E-ispiti: Model baze podataka i implementacija poslovnih pravila koristeći Oracle ADF

Glavaš, Tihana

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli
Preddiplomski sveučilišni studij informatika

TIHANA GLAVAŠ

E-ISPITI: MODEL BAZE PODATAKA I IMPLEMENTACIJA POSLOVNIH PRAVILA KORISTEĆI ORACLE ADF

Završni rad

Pula, 2018.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli
Preddiplomski sveučilišni studij informatika

TIHANA GLAVAŠ

E-ISPITI: MODEL BAZE PODATAKA I IMPLEMENTACIJA POSLOVNIH PRAVILA KORISTEĆI ORACLE ADF

Završni rad

JMBAG: 0303010993, izvanredni student
Smjer: Informatika

Predmet: Informatički praktikum
Mentorica: prof. dr. sc. Vanja Bevanda

Pula, rujan 2018.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisana Tihana Glavaš, kandidatkinja za prvostupnicu informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student:

U Puli, _____ 2018. _____

IZJAVA

o korištenju autorskog djela

Ja, Tihana Glavaš dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „E-ispiti: Model baze podataka i implementacija poslovnih pravila koristeći ADF„ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____ 2018.

Potpis

_____

Sadržaj

1. Uvod

U ovom radu biti će prikazan model baze podatka „e-ispiti“ za potrebe Termoelektrane Plomin (TE Plomin) za polaganje ispita iz zaštite na radu.

TE Plomin je sastavni dio HEP-proizvodnje d.o.o. koja je jedna od članica Hrvatske elektroprivrede (HEP grupe), jedne od najvećih poslovnih organizacija u Republici Hrvatskoj. Zbog svoje organizacijske složenosti, velikog broja zaposlenih te poslova koje zaposlenici obavljaju od izuzetne je važnosti briga o sigurnosti radnika te njihova primjerena zaštita.

Kod izvođenja remontnih radova na svojim postrojenjima TE Plomin ima potrebu osigurati polaganje zaštite na radu svim zaposlenicima vanjskih tvrtki kako bi oni uopće mogli ući u samo postrojenje te mora angažirati vanjske ovlaštene tvrtke za zaštitu na radu kako bi pojedinci mogli položiti ispit. Ovim projektom bi TE Plomin sasvim eliminirala bilo kakvu ovisnost izvana te bi zainteresirane strane mogle samostalno u samome postrojenju položiti zaštitu na radu, brzo i efikasno, a samim time upoznati se s radom u pogonu kako bi se uopće znali kretati unutar jednog tako kompleksnog postrojenja kao što je TE Plomin.

Nakon uvoda će biti opisana analiza korisničkog zahtjeva te razlozi zašto bi jedan takav projekt unaprijedio poslovanje.

U trećem dijelu rada je opisano konceptualno modeliranje baze podataka, odnosno izrada odgovarajuće sheme entiteta, atributa i njihovih veza.

Četvrti dio rada se bavi logičkim modeliranjem baze podataka gdje je opisan relacijski model baze podataka, relacijska shema i napravljen detaljan pregled tablica koje se mogu implementirati u sustav za upravljanje bazom podataka.

Peti dio rada se odnosi na fizičko modeliranje baze podataka odnosno na opis objekata u bazi, ali i na pregled poslovnih pravila te kako su ona rješena PL/SQL-om koji smanjuje opterećenje aplikacije budući da se obrada vrši u bazi podataka.

U šestom dijelu ovoga rada opisan je Oracle ADF (engl. *Application Development Framework*), njegova arhitektura te jedna od njegovih sastavnih komponenti ADF *Business Components* (ADF BC) koji se zapravo koristi za implementaciju modela baze podataka što je ujedno i tema ovoga rada. Nakon toga slijedi slaganje modela u Oracle ADF-u.

Baza je napisana i testirana u Oracle SQL Developeru, a za izradu modela baze podataka se koristila verzija Jdevelopera 12.1.3 koja nije najnovija, ali se koristi u HEP-u.

U ovom završnom radu korištena je metoda deskripcije te se rad fokusira isključivo na potrebe TE Plomin za upoznavanje rada u pogonu, a projekt „e-ispiti“ se lako može proširiti s dodatnim funkcionalnostima kakva inače imaju slična rješenja.

2. Analiza korisničkog zahtjeva

Prije nego što su uopće krene u dizajniranje baze podataka potrebno je prepoznati i okarakterizirati potrebe krajnjih korisnika. Radi toga je od presudne važnosti da dizajner baze podataka u samoj inicijalnoj fazi dođe u doticaj sa stručnjacima iz prakse, odnosno onim korisnicima koji su u doticaju s razno raznim problematikama koje bi baza podataka, a kasnije i sama aplikacija trebale rješiti. Krajnji korisnik ukratko objašnjava vrste operacija ili transakcija koje će se izvoditi s podacima, a dizajner skicira i dobiva bolji i širi uvid u samo poslovanje. Rezultat takvih intervjua bi trebao dovesti do detaljne specifikacije korisničkog zahtjeva na temelju kojeg se može krenuti u modeliranje same baze podataka.

Nakon toga se u sljedećoj fazi izrađuje model entiteti veze koji je dio konceptualnog modeliranja i u kojem se utvrđuju entiteti, atributi, njihove veze i ograničenja o kojima će biti riječi u nastavku.

Fizički model baze se može relativno lako izmijeniti nakon što je aplikacija izgrađena, međutim promjene na logičkoj razini su teške ili gotovo nemoguće budući da one mogu utjecati na veliki broj upita, a samim time i na kod aplikacije. Važno je isto tako obratiti pažnju na redundanciju i nepotpunost podataka odnosno na to da se podaci ne ponavljaju te da budu potpuni.

Nakon razgovora s odgovornim osobama u TE Plomin došli smo do sljedeće specifikacije korisničkog zahtjeva:

Prilikom obavljanja određenih radova u samom postrojenju kao što su izgradnja novih hala, servisiranja cijevi ili što je najčešći slučaj prilikom remontnih radova na generatoru za koje TE Plomin nema ovlaštene djelatnike isti ima potrebu dovođenja vanjskih firmi kako bi one odradile određene poslove. To su usko specijalizirana poduzeća koja vrše remontne radove na električnim generatorima veličine omanje zgrade te se na takvim i sličnim poslovima skriva pregršt opasnosti budući da se tu radi o radovima pod visokim naponom, velikim metalnim elementima i konstrukcijama.

Takve vrste radova se ne viđaju svaki dan te se prilikom njihovog izvođenja pozivaju i fakulteti da dođu pratiti same radove kako bi mogli vidjeti impozantne električne elemente koje inače nebi imali prilike vidjeti. TE Plomin u tom slučaju treba za svakog pojedinog posjetioca, radnika ili pak inženjera osigurati položen ispit iz zaštitu na radu kako bi cijeli proces prošao u najboljem redu i bez opasnosti za ikoga.

Ukratko tu je riječ o stotinama ljudi koji se trebaju kretati jednim vrlo kompleksnim postrojenjem te trebaju biti upoznati s određenim brojem pravila ponašanja.

Budući da se sada po zakonu o zaštiti na radu ispit treba polagati na mjestu rada, zaposlenici vanjskih tvrtki, bez obira na to što već imaju položen ispit iz zaštite na radu moraju ga polagati i u TE Plomin, odnosno na novom mjestu rada. Osim toga postoje i različiti tipovi ispita koji su usko specijalizirani za rad pod visokim naponom, rad s posebnim strojevima, rad s električnom energijom i slični.

Kako nadređene osobe u TE Plomin ne mogu osposobljavati ljude iz vanjskih tvrtki, oni bi trebali zvati drugu vanjsku tvrtku koja za to ima ovlaštenja. Tu naime dolazi do raznoraznih problema i nezadovoljstva zaposlenika, ali i njihovih nadređenih.

Kako bi se spriječili takvi i slični problemi odlično rješenje bi bio jedan e-ispit u kojem bi svaki zaposlenik iz vanjski tvrtke najprije pročitao određene upute te se upoznao s opasnostima koje se nalaze na novom mjestu rada, a nakon toga odgovorio na tri ili više pitanja s ponuđenim odgovorima od kojih je jedan odgovor točan. U slučaju da zaposlenik pogriješi, odnosno netočno odgovori, on bi se ponovno trebao vratiti na tekst gdje će opet pročitati upute te ponovno pokušati rješiti ispit. To znači kako nije cilj da zaposlenici ne prođu ispit već im on može dati dobru podlogu za pravilnu reakciju u na žalost mogućoj opasnoj situaciji.

3. Konceptualno modeliranje baze podataka

Prva faza modeliranja baze podataka je konceptualno modeliranje u kojem je glavni cilj stvoriti konceptualnu shemu baze koja je sastavljena od entiteta, veza i atributa. Ta faza oblikovanja je oslobođena tehničkih detalja te je to jedan od najvažnijih, ako ne i najvažniji proces u samom razumijevanju baze podataka. U slučaju da baza konceptualno nije dobro postavljena, odnosno ako je neki entitet, atribut ili veza krivo postavljen ili izostavljen kasnije je teško ili čak nemoguće ispraviti tu pogrešku jer cijeli koncept baze podataka može otići u krivom smjeru. Važno svojstvo konceptualne sheme je da je ona razumljiva ljudima svih struka, te da ona može služiti kao sredstvo projektanata i korisnika. Nadalje, konceptualna shema se ne može automatski implementirati pomoću današnjih sustava za upravljanje bazom podataka radi toga što joj nedostaju brojni detalji, no to ne umanjuje njezinu uporabljivost.

3.1. Entiteti, veze i njihovi atributi

Entitet (engl. *entity*) je objekt u bazi podataka o kojem želimo spremati podatke i koji je od krucijalne važnosti za bazu podataka. Pod entitetom se može smatrati osoba, stvar, događaj ili pojava koja je sama po sebi apstraktna, ali bez koje baza nebi funkcionirala. Entitet je nešto o čemu želimo spremati podatke, nešto što je u stanju postojati ili ne postojati, te se može identificirati (Manger, 2011).

Veze (engl. *relationship*) predstavljaju odnos između dva ili više entiteta. U stvarnom životu najčešće su u primjenama binarne veze, odnosno veze koje se uspostavljaju između točno dva tipa entiteta.

Svaki entitet ima svoje atribute (engl. *attribute*) koji ga opisuju. Na primjer, atributi entiteta *Korisnik* su *šifra*, *ime*, *prezime*, *adresa*, *email* i *rola*. Vrijednosti atributa koje opisuju entitet postaju glavni dio podataka koji se trebaju pohraniti u bazi. U nekim slučajevima pojedini atribut može ili ne mora imati svoju vrijednost, odnosno može biti *null*, kao što u primjeru baze podataka „e-ispiti“ atribut *napomena* može ostati i prazna no međutim korisnik ima mogućnost upisivanja napomene.

Kada spremimo instancu određenog entiteta u bazu podataka cilj je da taj isti entitet bude jedinstven. Kako je tablica skup sličnih objekata i događaja treba se osigurati jedinstvenost svakog retka, a za to služe ključevi. Zato je od presudne važnosti da se definiraju pravi

ključevi za svaki entitet budući da to osigurava dobro strukturirane tablice, minimalnu redundanciju te čvrste relacije među tablicama.

Atribut ili skup atributa koji jednoznačno određuje neki entitet može se smatrati kandidatom za ključ. Kandidat za ključ se mora sastojati od jedinstvenih podataka te se pod svaku cijenu mora izbjeći dupliciranje podataka unutar jedne tablice, ali i kompletne baze podataka. Također ni u kojem slučaju ne smije imati vrijednost *null* budući da to predstavlja nedostatak vrijednosti. Nije dobro niti da atributi poput OIB-a, lozinka i sličnih budu kandidati za ključ radi sigurnosnih razloga organizacije ili pak privatnosti samog korisnika. Kada entitet ima više kandidata za ključ tada se jedan od njih proglašava primarnim ključem.

U slučaju da entitet nema kandidata za ključ može se kreirati takozvani umjetni ili surogat ključ (engl. *artificial* ili *surrogate*). Njega kreiramo tako što izmislimo novi redak u tablici koji ima sve elemente primarnog ključa, a to najčešće u praksi bude *id*. Umjetni ključ između ostalog možemo dodati i entitetu koji ima kandidate za ključ te je tada njegova jedina svrha biti primarnim ključem. U primjeru entiteta *korisnik*, kandidati za ključ mogu biti korisnikova šifra ili e-mail budući da jednoznačno određuju korisnika, no kao što je ranije navedeno šifra nije dobar primarni ključ radi sigurnosnih razloga organizacije, a e-mail na primjer može imati *null* vrijednost, te se može modificirati, odnosno korisnik u određenim situacijama može promijeniti e-mail stoga niti on nije dobar kandidat za ključ. Radi toga se svim entitetima u bazi podataka „e-ispiti“ dodaje umjetni ključ *id* kao primarni ključ.

Slično kao tipovi entiteta, i veze unutar istog modela, ali i atributi moraju imati različita imena. Međutim Manger (2011) napominje kako se dopušta da dvije veze ili entitet i veza imaju atribute s istim imenom, no tada se podrazumijeva da su to ustvari atributi s istim značenjem i istim tipom vrijednosti.

Svakoj vezi treba odrediti njezinu funkcionalnost, obaveznosti članstva te kardinalnosti.

Najčešće veze u relacijskim bazama podataka imaju funkcionalnost jedan-naprama-mnogo (1:M), a veze s funkcionalnošću jedan-naprama-jedan (1:1) se u praksi mogu naći u manjem broju. Ako postoje instance dvaju entiteta (A i B) i ako postoji veza jedan-naprama-mnogo između dvije instance (A_i i B_i) tada je A_i povezan s nijednom, jednom ili više instanci entiteta B, a B_i je povezan s nijednom ili jednom instancom entiteta A. U primjeru baze podataka „e-ispiti“ čiji model možemo vidjeti na Slici 1. veza *definira* između entiteta *Definirani ispit* i *Definirano pitanje* ima funkcionalnost 1:M što znači da ispit može sadržavati jedno ili više

pitanja, a da pitanje može pripadati samo jednom ispitu, dok veza *sadrži* između entiteta *Postavljeno pitanje* i *Odgovor polagača* ima funkcionalnost 1:1 što znači da pitanje može sadržavati samo jedan odgovor koji pak može pripadati samo tom pitanju.

Veze mnogo-naprama-mnogo (M:M) su također česte u relacijskim bazama podataka. Ako postoji veza M:M između dvije instance entiteta (A_i i B_i) tada je A_i povezan s nijednom, jednom ili više instanci entiteta B, a B_i je povezan s nijednom, jednom ili više instanci entiteta A.

Opet promatramo vezu između instanci entiteta A_i i B_i. Kažemo da A_i ima obavezno članstvo u toj vezi ako svaki primjerak od A_i mora sudjelovati u vezi, dakle mora biti povezan barem s jednim primjerkom od B_i. Analogno se definira i obaveznost članstva za B_i. U ranije spomenutoj vezi *sadrži*, instanca entiteta *Odgovor polagača* i instanca entiteta *Postavljeno pitanje* imaju obavezno članstvo jer pitanja moraju sadržavati odgovor.

Svojstvo funkcionalnosti i opcionalnosti mogu se otprilike izraziti samo jednim svojstvom koje se naziva kardinalnost. Tada moguće veze postaju 1,1 (jedan i samo jedan) i 1,M (jedan ili više) koje imaju obaveznost članstva, te 0,1 (nijedan ili jedan) i 0,M (nijedan, jedan ili više) koje su opcionalne. Kardinalnosti veze baze podataka „e-ispiti“ su objašnjene na primjeru Chenovog dijagrama koji je opisan u sljedećem poglavlju.

3.2. Model entiteti veze

Nako što se utvrde entiteti, veze te njihovi atributi oblikuje se konceptualna shema baze.

Model entiteti veze ili E-R model (engl. *Entity-Relationship*) za „e-ispite“ sastoji se od deset međusobno povezanih entiteta te se može podijeliti na dva dijela. U prvom dijelu se nalaze pitanja koja je ispitivač postavio i svi ponuđeni točni ili netočni odgovori na ta pitanja, a u drugom dijelu se nalaze pitanja koja je polagač dobio metodom slučajnog odabira te samo jedan polagačev odgovor, bio on točan ili netočan.

U ovom radu korišten je reducirani Chenov dijagram u kojem možemo vidjeti entitete, veze među entitetima te njihove kardinalnosti, a koji je prikazan na Slici 1.

Slika 1. Dijagram entiteta i veza za bazu podataka „e-ispiti“ [izvor: autor]

Dijagram prikazuje entitete i veze među entitetima te njihove kardinalnosti:

- Veza *pripada* ima kardinalnost 1,M:1,1. Korisnik može pripadati samo jednoj organizaciji, a organizacija može imati više korisnika.
- Veza *zadaje* ima kardinalnost 1,1:1,M. Organizacija može zadati više ispita, ali ispit može pripadati samo jednoj organizaciji.
- Veza *ima* ima kardinalnost 1,1:1,M. Tip ispita može imati više definiranih ispita, a ispit može pripadati samo jednom tipu ispita.
- Veza *definira* ima kardinalnost 1,1:1,M. U definirani ispitu se može definirati više definiranih pitanja, međutim ta pitanja mogu pripadati samo jednom ispitu.
- Veza *čini* ima kardinalnost 1,1:1,M. Definirano pitanje čini više definiranih odgovora, no ponuđeni odgovori mogu pripadati samo jednom pitanju.
- Veza *polaže* ima kardinalnost 1,1:1,M. Polagač ispita ima mogućnost više puta polagati ispit, a taj ispit može pripadati samo određenom polagaču.
- Veza *generira* ima kardinalnost 1,1:1,M. U polaganju ispita se može izgenerirati više postavljenih pitanja, međutim ta pitanja mogu pripadati samo jednom ispitu.
- Veza *sadrži* ima kardinalnost 1,1:1,1. Polagač ispita može na postavljeno pitanje dati samo jedan odgovor, a taj odgovor može pripadati samo jednom pitanju.

Osim veza s funkcionalnošću 1:1 i 1:M u dijagramu su prikazani pod-tipovi i nad-tipovi, kao veze s nazivom JE, odnosno tipovi entiteta *Postavljeno pitanje* i *Polaganje ispita* su pod-tipovi entiteta *Definirano pitanje i Definirani ispit*, dok je tip entiteta *Odgovor polagača* nad-tip entiteta *Definirani odgovor*.

Reducirani Chenov dijagram koji se ovdje koristi opisuje samo entitete i veze, a ne sadrži informacije o atributima zbog jednostavnosti i preglednosti dijagrama. Budući da konceptualna shema nije u potpunosti opisana samim dijagramom na Slici 2. se nalazi popis entiteta i svih njihovih atributa.

Tip entiteta KORISNIK ima atribute:
ID_KORISNIKA, ŠIFRA, IME, PREZIME, EMAIL, ROLA (administrator, user, superuser)

Tip entiteta ORGANIZACIJA ima atribute:
ID_ORGANIZACIJE, ŠIFRA, NAZIV, ADRESA

Tip entiteta TIP ISPITA ima atribute:
ID_TIPA, ŠIFRA, NAZIV, ADRESA

Tip entiteta DEFINIRANI ISPIT ima atribute:
ID_DEF_ISPITA, ŠIFRA, NAZIV, TRAJANJE, BROJ PITANJA, BROJ POKUŠAJA

Tip entiteta DEFINIRANO PITANJE ima atribute:
ID_DEF_PITANJA, REDNI BROJ PITANJA, TEKST PITANJA, OPIS ODGOVORA, BODOVI, SLIKA

Tip entiteta DEFINIRANI ODGOVOR ima atribute:
ID_DEF_ODGOVORA, TEKST ODGOVORA, TOČNO (T/N), SLIKA

Tip entiteta POLAGAČ ISPITA ima atribute:
ID_POLAGAČA, IME, PREZIME, TVRTKA, LOZINKA

Tip entiteta POLAGANJE ISPITA ima atribute:
ID_ISPITA, DATUM, NAPOMENA

Tip entiteta POSTAVLJENO PITANJE ima atribute:
ID_POSTAVLJENOG_PITANJA, ODGOVORENO (T/N)

Tip entiteta ODGOVOR POLAGAČA ima atribute:
ID_ODGOVORA_POLAGAČA, ODGOVOR, DATUM

Slika 2. Popratni tekst uz dijagram sa Slike 1. [izvor: autor]

4. Logičko modeliranje baze podataka

Osnovni cilj logičkog modeliranja baze podataka je stvoriti relacijski model odnosno relacijsku shemu baze koja opisuje njezinu logičku strukturu u skladu s pravilima. Relacijski model je manje razumljiv korisnicima budući da su entiteti i veze među njima pretvoreni u relacije, međutim za razliku od E-R modela on se može izravno implementirati pomoću današnjih sustava za upravljanje bazom podataka te ga to u današnje vrijeme čini primarnim modelom podataka. U nastavku će biti objašnjeno kako se pojedini elementi E-R modela baze podataka „e-ispiti“ pretvaraju u relacijski model.

4.1. Relacijski model

Model relacijske baze podataka je teoretski osmislio dr. Edgar F. Codd koji je kasnih šezdesetih godina 20. stoljeća radio kao IBM-ov znanstvenik i bio u potrazi za novim načinima kako uspješno manipulirati velikim količinama podataka. Njegovo nezadovoljstvo tadašnjim modelima baza podataka ga je dovelo do zaključka o implementaciji matematičkih načela i struktura kako bi rješio mnoštvo problema s kojima se susretao. Budući da je bio matematičar po struci, bio je od dubokog uvjerenja kako se mogao osloniti na specifične grane matematike u rješavanju problema kao što su redundancija, slabi integritet podataka te prekomjerena ovisnost same baze podataka o fizičkoj implementaciji. Dr. Codd je svoj revolucionarni model relacijske baze podataka predstavio u znanstvenom radu nazvanom „*A Relational Model of Data for Large Shared Databanks*“ u lipnju 1970. (Hernandez, 2013), međutim relacijski model tek osamdesetih godina prošlog stoljeća postaje prevladavajući te je i dan danas jedan od najrasprostranjenijih modela baza podataka. Ime relacijskog modela dolazi od matematičkog pojma „relacija“, a ne kako se krivo tumači, od veza (zbog engl. *relations*) između tablica unutar baza podataka.

4.2. Pretvorba E-R modela u relacijski model

Kod pretvorbe E-R modela u relacijski model svaki tip entiteta prikazuje se jednom relacijom, a njegovi atributi postaju atributi relacije. Jednako tako, jedan primjerak entiteta prikazan je jednom n-torkom, a primarni ključ entiteta postaje primarni ključ relacije

(Manger, 2011). Na primjer, tip entiteta *Definirano pitanje* iz baze podataka „e-ispiti“ sa Slike 1. prikazuje se relacijom:

def_pitanja (id_pitanja, redni broj pitanja, tekst pitanja, bodovi, slika).

Također možemo primjetiti da tip entiteta *Definirano pitanje* je u E-R modelu zapisano u jednini dok se relacije pišu u množini budući da relacije predstavljaju više instanci entiteta, a ne samo jedan entitet.

Kod pretvorbe veza jedan-naprama-mnogo (1:M) gledamo obavezno članstvo između dva tipa entiteta. Na primjer, na Slici 1. tip entiteta *Definirano pitanje* ima obavezno članstvo u tipu entiteta *Definirani odgovor*. Budući da svaki odgovor mora pripadati nekom pitanju veza između ta dva tipa entiteta se svede na to da u relaciju *def_odgovori* ubacimo ključ relacije *def_pitanja*:

def_odgovori (id_odgovora, *id_pitanja*, tekst odgovora, tocno/netocno, slika).

Važno je spomenuti kako se primarni ključ jedne relacije, u ovom primjeru *id_pitanja*, koji je prepisan u drugu relaciju zove strani ključ u toj drugoj relaciji. Pravilo za pretvorbu veza s funkcionalnošću 1:M analogno se primjenjuje i na veze s funkcionalnošću M:1 i 1:1.

Veza s funkcionalnošću mnogo-naprama-mnogo (M:M) uvijek se prikazuje posebnom relacijom, koja se sastoji od primarnih atributa za oba tipa entiteta zajedno s eventualnim atributima veze (Manger, 2011), no u primjeru baze podataka „e-ispiti“ nema takvih veza.

Relacijski model je zapravo redak koji opisuje građu relacije, a sastoji se od imena relacije te atributa relacije koji se nalaze u zagradi. U skladu s prethodno navedenim pravilima pretvaramo E-R model u skup relacija te dobivamo relacijski model u kojem su primarni atributi odnosno primarni ključevi podvučeni, a strani ključevi ukošeni:

organizacija (id, sifra, naziv, adresa)
korisnici (id, *id_org*, sifra, ime, prezime, email, rola)
tip_ispita (id, naziv)
def_ispiti (id, *id_tipa*, *id_org*, sifra, naziv, trajanje, broj_pitanja, broj_pokusaja)
def_pitanja (id, *id_ispita*, r_br_pitanja, tekst_pitanja, opis_odgovora, bodovi, slika)
def_odgovori (id, *id_pitanja*, tekst_odgovora, tocno, slika)
polagac_ispita (id, lozinka, ime, prezime, tvrtka)
polaganje_ispita (id, *id_ispita*, *id_polagaca*, datum, napomena)

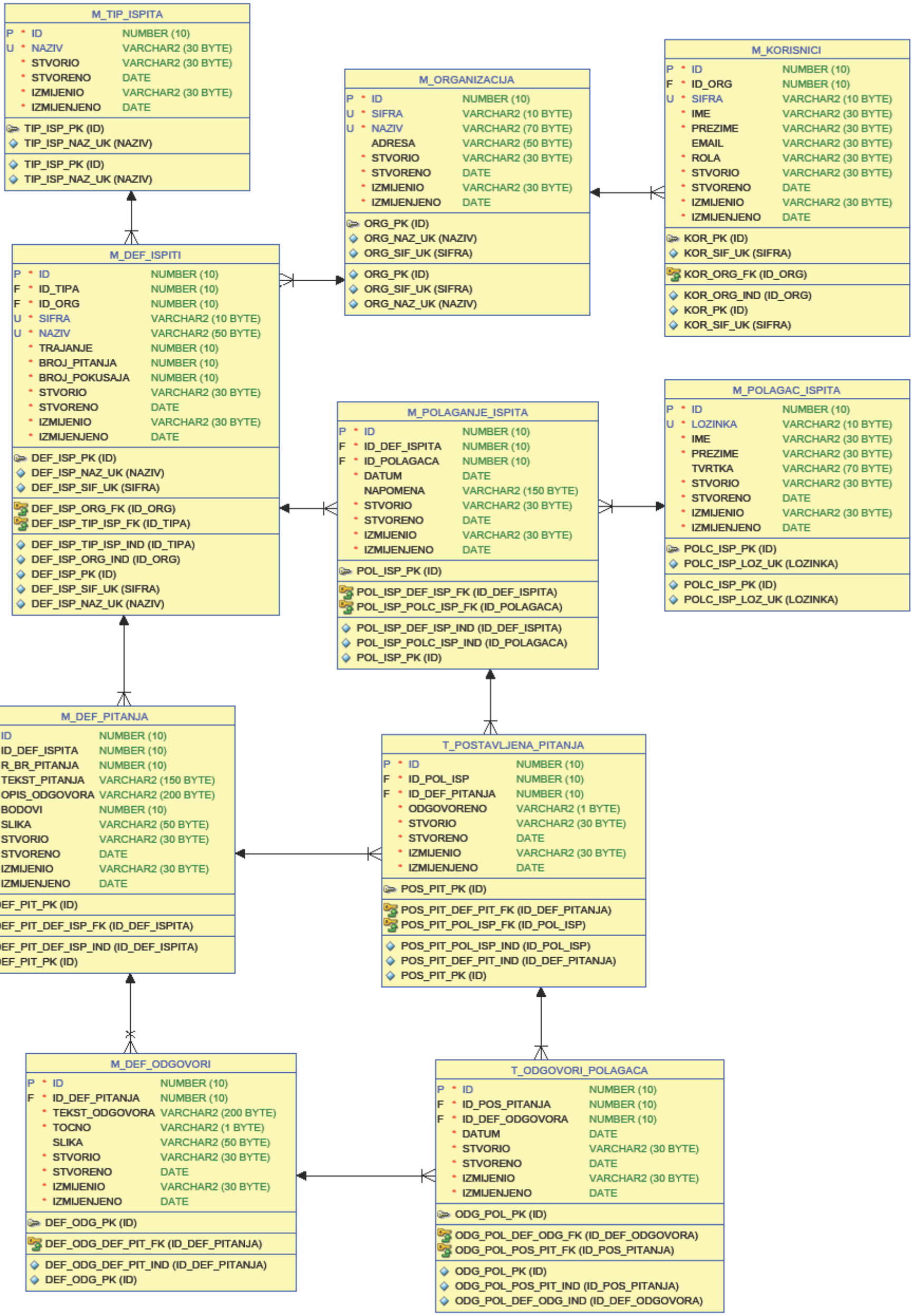
postavljena_pitanja (id, *id_pol_isp*, *id_pitanja*)

odgovori_polagaca (id, *id_pos_pitanja*, *id_odgovora*)

Normalizacija baze podataka je ključ za neometano formiranje i održavanje bilo koje baze podataka. U nekim slučajevima normalizacija ne treba biti u potpunosti izvedena, ali u većini slučajeva se treba provesti. Najčešće je dovoljno dovesti bazu podataka do treće normalne forme (3NF), a ostale ili slijede ili uopće nisu potrebne za normalno funkcioniranje baze podataka. Zato je važno napomenuti kako su sve relacije baze podataka „e-ispiti“ u trećoj normalnoj formi, budući da su svi atributi unutar pojedinih relacija jedinstveni te ne postoje nikakve parcijalne i tranzitivne ovisnosti.

4.3. Opis relacijskog modela

Baza podataka se sastoji od deset međusobno povezanih relacija, odnosno tablica. S jedne strane imamo tablice koje se odnose na definiranje pojedinog ispita, takozvane matične tablice, a to su redom *Organizacija*, *Korisnici*, *Tip_ispita*, *Def_ispiti*, *Def_pitanja*, *Def_odgovori*, *Polagac_ispita*, dok s druge strane imamo tablice koje se odnose na transakcijski dio, odnosno na polaganje ispita, a to su *Polaganje_ispita*, *Postavljena_pitanja* te *Odgovori_polagaca*. Na Slici 3. mogu se vidjeti sve tablice, njihove veze te primarni, strani i jedinstveni ključevi. Matične tablice imaju prefiks *M_*, dok transakcijske tablice imaju prefiks *T_*.



Slika 3. Relacijski dijagram baze podataka „e-ispiti“ [izvor: autor]

U nastavku su opisane sve tablice koje smo dobili u skladu sa svim prethodno objašnenim pravilima.

ORGANIZACIJA – predstavlja tablicu u bazi podataka gdje su upisane sve termoelektrane koje pripadaju HEP-u.

KORISNICI – je tablica u kojoj su upisani oni korisnici koji mogu upravljati aplikacijom, odnosno useri, superuseri i administratori.

TIP_ISPITA – tablica koja sadrži podatke o tipu ispita koji može biti bodovan ili ne bodovan.

DEF_ISPITI – predstavlja tablicu u kojoj su upisani ispiti za pojedino područje zaštite na radu, kao na primjer ispit iz električne struje ili pak ispit o rukovanju s posebnim strojevima. U ovu tablicu se upisuje i koliko će ispit imati ponuđenih pitanja te koliko će biti njegovo trajanje.

DEF_PITANJA – tablica koja sadrži sva moguća pitanja koja pripadaju jednom od definiranih ispita, ali i sadrže cjelovite odgovore koje polagač ispita mora pročitati prije pristupanja ispitu.

DEF_ODGOVORI – tablica koja sadrži sve odgovore na jedno pitanje, odnosno tri odgovora od kojih je samo jedan odgovor točan.

POLAGAC_ISPITA – tablica koja sadrži popis osoba koje polažu ispit.

POLAGANJE_ISPITA – izvedena tablica koja sadrži podatke o ispitu i o polagaču ispita na točno određeni datum i u točno određeno vrijeme.

POSTAVLJENA_PITANJA – izvedena tablica koja sadrži definirana pitanja koja su metodom slučajnog odabira postavljena polagaču ispita.

ODGOVORI_POLAGACA – tablica koja sadrži odgovore na koje je polagač ispita odgovorio bili oni točni ili netočni.

4.4. Rječnik podataka

Struktura relacijskog modela baze podataka se sprema u takozvani rječnik podataka (engl. *data dictionary*). Rječnik podataka je sačinjen od popisa svih atributa koji se pojavljuju u

pojedinim relacijama te sadrži njihove neformalne opise i tipove vrijednosti koje mogu poprimiti. Budući da relacijski model ne sadrži opise i tipove vrijednosti atributa ponekad se može dogoditi da iz imena atributa nije lako odrediti njihovo značenje te se radi toga sastavlja rječnik podataka koji se nalazi u Tablici 1 u kojem možemo vidjeti detaljan pregled relacija, njihovih atributa, tipove podataka, da li su polja obavezna ili neobavezna te kratki opis pojedinih atributa.

Tablica 1 Rječnik podataka baze podataka „e-ispiti“

Ime tablice				
Atribut	**Tip**	**NULL**	**Komentari**	**Opis**
ORGANIZACIJA				
ID	number(10)	NE	Primary key	Jednoznačno određuje organizaciju
SIFRA	varchar2(10)	NE	Unique key	Jedinstvena šifra organizacije
NAZIV	varchar2(70)	NE	Unique key	Naziv organizacije
ADRESA	varchar2(50)	DA		Adresa organizacije
STVORIO	varchar2(30)	NE		Korisnik dodavanja
STVORENO	date	NE		Datum dodavanja
IZMIJENIO	varchar2(30)	NE		Korisnik izmjene
IZMIJENJENO	date	NE		Datum izmjene
KORISNICI				
ID	number(10)	NE	Primary key	Jednoznačno određuje korisnika
ID_ORG	number(10)	NE	Foreign key	Referenca na organizaciju
SIFRA	varchar2(10)	NE	Unique key	Šifra (username) korisnika
IME	varchar2(30)	NE		Ime korisnika
PREZIME	varchar2(30)	NE		Prezime korisnika
EMAIL	varchar2(30)	DA		E-mail korisnika
ROLA	varchar2(30)	NE		Superuser, administrator ili user
STVORIO	varchar2(30)	NE		Korisnik dodavanja
STVORENO	date	NE		Datum dodavanja
IZMIJENIO	varchar2(30)	NE		Korisnik izmjene
IZMIJENJENO	date	NE		Datum izmjene
POLAGAC_ISPITA				
ID	number(10)	NE	Primary key	Jednoznačno određuje polagača ispita
LOZINKA	varchar2(10)	NE	Unique key	Lozinka koju polagač dobije kako bi mogao pristupiti ispitu
IME	varchar2(30)	NE		Ime polagača
PREZIME	varchar2(30)	NE		Prezime polagača

TVRTKA	varchar2(70)	NE		Vanjska tvrtka u kojoj polagač radi
STVORIO	varchar2(30)	NE		Korisnik dodavanja
STVORENO	date	NE		Datum dodavanja
IZMIJENIO	varchar2(30)	NE		Korisnik izmjene
IZMIJENJENO	date	NE		Datum izmjene
TIP_ISPITA				
ID	number(10)	NE	Primary key	Jednoznačno određuje tip ispita
NAZIV	varchar2(30)	NE	Unique key	Ispit može biti bodovan ili ne bodovan
STVORIO	varchar2(30)	NE		Korisnik dodavanja
STVORENO	date	NE		Datum dodavanja
IZMIJENIO	varchar2(30)	NE		Korisnik izmjene
IZMIJENJENO	date	NE		Datum izmjene
DEF_ISPITI				
ID	number(10)	NE	Primary key	Jednoznačno određuje definirani ispit
ID_TIPA	number(10)	NE	Foreign key	Referenca na tip ispita
ID_ORG	number(10)	NE	Foreign key	Referenca na organizaciju
SIFRA	varchar2(10)	NE	Unique key	Šifra ispita
NAZIV	varchar2(50)	NE	Unique key	Naziv, vrsta ispita
TRAJANJE	number(10)	NE		Trajanje ispita
BROJ_PITANJA	number(10)	NE		Broj pitanja u ispitu (3 od mogućih 15)
BROJ_POKUSAJA	number(10)	NE		Koliko puta korisnik može polagati ispit
STVORIO	varchar2(30)	NE		Korisnik dodavanja
STVORENO	date	NE		Datum dodavanja
IZMIJENIO	varchar2(30)	NE		Korisnik izmjene
IZMIJENJENO	date	NE		Datum izmjene
DEF_PITANJA				
ID	number(10)	NE	Primary key	Jednoznačno određuje definirana pitanja
ID_DEF_ISPITA	number(10)	NE	Foreign key	Referenca na definirani ispit
R_BR_PITANJA	number(10)	NE		Redni broj pitanja
TEKST_PITANJA	varchar2(150)	NE		Postavljen pitanje
OPIS_ODGOVORA	varchar2(200)	NE		Cjeloviti točan odgovor
BODOVI	number	NE		Koliko bodova donosi jedno pitanje
SLIKA	varchar2(20)	DA		Popratna slika
STVORIO	varchar2(30)	NE		Korisnik dodavanja
STVORENO	date	NE		Datum dodavanja
IZMIJENIO	varchar2(30)	NE		Korisnik izmjene
IZMIJENJENO	date	NE		Datum izmjene
DEF_ODGOVORI				
ID	number(10)	NE	Primary key	Jednoznačno određuje definirane odgovore

ID_DEF_PITANJA	number(10)	NE	Foreign key	Referenca na definirana pitanja
TEKST_ODGOVORA	varchar2(20)	NE		Mogući odgovori na definirana pitanja
TOCNO	varchar2(1)	NE		Je li odgovor točan ili netočan
SLIKA	varchar2(20)	DA		Popratna slika
STVORIO	varchar2(30)	NE		Korisnik dodavanja
STVORENO	date	NE		Datum dodavanja
IZMIJENIO	varchar2(30)	NE		Korisnik izmjene
IZMIJENJENO	date	NE		Datum izmjene
POLAGANJE_ISPITA				
ID	number(10)	NE	Primary key	Jednoznačno određuje polaganje ispita
ID_DEF_ISPITA	number(10)	NE	Foreign key	Referenca na definirani ispit
ID_POLAGACA	number(10)	NE	Foreign key	Referenca na polagača ispita
DATUM	date	NE		Datum i vrijeme polaganja ispita
NAPOMENA	varchar2(150)	DA		Napomena
STVORIO	varchar2(30)	NE		Korisnik dodavanja
STVORENO	date	NE		Datum dodavanja
IZMIJENIO	varchar2(30)	NE		Korisnik izmjene
IZMIJENJENO	date	NE		Datum izmjene
POSTAVLJENA_PITANJA				
ID	number(10)	NE	Primary key	Pitanja koja se pojavljuju u ispitu metodom slučajnog odabira
ID_POL_ISP	number(10)	NE	Foreign key	Referenca na polaganje ispita
ID_DEF_PITANJA	number(10)	NE	Foreign key	Referenca na definirana pitanja
ODGOVORENO	varchar2(1)	NE	Default='N'	U trenutku kada se odgovori točno 'N' se postavi u 'T'
STVORIO	varchar2(30)	NE		Korisnik dodavanja
STVORENO	date	NE		Datum dodavanja
IZMIJENIO	varchar2(30)	NE		Korisnik izmjene
IZMIJENJENO	date	NE		Datum izmjene
ODGOVORI_POLAGACA				
ID	number(10)	NE	Primary key	Odgovor/i kojeg je korisnik označio
ID_POS_PITANJA	number(10)	NE	Foreign key	Referenca na postavljeno pitanje
ID_DEF_ODGOVORA	number(10)	NE	Foreign key	Referenca na definirani odgovor
DATUM	date	NE		Datum i vrijeme odgovora polagača
STVORIO	varchar2(30)	NE		Korisnik dodavanja
STVORENO	date	NE		Datum dodavanja

IZMIJENIO	varchar2(30)	NE		Korisnik izmjene
IZMIJENJENO	date	NE		Datum izmjene

U Tablici 1 možemo vidjeti kako sve tablice iz baze podataka „e-ispiti“ imaju takozvana audit polja, a to su *stvorio, stvoreno, izmijenio, izmijenjeno*. Budući da mora postojati zapis kod stvaranja novog ili ažuriranja postojećeg zapisa ta se polja automatski pune te se na taj način njima ne može manipulirati zato jer njihove vrijednosti baza sama mijenja, najčešće pomoću okidača o kojima će biti riječi u nastavku.

Također treba napomenuti kako danas postoje i programi za rječnike podataka, takozvane „baze o bazi“, koji su integrirani u današne sustave za upravljanje bazom podataka, međutim u ovom slučaju se rječnik podataka odnosi na prethodno opisani dokument.

5. Pregled poslovnih pravila

Prije nego što se krene u fizičko modeliranje baze podataka treba definirati neka poslovna pravila koja omogućavaju funkcionalnost baze podataka.

Kao što je i ranije navedeno, sve tablice u bazi moraju imati jedinstveni identifikator, odnosno takozvani umjetni ključ *id*, te audit polja koja daju informacije o korisniku koji je i kada unio neki podatak ili o korisniku koji je zadnji mijenjao podatke, te datumu mijenjanja.

Sva polja, osim polja *adresa* u tablici *Organizacija*, *email* u tablici *Korisnici*, *napomena* u tablici *Polaganje_ispita*, *slika* u tablicama *Def_pitanja* i *Def_odgovori* su obavezna (NOT NULL).

Šifre ispita, organizacije i korisnika, kao i lozinka polagača te nazivi organizacije, tipa ispita i definiranog ispita moraju biti jedinstveni unutar tablice, a datum polaganja ispita ne smije biti veći ili manji od današnjeg.

Kada se brišu pitanja, automatski se moraju brisati i odgovori koji pripadaju tom pitanju.

Polagaču ispita se prije polaganja ispita trebaju prikazati svi opisi odgovora za odabrani ispit.

Korisnik koji zadaje ispit mora odabrati koja će se pitanja pojaviti u ispitu, a pritom se ona ne smiju ponavljati, odnosno ne smiju se pojaviti dva ista pitanja.

Polagač ispita najprije mora odabrati ispit koji mora polagati, na primjer ispit iz zaštite na radu o posebnim stojevima ili pak ispit o električnoj struji te se na temelju njegovog odabira trebaju izgenerirati ona pitanja koja pripadaju tom određenom ispitu u onom broju koliko je definirano, na primjer pet od mogućih dvadeset i to metodom slučajnog odabira.

Kada polagač ispita odgovori točno na postavljeno pitanje u tablici postavljena_pitanja se redak odgovoreno mijenja iz 'N' (netočno) u 'T' (točno).

5.1. Objekti u bazi podataka

Fizičko modeliranje relacijskih baza podataka se sastoji od definiranih objekata i veza među njima. To su objekti koje treba stvoriti i koji zajedno tvore bazu podataka. Najčešće korišteni objekt je tablica, dok ostali objekti mogu biti pogledi, indeksi, sekvence, procedure, okidači, odnosno svi objekti koji počinju naredbom CREATE. Razumijevanje značajki i

funkcionalnosti koje svaki od tih objekata baze podataka pruža je važno za implementaciju dobro dizajnirane baze podataka koja zadovoljava potrebe pohrane podataka u skladu s poslovnim pravilima.

Naredbom CREATE TABLE definira se jedna relacija iz baze to jest njeno ime, te imena i tipovi atributa. Potrebno je definirati koji atributi čine primarni i jedinstveni ključ relacije, a također i atribute koji povezuju dvije tablice, odnosno strane ključeve.

Primjer kreiranja tablice *Korisnici*:

```
CREATE TABLE korisnici (
   id           NUMBER(10)      NOT NULL,
   id_org       NUMBER(10)      NOT NULL,
   sifra        VARCHAR2(10)    NOT NULL,
   ime          VARCHAR2(30)    NOT NULL,
   prezime      VARCHAR2(30)    NOT NULL,
   email        VARCHAR2(30),
   rola         VARCHAR2(30)    NOT NULL,
   stvorio      VARCHAR2(30)    NOT NULL,
   stvoreno     DATE            NOT NULL,
   izmijenio    VARCHAR2(30)    NOT NULL,
   izmijenjeno  DATE            NOT NULL,
   CONSTRAINT kor_pk PRIMARY KEY (id),
   CONSTRAINT kor_org_fk FOREIGN KEY (id_org) REFERENCES
      organizacija(id),
   CONSTRAINT kor_sif_uk UNIQUE (sifra)
);
```

Podaci unutar pojedine tablice ponekad mogu imati dodatna ograničenja. U tablici *Definirani_odgovori* polje *tocno* te u tablici *Postavljena_pitanja* polje *odgovoreno* označavaju koji su odgovori točni, a koji nisu, odnosno na koja je polagač točno ili netočno odgovorio. Zato je potrebno dodati ograničenja koja moraju biti 'T' ili 'N' kojima se onemogućava unošenje nekog drugog znaka.

```
CONSTRAINT def_odg_toc_ck CHECK (tocno IN ('T', 'N'));

CONSTRAINT post_pit_odg_ck CHECK (odgovoreno IN ('T', 'N'));
```

Jedno od poslovnih pravila nalaže da ako se u bazi podataka „e-ispiti“ izbrišu definirana pitanja da se automatski izbrišu i odgovori koji pripadaju tom pitanju. To omogućava opcija

ON DELETE CASCADE koju može sadržavati vanjski ključ na relaciji roditelj dijete (engl. *parent-child*). Znači ako se obriše roditelj, u ovom slučaju je to tablica *Definirana_pitanja*, obrisat će se i potomak *Definirani_odgovori* koji je referenciran na primarni ključ tablice:

```
CONSTRAINT def_odg_def_pit_fk FOREIGN KEY (id_def_pitanja)
   REFERENCES def_pitanja(id) ON DELETE CASCADE;
```

Indeksi su objekti u bazi podataka koji se koriste za brže dohvaćanje vrijednosti najčešće korištenih redaka u tablici. Kod tablica s jako puno podataka njihovo dohvaćanje bi moglo dosta dugo potrajati te se zato na stupcima kojima se često pristupa stvaraju indeksi. Budući da Oracle baza automatski sama stvara indekse na ranije spomenutim PK i UK stupcima indekse treba dodati i na strane ključeve.

Primjer kreiranja indeksa za strane ključeve iz tablice *Polaganje_ispita*:

```
CREATE INDEX pol_isp_def_isp_ind
   ON polaganje_ispita (id_def_ispita);

CREATE INDEX pol_isp_polc_isp_ind
   ON polaganje_ispita (id_polagaca);
```

Također je potrebno osigurati automatski rast vrijednosti primarnog ključa, a tome služi objekt u bazi podataka koji se naziva sekvenca:

```
CREATE SEQUENCE sekvenca_za_id
INCREMENT BY 1 START WITH 1
MAXVALUE 99999999999 MINVALUE 1
NOCACHE;
/
```

5.2. Triggeri i paketi

Triggeri ili okidači (engl. *triggers*) su PL/SQL skripte pohranjene u bazi podataka koji se izvršavaju, odnosno okidaju prilikom izvršenja DML (*Data Manipulation Language*) naredbi INSERT, UPDATE, DELETE ili DDL (*Data Definition Language*) naredbi CREATE, ALTER. Omogućavaju efikasnije korištenje i upisivanje u bazu podataka te njima možemo automatski generirati neke vrijednosti u stupcima, možemo bilježiti prijave korisnika, prikupljati podatke o pristupima određenoj tablici, spriječiti nevažeće transakcije i slično.

Trigger se kreira naredbom CREATE TRIGGER, a zatim treba odrediti koji će događaj biti pokretač triggera i na koji će način djelovati trigger. U bazi podataka „e-ispiti“ korišteni su retčani (ROW) okidači koji se automatski okidaju za DML naredbu INSERT.

Kreirani okidač će automatski prije nego se spremi novi unos (BEFORE INSERT ON) umetnuti sljedeću vrijednost iz sekvence za svaki redak (FOR EACH ROW). Time će se za svaki unos povećavati vrijednost primarnog ključa. Jednako tako će se okinuti i radi implementacije audit polja, a budući da je retčani okida se nakon svakog retka koji se unosi:

```
CREATE OR REPLACE TRIGGER bir_def_ispiti
   BEFORE INSERT
   ON def_ispiti
   FOR EACH row
BEGIN
   :NEW.id := pomocni_pak.sekvenca (:NEW.id);
   pomocni_pak.upisi_audit_polja (
      :NEW.stvorio,
      :NEW.stvoreno,
      :NEW.izmijenio,
      :NEW.izmijenjeno );
END;
/
```

Okidaju se također za svaku tablicu radi implementacije audita, ali samo za DML naredbu UPDATE:

```
CREATE OR REPLACE TRIGGER bur_def_ispiti
   BEFORE UPDATE
   ON def_ispiti
   FOR EACH row
BEGIN pomocni_pak.upisi_audit_polja (
   :NEW.stvorio,
   :NEW.stvoreno,
   :NEW.izmijenio,
   :NEW.izmijenjeno );
END;
/
```

Kako bi se spriječila mogućnost da korisnik sam unosi da li je odgovor na postavljeno pitanje točan ili ne u okidaču *bir_postavljena_pitanja* se unosi još jedna naredba:

```
    :NEW.odgovoreno := 'N';
```

kojom je spriječena tranzicija NULL → 'T'.

Procedura je objekt u bazi podataka koja sadrži određeni dio koda, odnosno proceduru koja se sastoji od deklarativnih SQL naredbi, kao što su CREATE, UPDATE i SELECT te mogućih proceduralnih naredbi, poput IF-THEN-ELSE i WHILE-DO koje su spremljene u bazi podataka i koje se mogu pozvati iz programa, triggera ili čak druge procedure (Lans, 2006).

U bazi podataka „e-ispiti“ kreirana su i dva paketa, paket *random* koji sadrži proceduru za generiranje postavljenih pitanja metodom slučajnog odabira za određeni ispit, te paket *provjera* koji sadrži proceduru koja provjerava je li neki odgovor točan ili nije te ažurira tablicu *Postavljena_pitanja*.

U paketu *random* najprije je trebalo definirati kursor koji dohvaća nasumično odabrana pitanja iz tablice *Def_pitanja*, ali točno onoliko pitanja koliko je definirano u tablici *Def_ispiti*:

```
CURSOR pitanje_cursor (p_id number) IS
    SELECT id
    FROM (
        SELECT di.broj_pitanja, dp.id
        FROM def_ispiti di, def_pitanja dp
        WHERE di.id = dp.id_def_ispita
        AND di.id = p_id
        ORDER BY DBMS_RANDOM.VALUE )
    WHERE rownum <= broj_pitanja;
```

Nakon toga je kreirana procedura *generiraj_pitanja* u kojoj su najprije deklarirani parametri, odnosno ulazni podaci koji kreiraju objekt, u ovom slučaju tablicu *Polaganje_ispita*:

```
procedure generiraj_pitanja (
    p_id_def_ispita    IN def_ispiti.id%TYPE,
    p_id_polagaca      IN polaganje_ispita.id_polagaca%TYPE,
    p_id_datum         IN polaganje_ispita.datum%TYPE
) AS
...
```

Zatim se dohvaća kursor na temelju parametra *p_id_def_ispita* te se svaka pojedina instanca objekta sprema u varijablu *v_id_pitanja*, a zatim se dobiveni podaci upisuju u tablicu *Postavljena_pitanja*:

```
...
OPEN pitanje_cursor(p_id_def_ispita);
   LOOP
   FETCH pitanje_cursor
   INTO v_id_pitanja;
   EXIT WHEN pitanje_cursor%NOTFOUND;
   INSERT INTO postavljena_pitanja (id_pol_isp, id_def_pitanja)
      VALUES (v_id_polaganja_ispita, v_id_pitanja);
   END LOOP;
CLOSE pitanje_cursor;
...
```

Jednako kao kod paketa *random*, i u paketu *provjera* treba kreirati kursor. Kursor u ovom slučaju dohvaća *id* postavljenog pitanja iz tablice *Odgovori_polagaca*, ali samo ona koja su definirana kao točna, što se može vidjeti u tablici *Def_odgovori*:

```
CURSOR odgovoreno_cursor (p_id number) IS
   SELECT op.id_pos_pitanja
   FROM odgovori_polagaca op, def_odgovori do
   WHERE op.id_def_odgovora = do.id
   AND op.id_pos_pitanja = p_id
   AND do.tocno = 'T';
```

U proceduri *provjeri_odgovor* su definirani parametri koje unosi polagač ispita prilikom odgovaranja na pitanja, a to su polagačev odgovor i polja koja će se automatski unositi *id_pos_pitanja* i *datum*:

```
procedure provjeri_odgovor (
   p_id_pos_pitanja  IN postavljena_pitanja.id%TYPE,
   p_id_def_odgovora IN odgovori_polagaca.id_def_odgovora%TYPE,
   p_datum           IN odgovori_polagaca.datum%TYPE
) AS
...
```

Na temelju parametra *p_id_pos_pitanja* se dohvaća kursor koji u tablici *Postavljena_pitanja* mijenja iz 'N' u 'T' ona pitanja na koja je dan točan odgovor:

```
...
OPEN odgovoreno_cursor(p_id_pos_pitanja);
LOOP
   FETCH odgovoreno_cursor
   INTO v_id_pos_pitanja;
   EXIT WHEN odgovoreno_cursor%NOTFOUND;
   UPDATE postavljena_pitanja
   SET odgovoreno = 'T'
   WHERE id = v_id_pos_pitanja;
END LOOP;
CLOSE odgovoreno_cursor;
...
```

Sigurnost baze podataka se može promatrati kroz aspekt očuvanja poslovnih podataka, ali i ograničenja pristupa određenim korisnicima. Radi toga su kreirana audit polja u svim matičnim i transakcijskim tablicama koja čuvaju podatke o tome koji korisnik je unio novi podatak ili izvršio ažuriranje nekog podatka.

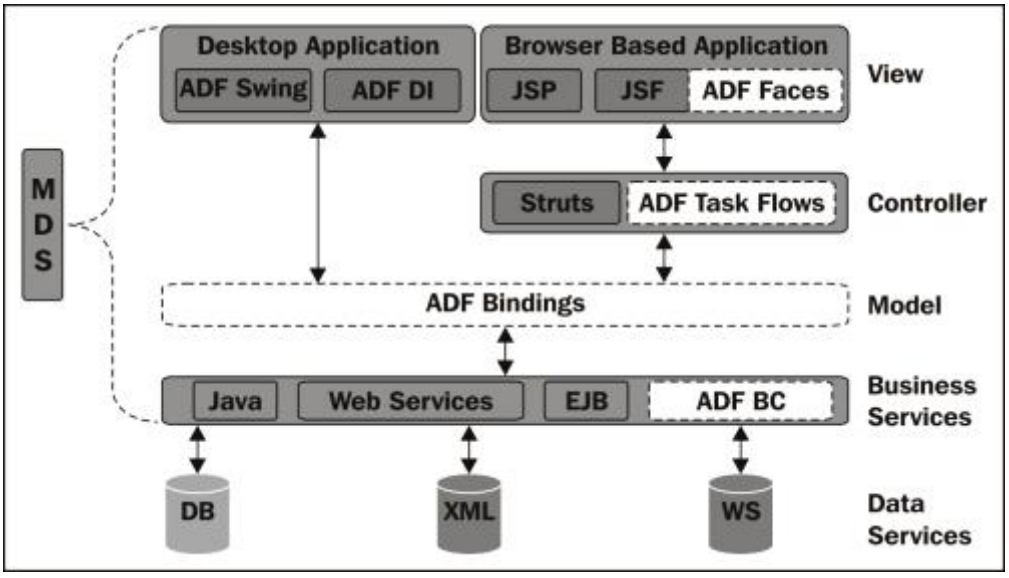
To je omogućeno PL/SQL paketom koji automatski i bez znanja korisnika kreira vrijednosti audit polja i automatski ih pohranjuje u bazu. PL/SQL paket predstavlja objekt u koji su pohranjene procedure i to one koje su zajedničke za više tablica. Radi toga nije potrebno za svaku tablicu kreirati posebnu proceduru već je dovoljna samo jedna i onda je ta jedna procedura zajednička za sve tablice baze što ujedno omogućuje efikasan rad baze.

Sama izvedba poslovnih pravila zamišljena je na način da je jedan dio baze kreiran kroz SQL i Oracleov PL/SQL, a jedan dio funkcionalnosti prepušten je aplikacijskim rješenjima koja će raditi s bazom. Na taj način korisnici koji definiraju ispite, ali i polagači imaju ograničene mogućnosti koje su zadane poslovnim pravilima.

6. Oracle ADF

Oracle *Application Development Framework* (Oracle ADF) je razvojni framework koji je građen na Java EE standardima i *open-source* tehnologijama kako bi se pojednostavio razvoj i ubrzala implementacija poslovnih aplikacija te se fokusira se na vizualnom i deklarativnom pristupu razvoja. Oracle ADF je pogodan za razvojne programere koji žele kreirati takozvane CRUD (*Create, Read, Update, Delete*) aplikacije koristeći pritom web, mobilna ili desktop sučelja. Također treba spomenuti kako razvojni programeri imaju mogućnost korištenja cjelokupnog ADF frameworka za kreiranje aplikacije, ali mogu koristiti i samo neke njegove dijelove. Oracle ADF pruža ogromni stupanj fleksibilnosti budući da može biti povezan s Oracle bazom podataka ili nekom drugom bazom podataka koja nije Oracleova.

Baziran je na takozvanoj *Model-View-Controller* (MVC) arhitekturi koja odvaja poslovnu logiku (*Model*) od klijenta (*View*) i logike upravljačkog toka (*Controller*), međutim ADF proširuje svoju arhitekturu na četiri sloja (Krishnan, 2013) koji su prikazani na Slici 4.



Slika 4. ADF arhitektura
[Izvor: Vinod Krishnan: *Oracle ADF 11gR2 Development Beginner's Guide*]

Prvi sloj koji je odgovoran za interakciju između baze podataka i sloja *Model* te za većinu poslovne logike, odnosno performanse same aplikacije je *Business Service* sloj koji dostavlja podatke i prihvaća instrukcije za unos, izmjenu, brisanje podataka te ostala poslovna pravila.

Na njega se nadovezuje *Model* sloj koji povezuje korisničko sučelje aplikacije s poslovnim servisom te omogućava njegovu promjenu, a da to ne utječe na ostale dijelove aplikacije. *Controller* sloj kontrolira tok korisničkog sučelja i često je uključen u navigaciju s jedne stranice na drugu putem ADF *task flow*-a te određuje što će se dogoditi u interakciji korisnika s nekom od komponenti. *View* sloj je zapravo prezentacijski sloj budući da njega koristi krajnji korisnik za pregledavanje i interakciju s aplikacijom.

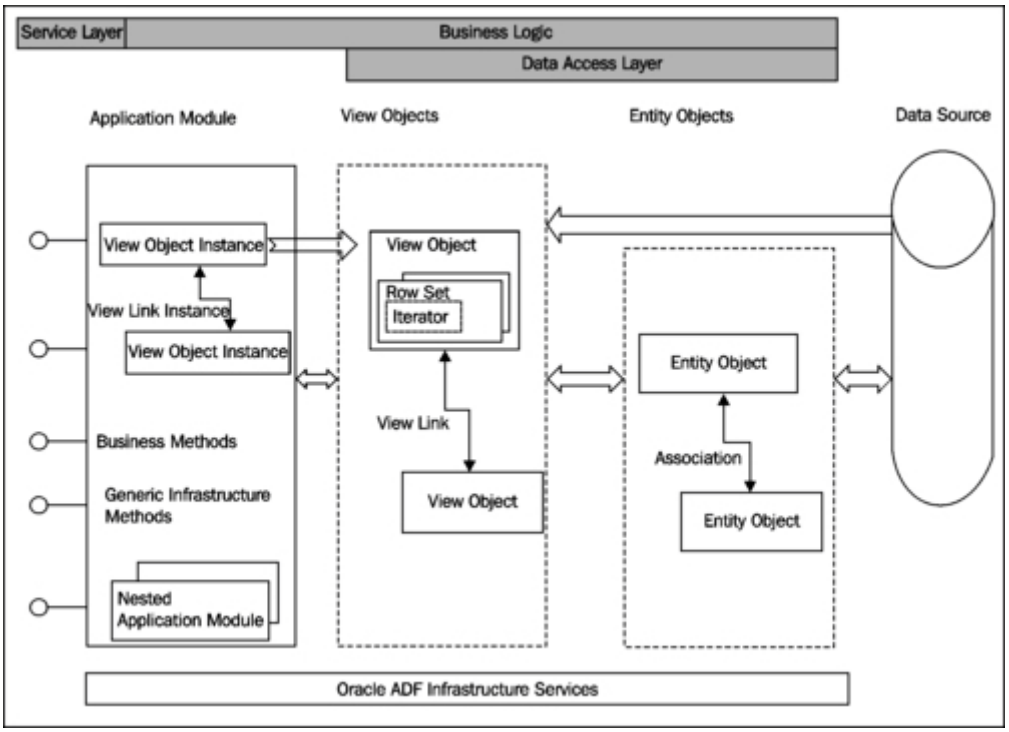
Ukratko, Oracle ADF se može podijeliti na dva dijela, odnosno *frontend* dio u koji spada MVC arhitektura i *backend* dio u koji spada ADF *Business Service*. Web aplikacije koje koriste ADF tehnologiju kroz sloj poslovnog servisa te MVC sloja se nazivaju Fusion web aplikacije. Oracle ADF između ostalog pojednostavljuje razvoj Java aplikacija minimizirajući potrebu za pisanjem koda te na taj način omogućuje razvojnim programerima da se fokusiraju na potrebe aplikacije i potrebe krajnjih korisnika, a ne na njezinu konfiguraciju.

6.1. ADF Business Components

ADF *Business Components* (ADF BC) je dio ADF frameworka, odnosno jedan od dijelova sloja poslovnih servisa koji direktno surađuje sa slojem *Model*. Idealan je za razvoj poslovnih aplikacija, osobito onih koje u pozadini imaju relacijski, a ne objektni model baze te je radi toga pogodan i za programere koji potječu iz 4. generacije programskih alata kao što su Oracle Forms, PeopleTools, PowerBuilder ili Visual Basic.

ADF BC za razliku od ostalih tehnologija poslovnih servisa pruža deklaratvni framework za implementaciju poslovnih pravila i poslovne logike baze podataka u aplikaciju. Upravo radi tog deklarativnog pristupa u kreiranju poslovnih komponenti olakšava posao razvojnim programerima budući da oni samo povremeno trebaju pisati proceduralni kod za validaciju ili pak samo intervenirati u kodu te ga oblikovati po želji.

Na Slici 5. možemo vidjeti ADF BC arhitekturu gdje se jasno može razlučiti svaka odvojena komponenta.



Slika 5. ADF BC arhitektura
[izvor: Jobinesh Purushothaman: *Oracle ADF Real World Developer's Guide*]

Prvi sloj ADF BC arhitekture čine entity objekti (engl. *Entity Objects*) koji predstavljaju tablice u bazi podataka i instance entity objekata koje predstavljaju jedan redak u tablici. Entity objekti mogu biti međusobno povezani asocijacijama (engl. *Associations*) koje predstavljaju strane ključeve, ali i ostala ograničenja u bazi podataka.

Drugi sloj BC-a je sastavljen od view objekata (engl. *View Objects*) koji se mogu temeljiti na jednom ili više entity objekata ili pak na SQL upitu. Kao što entity objekti mogu biti spojeni asocijacijama, tako view objekti mogu biti povezani linkovima (engl. *View Links*) koji se ili automatski generiraju ili ih se može kreirati. Oni predstavljaju takozvanu master-detalj (engl. *master-detail*) vezu između objekata.

Posljednji sloj ADF BC-a je aplikacijski modul (engl. *Application Module*) koji predstavlja transakcijski spremnik instanci view objekata pojedinog poslovnog zadatka.

Izvor podataka za ADF aplikaciju može biti bilo koji tradicionalni RDBMS kao što su Oracle, MySQL, MS SQL Server, DB2 ili slični.

6.1.1. Entity objekti i asocijacije

Entity objekti brinu o objektno-relacijskom mapiranju na način da relacije, odnosno tablice iz baze podataka postaju dostupne aplikaciji kao Java objekti te samim time formiraju temelje na kojima su izgrađeni view objekti, o kojima će biti riječi u nastavku.

Sve modifikacije podataka u biti prelaze preko entity objekata ili u prevedenom značenju za svaku tablicu ili pogled iz baze podataka postoji recipročan entity objekt na koji se aplikacija oslanja te je taj isti objekt zaslužan za izvođenje točno određenih naredbi kao što su *insert, update* ili *delete*.

Entity objekti pomažu izgraditi skalabilne aplikacije s dobrim performansama tako što se ponašaju kao predmemorija (engl. *cache*) u kojoj se spremaju podaci iz baze, ali na aplikacijskom serveru te na taj način minimiziraju opterećenost koju aplikacija stvara na bazi podataka.

Kao što su entity objekti međuslojna reprezentacija tablica i pogleda iz baze podataka tako su asocijacije reprezentacija stranih ključeva, odnosno veza između dvije ili više tablica. Drugim riječima, asocijacija predstavlja vezu između dva entity objekta i dopušta ADF-u spajanje podataka jednog entity objekta s drugim. Jdeveloper je u mogućnosti kreirati asocijacije automatski, pretražujući bazu podataka, ali i u slučaju da baza podataka ne sadrži strane ključeve, asocijacije se mogu dodati ručno te se i na taj način može uspostaviti veza između dva entity objekta.

6.1.2. View objekti i linkovi

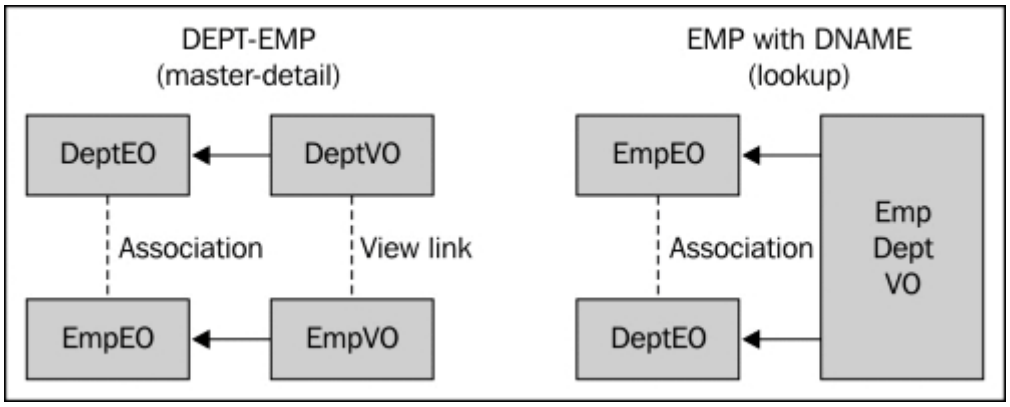
Kod izrade entity objekata ne moraju se donositi nikakve važne odluke, međutim kod izrade view objekata mora se razmisliti o tome koje će se informacije prikazvati krajnjem korisniku.

View objekti se temelje na entity objektima te se osim za prikaz podataka koriste i za liste vrijednosti (engl. *List of Values*). Jedan prikaz može sadržavati više view objekata kao u slučaju master-detalja (engl. *master-detail*) koji na primjer služi tome da se u jednom prikazu može vidjeti pojedina organizacija te popis svih njenih korisnika, odnosno da jedan prikaz sadrži i view objekt organizacije i view objekt korisnika.

Jedan view objekt može biti sastavljen od nekoliko entity objekata. Na primjer entity objekt *Korisnik* sadrži *id_organizacije* koji krajnjem korisniku ništa ne znači te bi bilo dobro da umjesto *id_organizacije* vidi naziv organizacije koji se nalazi u entity objektu *Organizacija*. Zato nije potrebno kreirati zasebni view objekt već samo treba uključiti entity objekt organizacije u view objekt korisnika.

Liste vrijednosti se koriste za padajuće menije na korisničkom sučelju te se najčešće definiraju samo za čitanje (engl. *read only*) te ih se može definirati samo jednom i ponovno koristiti gdje god postoji potreba za time.

Linkovi se koriste za definiranje *master-detail* veza između view objekata koji se najčešće temelje na asocijacijama što se može vidjeti na Slici 6.



Slika 6. Dva načina prikaza view objekata
[izvor: Sten E. Vesterli: *Oracle ADF Enterprise Application Development – Made Simple*]

Osim linkova na Slici 6. mogu se vidjeti dva prethodno opisana načina prikaza view objekata. Lijeva strana slike prikazuje situaciju gdje je na ekranu *master-detail* veza organizacije sa svim korisnicima te se u tom slučaju se kreiraju dva view objekta koji su povezani linkovima, dok desna strana slike prikazuje situaciju gdje je potrebno vidjeti prikaz svih zaposlenika s nazivom organizacije kojoj pripadaju. U tom slučaju je potreban samo jedan view objekt koji je temeljen na SQL upitu i koji povezuje korisnike i organizaciju kroz entity objekte.

6.1.3. Aplikacijski modul

Aplikacijski modul enkapsulira instance view objekata i metode poslovnih servisa koje su potrebne za njegov rad. Svaki aplikacijski modul ima svoj vlastiti transakcijski kontekst i vlastitu konekciju s bazom podataka. To znači da je sav posao koji korisnik izvodi pomoću view objekata iz jednog aplikacijskog modula dio jedne transakcije baze podataka (Vesterli, 2014). Aplikacijski modul može sadržavati i druge ugniježđene (engl. *nested*) aplikacijske module, no malu aplikacije je dovoljno izgraditi na samo jednom aplikacijskom modulu.

6.2. Praktični primjer – Slaganje modela u ADF-u

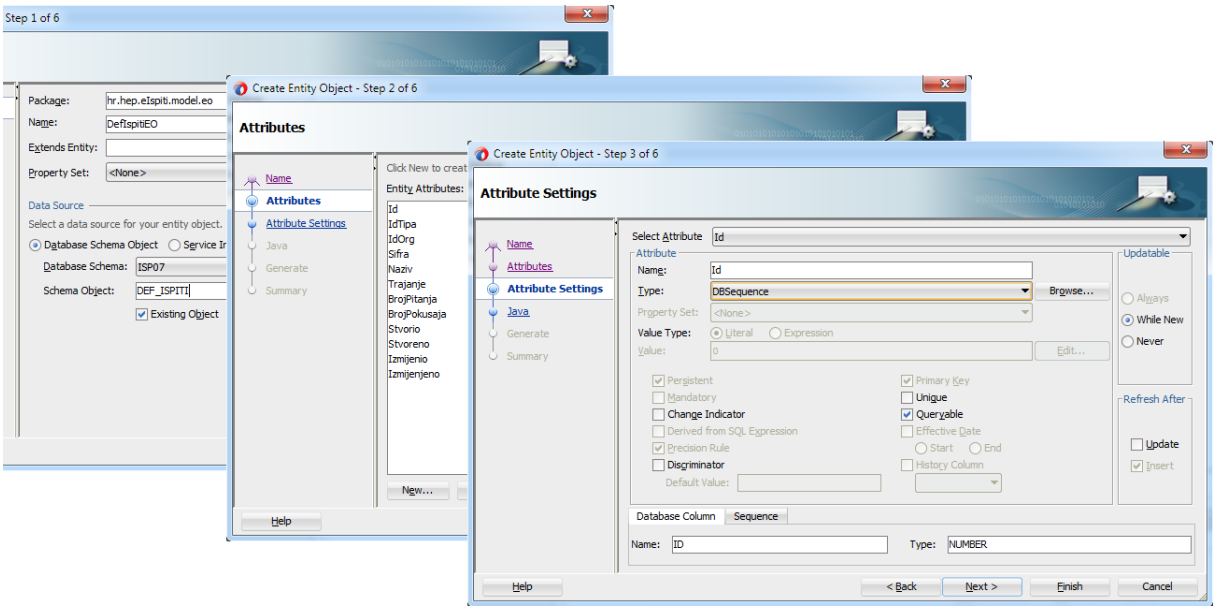
Kako bi se izgradio ADF *Business Components* sloj najprije je potrebno kreirati entity objekte koji zapravo predstavljaju tablice iz baze podataka i koji čine temelj buduće aplikacije. Kod kreiranja entity objekata važno je obratiti pažnju i na validacije koje se rješavaju deklarativno te na taj način smanjuju kompleksnost izrade validacijske logike. Nakon toga slijedi izgradnja view objekata koji predstavljaju one podatke koje krajnji korisnik treba vidjeti te ih se prilagođava u skladu s potrebama. Na kraju slijedi izrada aplikacijskog modula u kojem se zapravo može istestirati funkcionalnost buduće aplikacije i to bez potrebe dizajna.

6.2.1. Kreiranje i konfiguracija entity objekata

ADF-ovi entity objekti developerima skrivaju fizičke objekte iz baza podataka te im omogućuju rad s podacima u formi Java objekata. Poslovna logika koja je ugrađena u ADF *Business Components* entity objekt predstavlja kao tablicu iz baza podataka dok instanca entity objekta predstavlja redak u tablici.

Entity objekti štite developera od kompleksnosti izvornog koda podataka koristeći koncept objektno relacijskog mapiranja (Purushothaman, 2012). Postoje dva načina kreiranja entity objekta, koristeći opciju *Create Business Components from Table* koja omogućuje kreiranje više entity objekata od jednom ili *Create Entity Object* opciju koja omogućuje kreiranje svakog entity objekta zasebno. Takvim načinom kreiranja mapiraju se objekti iz baze podatka u entity objekte, te im se dodaje sufiks `EO` radi bolje organizacije i lakšeg snalaženja u kasnijim fazama razvoja aplikacije. Na primjer tablica `DEF_ISPITI` iz baze podataka sada

postaje entity objekt `DefIspitiEO`. Na Slici 7. možemo vidjeti kako se entity objekt kreira u nekoliko jednostavnih koraka.

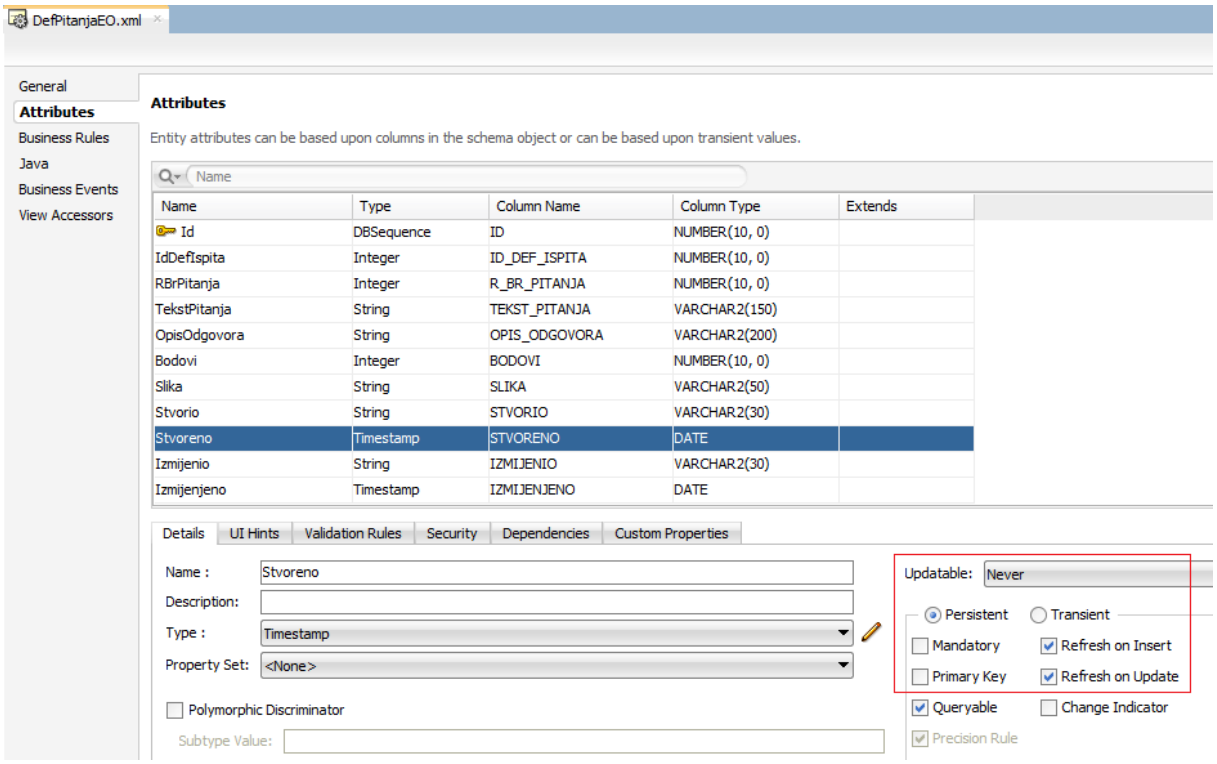


Slika 7. Kreiranje entity objekta [izvor: autor]

U prvom koraku se mapira objekt iz baze podataka `DEF_ISPITI` u entity objekt `DefIspitiEO` kojeg se smješta u paket `hr.hep.eIspiti.model.eo`. Nakon toga se u sljedećem koraku pored postojećih atributa mogu stvoriti i takozvani *transient* atributi, odnosno prijelazni atributi koji se ne temelje na tablici iz baze podataka već su njihove vrijednosti izvedene iz drugih atributa i koji se ponekad koriste kao privremene varijable radi implementacije poslovne logike. U trećem koraku za svaki atribut moramo odrediti njegova svojstva, odnosno provjeriti tip podataka, primarne ključeve, može li se ažurirati i slično.

Za tip primarnog ključa u svim entity objektima odabran je `DbSequence` kako bi se mogao koristiti trigger za sekvencu koji je prethodno kreiran u bazi podataka i opisan u petom dijelu rada.

Osim triggera za sekvencu, također je u bazi kreiran i trigger za unos audit polja, stoga je za svaki atribut `Stvorio`, `Stvoreno`, `Izmijenio`, `Izmijenjeno` u svim entity objektima potrebno označiti kako se vrijednost atributa treba osvježiti nakon unosa ili izmjene, što se može vidjeti na Slici 8.

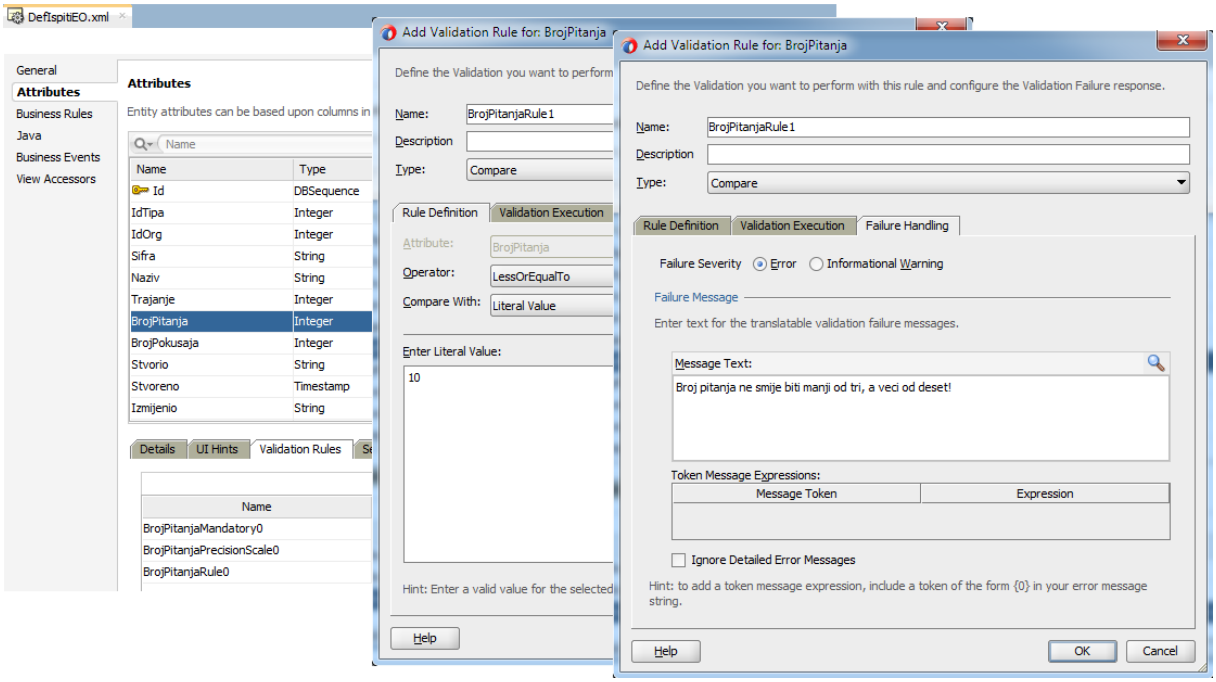


Slika 8. Svojstva atributa entity objekta `DefPitanjaEO` [izvor: autor]

Kod mapiranja entity objekata iz baze podataka svi atributi imaju predefinirano svojstvo *updateable – always*, međutim kada je potrebno onemogućiti korisniku mijenjanje podataka kod novog unosa ili ažuriranja nekog atributa tada se može odabrati opcija *updateable* – never što se također može vidjeti na gore prikazanoj slici. Treba napomenuti kako osim opcija *updateable – never*, odnosno *always* postoji i opcija *while new* za redak u tablici koji se u tom slučaju ne može ažurirati, međutim moguć je unos novog retka.

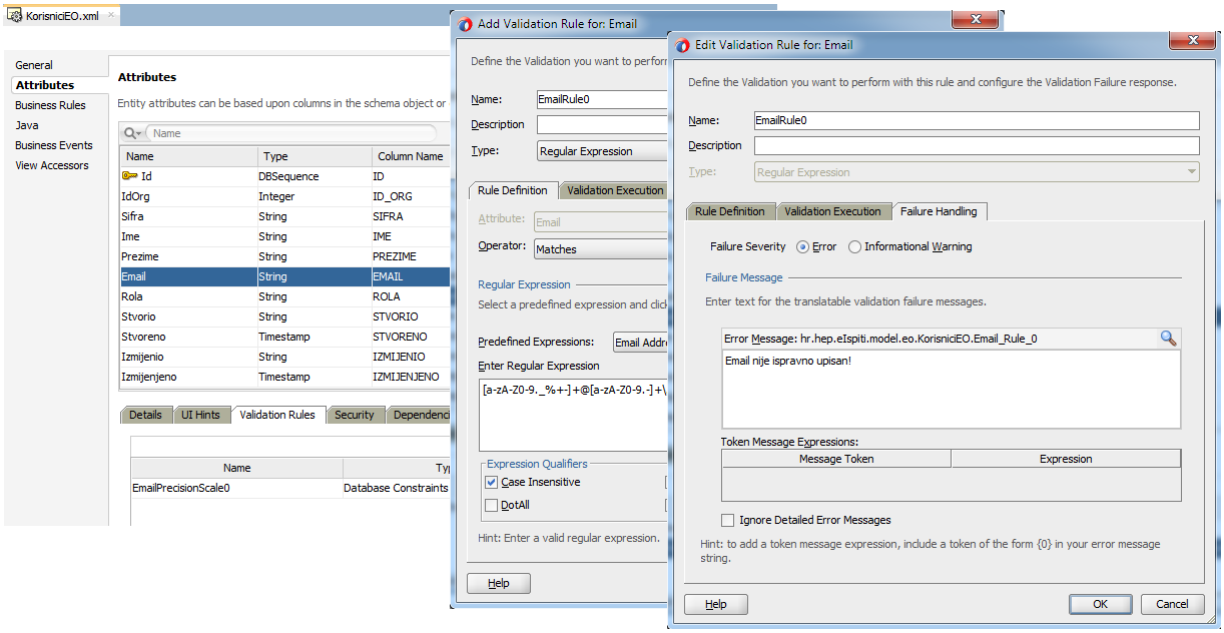
Atributi mogu imati i zadane vrijednosti koje se mogu postaviti na tri različita načina. Prvi od njih je *Literal*, gdje je vrijednost atributa statička, drugi način je *Expression* gdje se na primjer umjesto atributa `Datum` u `PolaganjeIspitaEO` može upisati `adf.currentDate`, a treći način je korištenje izvedenog SQL izraza.

Budući da entity objekti osim što preslikavaju objekte iz baze podataka služe i za konfiguriranje validacije mogu im se dodijeliti još neke dodatne validacije, kao na primjer da broj pitanja u ispitu koji se polaže ne smije biti manji od tri, a veći od pet. Primjer te validacije možemo vidjeti na slici 9.



Slika 9. Validacija za provjeru broja pitanja u `DefIspitiEO` [izvor: autor]

Osim validacije tipa *Compare*, postoji i *Regular Expression* tip validacije u kojem se provjerava da li unešena vrijednost sadrži odgovarajuće znakove, u ovom slučaju da li je korisnik unio ispravnu e-mail formu kao što se može vidjeti na Slici 10.

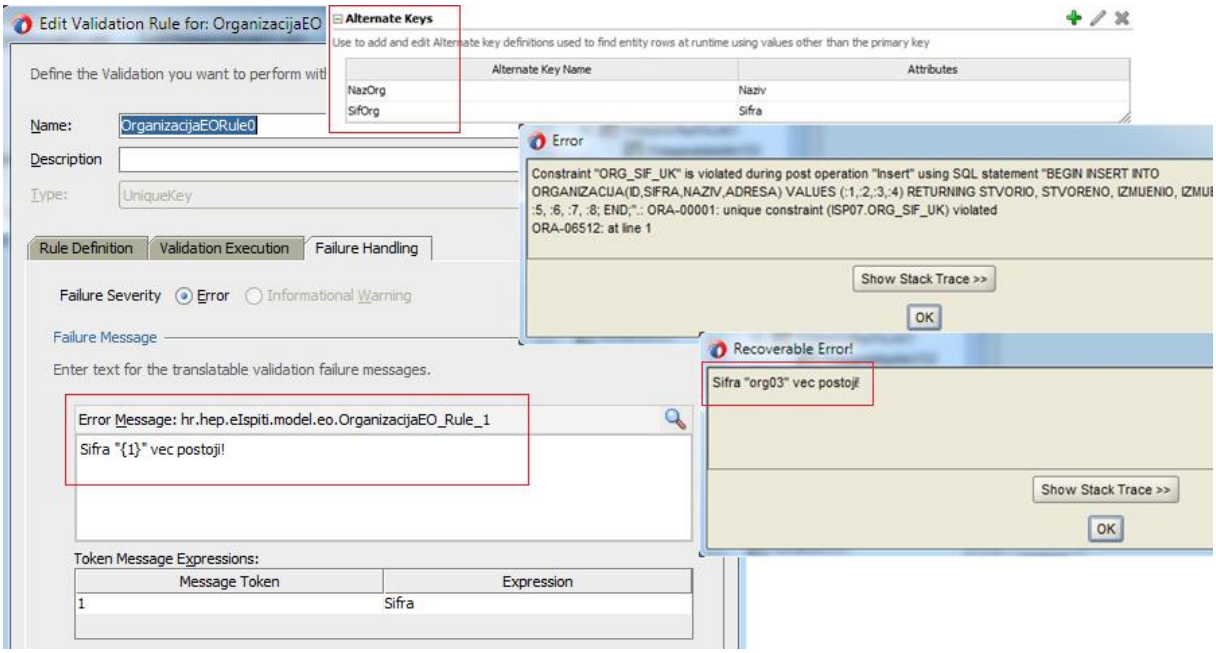


Slika 10. Validacija za provjeru email-a u `KorisniciEO` [izvor: autor]

Jedno od poslovnih pravila nalaže da šifre i nazivi u pojedinim tablicama moraju biti jedinstveni. Bez obzira na to što je jedinstvenost ovih atributa već deklarirana u bazi podataka kao *unique key*, dobro je napraviti i ovu validaciju budući da se u tom slučaju jedinstvenost provjerava na samome retku prije *commit*-a, a unos korisnika ostaje u *cache*-u te ona niti ne dospijeva do provjere ograničenja u bazi podataka.

Osim jedinstvenosti još jedno svojstvo *Alternate Key* definicije je prikazivanje čitljivije poruke krajnjem korisniku, za razliku od one koja dolazi iz baze podataka.

Kao primjer ćemo navesti entity objekt `OrganizacijaEO` čiji su atributi `Naziv` i `Sifra` dodani u *Alternate Key* definiciju kojeg možemo vidjeti na Slici 11.



Slika 11. Alternate Key atributa `Naziv` i `Sifra` u `OrganizacijaEO` [izvor: autor]

Kada u bazi podataka postoji ograničenje za strani ključ, automatski se kreiraju i asocijacije među entity objektima. Tako je kod kreiranja `DefOdgovoriEO` nastala asocijacija `DefOdgDefPitFkAssoc` koju možemo vidjeti na Slici 12.

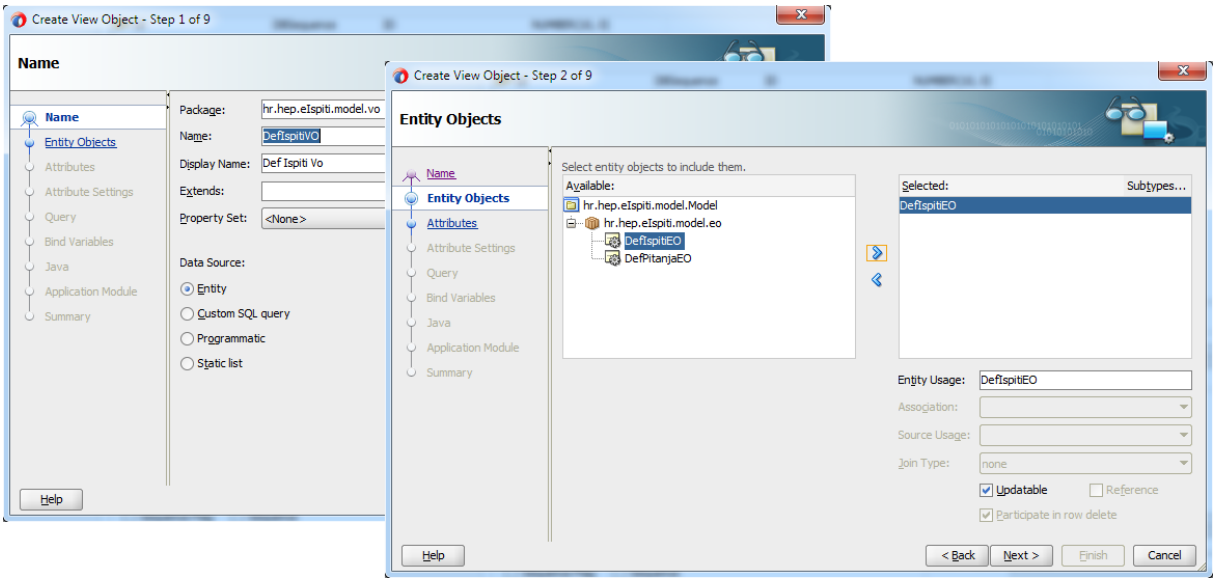
Slika 12. Asocijacija između `DefPitanjaEO` i `DefOdgovoriEO` [izvor: autor]

Na gore prikazanoj slici također možemo i vidjeti ograničenje koje je kreirano u bazi podataka, ON DELETE CASCADE, odnosno pravilo da kada se obriše pitanje, obrišu se i svi odgovori koji pripadaju tom pitanju.

6.2.2. Kreiranje i konfiguracija view objekata

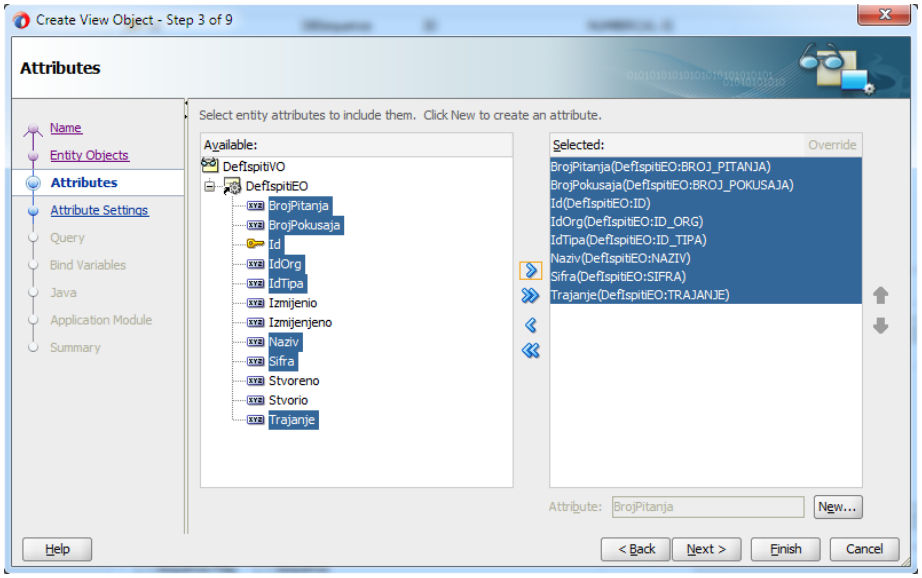
Nakon što su kreirani entity objekti, na temelju njih se mogu kreirati i takozvani *entity-based* view objekti koji mapiraju atribute jednog ili više entity objekata te na taj način omogućuju model podataka koji je moguće ažurirati. Za razliku od *entity-based* view objekata postoje i *entity-less* view objekti. To su read only view objekti koji se ne temelje na entity objektima već isključivo na SQL upitu, stoga se ne mogu ažurirati te se koriste u rijetkim i specifičnim situacijama. U svakom slučaju, pa čak i ako će se podaci view objekata koristiti samo za pregled, bolje je koristiti *entity-based* view objekte budući da oni upravljaju upitima koristeći predmemoriju entity objekata te samim time imaju bolje performanse.

View objekti imaju sufiks `VO` te se smješaju u paket `hr.hep.eIspiti.vo`, a kreiranje *entity-based* view objekta se može vidjeti na Slici 13.



Slika 13. Kreiranje view objekta `DefIspitiVO` [izvor: autor]

U prvom koraku izabiremo izvor podataka, odnosno hoće li se view objekt temeljiti na entity objektu, SQL upitu, da li će biti napisan programski ili se pak samo koristiti kao statička lista. Budući da je u ovom slučaju kao izvor podataka izabran entity objekt u sljedećem koraku treba odabrati na temelju kojeg entity objekta (ili kojih) želimo kreirati view objekt. U ovom slučaju je to `DefIspitiEO`. Nakon toga u sljedećem koraku odabiremo samo one atribute koji će biti potrebni krajnjem korisniku, a što se može vidjeti na Slici 14.

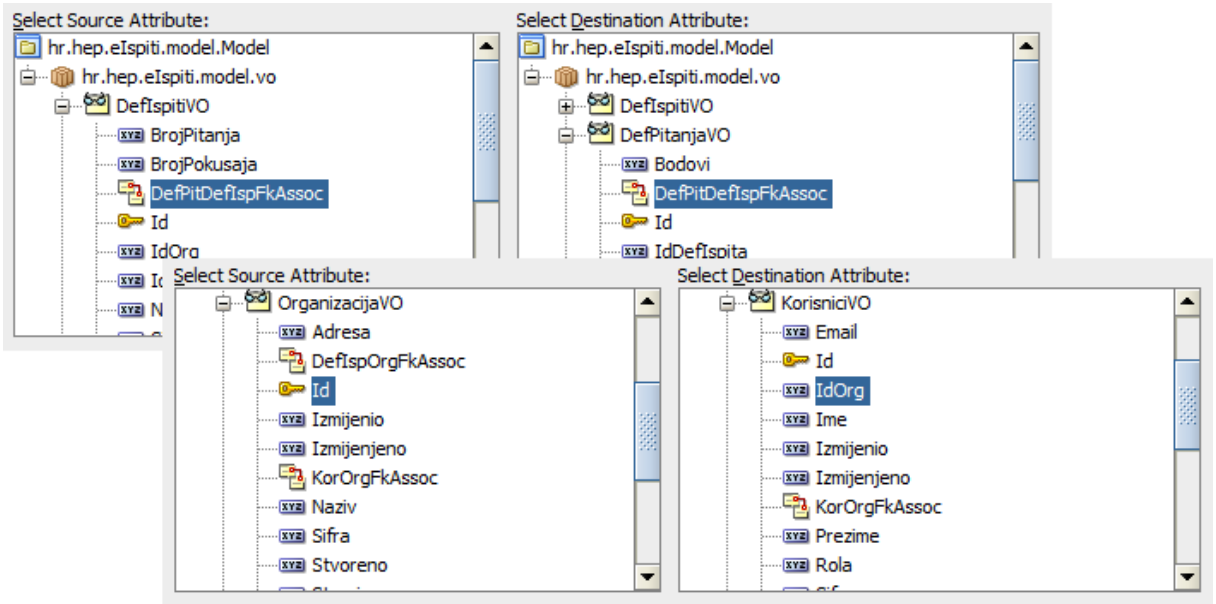


Slika 14. Izbor atributa za `DefIspitiEO` [izvor: autor]

Nakon odabira atributa se u sljedećem koraku mogu izmijeniti i njihove postavke, međutim u ovom slučaju to nije učinjeno.

Kod kreiranja view objekata u daljnjim koracima postoji i mogućnost dodavanja WHERE i/ili ORDER BY uvjeta. U slučaju da treba prikazati samo one ispite koji pripadaju određenoj organizaciji trebalo bi u WHERE uvjet staviti `IdOrg = 1` ili ako je potrebno da prikaz svih ispita bude poredan po nazivu ispita abecednim redoslijedom tada je potrebno u uvjet ORDER BY staviti atribut `Naziv` i slično. Naravno, to se sve može učiniti i naknadno, nakon što je view objekt već kreiran.

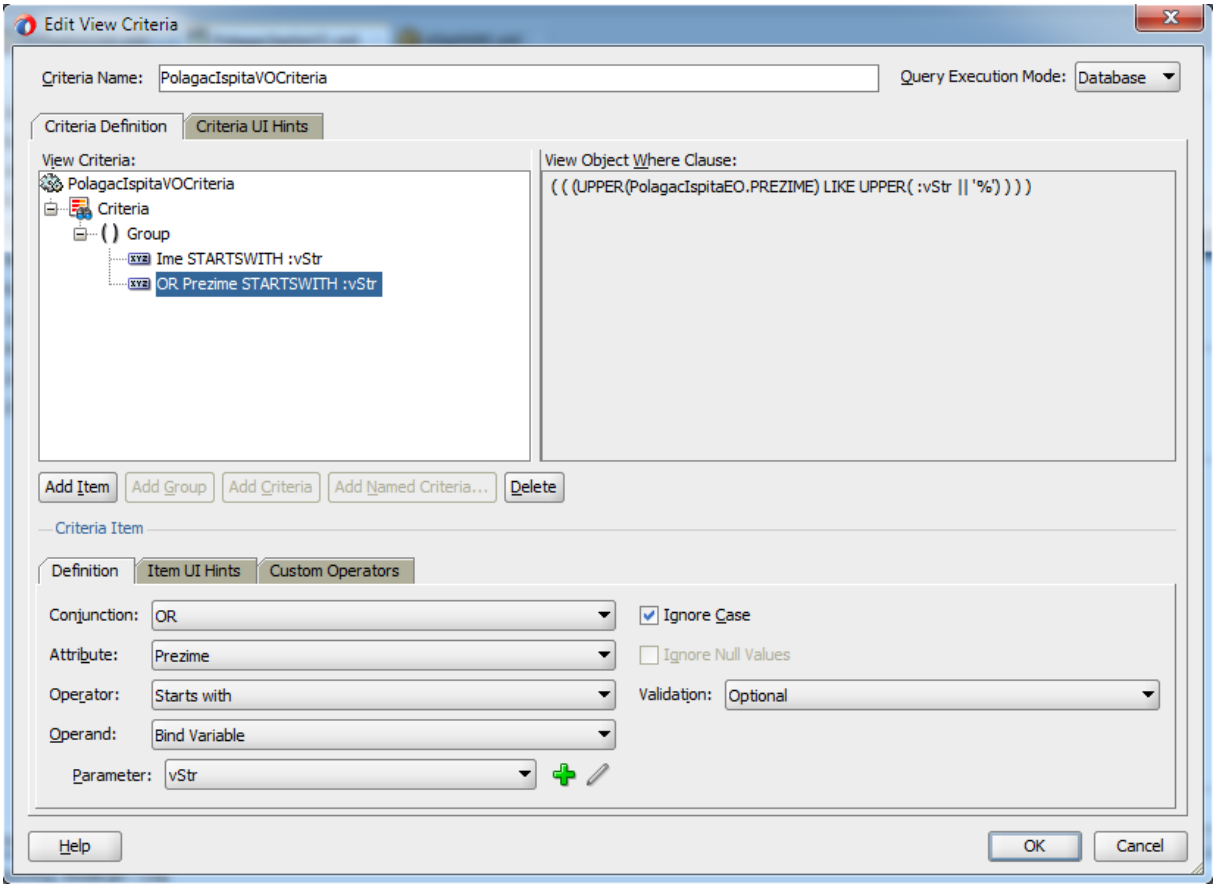
View linkovi slično kao asocijacije kod entity objekata predstavljaju veze između dva view objekta. Ako postoje dva view objekta koja se baziraju na entity objektima tada se na temelju njihovih asocijacija mogu mapirati view linkovi koji tada predstavljaju *master-detail* hijerarhiju na nivou entity objekta. Međutim, za razliku od asocijacija view linkovi ne moraju nužno povezivati view objekte kao što asocijacije povezuju entity objekte, stranim ključevima, već ih mogu povezati na način da se povežu dva ista atributa. Na Slici 15. možemo s lijeve strane vidjeti view linkove koji se temelje na asocijacijama, a s desne strane koji se temelje na atributima:



Slika 15. Kreiranje view linkova [izvor: autor]

Prethodno je spomenuto kako se view objektu naknadno može dodati WHERE uvjet, međutim još u samoj izradi view objekta postoji mogućnost kreiranja *bind* varijable koja dohvaća vrijednosti u trenutku izvođenja. *Bind* varijabla se koristi kao *placeholder* u WHERE uvjetu u kombinaciji s *view criteriom* koja sadrži filtrirane informacije instance view objekta. Na taj se način mijenja WHERE uvjet u skladu s potrebama korisnika.

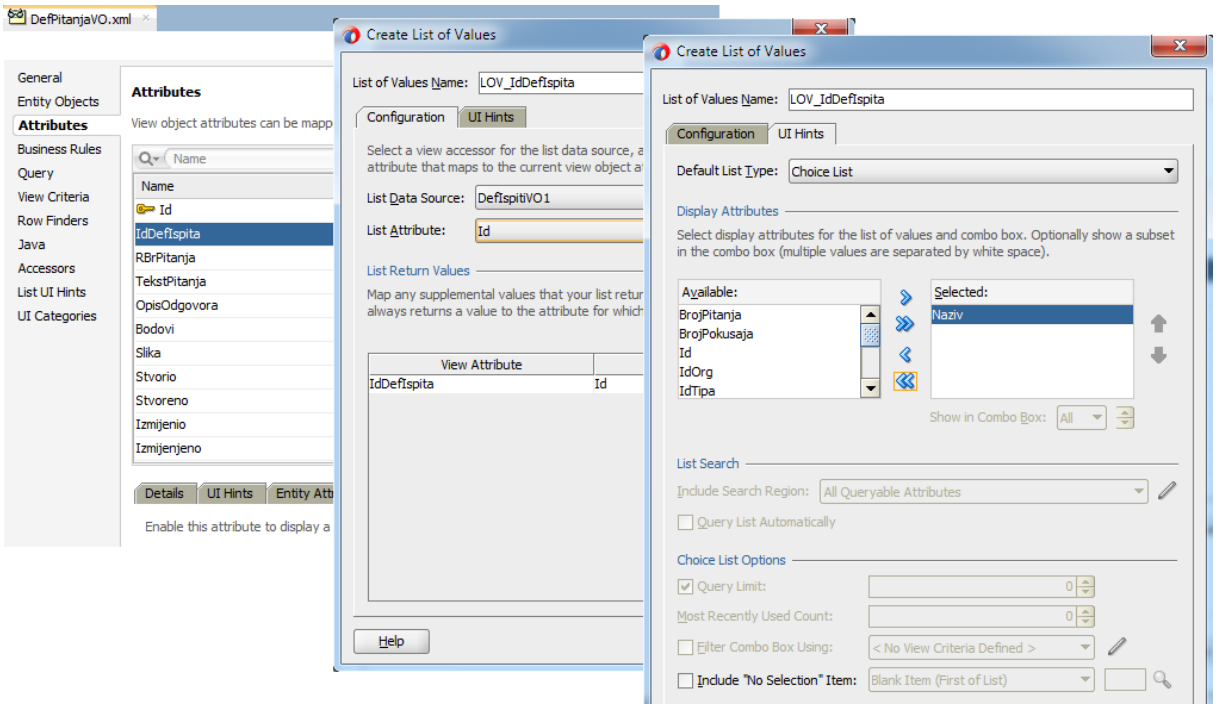
Na Slici 16. je prikazana *View Criteria* koja koristi dvije *bind* varijable `:vStr` tipa string te na temelju njih filtrira polagače ispita po njihovom imenu ili prezimenu u view objektu PolagaciIspitaEO.



Slika 16. View criteria i bind varijable [izvor: autor]

View criteria je korisna iz tog razloga što se ne mora raditi poseban view objekt s filtriranim podacima, nego se na već postojećem objektu može kreirati filter, koji se tada koristi zavisno o potrebi.

Kod view objekata potrebno je još spomenuti i liste vrijednosti (engl. *List of Values*, LOV). Njihova svrha je da krajnjem korisniku olakšaju pretragu prikazujući podatke na razumljiv način, kao na primjer da umjesto id-a organizacije korisnik vidi naziv organizacije i slično. Kako bi se definirale liste vrijednosti (LOV) treba najprije odlučiti gdje se nalazi izvor podataka koji se treba prikazivati u listi. U primjeru `OrganizacijaEO` – `KorisnikEO`, ako je potrebno prikazati atribut `Naziv` iz tablice `OrganizacijaEO` u tablici `KorisnikEO`, tada je `OrganizacijaEO` izvor podataka. U primjeru na Slici 17. može se vidjeti kako na jednostavan način kreirati liste vrijednosti. Najprije se odabire atribut u `DefPitanjaEO` za koji je potrebna lista vrijednosti, u ovom slučaju je to `IdDefIspita`, nakon toga je potrebno odabrati izvor podataka koji se nalazi u `DefIspitiVO` te na kraju u samom izvoru podataka treba odabrati atribut koji se treba prikazivati.



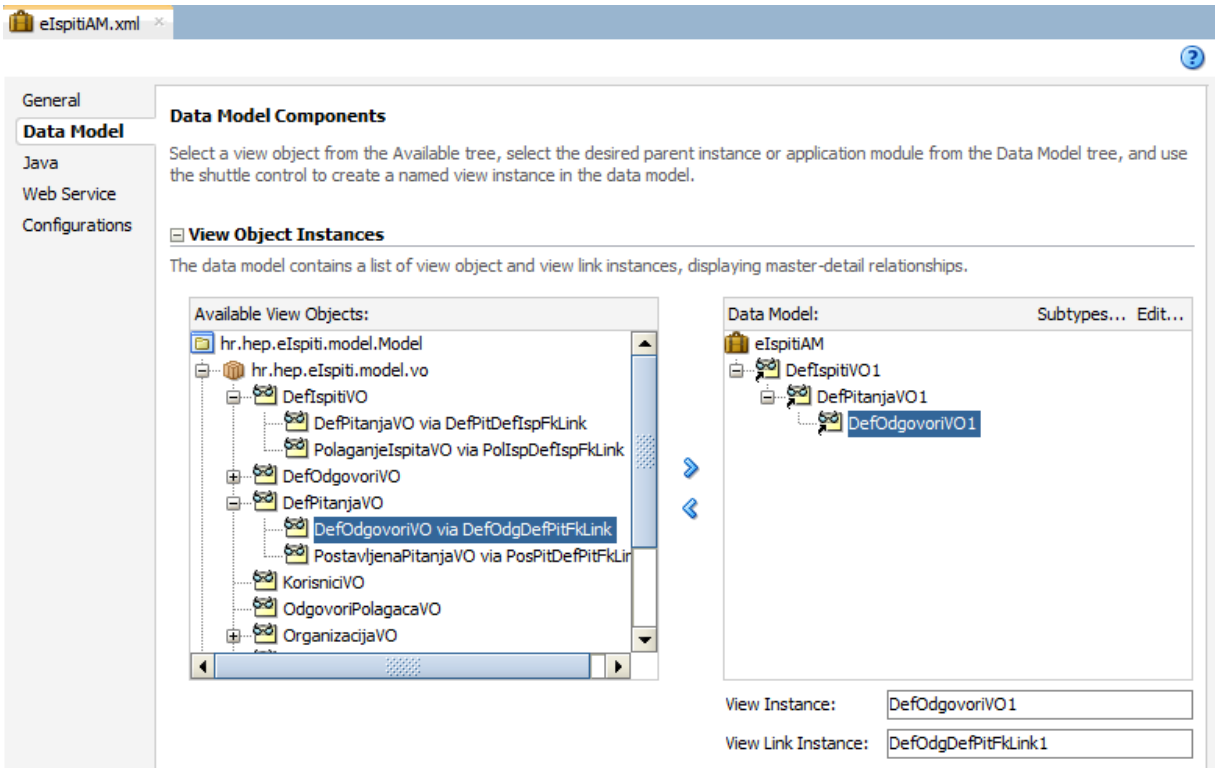
Slika 17. Lista vrijednosti `IdDefIspita` u `DefPitanjaVO` [izvor: autor]

6.2.3. Kreiranje i konfiguracija aplikacijskog modula

Na kraju, nakon što su kreirani svi view objekti koji će se koristiti u radu same aplikacije potrebno je kreirati aplikacijski modul kako bi se mogli testirati ulazi i izlazi te ostale validacije koje su napravljene na entity, a samim time i na view objektima.

Aplikacijski modul je transakcijska poslovna komponenta koja zapravo predstavlja određeni aplikacijski zadatak na način da pruža model podataka (engl. *Data Model*) povezujući instance view objekata s instancama view linkova.

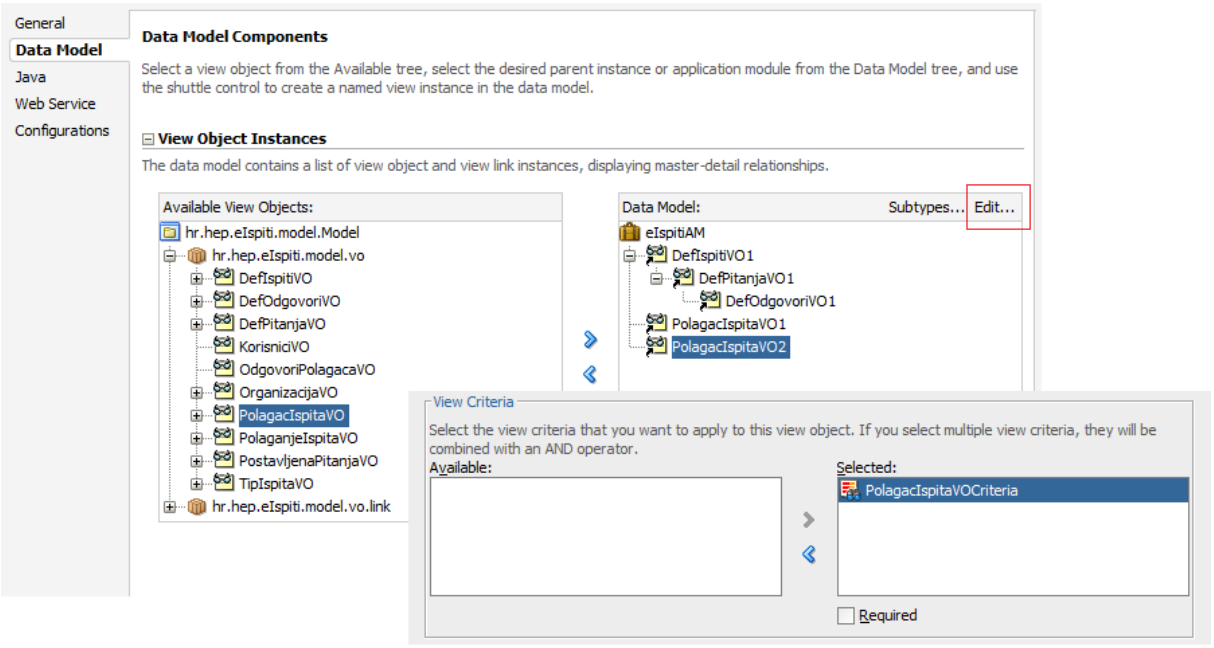
Na Slici 18. je prikazan aplikacijski modul `eIspitiAM` gdje se s lijeve strane nalaze prethodno kreirani view objekati kao i view linkovi s kojima se izgrađuje takozvani *Data Model*. Osim toga može se vidjeti kako je na primjer `DefPitanjaVO` prikazan dva puta, jednom zasebno, a jednom kao *detail* view objekta `DefIspitiVO`.



Slika 18. Aplikacijski modul `eIspitiAM` [izvor: autor]

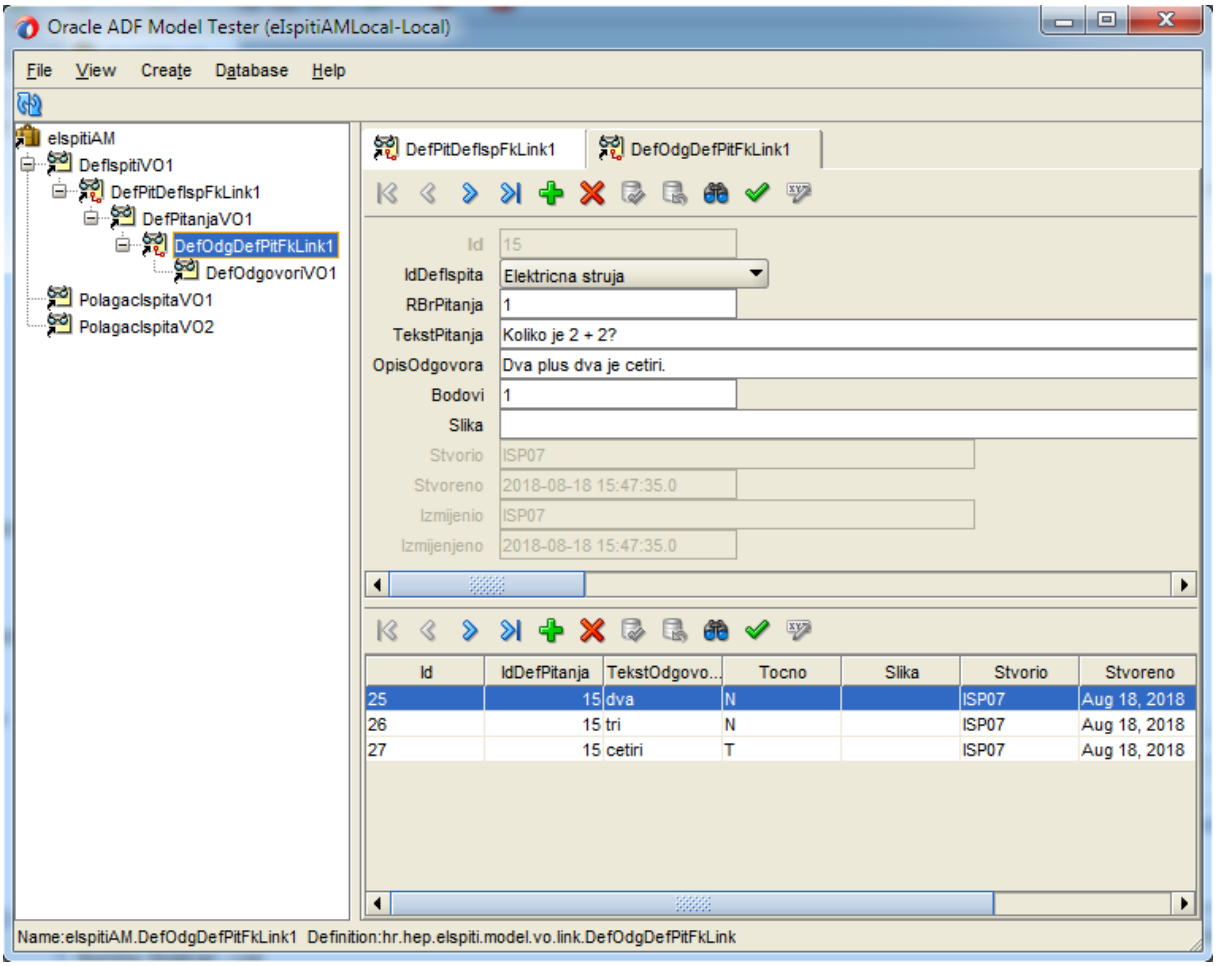
S desne strane *Data Model*-a se nalaze instance view objekata koji može sadržavati onoliko razina hijerarhije koliko ima relacija između view objekata. Na primjer na desnoj strani Slike 18. je prikazan *master-detail-detail* model s instancom `DefIspitiVO1` koja je master od `DefPitanjaVO1` (preko `DefPitDefIspFkLink`), a koja je pak master od `DefOdgovoriVO1` (preko `DefOdgDefPitFkLink`). Instanca view objekta `DefOdgovoriVO1` je posljednja budući da više nema instanci, odnosno *detail*-a, ispod sebe.

Ranije je spomenuto kako su view criterie dobre iz tog razloga što isti view objekt možemo iskoristiti na više načina. Na Slici 19. se može vidjeti jedan takav primjer gdje se koriste dvije instance istog view objekta, `PolagacIspitaVO1` koja ne sadrži nikakve filtere, te `PolagacIspitaVO2` koja sadrži View Criteriu i u kojoj radi toga je moguće pretraživanje polagača po imenu i prezimenu.



Slika 19. Dodavanje View Criterie na instancu view objekta [izvor: autor]

Na kraju je još potrebno prikazati i ADF *Model Tester* koji se nalazi na Slici 20. U njemu se može istestirati prethodno kreirani aplikacijski modul koji sadrži ranije opisani *master-detail-detail* model, točnije može se vidjeti *master-detail* hijerarhija definiranih pitanja i odgovora koji pripadaju tom pitanju, zatim prethodno kreirana lista vrijednost za `IdDefIspita` te koja polja se mogu ažurirati, a koja ne. U ADF *Model Testeru* je također moguć unos novog retka, brisanje ili ažuriranje te provjera implementiranih poslovnih pravila.



Slika 20. Oracle ADF Model Tester [izvor: autor]

Međutim, sva poslovna pravila se nisu mogla rješiti deklarativno, kao što je to slučaj kod procedura `generiraj_pitanja` i `provjeri_odgovor` koje su kreirane u bazi podataka i za koje bi u daljnjem razvoju aplikacije trebalo napisati Java metode te pozvati procedure koje bi zatim mogle izvršiti određena poslovna pravila.

7. Zaključak

Ovaj završni rad baziran je na konkretnom primjeru odnosno na stvarnoj problematici i potrebi jedne kompanije za e-ispitom iz zaštite na radu, koja uz potrebu da zaštiti i osigura uvjete za vanjske suradnike i posjetioce, ima potrebu za smanjenjem birokracije, ali i troškova kao što je to navedeno u korisničkim zahtjevima.

Sam model baze podataka je osmišljen na način da usvoji sva poslovna pravila te osigura sigurnost podataka bez redundancije kako bi se jedan ispit, u ovom slučaju zaštite na radu, odvio što transparentnije i unutar svih pravila i ograničenja, kako za polagača, tako i za ispitivača, od samog osmišljavanja ispita, dodavanja pitanja i označavanja odgovora.

Oracle ADF se zapravo može podijeliti na dva dijela, odnosno *frontend* dio koji se temelji na takozvanoj MVC arhitekturi te *backend* dio, odnosno *Business Service* sloj koji omogućuje implementaciju objekata i poslovnih pravila iz baze podataka kroz ADF *Business Components framework* što je ujedno i fokus ovoga rada.

Budući da se ADF fokusira na vizualni i deklarativni pristup razvoja samim time pojednostavljuje razvoj Java aplikacija minimizirajući potrebu za pisanjem koda te na taj način omogućuje razvoj aplikacije kojom je vrlo jednostavno manipulirati te ju prilagođavati i gdje se više pažnje može posvetiti potrebama krajnjih korisnika, a ne samo na konfiguraciju.

Dobar model baze ne jamči siguran i efikasan rad. Za postizanje optimalnog rješenja baze potrebno je ispitati funkcionalnost svakoga elementa baze zasebno, a zatim i funkcionalnost svega u cjelini. U tome je koristan ADF Model Tester u kojem se već u ranoj fazi razvoja aplikacije, odnosno čim su mapirani objekti iz baze podataka, može istestirati funkcionalnost baze podataka i to bez potrebe korisničkog sučelja. Jedna od prednosti je što se može koristiti i u kasnijim fazama razvoja aplikacije.

Ovaj projekt bi trebao olakšati i automatizirati cijeli proces osposobljavanja radnika za ulazak u postrojenje, ali i za ostala mjesta rada gdje je to potrebno te se u budućnosti može i nadograditi s novim funkcionalnostima za kojima ovo poduzeće može imati potrebu.

Literatura

ELMASRI, R. i NAVATHE, S. B. (2011) *Fundamentals of Database Systems*. 6th edition. Pearson.

HERNANDEZ, M. J. (2013) *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design*. 3rd edition. Addison-Wesley Professional.

HARRINGTON, J. L. (2016) *Relational Database Design and Implementation*. 4th edition. Morgan Kaufmann.

KRISHNAN, V. (2013) *Oracle ADF 11gR2 Development Beginner's Guide*. Packt Publishing.

LANS, R. F. Van der (2006) *Introduction to SQL: Mastering the Relational Database Language*. 4th edition. Addison-Wesley Professional.

MANGER, R. (2012) *Baze podataka*. Zagreb: Element.

PURUSHOTHAMAN, J. (2012) *Oracle ADF Real World Developer's Guide*. Packt Publishing.

RONALD, G. (2011) *Quick Start Guide to Oracle Fusion Development: Oracle JDeveloper and Oracle ADF*. 1st edition. McGraw-Hill Education.

VESTERLI, S. E. (2014) *Oracle ADF Enterprise Application Development – Made Simple*. 2nd edition. Packt Publishing.

Popis slika

Popis tablica

Sažetak

U radu je prikazan model baze podataka „e-ispiti“ za potrebe TE Plomin kojim bi se olakšao proces izvođenja ispita iz zaštite na radu i koji se opisuje kroz tri faze oblikovanja, konceptualnog, logičkog i fizičkog. Prije same izrade konceptualnog modela baze podataka potrebno je utvrditi korisničke zahtjeve, a zatim u konceptualnom dijelu na temelju njihove analize utvrditi entitete, veze i njihove atribute te izgraditi E-R model. Na taj način se dolazi do relacijske faze oblikovanja u kojoj se svaki tip entiteta iz E-R modela prikazuje jednom relacijom dok njegovi atributi postaju atributi relacije. Također su definirana i poslovna pravila koja omogućuju funkcionalnost baze podataka, a dio programske logike se rješava u samoj bazi kako bi se smanjila opterećenost aplikacije. Nakon toga je opisana arhitektura ADF-a te je u sloju poslovnog servisa, ADF *Business Components*, izrađen model u kojem su na deklarativan način implementirana poslovna pravila.

Ključne riječi: E-ispiti, model baze podataka, Chenov dijagram, relacijski dijagram, poslovna pravila, Oracle ADF Business Components

Summary

The Thesis presents an „e-tests“ database model for the purpose of Plomin Power Plant in order to facilitate the examination process concerning security at work, and which is described throuh three framing phases, the conceptual, the logical and the phisycal. Before drafting of the conceptual database model user's requirements should be established, and afterwards in the conceptual part on the basis of theirs analysis their entities, connections and attributes should be determined as well as developed the E-R model. This leads to a relational phase of formation in which every type of entity from E-R model is presented by one relation whilst its attributes became relation's attributes. In addition business rules that ensure the data base functionality are also defined, and part of the programming logics is solved in the sole base in order to minimize the application burden. Subsequently the ADF architecture is described being in the class of business service, ADF Business Componenets, a developped model in which the business rules are implemented in a declarative way.

Key words: E-tests, database model, Chen's diagram, relational diagram, business rules, Oracle ADF Business Components.