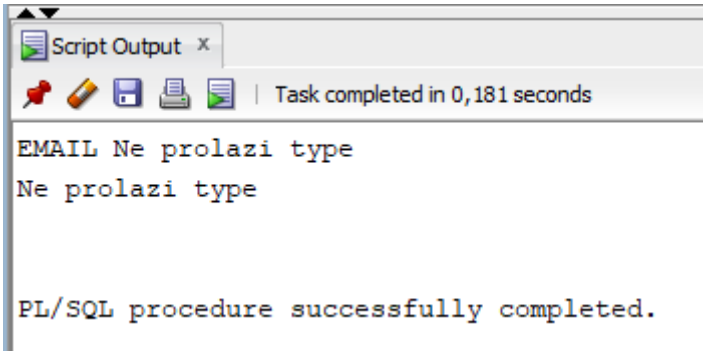
Slika 8.1: Uspješna kontrola

8.2. Neuspješna kontrola

Programski kod 8.2: Poziv procedure s neuspješnim rezultatom

```
l_input clob := '{
   "ID": 1,
        "IME": "Matej",
        "PREZIME": "Somljačan",
        "EMAIL": "msomljacanvub.hr",
    "GODINE": 21,
    "SPOL": 5
   }';
```

U programskom kodu 8.2 je promijenjen email tako da više nema znak @ u sebi što znači da ne prolazi kontrolu te će korisnik na ekranu dobiti poruku greške prikazanu na slici 8.2.



Slika 8.2: Neuspješna kontrola

8.3. Korištenje tipa za poziv procedure

Kao što je prethodno već spomenuto za poziv programa može se koristiti i tip. Jedini preduvjet za korištenje tipa jest pretvoriti ga prethodno u JSON pomoću *member* funkcije. Member funkcija se definira pri kreiranju tipa kao što je vidljivo u programskom kodu 8.3. dok se sama programska logika te metode nalazi unutar tijela tog tipa. *Programski kod* 8.4. prikazuje programsku logiku pretvorbe tipa u JSON.

Programski kod 8.3: Kod za kreiranje tipa

```
CREATE OR REPLACE TYPE OSOBA AS OBJECT (
  ID              NUMBER,
  IME             VARCHAR2(50),
  PREZIME         VARCHAR2(50),
  GODINE          NUMBER,
  EMAIL           VARCHAR2(50),
  SPOL            NUMBER,
  MEMBER FUNCTION to_json RETURN clob
);
```

Programski kod 8.4: Kod za pretvorbu tipa u JSON

```
MEMBER FUNCTION to_json RETURN CLOB IS
  l_json clob;
BEGIN
  l_json := l_json || case
                        when self.ID is not null then
                          (case
                            when l_json is not null then
                              ','
                           end) || '"ID":"' || self.ID || ""
                      end;

  l_json := l_json || case
                        when self.IME is not null then
                          (case
                            when l_json is not null then
                              ','
                           end) || '"IME":"' || self.IME || ""
                      end;

  l_json := l_json || case
                        when self.PREZIME is not null then
                          (case
```

```
                        when l_json is not null then
                          ','
                       end) || '"PREZIME":"' || self.PREZIME || '"'
                  end;

l_json := l_json || case
                    when self.GODINE is not null then
                     (case
                       when l_json is not null then
                         ','
                       end) || '"GODINE":"' || self.GODINE || '"'
                  end;

l_json := l_json || case
                    when self.EMAIL is not null then
                      (case
                       when l_json is not null then
                         ','
                       end) || '"EMAIL":"' || self.EMAIL || '"'
                   end;

l_json := l_json || case
                    when self.SPOL is not null then
                      (case
                        when l_json is not null then
                          ','
                       end) || '"SPOL":"' || self.SPOL || '"'
                  end;

if l_json is not null then
  l_json := '{' || l_json || '}';
end if;

RETURN l_json;
```

```
END to_json;
```

Programski kod 8.5: Poziv procedure s tipom

```
begin
kontrole.p_kontrola(l_input.to_json, l_kontrola);
end;
```

Programski kod 8.5 prikazuje na koji način se poziva procedura ako se ulazni podatci šalju u obliku tipa. Jedina razlika jest što se unutar parametra procedure uz tip poziva i *member* funkcija *to_json*.

9. PRIMJENA NA POSTOJEĆE SUSTAVE

Ukoliko neki drugi sustav ili program koristi JSON objekte za svoj rad može koristiti ovaj program za dodatnu kontrolu svojih podataka. Jedino što korisnik mora napraviti prije nego što pokrene program jest napisati kontrolni JSON objekt po principu JSON Schema-e u kojemu će se nalaziti kontrole koje će korisnički podatci morati poštivati. Primjer kontrolnog JSON objekta je prikazan u programskom kodu 4.1. Nakon toga korisnik pozove proceduru s dva parametra, prvi parametar, njegov JSON objekt koji sadrži njegove podatke i drugi parametar kontrolni JSON objekt kojeg je korisnik naknadno napisao i koji sadrži kontrole koje njegovi podatci moraju proći.

10. ZAKLJUČAK

Tema ovog završnog rada je Oracle PL/SQL proširenje za kontrolu ulaznih podataka koja je uspješno realizirana. Izrađen je program koji obavlja uspješnu kontrolu nad ulaznim podatcima korištenjem dva JSON objekta, jedan s ulaznim podatcima i jedan koji sadržava kontrole koje ti podatci moraju poštivati. Projekt u sklopu završnog rada je izrađen uz pomoć programskog jezika PL/SQL koji je proceduralna nadogradnja na široko rasprostranjeni SQL. Program radi na principu JSON Schema-e koja je rasprostranjena u velikom broju programskih jezika kao sastavni dio i pomoću koje neki programski jezici mogu kroz ugrađene metode izvršiti kontrolu. Za razliku od drugih jezika Oracle PL/SQL nema direktnu podršku za JSON Schema-u nego se s ovim završnim radom ručno izradila kontrola na temelju nje. Rad će biti vrlo koristan za sustave i programe koji koriste JSON objekt jer se poziv izrađene procedure lako mogu provjeriti ulazni podatci prije nego što uopće dođe do korištenja tih podataka u daljnjem radu sustava ili programa. Ukoliko podatci ne prolaze kontrolu program će se automatski zaustaviti i korisniku prikazati poruku greške tako da odmah zna koji podatak ne prolazi koju kontrolu. Program se zbog svoje jednostavnosti može koristiti i u sustavima koje koriste tipove jer sve što je potrebno napraviti jest pretvoriti tip u JSON što se može izvršiti pomoću *member* funkcije unutar definicije tipa. Ovim radom uspješno je realizirana kontrola ulaznih podataka uz pomoć JSON objekta kako za nove sustave tako i za postojeće koji koriste JSON ili tip kao primarni resurs za rad.

11. LITERATURA

TechTarget, Structured Query Language (SQL), 2005., Dostupno na: What is Structured Query Language (SQL)? (techtarget.com), 24.06.2022.

TutorialsPoint, PL/SQL, 2022., Dostupno na: PL/SQL Tutorial (tutorialspoint.com), 23.06.2022.

Oracle, PL/SQL Dynamic SQL, 2014., Dostupno na: PL/SQL Dynamic SQL (oracle.com), 23.06.2022.

Oracle, Oracle Database JSON Developer's Guide, 2022., Dostupno na: Oracle Database JSON Developer's Guide, 21c, 23.06.2022.

Michael Droettboom, Understanding JSON Schema, 2022., Dostupno na: Understanding JSON Schema — Understanding JSON Schema 2020-12 documentation (json-schema.org), 23.06.2022.

12. OZNAKE I KRATICE

DCL – Data Control Language

DDL – Data Definition Language

DML – Data Manipulation Language

DQL – Data Query Language

JSON – JavaScript Object Notation

PL/SQL – Procedural Language for SQL

SQL - Structured Query Language

XML – Extensible Markup Language

13. SAŽETAK

Ovaj se rad bavi problematikom manjka kontrole nad ulaznim podatcima, a programom u sklopu završnog rada koji to rješava. Program je izrađen u programskom jeziku PL/SQL s više funkcija koje služe za samo izvršavanje kontrole nad ulaznim podatcima te glavna procedura u kojoj se nalazi programska logika. U radu su opisani sve potrebne mogućnosti SQL-a i JSON objekta koje su bile potrebne za izradu programa. U radu je također opisana JSON Schema na temelju koje se automatski izvršavaju kontrole. U radu su se koristili dinamički *select* izrazi za dohvaćanje podataka iz JSON objekta. Glavna prednost programa jest da može povećati kontrolu ulaznih podataka kako bi se izbjegli problemi koji mogu nastati ako dođe do ulaza neispravnih podataka u bazu podataka. U radu su prikazani primjeri na koji način program ispravno funkcionira te kako program reagira ukoliko dobije neispravne podatke kao ulazne parametre.

Ključne riječi: JSON, SQL, Dinamički SQL, Kontrola ulaznih podataka

14. ABSTRACT

This paper deals with the issue of lack of control over input data, and the program as part of the final paper that solves it. The program was created in the PL/SQL programming language with several functions that serve only to control the input data and the main procedure that contains the program logic. The paper describes all the necessary SQL and JSON object capabilities that were needed to create the program. The paper also describes the JSON Schema based on which the controls are automatically executed. The paper used dynamic select expressions to retrieve data from the JSON object. The main advantage of the program is that it can increase the control of input data in order to avoid problems that may arise if incorrect data is entered into the database. The paper presents examples of how the program functions correctly and how the program reacts if it receives incorrect data as input parameters.

Key words: JSON, SQL, Dynamic SQL, Input data control

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, 15.09.2022.	MATEJ SOMLYAČAN	Somlyačan

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

MATEJ SOMLYAČAN

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, 15.09.2022.

Somlyačan

potpis studenta/ice