Sloj za validaciju podataka na Oracle bazi

Živković, Iva

VELEUČILIŠTE U BJELOVARU
PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

SLOJ ZA VALIDACIJU PODATAKA NA ORACLE BAZI

Završni rad br. 10/RAČ/2020

Iva Živković

Bjelovar, listopad 2020.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Kandidat: **Živković Iva** Datum: 31.08.2020. Matični broj: 001875

JMBAG: 0314018080

Kolegij: **BAZE PODATAKA**

Naslov rada (tema): **Sloj za validaciju podataka na Oracle bazi**

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

Grana: **Programsko inženjerstvo**

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

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

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

2. ZADATAK ZAVRŠNOG RADA BROJ: 10/RAČ/2020

U radu je potrebno izraditi Oracle proceduru za unos ulaznih podataka u zajedničku tablicu te preko triggera za tu tablicu aktivirati sustav za validaciju ulaznih podataka u dostupne tablice. Validacija podataka će se temeljiti na opisima kontrola unutar sistemskih tablica. Procedura mora biti minimalno invazivna što podrazumijeva vrlo lako uključivanje i isključivanje procedure, autonomnu transakciju, predefinirane poruke o greškama i rukovanje iznimkama.

Zadatak uručen: 31.08.2020. Mentor: **Tomislav Adamović, mag.ing.el.**

Zahvala

Zahvaljujem svojem mentoru završnog rada Tomislavu Adamoviću te mentoru stručne prakse Nenadu Kovačeviću na vanrednom vodstvu i upečatljivoj suradnji. Također zahvaljujem svojim roditeljima te svim kolegama i prijateljima koji su mi svakodnevno pružali podršku i motivaciju kroz cijeli tok ovog nezaboravnog perioda mojeg školovanja.

Sadržaj

1. Uvod

Svrha ovog završnog rada jest pojednostaviti unos podataka u raspoložive tablice unutar podatkovne baze te omogućiti autonomnost validacijskih procesa nad tim podacima. Funkcija validacijskog sloja biva pokrenuta u trenutku prosljeđivanja podataka s klijentske strane. To su procesi skriveni od korisnika koji svojim izvršavanjem brinu o tome da zaprimljeni podaci budu konzistentni sa strukturom podataka koji se na strani baze očekuju. Podaci se s *frontend-a* šalju u JSON formatu prilikom čega korisnik mora voditi brigu o tome da atributni dijelovi atribut-vrijednost parova budu nazivom usklađeni imenima stupaca u tablici za koju je objekt namijenjen. Uvođenje JSON podrške u Oracle donosi fleksibilnost u izvođenju ovakvog zadatka. S obzirom na to da sadrži funkcije i metode koje omogućavaju pristup podacima na iznimno jednostavan način, lako je programski ispitivati traženu podudarnost. Usporedno JSON podršci, detaljnije informacije o tablicama i njihovim ograničenjima pruža tzv. Data Dictionary – repozitorij podataka o svim objektima unutar baze [8]. Sama literatura ne sadrži puno informacija o pristupu validaciji na ovaj način. Vjerojatnost je da nije često primjenjivan s obzirom na to da ovakve vrste provjera većim dijelom već postoje na razini same baze, a njihov rezultat u slučaju neispravnih unosa djeluje u obliku predefiniranih iznimaka. Bitno je napomenuti glavnu razliku: navedene provjere funkcioniraju samo za jednostavne oblike podataka i pojedinačne unose, suprotno validacijskom sloju koji dinamičkim SQL-om navedeno provodi nad kompleksnijim strukturama i eliminira ovisnost o definiranju konkretnih objekata.

O pobližim definicijama korištenih alata i njihovoj namjeni moguće je pročitati u poglavlju programske podrške. Sama srž primijene spomenutih tehnologija nalazi se u nastavku pod naslovom „Program“; dio dokumentacije koji sadrži prikaz cjelokupnog koda kao i sveobuhvatan opis njegove funkcije.

2. Korištene tehnologije

2.1 Programska podrška

S obzirom na to da se radi o programu koji se nalazi i pokreće na strani baze, za potrebe ovog projekta korištena su izričito programska okruženja u kojima se rukovodi pozadinskim dijelom aplikacija (engl. *backend*). U ovom slučaju se Oracle programska podrška pokazala najpogodnijom opcijom.

Oracle je globalna američka IT korporacija koja nudi i prodaje pregršt programskih rješenja i ostalih tehnoloških proizvoda [1]. Poznat je po vlastitim sustavima za održavanje baza podataka i aplikacijskih servisa u oblaku (engl. *cloud*). Kao jedan od takvih sustava jest Oracle Database čija je specifična karakteristika njegovo „multi-model“ svojstvo. Ono omogućava podršku više od jednog modela podataka, po čemu se razlikuje od većine drugih *database* sustava [2]. Njegova glavna namjena je olakšano pokretanje i održavanje online transakcijskih procesa te skladištenje podataka [4].

2.1.1 ***Oracle Database XE i SQL Developer***

U ovom radu koristi se jedna od značajnijih inačica Oracleove baze podataka. Oracle Database Express Edition (skraćeno Oracle Database XE) je manje opsežna, besplatna verzija navedenog podatkovnog sustava. S njime je moguće:

- voditi administraciju baze,
- kreirati tablice, poglede (engl. *view*) i ostale objekte,
- obavljati uvoz, izvoz te prikaz podataka u tablicama,
- pokretati upite i SQL skripte [3].

Verzija korištena za ovaj projekt jest pretposljednja u nizu, dostupna za korištenje od veljače 2018. godine; verzija 18c. Kao povezani alat, uz njega dolazi također besplatan programski paket SQL Developer-a. SQL Developer je integrirano razvojno okruženje za olakšan razvoj i održavanje Oracleovih baza. Njegovo sučelje sadrži sve potrebno za neometan *end-to-end development* PL/SQL aplikacija. Samo neke od funkcionalnosti su konekcijsko okno, radni list (engl. *worksheet*), DBA konzola, izvještajno sučelje (engl. *report interface*) uz mnoštvo drugih elemenata [9]. Oba softvera nude korisniku jednostavnost instalacije i upravljanja programskim postavkama.

2.2 Data Dictionary

Data Dictionary je jedna od najvažnijih komponenti; kako u Oracle-u, tako i u ovome projektu. To je set tablica dostupan izričito za čitanje (engl. *read-only*) koje sadrže kompletne informacije o cijeloj bazi; definicije svih shematskih objekata u bazi, informacije o alociranim prostorima za te objekte, podaci o ograničenjima (engl. *constraints*) i slično [7].

Postoje tri različita prefiksa pomoću kojih pristupamo određenim pogledima: USER čiji je „vidokrug“ samo ono unutar njegove vlastite sheme, ALL čiji opseg obuhvaća sve ono čemu korisnik može pristupiti te DBA prefiks koji predstavlja pogled iz perspektive administratora baze - svi objekti u svim korisničkim shemama [7]. U svrhu pristupanja podacima neizbježnima za ostvarivanje ovog zadatka, korišten je pogled ALL_TAB_COLUMNS koji opisuje sve stupce tablica, pogleda i skupova podataka (engl. *clusters*) [8]. Pomoću njega su dobiveni podaci o stupcima u tablicama kao što su „vlasnik“ tablice u kojima se ti stupci nalaze, tipovi podataka koje primaju, nužnost dodjeljivanja vrijednosti (engl. *nullable*) i dozvoljena duljina podatka.

2.3 JSON

JSON (JavaScript Object Notation) je datotečni format otvorenog standarda koji služi za razmjenu podataka u ljudski čitljivom obliku. Njegova sintaksa podržava serijalizaciju tekstualnih oblika podataka od kojih tipovi mogu biti polja, brojevi, tekstovi (engl. *string*), *boolean* (logičke istinite i neistinite vrijednosti) te *null* (tip koji opisuje nepostojeću vrijednost ili referencu na objekt) [10]. Atribut-vrijednost, također nazivan i ključ-vrijednost par (engl. *key-value pair*), je način prikaza podataka u računalnim sustavima gdje *key*, najčešće proizvoljnog naziva, predstavlja ime koje služi kao opis, a *value* vrijednost dodijeljenog podatka [11]. Na sljedećoj slici *Slika 2.1* moguće je vidjeti kako ključ-vrijednost par djeluje unutar JSON formata na primjeru objekta u kojem su pohranjene informacije o nekom korisniku.

Slika 2.1 Primjer JSON formata podataka

2.3.1 ***JSON u Oracle-u***

Kao što je ranije spomenuto, funkcionalnost ovog zadatka postiže se programskim proširenjem u Oracle bazi za JSON oblike podataka [5]. Prva verzija ove podrške predstavljena je s izlaskom Oracle Database 12c verzije. Ona uvodi brojne JSON uvjete i funkcije od kojih su u ovom slučaju korištene jedino sljedeće:

- JSON_OBJECT_T + .put( ) – tip objekta koji odgovara JSON strukturi pomoću kojeg je moguće kreirati njegove instance te put( ) procedura kojom se postavljaju vrijednosti JSON objekta [6] kao što je vidljivo u sljedećem primjeru (*Slika 2.2* kao primjer stvaranja instance JSON objekta te *Slika 2.3* koja opisuje rezultat primjene metode put( ) na ulazni JSON parametar)

```
create or replace NONEDITIONABLE procedure procedura(ulazni_string in CLOB) AS
    json_objekt JSON_OBJECT_T;
  BEGIN
    json_objekt := JSON_OBJECT_T(ulazni_string);
    json_objekt.put('KEY PRIMJER', 'VALUE PRIMJER');

    dbms_output.put_line(json_objekt.TO_STRING);
  END;
```

Slika 2.2 Primjer korištenja JSON_OBJECT_T i put( ) metode

{"projekt":"p_izivkovic","session_id":"eal2fc36a71da30289447ae6887064d6","KEY PRIMJER":"VALUE PRIMJER"}

Slika 2.3 Rezultat nadodanih vrijednosti na JSON objekt

- IS JSON – uvjet koji vraća istinitu (engl. *true*) ili neistinitu (engl. *false*) vrijednost ovisno je li format podataka uistinu JSON oblika; također osigurava formiranje podatka u JSON oblik unutar stupaca tablice koje su određene ovim uvjetom ako je njihova sintaksa validna [5]. *Slika 2.4* prikazuje dodavanje ograničenja stupca na JSON format koristeći IS JSON uvjet.

```
CREATE TABLE JSON_PRIMJER
   (
    JSON_SADRZAJ CLOB
    CONSTRAINT ensure_json CHECK (JSON_SADRZAJ IS JSON)
   );
```

Slika 2.3 Primjer kreiranja tablice sa stupcem čiji sadržaj mora biti JSON formata

- JSON_VALUE – funkcija koja uzima određenu skalarnu vrijednost iz zadanog JSON objekta te ju vraća u nekom od SQL-ovih tipova podataka. Kao prvi parametar prima JSON objekt, a kao drugi parametar specificirano svojstvo (engl. *property*) čiju vrijednost želimo dobiti. Klauzula RETURNING kojom se određuje povratni tip podatka može, a i ne mora biti definirana. Ako ne biva određena, vrijednost će biti vraćena u tipu VARCHAR2. Druga dostupna opcija koju možemo definirati je tip NUMBER [5]. *Slika 2.5* prikazuje primjer poziva funkcije nad proizvoljnim JSON podacima, a *Slika 2.6* rezultat pokrenutog upita.

```
SELECT JSON_VALUE('{"a":"100", "b":"200"}', '$.a') FROM DUAL;
```

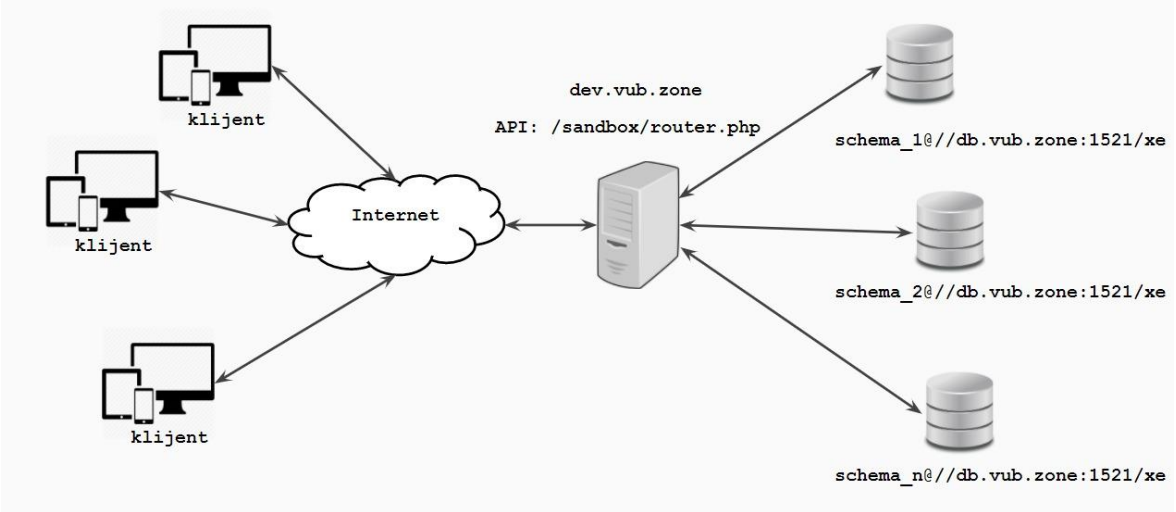
Slika 2.4 Primjena JSON_VALUE funkcije nad nekim JSON objektom

```
JSON_VALUE('{"A":"100","B":"200"}','$.A')
------------------------------------------
100
```

Slika 2.5 Rezultat SQL upita u tipu VARCHAR2

3. Program

3.1 Arhitektura



Slika 3.1 Shema arhitekture aplikacije

Arhitektura programa razrađena je u tri razine. *Slika 3.1* prikazuje interakciju različitih slojeva aplikacije. Kako bi pozadinska provjera podataka ispravno funkcionirala, potrebno je odrediti središnju točku između prednje (engl. *frontend*) i pozadinske strane (engl. *backend*). Ovo se postiže na način da se prednjem dijelu aplikacije prije samog zahtjeva za unos podataka osigura informacija o tome koji stupci u željenoj tablici postoje i kakav tip podatka mogu primiti. Tako osoba koja podatke unosi u samome početku može umanjiti mogućnost neispravnih unosa, a time i skratiti vrijeme izvršavanja željenih upita. Nakon što je definirao za koju tablicu su podaci namijenjeni, oni se šalju kao parametar glavnoj funkciji koja ima ulogu gore navedene središnje točke. Ona će razraditi ulazne podatke na način da će iščitati i spremiti sistemske informacije u lokalne varijable. Pomoću

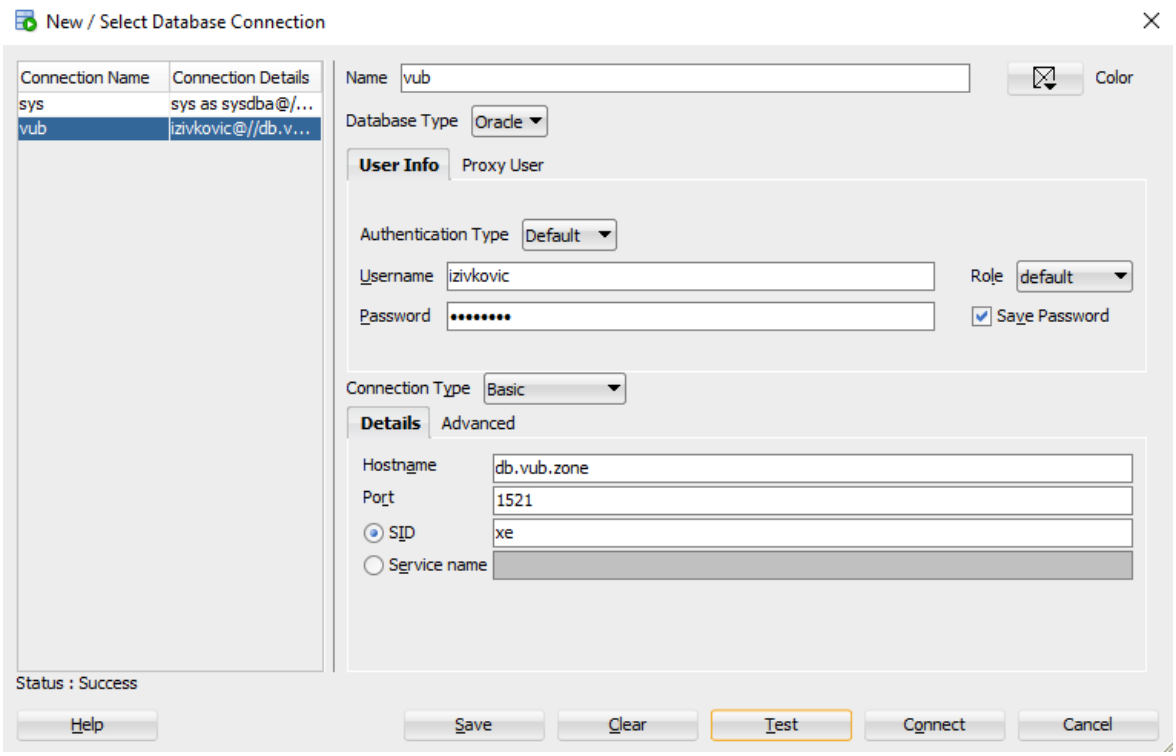
tih informacija može odrediti kamo dalje usmjeriti objekt. Neke od njih su pošiljatelj, njegov identifikacijski broj sesije, procedura kojoj želi poslati objekt te sam JSON sadržaj. U slučaju da informacija o ciljanoj proceduri zadovoljava uvjet nekog od izvršnih blokova u glavnoj funkciji, podaci se prosljeđuju tom dijelu koda i nadalje izvršavaju procese koje on zadaje. U ovome slučaju oni su najprije proslijeđeni na spremanje u tablicu koja evidentira promjene na shemi. Spremanje u tablicu okida *trigger* koji će pozvati funkciju zaduženu za detaljnu provjeru podataka. Ona osigurava podudarnost ulaznog objekta s objektima na bazi. Funkcija daje povratnu informaciju o tome je li poslana struktura zadovoljila sve tražene uvjete. U slučaju da su podaci prošli provjeru bez odstupanja, omogućen im je unos u tablicu za koju su prvotno bili namijenjeni. U slučaju da provjere ipak ne prođu, za svaki potencijalan neispravan unos unaprijed je definirana varijabla koja skladišti podatke s preciziranim neispravnim parametrima ovisno o grešci koju su izazvali. Više o ovome pristupu u samom opisu koda.

3.2 Opis koda

Kod programa pisan je engleskim jezikom radi bolje usklađenosti sintakse s integriranim programskim svojstvima. Također se postiže univerzalnost programa te potencijalna distribucija projekta za primjenu u pravim aplikacijama. Pojedini dijelovi sintakse na nekim linijama prelaze u novi red kako bi se čitljivost koda podudarala s formatom ovog dokumenta.

3.2.1 ***Konekcija i postizanje konekcije***

Za početak, najprije treba ostvariti konekciju na bazu podataka. Za potrebe ovog projekta koristi se konekcija na server Veleučilišta u Bjelovaru. Naziv konekcije je proizvoljan dok su korisničko ime i lozinka prethodno dodijeljeni od strane administratora. Ime *host-a*, *port* koji se koristi te SID unaprijed su definirani i samo ih je potrebno ispravno upisati. Na slici *Slika 3.2* je prikazan prozor povezivanja na bazu Veleučilišta s navedenim podacima. Pritiskom na gumb „*Test*“ moguće je provjeriti ispravnost vjerodajnica, a spajanje se izvršava odabiranjem gumba „*Connect*“.

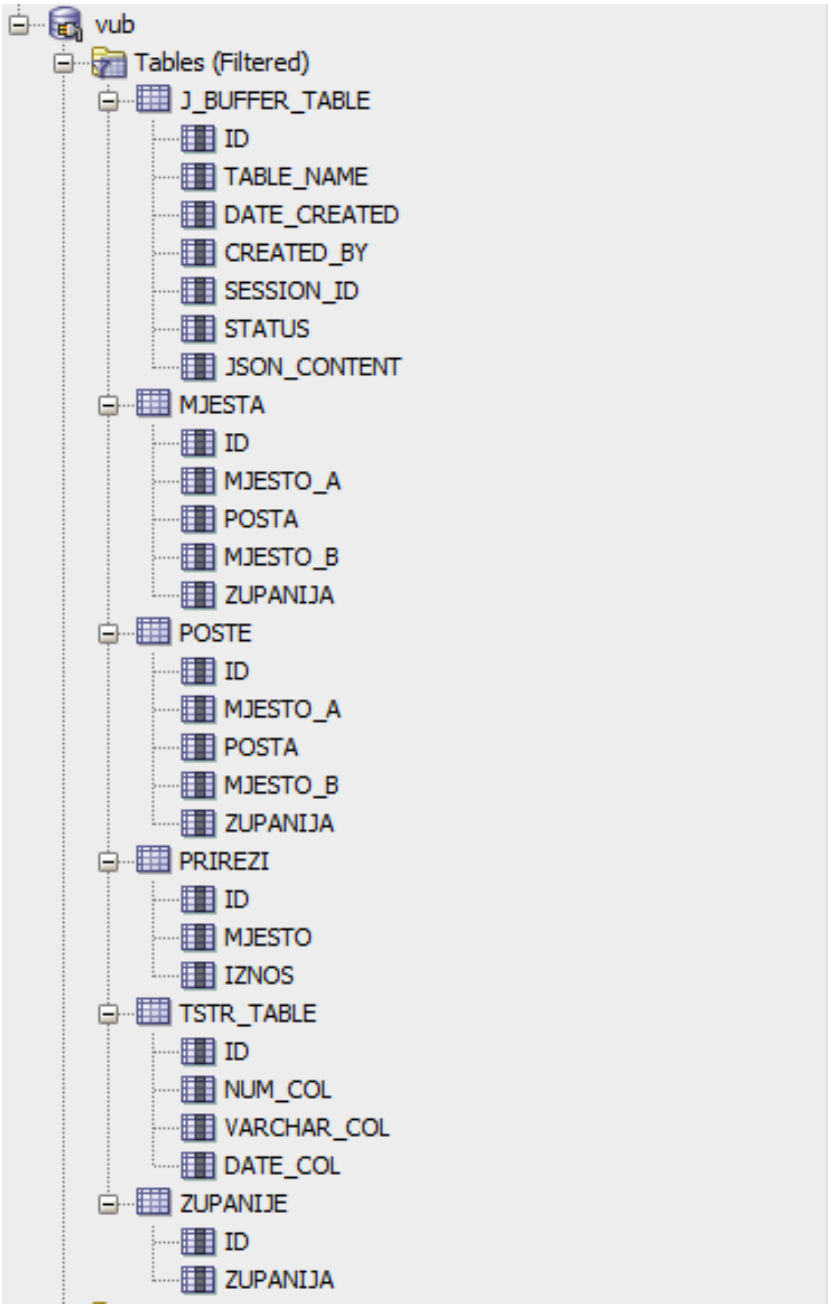


Slika 3.2 Spajanje na bazu

3.2.2 ***Objekti na bazi***

3.2.2.1 ***Tablice***

Nakon uspješnog spajanja na bazu, u prozoru gdje se nalazi popis spremljenih konekcija sada je omogućeno proširivanje padajućeg izbornika konekcije s kojom smo povezani. U tom izborniku se nalazi popis svih objekata na bazi. Sljedeća *Slika 3.3* prikazuje dostupne tablice i nazive njihovih stupaca unutar korištene sheme.



Slika 3.3 Prikaz dostupnih tablica

Prva po redu, tablica pod imenom „J_BUFFER_TABLE“, je od prikazanih tablica najbitnija ovome zadatku. Ona predstavlja *log* spremnik za sve dolazne podatke u bazu. Svaki pokušaj okidanja *insert-a* nad bilo kojom drugom tablicom rezultirat će najprije spremanjem u J_BUFFER_TABLE. Svrha ove tablice je voditi detaljan zapisnik o podacima popratnima činu unosa. Iz prikazanog je moguće vidjeti da ti podaci odgovaraju stupcima u tablici. Svaka tablica će imati svoj identifikacijski stupac koji će najčešće biti postavljeni kao primarni ključ, pa tako i tablica u pitanju. TABLE_NAME je VARCHAR2

tipa i predstavlja ime tablice za koju je korisnik odredio da su podaci namijenjeni. Kod DATE_CREATED stupca sam naziv govori kako se radi o detaljnom zapisu datuma i vremena nastanka unosa koristeći TIMESTAMP tip podatka. CREATED_BY bilježi korisnika koji unosi promjenu. SESSION_ID je znakovni niz identifikacijskog broja sesije koji se prosljeđuje s *frontend-a*. STATUS je vrlo bitna informacija i ujedno jedini stupac u ovoj tablici kojem je omogućeno ažuriranje (engl. *update*). Njegova vrijednost (0 ili 1) nam govori što se dogodilo po završetku izvođenja provjera nad *input* podacima. Ako se vrijednost obnovi nulom, to znači da su provjere pravilno izvršene i podaci su uspješno spremljeni na svoje mjesto. U slučaju da se vrijednost ažurira na broj 1, u nekom dijelu provjera je program zakazao te je potrebno obratiti pažnju na greške na koje dignute iznimke ukazuju. Finalno, JSON_CONTENT, sadrži JSON format ulaznih podataka koji se raspodjeljuju prema odgovarajućim stupcima. JSON format je osiguran ranije spomenutim IS JSON uvjetom u obliku *constraint-a* što je vidljivo na slici *Slika 3.5*, dok su svi ostali podaci o tablici vidljivi na *Slika 3.4.* Navedene informacije dostupne su odabirom opcija „*Columns*“ i „*Constraints*“, a odabirom „*Data*“ izbornika dobivamo uvid u sve zapise u tablici. *Slika 3.6* je primjer polja upisanih tijekom ostvarivanja zadatka.

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(1,0)	Yes	(null)	1	(null)
2	NUM_COL	NUMBER(2,0)	Yes	(null)	2	(null)
3	VARCHAR_COL	VARCHAR2(5 BYTE)	Yes	(null)	3	(null)
4	DATE_COL	DATE	Yes	(null)	4	(null)

Slika 3.4 Prikaz izbornika Columns

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS
1	ENSURE_JSON	Check	JSON_CONTENT IS JSON	(null)	(null)	(null)	(null)	ENABLED
2	PK_J_BUFFER_ID	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED

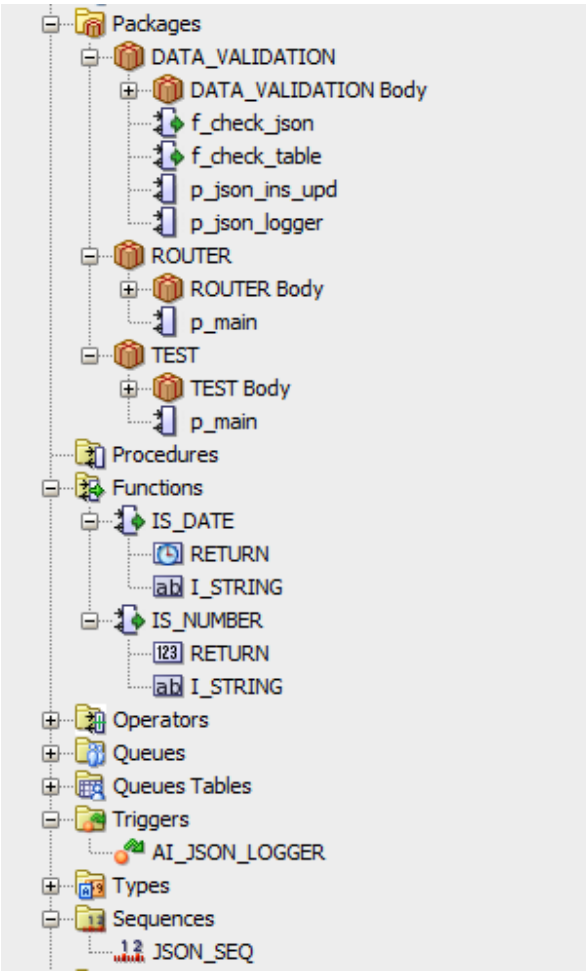
Slika 3.5 Prikaz izbornika Constraints

	ID	TABLE_NAME	DATE_CREATED	CREATED_BY	SESSION_ID	STATUS	JSON_CONTENT
1	9000	TSTR_TABLE	(null)	(null)	(null)	(null)	{"ID":"6", "NUM_COL":"2", "VARCHAR_COL":"new"}
2	800	TSTR_TABLE	(null)	(null)	(null)	(null)	{"ID":"6", "NUM_COL":"2", "VARCHAR_COL":"new"}
3	507	PRIREZI	21-OCT-20 08.20.31.726061000 PM	ggrubisic@gmail.com	48c7d3271492f54ff0bcb06071fbfa86	0	{"ID":"502","MJESTO":"test","IZNOS":"43"}
4	497	PRIREZI	21-OCT-20 02.30.25.716727000 PM	IZIVKOVIC	48c7d3271492f54ff0bcb06071fbfa86	0	{"ID":"502","MJESTO":"test","IZNOS":"1"}
5	496	PRIREZI	21-OCT-20 02.18.01.649110000 PM	IZIVKOVIC	48c7d3271492f54ff0bcb06071fbfa86	0	{"ID":"502","MJESTO":"test","IZNOS":"1"}
6	486	PRIREZI	21-OCT-20 01.43.36.901533000 PM	IZIVKOVIC	48c7d3271492f54ff0bcb06071fbfa86	0	{"ID":"500","MJESTO":"test","IZNOS":"2"}
7	485	PRIREZI	21-OCT-20 01.36.27.347803000 PM	IZIVKOVIC	48c7d3271492f54ff0bcb06071fbfa86	(null)	{"ID":"500","MJESTO":"NOVA","IZNOS":"2"}
8	484	PRIREZI	21-OCT-20 01.36.10.955279000 PM	IZIVKOVIC	48c7d3271492f54ff0bcb06071fbfa86	(null)	{"ID":"500","MJESTO":"NOVA","IZNOS":"2"}
9	401	TSTR_TABLE	17-OCT-20 11.09.11.728317000 PM	IZIVKOVIC	f557bf7d39d1db8ab4eb14559lcbba16	(null)	{"ID":7,"VARCHAR_COL":"BRA"}
10	381	TSTR_TABLE	17-OCT-20 10.56.21.244795000 PM	IZIVKOVIC	f557bf7d39d1db8ab4eb14559lcbba16	(null)	{"ID":9,"NUM_COL":"8"}

Slika 3.6 Prikaz izbornika Data

Preostale tablice služe za skladištenje legitimnih podataka koji se koriste za potrebe ostalih aplikacija, osim tablice TSTR_TABLE. Ona razvojnom inženjeru pomaže postići funkcionalnost koda tako da ne mora dirati tablice koje sadrže osjetljive podatke.

Na sljedećoj slici *Slika 3.7* prikazan je ostatak objekata na bazi i struktura nekih od njih. U te objekte, od dostupnih, spadaju paketi (engl. *packages*), funkcije (engl. *functions*), trigeri (engl. *triggers*) te sekvence (engl. *sequences*).



Slika 3.7 Objekti Packages, Triggers, Sequences

3.2.2.1 ***Funkcije***

Funkcije IS_DATE i IS_NUMBER kreirane su s namjerom da se prilikom provjera JSON objekata automatizira sam postupak. Funkcija IS_DATE, kao što je vidljivo na slici *Slika 3.8*, parsira ulazni *string* i provjerava odgovara li tipom ijednom formatu datuma. Ako je format ispravan, vratit će datum u tome obliku, a u slučaju neispravnog formata vratit će NULL vrijednost što će izazvati podizanje iznimke na strani provjere.

```
create or replace NONEDITIONABLE FUNCTION IS_DATE (i_string varchar2)
RETURN DATE AS
BEGIN

  RETURN TO_DATE(i_string);

  EXCEPTION
    WHEN OTHERS THEN
        RETURN NULL;

END IS_DATE;
```

Slika 3.8 Funkcija IS_DATE

Funkcija IS_NUMBER na slici *Slika 3.9* kao parametar također prima *string* vrijednost iz JSON objekta, no vraća vrijednosti 0 ili 1 ovisno o rezultatu integrirane funkcije TO_NUMBER koja ulazni tekst pokuša pretvoriti u brojku. Za prolazak provjere kao povratnu vrijednost dobit ćemo nulu, a u slučaju iznimke broj jedan koji će na strani provjere JSON objekta također obavještavati kako tip dobivenog podatka nije broj, a trebao bi biti.

```
create or replace NONEDITIONABLE FUNCTION IS_NUMBER (i_string IN VARCHAR2)
   RETURN INT
AS

   l_new_num number;

BEGIN
    l_new_num := TO_NUMBER(i_string);

    RETURN 0;


    EXCEPTION
    WHEN VALUE_ERROR THEN
       RETURN 1;
    WHEN OTHERS THEN
       RETURN 1;

END is_number;
```

Slika 3.9 Funkcija IS_NUMBER

3.2.2.2 ***Sekvence***

Uloga sekvence JSON_SEQ je uvećavati vrijednost ID stupca u J_BUFFER_TABLICI za 1 kako bi pri svakom unosu bila osigurana jedinstvena nova vrijednost ID polja. Vrijednost sekvence ima mogućnost uvećavanja počevši od broja jedan do maksimalnog dozvoljenog raspona. *Slika 3.10* je prikaz sekvence i njezinih detalja.

	Name	Value
1	CREATED	12-OCT-20
2	LAST_DDL_TIME	12-OCT-20
3	SEQUENCE_OWNER	IZIVKOVIC
4	SEQUENCE_NAME	JSON_SEQ
5	MIN_VALUE	1
6	MAX_VALUE	999999999999999999999999999
7	INCREMENT_BY	1
8	CYCLE_FLAG	N
9	ORDER_FLAG	N
10	CACHE_SIZE	20
11	LAST_NUMBER	441
12	SCALE_FLAG	N
13	EXTEND_FLAG	N
14	SESSION_FLAG	N
15	KEEP_VALUE	N
16	DUPLICATED	N
17	SHARDED	N

Slika 3.10 Sekvenca JSON_SEQ

3.2.2.3 ***Trigeri***

Jedini postojeći *trigger*, a ključan za pravilno funkcioniranje sustava validacije je *trigger* AI_JSON_LOGGER. Ovaj *trigger* implementira komponente DATA_VALIDATION paketa i odgovoran je za kronološki redoslijed radnji odnosno izvršavanja provjera. Kao što je moguće iščitati iz njegovog prefiksa, *trigger* je tipa AFTER INSERT i vrši se nad tablicom J_BUFFER_TABLE što znači da će pri svakom unosu u tablicu biti pokrenut i za svaki unos izvršavati naredbe koje sadrži.

```
create or replace NONEDITIONABLE TRIGGER AI_JSON_LOGGER
AFTER INSERT ON J_BUFFER_TABLE
FOR EACH ROW
DECLARE
    i_id j_buffer_table.id%type;
    i_table_name all_tab_columns.column_name%type;
    i_json CLOB;
    o_error varchar2(32767);
    f_check_res number;
    e_iznimka EXCEPTION;
BEGIN
  i_id := :NEW.id;
  i_table_name := :NEW.table_name;
  i_json := :NEW.json_content;

  f_check_res := data_validation.f_check_json(i_id, i_table_name, i_json, o_error);

  IF f_check_res = 0 THEN
    data_validation.p_json_ins_upd(i_id, i_table_name, i_json);
  ELSE
    RAISE e_iznimka;
  END IF;

  EXCEPTION
   WHEN e_iznimka THEN
     raise_application_error(-20001, o_error);
   WHEN OTHERS THEN
     raise_application_error(-20000, 'Trigger error.');
END;
```

Slika 3.11 Trigger AI_JSON_LOGGER

Slika 3.11 predstavlja kod kreiranja *triggera*. Varijable *i_id, i_json* i *i_table_name* sve odgovaraju stupcima tablice J_BUFFER_TABLE. Pomoću operatora :NEW dodjeljujemo im posljednju vrijednost čije je unošenje pokrenulo sam *trigger*. Unutar *triggera* dobivene se vrijednosti prosljeđuju funkciji F_CHECK_JSON te proceduri P_JSON_INS_UPD. Funkcija F_CHECK_JSON će pomoću proslijeđenih podataka provesti provjeru nad JSON objektom kako bi dala povratnu informaciju zadovoljavaju li svi podaci odgovarajuće uvjete i odgovaraju li tipovima stupaca u koje bi trebali biti uneseni. Ako se tijekom rada funkcije ne jave nikakve programske iznimke, ona će vratiti vrijednost 0 što je korisniku informacija kako su svi podaci ispravni te će dalje omogućiti izvršavanje IF uvjeta. Procedura P_JSON_INS_UPD će u tom slučaju provjeriti postoji li u tablici zapis s ID vrijednosti koja se nalazi u JSON-u te će za nađeno polje obaviti *update*, a za nenađeno *insert*. Ako bilo što drugo prilikom obavljanja procesa u *triggeru* pođe po krivu, korisniku će biti javljena iznimka na razini *triggera*.

U narednim poglavljima bit će detaljnije objašnjeno kako točno ova funkcija i procedura koje su dio DATA_VALIDATION paketa rade svoj zadatak i koje sve podatke koriste kako bi to postigle.

3.2.2.4 ***Paketi***

DATA_VALIDATION paket je glavni dio zadatka koji sadrži sve funkcije i procedure odgovorne za validaciju podataka kao i njihov unos i ažuriranje.
Paketu se definiraju zaglavlje (engl. *header*) i tijelo (engl. *body*). Zaglavlje se sastoji od popisa svih funkcija i procedura koje su dostupne i izvan paketa. *Slika 3.12* sadrži specifikacije za paket DATA_VALIDATION. Njihov format mora opisivati vrstu (procedura ili funkcija), imena tih elemenata, njihove parametre i po potrebi *return* tipove.

```
create or replace NONEDITIONABLE package data_validation as

  function f_check_table(i_table_name in varchar2) RETURN varchar2;

  procedure p_json_logger(i_userid in number, i_table_name in varchar2,
                          i_session_id in varchar2, i_json in CLOB,
                              o_error out nocopy varchar2);

  function f_check_json(j_buff_id number, i_table_name in varchar2,
                          i_json in CLOB,
                              o_error out nocopy varchar2) RETURN number;

  procedure p_json_ins_upd(j_buff_id number, i_table_name in varchar2,
                              i_json in CLOB);

end data_validation;
```

Slika 3.12 Zaglavlje DATA_VALIDATION paketa

Tijelo paketa sadrži definicije gore prikazanih funkcija. Tijelo DATA_VALIDATION paketa otvaramo funkcijom F_CHECK_TABLE čiji je kod prikazan na slici *Slika 3.13.*
Kao što je vidljivo iz slike, funkcija kao ulazni parametar prima ime tablice, a vraća VARCHAR2 vrijednost.

```
function F_CHECK_TABLE(i_table_name in varchar2) RETURN varchar2 AS
      l_data_type all_tab_columns.data_type%type;
      l_column_name all_tab_columns.column_name%type;
      l_json_table_query varchar2(32767);
  BEGIN
      FOR x IN (
          SELECT
              table_name,
              data_type,
              column_name
          FROM
              ALL_TAB_COLUMNS
          WHERE
              table_name = i_table_name)
      LOOP
          l_data_type := x.data_type;
          l_column_name := x.column_name;
          l_json_table_query := l_json_table_query || CHR(10) || '"' ||
                                l_column_name || '":"' || l_data_type || '",';
      END LOOP;

    l_json_table_query := RTRIM(l_json_table_query, ',');
    l_json_table_query := TRIM(LEADING CHR(10) FROM l_json_table_query);
    l_json_table_query := '{' || l_json_table_query || '}';

    RETURN l_json_table_query;

    EXCEPTION
    WHEN NO_DATA_FOUND THEN
      raise_application_error(-20000, 'No data was found at checking table.');
      RETURN NULL;
    WHEN OTHERS THEN
      raise_application_error(-20001, 'Dev error at checking table.');
      RETURN NULL;

END F_CHECK_TABLE;
```

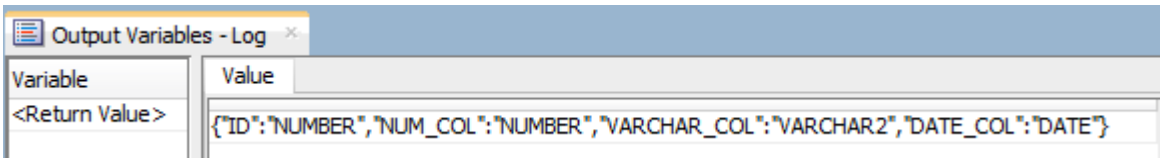
Slika 3.13 Funkcija F_CHECK_TABLE

U opisu korištene programske podrške objašnjeno je kako je za potrebe dobivanja krucijalnih podataka o tablicama korišten Data Dictionary, konkretnije set podataka pod imenom ALL_TAB_COLUMNS. Ovdje se prvi put primjenjuje navedeno. Varijablama *l_data_type* i *l_column_name* dodjeljujemo tip stupaca iz spomenute tablice pomoću %TYPE atributa. Njime se deklariraju varijable na temelju već postojećih koje svojstvima odgovaraju jedne drugima. Na isti način će i u svim ostalim funkcijama i procedurama ovog paketa biti deklarirane lokalne varijable koje će predstavljati neki podatak iz tablice ALL_TAB_COLUMNS. Varijabla *l_json_table_query* je tekstualni spremnik u kojeg ćemo naknadno spremati rezultat programa te na posljetku, u slučaju ispravnog izvršavanja

funkcije, njegovu vrijednost i vraćati. U suprotnom će biti dignuta prilagođena iznimka, a povratna vrijednost će biti *null.*

Svakom iteracijom kroz stupce u ALL_TAB_COLUMNS i prikupljanjem podataka iz nje prvotno praznu varijablu *l_json_table_query* punit će „konkatenacija“ tih podataka zajedno sa *string-ovima* koji služe kao delimiteri. Nakon dohvaćanja svih dostupnih stupaca, izlazi se iz petlje i uređuje dobiveni *string* funkcijama RTRIM koja će maknuti nepotreban zarez s desne strane teksta te TRIM koji će maknuti početni karakter za prelaženje u novi red. Kada mu je višak znakova oduzet te su mu dodijeljene vitičaste zagrade, kako priliči JSON formatu, *string* je spreman za prikaz korisniku.

Namjena ove funkcije je dobiti strukturu koju valja zadovoljiti s klijentske strane. Korisnik mora biti svjestan kakav oblik JSON-a i kakve podatke slati bazi pošto će mu ispravan rezultat vraćati JSON format atribut-vrijednost parova; atributni dijelovi će biti nazivi stupaca, a vrijednosti njihovi tipovi podataka. Rezultat funkcije na primjeru tablice J_BUFFER_TABLE može se proučiti na slikama *Slika 3.14* i *Slika 3.15.*



Slika 3.14 Output Variables log

```
{"ID":"NUMBER",
"TABLE_NAME":"VARCHAR2",
"DATE_CREATED":"TIMESTAMP(6)",
"CREATED_BY":"VARCHAR2",
"SESSION_ID":"VARCHAR2",
"STATUS":"NUMBER",
"JSON_CONTENT":"CLOB"}
```

Slika 3.15 Rezultat funkcije F_CHECK_TABLE

Nadalje, kada je korisnik dobio uvid u dozvoljenu strukturu, procedura P_JSON_LOGGER će se pozivati pri samom primitku nekog JSON-a na bazu. Ona je odgovorna za spremanje JSON-a i popratnih podataka u J_BUFFER_TABLE. Na *Slika 3.16* nalazi se kod procedure. Suprotno funkciji, procedure ne podržavaju vraćanje vrijednosti već su zadužene samo za okidanje određenih akcija.

```
procedure P_JSON_LOGGER(i_userid in number, i_table_name in varchar2,
                        i_session_id in varchar2, i_json in CLOB,
                                o_error out nocopy varchar2) AS
    l_json CLOB;
    l_obj JSON_OBJECT_T;
    update_status varchar2(3200);
    l_rowtype j_buffer_table%rowtype;
    l_email common.korisnici.email%type;
  BEGIN
      SELECT email INTO l_email FROM common.korisnici WHERE ID = i_userid;
      update_status := 'UPDATE j_buffer_table SET STATUS=:1 WHERE ID=:2';
      l_obj := JSON_OBJECT_T(i_json);
      l_json := l_obj.TO_STRING;
      l_rowtype.id := JSON_SEQ.nextval;
      l_rowtype.table_name := i_table_name;
      l_rowtype.date_created := CURRENT_TIMESTAMP;
      l_rowtype.created_by := l_email;
      l_rowtype.session_id := i_session_id;
      l_rowtype.json_content := i_json;

      BEGIN
          INSERT INTO
             j_buffer_table
          VALUES
             l_rowtype;
          COMMIT;
          EXECUTE IMMEDIATE update_status USING 0, l_rowtype.id;
          EXCEPTION
           WHEN OTHERS THEN
              o_error := SUBSTR(SQLERRM, 12);
              EXECUTE IMMEDIATE update_status USING 1, l_rowtype.id;
              raise;
      END;

      EXCEPTION
        WHEN NO_DATA_FOUND THEN
          raise_application_error(-20002, 'No data found at logging table.');
          ROLLBACK;
        WHEN OTHERS THEN
          raise_application_error(-20003, 'Dev error at logging table.');
          ROLLBACK;

END P_JSON_LOGGER;
```

Slika 3.16 Procedura P_JSON_LOGGER

%ROWTYPE atribut može skladištiti jedno cijelo polje podataka na način kako je prikazano na *Slika 3.16.* Ulazni parametri su svi dolazni podaci tablice J_BUFFER_TABLE: ID korisnika, ime tablice, session ID i JSON podaci za tu tablicu, a izlazni greška funkcije za provjeru; ako se pojavi. Svakom stupcu pridružujemo njegovu vrijednost. Status provjere će se obnavljati nakon uspješnog *commit-a* ili nastale iznimke.

Broj 0 će upućivati na prolazak provjera, a broj 1 na neispravnost unosa. Primjer zapisa u tablicu J_BUFFER_TABLE postoji na *Slika 3.6.*

Sljedeća funkcija odgovorna je za sve potrebne provjere nad JSON podacima. Programski kod funkcije F_CHECK_JSON podijeljen je na više dijelova između slika *Slika 3.17*, *Slika 3.18*, *Slika 3.19* i *Slika 3.20* radi bolje preglednosti.

```
function F_CHECK_JSON(j_buff_id in number, i_table_name in varchar2, i_json in CLOB,
                                  o_error out nocopy varchar2)
                                              RETURN number AS
      l_json CLOB;
      l_json_obj JSON_OBJECT_T;
      l_column_name all_tab_columns.column_name%type;
      l_data_type all_tab_columns.data_type%type;
      l_data_length all_tab_columns.data_length%type;
      l_nullable all_tab_columns.nullable%type;
      l_property varchar(400);
      update_status varchar2(3200);
      error_number number;
      error_message varchar2(32767);
      e_exception EXCEPTION;
```

Slika 3.17 Prvi dio F_CHECK_JSON

Na slici *Slika 3.17* nalaze se ulazni parametri funkcije koji su: vrijednost ID stupca J_BUFFER_TABLE, ime tablice kojoj pripadaju JSON podaci i JSON objekt, izlazni parametar je greška, a povratna vrijednost broj 0 ili 1. Implementacija ove funkcije nalazi se na slici *Slika 3.11*. Vidljiv je i deklaracijski blok u kojem osim podataka iz ALL_TAB_COLUMNS deklariramo varijable *l_property* u kojem ćemo oblikovati *string* koji će služiti kao parametar funkciji JSON_VALUE; pomoću njega ćemo pristupati pojedinačnim podacima u JSON objektu.

```
BEGIN
    error_message := 'No error.';
    o_error := error_message;

    l_json_obj := JSON_OBJECT_T(i_json);
    l_json := l_json_obj.TO_STRING;

    FOR x IN (
        SELECT
            column_name,
            data_type,
            data_length,
            nullable
        FROM
            ALL_TAB_COLUMNS
        WHERE
            table_name = i_table_name)
    LOOP
        l_column_name := x.column_name;
        l_data_type := x.data_type;
        l_data_length := x.data_length;
        l_nullable := x.nullable;
        l_property := '$.' || l_column_name;
```

Slika 3.18 Drugi dio F_CHECK_JSON

Slika 3.18 prikazuje otvaranje naredbenog bloka izrazom BEGIN u kojem se ponovno vrši iteracija kroz stupce ALL_TAB_COLUMNS. Na samom početku je *error_message* definiran prikazanom porukom kako bi kao izlazni parametar po završetku izvođenja korisnik mogao primiti prilagođenu poruku. Poruka greške će se mijenjati ovisno o rezultatu provjera.

```
BEGIN
    --date conditions
    IF l_nullable = 'Y' AND l_data_type = 'DATE' AND (TRIM(json_value(l_json, l_property)) IS NULL
                                                      OR TRIM(json_value(l_json, l_property)) = '0') THEN
        CONTINUE;
    ELSIF l_data_type = 'DATE' AND IS_DATE(json_value(l_json, l_property)) IS NULL THEN --l_nullable = 'N'
        error_number := -20100;
        error_message := 'Column ' || l_column_name || ' has a type ' || l_data_type || ' and '''
                         || json_value(l_json, l_property) || ''' is not of the right date format.';
        o_error := error_message;
        RAISE e_exception;
    END IF;
    ----------------------------------------------------------------------------------------------------------
    --ID conditions
    IF l_property = '$.ID' AND (TRIM(json_value(l_json, l_property)) = '0' OR TRIM(json_value(l_json, l_property)) IS NULL) THEN
        error_number := -20101;
        error_message := 'Column ' || l_column_name || ' is an identifier and cannot be 0 nor empty.';
        o_error := error_message;
        RAISE e_exception;
    END IF;
    ----------------------------------------------------------------------------------------------------------
    --nullable
    IF l_nullable = 'N' AND TRIM(json_value(l_json, l_property)) IS NULL THEN
        error_number := -20102;
        error_message := 'Column ' || l_column_name || ' cannot be a null, but input JSON sends one.';
        o_error := error_message;
        RAISE e_exception;
    END IF;
```

Slika 3.19 Treći dio F_CHECK_JSON

Na slici *Slika 3.19* prikazani su uvjeti koje JSON vrijednosti moraju zadovoljavati. Prvi uvjet kaže da ALL_TAB_COLUMNS podaci na trenutnoj iteraciji moraju najprije istovremeno moći biti *null* vrijednost, biti tipa DATE te nemati dodijeljenu vrijednost. Za ovakav uvjet neće javljati grešku već će nastaviti izvođenje programa. Ovaj uvjet postoji ako se na strani baze dogodi pretvorba podatka tipa datuma u *string* 0 ili *null* vrijednost od kojih ni jedna nije ispravna, ali ih *nullable* svojstvo odobrava. U ovom trenutku bi se teoretski trebala dogoditi greška, no taj dio program neće pročitati kao grešku već će ga obraditi kasnije u proceduri P_JSON_INS_UPD kao što je prikazano na slici *Slika 3.22.* Ako u suprotnom vrijednost ipak ne smije biti *null* u tablici, a tip podatka je i dalje datum i funkcija IS_DATE vraća *null*, formirat će se prilagođena poruka i jedinstveni broj greške koji će se javiti prilikom dizanja iznimke. Ovakav pristup javljanju preciziranih grešaka korisniku primjenjuje se u svakom sljedećem uvjetu kroz programski kod.

Sljedeći uvjet sprječava postavljanje ID vrijednosti na nulu kako bi se očuvalo sekvencijsko svojstvo početka vrijednosti jedinicom.

Posljednji uvjet na ovoj slici osigurava neunošenje praznih vrijednosti za sve ostale tipove podataka ako njihova vrijednost ne može biti *null.*

```
            --data type condition
            IF l_data_type = 'NUMBER' AND is_number(json_value(l_json, l_property)) = 1 THEN
                error_number := -20103;
                error_message := 'Column ' || l_column_name || ' has a type ' || l_data_type || ' and '''
                                 || json_value(l_json, l_property) || ''' is not of the right numeric format.';
                o_error := error_message;
                RAISE e_exception;
            END IF;
            ------------------------------------------------------------------------------------------------------------
            --data length
            IF LENGTH(json_value(l_json, l_property)) > l_data_length THEN
                error_number := -20104;
                error_message := 'Column ' || l_column_name || ' has a length of ' || l_data_length
                                 || ' characters and ''' || json_value(l_json, l_property) || ''' is not of the right length.';
                o_error := error_message;
                RAISE e_exception;
            END IF;

            EXCEPTION
                WHEN e_exception THEN
                    raise_application_error(error_number, error_message);
                    RETURN 1;
        END;
    END LOOP;

    ------------------------------------------------------------------------------------------------------------
    RETURN 0; --if none of the conditions trigger, return a value that will make an ins/upd proc go through

END F_CHECK_JSON;
```

Slika 3.20 Četvrti dio F_CHECK_JSON

Na slici *Slika 3.20* prikazuje se slučaj ako trenutna iteracija sprema brojevni tip podatka, a funkcija IS_NUMBER vraća 1, gdje dolazi do greške s obzirom na to da funkcija u pitanju provjerava je li moguće ulazni *string* pretvoriti u broj. Uvjet nakon ograničava „*overflow*“

podataka tako da provjerava dozvoljenu duljinu podatka iz tablice i uspoređuje ju s duljinom ulaznog podatka. U bloku iznimke, ako do njega dođe, dići će se prilagođena iznimka te vratiti broj 1.

Posljednja procedura u paketu jest P_JSON_INS_UPD. Njezina uloga je umetanje ili ažuriranje polja u zadanoj tablici u slučaju prolaska svih provjera na razini F_CHECK_JSON funkcije. Poziv funkcije se također nalazi u *trigger-u* kao što pokazuje *Slika 3.11*, dok je tijelo ove funkcije također razloženo na sljedeći niz slika.

```
procedure P_JSON_INS_UPD(j_buff_id in number, i_table_name in varchar2, i_json in CLOB) AS
    l_property varchar(400); --variable that strings together a json property from table column name
    l_tab_values varchar(32767);
    l_data_type all_tab_columns.data_type%type;
    l_column_name all_tab_columns.column_name%type;
    l_insert_query varchar(32767);
    l_update_query varchar2(32767);
    ins_vals varchar(32767);
    upd_vals varchar(32767);
    l_json CLOB := i_json;
    l_json_id number;
    l_json_value varchar(800); --variable that stores value from input json for table insertion
    rows_found number;
    dynamic_select varchar2(400);
BEGIN
    FOR x IN (
        SELECT
            data_type,
            column_name
        FROM
            ALL_TAB_COLUMNS
        WHERE
            table_name = i_table_name)
```

Slika 3.21 Prvi dio P_JSON_INS_UPD

```
LOOP
    l_data_type := x.data_type;
    l_column_name := x.column_name;
    l_tab_values := l_tab_values || ', ' || x.column_name;
    l_property := '$.' || l_column_name;

    IF l_data_type = 'NUMBER' THEN
        l_json_value := json_value(l_json, l_property RETURNING NUMBER); --to_number()

        IF TRIM(l_json_value) IS NULL THEN
           ins_vals := ins_vals || ', ' || null;
        ELSE
           ins_vals := ins_vals || ', ' || l_json_value;
        END IF;

        IF TRIM(l_json_value) IS NULL OR l_property = '$.ID' THEN
           upd_vals := upd_vals || '';
        ELSE
           upd_vals := upd_vals || ', ' || l_column_name || '=' || l_json_value;
        END IF;

    ELSIF l_data_type = 'DATE' OR l_data_type = 'TIMESTAMP' THEN
        l_json_value := TO_DATE(json_value(l_json, l_property));
        ins_vals := ins_vals || ', ' || '''' || l_json_value || '''';

        IF TRIM(l_json_value) IS NULL OR TRIM(l_json_value) = '0' THEN
           upd_vals := upd_vals || '';
        ELSE
           upd_vals := upd_vals || ', ' || l_column_name || '=''' || l_json_value || '''';
        END IF;

    ELSE
        l_json_value := json_value(l_json, l_property); --no RETURNING clause will return in varchar(4000)
        ins_vals := ins_vals || ', ' || '''' || l_json_value || '''';

        IF TRIM(l_json_value) IS NULL OR TRIM(l_json_value) = '0' THEN
           upd_vals := upd_vals || '';
        ELSE
           upd_vals := upd_vals || ', ' || l_column_name || '=''' || l_json_value || '''';
        END IF;
    END IF;
END LOOP;
```

Slika 3.22 Drugi dio P_JSON_INS_UPD

Slika 3.21 pokazuje deklaracijski blok procedure i istu dosad korištenu iteraciju kroz stupce tablice ALL_TAB_COLUMNS. *Slika 3.22* je petlja zadužena za formiranje *insert* i *update* upita koristeći dinamički SQL. Kao što je ranije spomenuto u tekstu, za tip datuma kao i za brojevne tipove, uvjet ispravne vrijednosti se obrađuje u gore prikazanom kodu. Time se formira pravilna sintaksa upita jer bi u suprotnom vrijednosti ostale potpuno prazne. Ako je njihova vrijednost mogla biti *null*, onda se ti stupci jednostavno neće spremiti u upit za *update*. Što se tiče *update_vals* vrijedno je također spomenuti kako vrijednost ID-a isto neće moći biti nadodana u upit s obzirom da se prema ID vrijednosti vrši ažuriranje polja te je njegovu vrijednost u većini slučajeva zabranjeno mijenjati.

```
        l_tab_values := SUBSTR(l_tab_values, 3);
        ins_vals := SUBSTR(ins_vals, 3);
        upd_vals := SUBSTR(upd_vals, 3);
        l_json_id := json_value(l_json, '$.ID' RETURNING NUMBER);

        l_insert_query := 'INSERT INTO ' || i_table_name || ' (' || l_tab_values
                          || ') ' || 'VALUES (' || ins_vals || ')';
        l_update_query := 'UPDATE ' || i_table_name || ' SET ' || upd_vals
                          || ' WHERE ID=' || l_json_id;

        --checks if ID sent through JSON exists within a table
        dynamic_select := 'SELECT COUNT(*) FROM ' || i_table_name || ' WHERE ID = :1';

        EXECUTE IMMEDIATE dynamic_select INTO rows_found USING l_json_id;
        --if ID doesnt exist in the table, insert a new row and if it does,
        --update the existing one
        IF rows_found = 1 THEN
            EXECUTE IMMEDIATE l_update_query;
        ELSE
            EXECUTE IMMEDIATE l_insert_query;
        END IF;

        EXCEPTION
          WHEN NO_DATA_FOUND THEN
            raise_application_error(-20005, 'No data was found at insert/update.');
          WHEN OTHERS THEN
            RAISE_APPLICATION_ERROR(-20006, 'Dev error at insert/update.');

    END P_JSON_INS_UPD;
--------------------------------------------------------------------------------
END DATA_VALIDATION;
```

Slika 3.23 Treći dio P_JSON_INS_UPD

Na slici *Slika 3.23* prikazano je finalno formiranje *insert* i *update* upita. Dinamička SELECT COUNT tvrdnja u traženoj tablici traži vrijednost ID-a dobivenog iz JSON-a te sprema broj nađenih zapisa u varijablu *rows_found*. Ako u tablici zapis postoji, nad retkom gdje je upisan ID bit će izvršeno ažuriranje, a ako je ID nepostojeći, dolazni JSON podaci će se raspodijeliti prema odgovarajućim stupcima i umetnuti u tablicu.

3.2.3 ***Implementacija u ROUTER***

Od korištenih paketa preostaje ROUTER *package*. On je središnja točka arhitekture aplikacije koja zaprima podatke s *frontend-a* i usmjerava ih dalje po *backend-u*. *Slika 3.24* prikazuje skupljanje potrebnih podataka iz ulaznog JSON-a. S obzirom da ROUTER paket

ne pripada specifikaciji validacijskog sloja već je doslovno samo, kako mu sam naziv kaže, *router* aplikacije – služit će samo za pozivanje glavne procedure ovog projekta P_JSON_LOGGER čime će se dalje pokrenuti funkcija validacijskog sloja (*Slika 3.25*). Ovaj paket je napravljen tako da međuostalim svaku promjenu bilježi u *log* na „common“ shemi putem procedure P_LOGIRAJ kao i nastale iznimke koje se zajedno s rezultatom podprograma „backtrace“ koji *logira* iznimke spremaju u tzv. *error log* (*Slika 3.25*). Prilagođene greške nastale pri provjerama bilježit će se i dodavati kao novi *property* u lokalni JSON objekt. Kada kod ROUTER-a završi s izvršavanjem, prilagođeni JSON objekt će se vraćati *frontend*-u iz kojega se može pristupiti grešci (ako postoji).

```
create or replace NONEDITIONABLE PACKAGE BODY ROUTER AS
e_iznimka EXCEPTION;

  procedure p_main(p_in in varchar2, p_out out varchar2) AS
    l_obj JSON_OBJECT_T;
    l_string varchar2(4000);
    l_json_obj JSON_OBJECT_T;
    l_json_check varchar2(4000);
    l_procedura varchar2(40);
    l_tablica varchar2(60);
    l_session varchar2(60);
    l_userid number;
    o_error varchar2(1000);
  BEGIN
    l_obj := JSON_OBJECT_T(p_in);
    l_string := l_obj.TO_STRING;

    SELECT
        JSON_VALUE(l_string, '$.procedura' RETURNING VARCHAR2),
        JSON_VALUE(l_string, '$.table' RETURNING VARCHAR2),
        JSON_VALUE(l_string, '$.session_id' RETURNING VARCHAR2),
        JSON_VALUE(l_string, '$.json' RETURNING VARCHAR2),
        JSON_VALUE(l_string, '$.UserID' RETURNING NUMBER)
    INTO
        l_procedura,
        l_tablica,
        l_session,
        l_json_check,
        l_userid
    FROM DUAL;
```

Slika 3.24 Prvi dio ROUTER

```
    CASE l_procedura
    WHEN 'p_validate' THEN
        l_json_obj := JSON_OBJECT_T(l_json_check);
        l_json_check := l_json_obj.TO_STRING;

        BEGIN
            data_validation.p_json_logger(l_userid, l_tablica, l_session, l_json_check, o_error);

            EXCEPTION
                WHEN OTHERS THEN
                    common.p_logiraj('p_izivkovic', l_json_check);
                    COMMON.p_errlog('greska', dbms_utility.format_error_backtrace, SQLCODE, SQLERRM, l_string);
        END;
        l_obj.put('error', o_error);
    ELSE
        l_obj.put('h_message', ' Nepoznata metoda' || l_procedura);
        l_obj.put('h_errcod', 997);
    END CASE;
    p_out := l_obj.TO_STRING;
    commit;
  EXCEPTION
      when e_iznimka then
         null;
         rollback;
      when others then
         rollback;
         COMMON.p_errlog('router',dbms_utility.format_error_backtrace,SQLCODE,SQLERRM, l_string);

  END p_main;

END ROUTER;
```

Slika 3.25 Drugi dio ROUTER

3.3 Rezultati

Pod rezultate spadaju svi lokalni rezultati procedura i funkcija te oni globalni pri testiranju funkcije validacijskog sloja. U svrhu testiranja koriste se definicije unutar anonimnih blokova.

PL/SQL Block

```
1  DECLARE
2    J_BUFF_ID NUMBER;
3    I_TABLE_NAME VARCHAR2(200);
4    I_JSON CLOB;
5    O_ERROR VARCHAR2(200);
6    v_Return NUMBER;
7  BEGIN
8    J_BUFF_ID := 8;
9    I_TABLE_NAME := 'TSTR_TABLE';
10   I_JSON := '{"ID":"2", "NUM_COL":"4",
11                 "VARCHAR_COL":"ABCD", "DATE_COL":"prr"}';
```

Slika 3.26 Anonimni blok funkcije F_CHECK_JSON za "date condition"

```
ORA-20100: Column DATE_COL has a type DATE and 'prr' is not of the right date format.
ORA-06512: at "IZIVKOVIC.DATA_VALIDATION", line 164
ORA-06512: at "IZIVKOVIC.DATA_VALIDATION", line 164
ORA-06512: at line 13
```

Slika 3.27 Output iznimke za neispravan datum

Na slici *Slika 3.26* prikazan je *input* JSON definiran u anonimnom bloku s nasumično pisanim tekstom umjesto nekim formatom datuma, a na slici *Slika 3.27* rezultat izvršavanja bloka. Javlja se poruka greške nezadovoljenog uvjeta za datum kao što je opisano na *Slika 3.19*. Greška se javlja zato što prilikom provjere u F_CHECK_JSON funkciji funkcija IS_DATE ne nalazi način kako pretvoriti tekst „prr“ u ijedan od dozvoljenih formata datuma.

Pošto su u navedenoj testnoj tablici svi stupci *nullable*, u nekim od narednih blokova će pojedine vrijednosti biti izostavljene.

PL/SQL Block

```
1  DECLARE
2    J_BUFF_ID NUMBER;
3    I_TABLE_NAME VARCHAR2(200);
4    I_JSON CLOB;
5    O_ERROR VARCHAR2(200);
6    v_Return NUMBER;
7  BEGIN
8    J_BUFF_ID := 8;
9    I_TABLE_NAME := 'TSTR_TABLE';
10   I_JSON := '{"ID":"", "NUM_COL":"", "VARCHAR_COL":"ABCD", "DATE_COL":""}';
```

Slika 3.28 Anonimni blok funkcije F_CHECK_JSON za "id condition"

```
ORA-20101: Column ID is an identifier and cannot be 0 nor empty.
ORA-06512: at "IZIVKOVIC.DATA_VALIDATION", line 164
ORA-06512: at "IZIVKOVIC.DATA_VALIDATION", line 164
ORA-06512: at line 12
```

Slika 3.29 Output iznimke za izostavljenu ID vrijednost

Na slici *Slika 3.28* prikazan je *input* JSON definiran u anonimnom bloku za izostavljenom ID vrijednošću, a na slici *Slika 3.29* rezultat izvršavanja bloka. Iznimku za izostavljen ID ili vrijednost 0 obuhvaćat će jedna te ista poruka.

PL/SQL Block

```
1  DECLARE
2    J_BUFF_ID NUMBER;
3    I_TABLE_NAME VARCHAR2(200);
4    I_JSON CLOB;
5    O_ERROR VARCHAR2(200);
6    v_Return NUMBER;
7  BEGIN
8    J_BUFF_ID := 8;
9    I_TABLE_NAME := 'TSTR_TABLE';
10   I_JSON := '{"ID":"4", "VARCHAR_COL":"Text should be too long."}';
```

Slika 3.30 Anonimni blok funkcije F_CHECK_JSON za "length condition"

```
ORA-20104: Column VARCHAR_COL has a length of 5 characters and 'Text should be too long.' is not of the right length.
ORA-06512: at "IZIVKOVIC.DATA_VALIDATION", line 164
ORA-06512: at "IZIVKOVIC.DATA_VALIDATION", line 164
ORA-06512: at line 13
```

Slika 3.316 Output iznimke za nedozvoljenu duljinu

Na slici *Slika 3.30* prikazan je *input* JSON definiran u anonimnom bloku s nedozvoljenom duljinom podatka, a na slici *Slika 3.31* rezultat izvršavanja bloka. Podatak o duljini dohvaća se iz Data Dictionary-ja.

PL/SQL Block

```
1  DECLARE
2    J_BUFF_ID NUMBER;
3    I_TABLE_NAME VARCHAR2(200);
4    I_JSON CLOB;
5    O_ERROR VARCHAR2(200);
6    v_Return NUMBER;
7  BEGIN
8    J_BUFF_ID := 8;
9    I_TABLE_NAME := 'ZUPANIJE';
10   I_JSON := '{"ID":"22", "ZUPANIJA":""}';
```

Slika 3.327 Anonimni blok funkcije F_CHECK_JSON za "nullable condition"

```
ORA-20102: Column ZUPANIJA cannot be a null, but input JSON sends one.
ORA-06512: at "IZIVKOVIC.DATA_VALIDATION", line 164
ORA-06512: at "IZIVKOVIC.DATA_VALIDATION", line 164
ORA-06512: at line 13
```

Slika 3.33 Output iznimke za izostavljenu vrijednost

Na slici *Slika 3.32* prikazan je *input* JSON definiran u anonimnom bloku s nedodijeljenom vrijednošću stupcu koja zahtjeva vrijednost, a na slici *Slika 3.33* rezultat izvršavanja bloka. Ovaj primjer koristi tablicu „ZUPANIJE“ jer u prethodno korištenoj tablici osim ID-a niti jedan drugi stupac nema uvjet *not nullable.*

PL/SQL Block

```
1  DECLARE
2    J_BUFF_ID NUMBER;
3    I_TABLE_NAME VARCHAR2(200);
4    I_JSON CLOB;
5    O_ERROR VARCHAR2(200);
6    v_Return NUMBER;
7  BEGIN
8    J_BUFF_ID := 8;
9    I_TABLE_NAME := 'TSTR_TABLE';
10   I_JSON := '{"ID":"4", "NUM_COL":"prr"}';
```

Slika 3.34 Anonimni blok funkcije F_CHECK_JSON za "number condition"

```
ORA-20103: Column NUM_COL has a type NUMBER and 'prr' is not of the right numeric format.
ORA-06512: at "IZIVKOVIC.DATA_VALIDATION", line 164
ORA-06512: at "IZIVKOVIC.DATA_VALIDATION", line 164
ORA-06512: at line 13
```

Slika 3.358 Output iznimke za neispravnu brojčanu vrijednost

Na slici *Slika 3.34* prikazan je *input* JSON definiran u anonimnom bloku s nasumičnom tekstualnom na mjestu brojčane vrijednosti, a na slici *Slika 3.35* rezultat izvršavanja bloka. Vraća ovakav rezultat na temelju provjere IS_NUMBER funkcije.

PL/SQL Block

```
1  DECLARE
2    J_BUFF_ID NUMBER;
3    I_TABLE_NAME VARCHAR2(200);
4    I_JSON CLOB;
5  BEGIN
6    J_BUFF_ID := 8;
7    I_TABLE_NAME := 'TSTR_TABLE';
8    I_JSON := '{"ID":"3","NUM_COL":"4","VARCHAR_COL":"ABCD"}';
```

Slika 3.36 Anonimni blok procedure P_JSON_INS_UPD

```
INSERT INTO TSTR_TABLE (ID, NUM_COL, VARCHAR_COL, DATE_COL) VALUES (3, 4, 'ABCD', '')
UPDATE TSTR_TABLE SET NUM_COL=4, VARCHAR_COL='ABCD' WHERE ID=3
```

Slika 3.37 Output upita

PL/SQL Block

```
1  DECLARE
2    J_BUFF_ID NUMBER;
3    I_TABLE_NAME VARCHAR2(200);
4    I_JSON CLOB;
5  BEGIN
6    J_BUFF_ID := 8;
7    I_TABLE_NAME := 'TSTR_TABLE';
8    I_JSON := '{"ID":"3","NUM_COL":"","VARCHAR_COL":"",
9                      "DATE_COL":"18-AUG-2020"}';
```

Slika 3.38 Anonimni blok procedure P_JSON_INS_UPD

```
INSERT INTO TSTR_TABLE (ID, NUM_COL, VARCHAR_COL, DATE_COL) VALUES (3, 0, '', '18-AUG-20')
UPDATE TSTR_TABLE SET NUM_COL=0, DATE_COL='18-AUG-20' WHERE ID=3
```

Slika 3.39 Output upita

Slike *Slika 3.36* i *Slika 3.38* prikazuju dva različita primjera unosa JSON-a. U trenutku kada su svi uvjeti za unos zadovoljeni, program će formirati konačne oblike upita za unos i ažuriranje tablica. Željeni cilj je kroz slike *Slika 3.37* i *Slika 3.39*. pokazati izlazne upite te usporediti kako se program ponaša ako se npr. ne dodijeli vrijednost.

4. ZAKLJUČAK

Zamisao rada je pojednostavljenje odnosa klijentske i serverske strane aplikacije te usklađivanje njihovih radnji. Korisnost programa očituje se osiguranim primitkom povratnih informacija u obliku prilagođenih tekstualnih i numeričkih iznimaka na jednostavan i razumljiv način. Automatika ove validacije također stvara neovisnost o ručnom upisivanju podataka u bazu što umanjuje fizičko vrijeme obavljanja neke radnje. Cilj rada bit će postignut samo ako pri samom kreiranju aplikacije budu jasno definirana ograničenja na oblike podataka koji se mogu slati i primati. Preduvjet je da klijentska strana bude dovoljno dobro upoznata s internom strukturom podataka na bazi stoga je potrebno voditi računa o pravilnoj formaciji JSON objekata koji se šalju. Iako se čini kako bi se dodavanjem još jednog programskog sloja aplikaciji moglo stvoriti dodatno zagušenje cjelokupnog sustava, uvjeti ograničenja na tablicama ionako moraju biti implementirani na serverskoj strani. Time se eliminira potreba da razvojni inženjer ove provjere definira samostalno. Za idealan rezultat bilo bi moguće uvesti opciju prikupljanja svih grešaka i njihovog slanja kao skupa povratnih informacija, a ne pojedinačnih poruka. Na ovaj način krajnji korisnik može dobiti informaciju o svim greškama pri unosu podataka kako ne bi morao za svaku potencijalnu grešku pozivati server. Ova implementacija bi kompenzirala za navedeno zagušenje i time skratila vrijeme izvođenja nekih radnji na strani aplikacije. Obzirom na to da rad sugerira samo jedan od mogućih načina implementacije, otvoren je za daljnji razvoj i prilagodbu bilo kojem sustavu koji podržava klijent-server arhitekturu komunikacije.

5. LITERATURA

[1] Hall M. (2019): *Oracle Corporation*, Preuzeto 23.09.2020. s https://www.britannica.com/topic/Oracle-Corporation

[2] Oracle Corporation (bez dat.): *Multimodel Database*, Preuzeto 23.09.2020. s https://www.oracle.com/a/tech/docs/multimodel19c-wp.pdf

[3] Oracle Corporation (bez dat.): *Introducing Oracle Database XE*, Preuzeto 27.09.2020. s https://docs.oracle.com/cd/E17781_01/server.112/e18804/getstart.htm#ADMQS110

[4] Laker K. (2009): *OLTP And Data Warehouse? How Does That Work?* Preuzeto 23.09.2020 s https://blogs.oracle.com/datawarehousing/oltp-and-data-warehouse-how-does-that-work

[5] Oracle Corporation (bez dat.): *JSON in Oracle Database*, Preuzeto 1.10.2020. s https://docs.oracle.com/database/121/ADXDB/json.htm#ADXDB6275

[6] Oracle Corporation (bez dat.): *JSON Data Structures*, Preuzeto 1.10.2020. s https://docs.oracle.com/en/database/oracle/oracle-database/12.2/arpls/json-types.html#GUID-10062646-E36F-48B1-9F24-751B613DFB5A

[7] Oracle Corporation (bez dat.): *The Data Dictionary*, Preuzeto 27.09.2020. s https://docs.oracle.com/cd/B10501_01/server.920/a96524/c05dicti.htm

[8] Oracle Corporation (bez dat.): *Tables Data Dictionary Views*, Preuzeto 27.09.2020. s https://docs.oracle.com/cd/B28359_01/server.111/b28310/tables014.htm#ADMIN01508

[9] Oracle Corporation (bez dat.): *Oracle SQL Developer*, Preuzeto 27.09.2020. s https://www.oracle.com/database/technologies/appdev/sqldeveloper-landing.html

[10] W3Schools (bez dat.): *JSON - Introduction*, Preuzeto 27.09.2020. s https://www.w3schools.com/js/js_json_intro.asp

[11] Javascript.info (2020): *Objects*, Preuzeto 27.09.2020. s https://javascript.info/object

6. OZNAKE I KRATICE

DBA - Database Administrator (Administrator baze podataka)

ID – Identifier (Identifikator)

IT - Information Technologies (Informacijske tehnologije)

JSON - JavaScript Object Notation

PL/SQL - Procedural Language/Structured Query Language

SEQ – Sequence (Sekvenca)

SID - Site Identifier (Identifikator stranice)

SQL - Structured Query Language

TSTR - Tester

XE - Express Edition

7. SAŽETAK

Naslov: Sloj za validaciju podataka na Oracle bazi

Svrha rada je izrada paketa na podatkovnoj bazi koji će služiti kao validacijska točka podataka koji se šalju s klijentske strane. Podaci će se bilježiti u zajedničku tablicu te će validacijski sustav biti pokrenut triggerom koji se okida po unosu u tablicu. Bit je provođenje validacija nad JSON formatima podataka te vraćanje prilagođenih preciziranih poruka putem programskih iznimaka. Validacija te usklađivanje ulaznih podataka sa strukturom na bazi postignuto je korištenjem podataka iz seta Oracle sistemskih tablica – Data Dictionary. Činjenice i informacije u radu većim dijelom su temeljene na samoj originalnoj Oracle dokumentaciji.

Ključne riječi: Oracle baza podataka, JSON, validacija podataka

8. ABSTRACT

Title: Oracle data validation layer

The objective of this project is to create a database package that will serve as a validating point for data sent from the client side. Data is stored inside a common table so the validating system could be initiated by a trigger set off on table insertion. Goal is to carry out validation over JSON data formats and return custom precise error messages through program exceptions. Validation and adaptation of input data to the architecture in the database is accomplished through using data from Oracle's set of system tables – Data Dictionary. Most of the facts and information in this paper is based on Oracle's original documentation itself.

Keywords: Oracle Database, JSON, data validation

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, 29.10.2020.	IVA ŽIVKOVIĆ	

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

IVA ŽIVKOVIĆ

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, 29.10.2020

potpis studenta/ice