



FAKULTET ZA POSLOVNU INFORMATIKU

RAZVOJ I KOMPARATIVNA ANALIZA KLASIČNE DESKTOP APLIKACIJE I VIŠESLOJNE KLIJENT/SERVER ARHITEKTURE

- Diplomski rad -

Mentor:
Prof. dr Mladen Veinović

Student:
Goran Nešić

Br. indeksa:
III-2/2005



FAKULTET ZA POSLOVNU INFORMATIKU

UNIVERZITET SINGIDUNUM
FAKULTET ZA POSLOVNU INFORMATIKU
Beograd, Danijelova 32

Broj: _____/2007

Kandidat: Goran Nešić

Broj indeksa: III-2/2005

Smer: Projektovanje i programiranje

Tema: RAZVOJ I KOMPARATIVNA ANALIZA KLASIČNE DESKTOP APLIKACIJE I
VIŠESLOJNE KLIJENT/SERVER ARHITEKTURE

Zadatak: Realizovati bazu podataka za vođenje evidencije o sredstvima za rad, opremi i potrošnom materijalu u jednoj ustanovi ili firmi. Baza treba da omogući vođenje precizne i ažurne evidencije o tome koja su sredstva, oprema, i druge vrste imovine izdati na korišćenje svakom od zaposlenih, zatim vođenje evidencije o potrošnom materijalu u smislu količina koje su izdate i preostalih količina na lageru, kao i štampanje odgovarajućih vrsta dokumenata i izveštaja. Potrebno je razviti i aplikaciju koja korisnicima omogućava efikasan i intuitivan rad sa bazom, i to u klasičnoj, desktop varijanti i u formi višeslojne klijent/server aplikacije. Komparativno analizirati ove dve vrste aplikacija.

MENTOR

Datum odobrenja teme:
05.02.2007.
Beograd

Prof. dr Mladen Veinović

DEKAN

Prof. dr Milan Milosavljević

Sažetak

Faze u razvoju baze podataka su:

1. Planiranje
2. Definicija sistema
3. Prikupljanje i analiza zahteva
4. Dizajniranje baze
5. Izbor DBMS platforme
6. Dizajniranje aplikacije
7. Implementacija
8. Punjenje baze podacima
9. Testiranje
10. Održavanje

Da bi se obezbedio precizan opis prirode podataka i načina na koji se oni koriste, potrebno je proizvesti jasan model koji nije striktno tehničke prirode. Jedan od najčešće korišćenih modela u praksi je *Entity-Relationship* (ER) model. Osnovne koncepte ER modela čine *entiteti*, *veze* i *atributi*.

Glavni cilj u procesu razvoja baze podataka je da se kreira sistem koji verno reprezentuje podatke, njihove veze i odnose, kao i ograničenja. Da bi se postigao ovaj cilj, moraju se pravilno uočiti odgovarajuće tabele, a metoda koja se za to koristi naziva se *normalizacija*. Normalizacija omogućava projektantu baze da izvrši optimalno grupisanje atributa po tabelama.

U metodologiji razvoja baze podataka definisane su tri osnovne faze, i to su:

1. Konceptualni dizajn
2. Logički dizajn
3. Fizički dizajn

Baza podataka **MAGACIN** predstavlja efikasno softversko rešenje za vođenje evidencije o sredstvima za rad, opremi, potrošnom materijalu i još nekim vrstama imovine jedne ustanove ili firme. Baza je originalno razvijena za potrebe ustanove u kojoj se oprema i imovina izdaju zaposlenima na korišćenje, a oni ih upotrebljavaju u svakodnevnom radu, kao osnovno ili pomoćno sredstvo.

Klijent/server arhitektura je nastala iz potrebe za prevazilaženjem ograničenja kao što su slaba ili nikakva podrška za grafičko korisničko okruženje, limitirani broj korisnika koje je sistem u stanju da podrži i otežan pristup fizički razmeštenim izvorima podataka.

Višeslojna klijent/server arhitektura predstavlja praktičnu realizaciju klijent/server arhitekture koja je tokom vremena evoluirala u skladu sa korisničkim zahtevima i kao rezultat težnje ka boljim performansama sistema. Višeslojne klijent/server arhitekture se najčešće realizuju kao *2-slojne*, *3-slojne* i *n-slojne*.

I klasične desktop aplikacije i moderne višeslojne klijent/server arhitekture imaju svoje prednosti i mane. U poslovnim okruženjima u kojima se koriste, za obe vrste aplikacija postoje poslovi za koje je primena jedne ili druge vrste aplikacija jedino logično rešenje, pa ih u tom kontekstu ne treba posmatrati kao direktnu konkurenciju.

Abstract

The stages of the database application lifecycle are:

1. Database planning
2. System definition
3. Requirements collection and analysis
4. Database design
5. DBMS selection
6. Application design
7. Implementation
8. Data loading
9. Testing
10. Operational maintenance

To ensure a precise understanding of the nature of the data and how it is used by the enterprise, it is necessary to create a non-technical model free of ambiguities. One of the most commonly used models is the *Entity-Relationship* (ER) model. Basic ER concepts are *entities*, *relationships* and *attributes*.

In a database design process, the main objective is to create an accurate representation of the data, its relationships, and constraints. To achieve this objective, a suitable set of relations must be identified. *Normalization* is a technique used for identification such relations. Normalization enables the database designer to identify optimal grouping of attributes for each relation in the schema.

There are three main phases in methodology of database design:

1. Conceptual database design
2. Logical database design
3. Physical database design

MAGACIN database is an efficient software solution for keeping track of equipment and other kinds of inventory in an institution or enterprise. Database was originally developed for an institution in which employees are given different kinds of equipment and inventory to use them in their daily activities.

As a result of the limitations like bad or no GUI support, limited number of supported users, and complicated access to data sources from geographically dispersed sites, client/server architecture emerged.

Multitier client/server architecture represents a practical implementation of the client/server architecture that has evolved according to user requirements and as a result of tendency towards constant improvement of system performances. Multitier client/server architectures are usually implemented as *2-tier*, *3-tier*, and *n-tier*.

Both traditional desktop applications and modern multitier client/server architectures have their advantages and weaknesses. They are frequently used in business environments, where is possible to identify jobs perfectly suitable for both types of applications, so in that sense they are not to considered as direct opponents.

SADRŽAJ

Uvod.....	1
1. Teorija baza podataka.....	2
FAZE U RAZVOJU BAZE PODATAKA	3
Planiranje.....	3
Definicija sistema.....	3
Prikupljanje i analiza zahteva.....	4
Proučavanje dokumentacije.....	4
Intervjuisanje.....	4
Posmatranje rada u okruženju za čije potrebe se razvija baza.....	4
Istraživanje.....	4
Upitnici	5
Dizajniranje baze	5
Faze u dizajniranju baze podataka.....	5
Izbor DBMS platforme	5
Dizajniranje aplikacije.....	5
Transakcije	6
Korisnički interfejs	6
Implementacija	7
Punjenje baze podacima	7
Testiranje.....	7
Održavanje	7
ENTITY-RELATIONSHIP MODEL.....	8
Entiteti	8
Veze	8
Atributi.....	9
Ključevi	9
Atributi veza.....	10
Strukturna ograničenja	10
Jedan-prema-jedan (1:1) veze.....	11
Jedan-prema-više (1:*) veze	11
Više-prema-više (*:*) veze	12
Kardinalnost i participacija (učesće).....	12
NORMALIZACIJA	13
Redundantnost (ponavljanje) podataka i anomalije ažuriranja	13
Anomalije unosa podataka	13
Anomalije brisanja podataka.....	14
Anomalije promene podataka	14
Funkcionalna zavisnost	14
Prva normalna forma (1NF).....	14
Druga normalna forma (2NF).....	15
Potpuna funkcionalna zavisnost	15
Definicija druge normalne forme	15
Treća normalna forma (3NF)	15
Tranzitivna zavisnost.....	15
Definicija treće normalne forme	16
Boyce-Codd normalna forma (BCNF).....	16
Četvrta normalna forma (4NF)	16
Zavisnost višestruke vrednosti	17
Definicija četvrte normalne forme	17
Peta normalna forma (5NF).....	18
Postojanost spajanja (lossless-join)	18
Definicija pete normalne forme.....	18
METODOLOGIJA DIZAJNIRANJA BAZE PODATAKA.....	19
Konceptualni dizajn.....	19
Logički dizajn	19
Fizički dizajn	20
2. Baza podataka Magacin.....	21
I faza: PLANIRANJE	21

II faza: DEFINICIJA SISTEMA	21
III faza: PRIKUPLJANJE I ANALIZA ZAHTEVA	21
Studija	22
IV faza: DIZAJNIRANJE BAZE	23
Entiteti	23
Primarni ključevi	24
Veze	25
V faza: IZBOR DBMS PLATFORME	26
VI faza: DIZAJNIRANJE APLIKACIJE	26
VII faza: IMPLEMENTACIJA	27
VIII faza: PUNJENJE BAZE PODACIMA	27
IX faza: TESTIRANJE	27
X faza: ODRŽAVANJE	27
3. Klijent/Server arhitektura	28
4. Višeslojna klijent/server arhitektura	30
2-slojna arhitektura	30
3-slojna arhitektura	32
N-slojna arhitektura	35
Sloj podataka	35
Sloj poslovnih pravila (business rules)	36
Sloj toka podataka (workflow layer)	37
Prezentacioni sloj	37
5. Komparativna analiza klasične desktop aplikacije i višeslojne klijent/server arhitekture	38
Bezbednost i integritet podataka	38
Stepen složenosti razvoja	39
Raspoloživost	39
Jasnoća/preglednost	40
Upotrebljivost	40
Brzina reagovanja	40
Zaključak	42
Literatura	43

Uvod

U poslednjih desetak godina svedoci smo rapidnog razvoja informacionih tehnologija koje su postale neizbežne u svim sferama poslovanja i života. Kao jedna od najznačajnijih IT oblasti, aplikativno programiranje je takođe pretrpelo mnoge promene i doživelo ekspanziju u mnogo različitih pravaca.

U ne tako dalekoj prošlosti, aplikacijom se smatrao program koji se izvršava na pojedinačnom desktop računaru, potpuno izolovano od ostatka sveta. Aplikaciji su na raspolaganju stajali samo resursi računara na kome se izvršava, dok je deljenje podataka i resursa bilo nezamislivo, a razmena podataka sa drugim korisnicima iz okruženja se obavljala na krajnje primitivan način, pomoću medijuma kao što je flopi disk.

Može se konstatovati da smo daleko odmakli od tog doba. Moderna aplikacija je tipično klijent/server tipa, svi neophodni podaci i resursi u mreži se dele između svih korisnika kojima su potrebni, poslovi se distribuiraju na posvećene servere koji raspolažu moćnim resursima koji su u stanju da odgovore na sve postavljene zahteve.

Ovaj rad ima za cilj da prikaže karakteristike nekih od najčešće zastupljenih arhitektura kod modernih klijent/server aplikacija i da izvrši komparativnu analizu tih arhitektura sa klasičnom desktop aplikacijom.

1.

Teorija baza podataka

U ovom poglavlju su obrađene odabrane teme iz teorije baza podataka. Materija je obrađena u potpuno opštem maniru, tako da se sve spomenute tehnike i metode mogu primeniti u procesu razvoja bilo koje baze podataka.

Poglavlje se sastoji iz četiri dela. Najpre se detaljno opisuju faze u razvoju baze podataka da bi se stekao opšti pregled celog procesa. Zatim se, u drugom, trećem i četvrtom delu respektivno, prikazuju izabrane oblasti teorije baza podataka, i to: *Entity-Relationship model*, *Normalizacija* i *Metodologija dizajniranja baze podataka*.

FAZE U RAZVOJU BAZE PODATAKA

Tabela 1 prikazuje hronološki redosled faza u razvoju baze podataka, uz kratak opis. Zatim sledi detaljan opis svake faze.

FAZA	OPIS
Planiranje	Planiranje najefikasnijeg načina za realizaciju svih faza u razvoju baze podataka
Definicija sistema	Određivanje opsega baze podataka, njenih korisnika i oblasti primene
Prikupljanje i analiza zahteva	Prikupljanje i analiza zahteva od korisnika; prikupljanje i analiza ostalih značajnih podataka iz okruženja u kome će se baza koristiti
Dizajniranje baze	Konceptualni, logički i fizički dizajn baze podataka
Izbor DBMS platforme	Izbor odgovarajuće DBMS platforme (DBMS = Database Management System)
Dizajniranje aplikacije	Dizajniranje korisničkog interfejsa i pratećih aplikacija
Implementacija	Fizička realizacija baze i pratećih aplikacija
Punjenje baze podacima	Punjenje baze podacima (ako je prethodno postojala starija verzija baze, vrši se i migracija na novu verziju, što ponekad zahteva i konverziju podataka)
Testiranje	Baza se testira u cilju otkrivanja i otklanjanja grešaka; proverava se da li su ispunjeni svi korisnički zahtevi
Održavanje	Baza je implementirana u celini; vrši se kontinualno kontrolisanje i održavanje; ako je potrebno, implementiraju se novi korisnički zahtevi

Tabela 1 – Faze u razvoju baze podataka

Planiranje

Najvažnija aktivnost tokom faze planiranja je jasno definisanje ciljeva koje baza podataka treba da ispuni. Ovo pomaže da se pojasni svrha projekta i da se sačini jasan plan za efikasnu realizaciju. Svaki od definisanih ciljeva identifikuje određeni zadatak koji baza mora da ispuni. Pretpostavlja se da će, ako baza ispunjava sve planirane zadatke, i svi postavljeni ciljevi biti ispunjeni. U ovoj fazi se mogu navesti i neki dodatni podaci, kao na primer tačna specifikacija poslova koji treba da se urade, potrebni resursi, novčana sredstva potrebna za realizaciju, itd.

Planiranje treba da obuhvati i razvoj standarda koji određuju kako će se sakupljati podaci, format podataka, potrebnu dokumentaciju, kao i metodologiju dizajniranja i implementacije.

Definicija sistema

Pre nego što se pristupi dizajniranju, neophodno je najpre identifikovati opseg i granice baze podataka, kao i način na koji će se ona uklopiti u informacioni sistem posmatranog okruženja, ako on već postoji. Važno je da se unutar opsega i granica baze nađu ne samo trenutni korisnici i postojeće oblasti primene, već da se predvidi i mogućnost prihvatanja korisnika i oblasti primene koji će se pojaviti u budućnosti.

U ovoj fazi se definišu glavni korisnici baze u smislu vrste posla koga obavljaju ili funkcije na kojoj se nalaze (npr. menadžer ili supervizor), što ujedno određuje podatke koje će korisnik upotrebljavati i transakcije koje će obavljati nad podacima. Takođe se definišu i zahtevi iz perspektive pojedinih oblasti primene u posmatranom okruženju (npr. marketing, ljudski resursi). Ovo se najčešće realizuje upotrebom "korisničkih pregleda" (*user views*) koji se kreiraju za sve glavne korisnike i oblasti primene.

Prikupljanje i analiza zahteva

Ova faza podrazumeva prikupljanje i analizu informacija o okruženju za čije potrebe se baza razvija. Informacije se prikupljaju za sve glavne funkcije i oblasti primene. Na osnovu analize prikupljenih podataka kreira se dokument koji se naziva *specifikacija zahteva* i iz njega, zapravo, proizilazi funkcionalnost baze podataka.

Postoji nekoliko uobičajenih tehnika za prikupljanje informacija relevantnih za razvoj:

- Proučavanje dokumentacije
- Intervjuisanje
- Posmatranje rada u okruženju za čije potrebe se razvija baza
- Istraživanje
- Upitnici.

Proučavanje dokumentacije

Proučavanje dokumentacije može da pruži korisne informacije o tome kako i zašto se javila potreba za bazom, o problematičnom delu (sektoru, odeljenju) posmatranog okruženja ili o već postojećem DBMS sistemu. Dokumentacija koja se proučava može biti vrlo raznolika: interna dokumenta, elektronska pošta, razne vrste formulara i izveštaja, dokumentacija o postojećem sistemu, itd.

Intervjuisanje

Intervjuisanje je najčešće korišćena i najkorisnija tehnika za prikupljanje informacija. Najbolji rezultati se postižu upotrebom tipskih, strukturiranih intervju sa tačno određenim pitanjima koja se, u zavisnosti od situacije, dopunjuju još nekim pitanjima koja mogu da daju korisne informacije. Ipak, i ova tehnika ima mane jer zahteva mnogo vremena i u mnogome zavisi od komunikativnosti učesnika i od dobre volje zaposlenih da daju informacije.

Posmatranje rada u okruženju za čije potrebe se razvija baza

Ovo je jedna od najefikasnijih tehnika za shvatanje sistema zato što podrazumeva direktno učešće u radu ili posmatranje osoba u radu. Ova tehnika je posebno korisna u situaciji kada krajnji korisnici nisu u stanju da daju jasna objašnjenja o sistemu i/ili procesu rada zbog njihove kompleksnosti.

Istraživanje

Istraživanje može da pruži korisne podatke o tome da li je i na koji način neki problem pred kojim se našao razvojni tim već rešen. Takođe se može doći i do informacije o nekom postojećem softverskom alatu ili aplikaciji koji delimično ili u potpunosti rešavaju taj problem.

Upitnici

Metoda popunjavanja upitnika je pandan intervjuisanju. Primenjuje se u situacijama kada bi bilo potrebno intervjuisati ogroman auditorijum, a to zbog utroška vremena nije praktično. Kao i kod intervjuisanja, najbolje je koristiti tipske upitnike tačno određenog formata sa ponuđenim odgovorima i mogućnošću izbora jednog ili više od njih.

Dizajniranje baze

Postoje dva glavna pristupa dizajniranju baze podataka.

Po prvom, dizajniranje počinje na osnovnom nivou – nivou atributa (svojstva entiteta i veza) koji se, posle analiziranja, grupišu u tabele (koje predstavljaju tipove entiteta) i veze između entiteta. Proces normalizacije je primer za ovakav pristup koji je pogodan za manje baze sa relativno malim brojem atributa.

Drugi pristup je pogodniji za dizajniranje složenih baza i po njemu se najpre kreira model koji sadrži samo nekoliko globalnih entiteta i veza, da bi se zatim prešlo na sukcesivno identifikovanje entiteta, veza i njima pripadajućih atributa na nižim nivoima. Primer za ovakav pristup je upotreba ER (*Entity-Relationship*) modela koji započinje identifikacijom globalnih entiteta i njihovih međusobnih veza.

Faze u dizajniranju baze podataka

Razvoj baze podataka se sastoji iz tri faze:

- I faza – Konceptualni dizajn
- II faza – Logički dizajn
- III faza – Fizički dizajn

Detaljni opis svih faza dat je u poglavlju *Metodologija dizajniranja baze podataka*.

Izbor DBMS platforme

Ako DBMS platforma još uvek ne postoji, pravi trenutak za njen izbor je između kreiranja konceptualnog i logičkog modela. Zapravo, platforma se može izabrati u bilo kom trenutku pre kreiranja logičkog modela, pod uslovom da se raspolaže dovoljnom količinom informacija po pitanju sistemskih zahteva kao što su performanse, bezbednost i integritet. Takođe, treba uzeti u obzir i eventualna proširenja sistema u budućnosti, pa na osnovu svih spomenutih kriterijuma doneti odluku o izboru platforme.

Dizajniranje aplikacije

Dizajniranje aplikacije podrazumeva kreiranje korisničkog interfejsa i programa (aplikacija) koji će raditi sa bazom. Cilj ove faze je da obezbedi da svi korisnički zahtevi navedeni u specifikaciji zahteva budu implementirani u aplikaciji. To podrazumeva razvoj programa i kreiranje *transakcija*, tj. metoda (funkcija) koje pristupaju bazi. Aplikacija baze podataka mora da ima takozvani "user-friendly" korisnički interfejs. Dizajneri često zanemaruju razvoj

korisničkog interfejsa, ali treba napomenuti da je to jedan od najvažnijih delova sistema. Ako je lak za učenje, jednostavan za upotrebu i otporan na greške, korisnički interfejs će onome ko ga upotrebljava dati mogućnost da podatke koje reprezentuje iskoristi na pravi način.

Transakcije

Transakcija se definiše kao *akcija ili više sukcesivnih akcija koje vrši korisnik, a koje podrazumevaju pristup bazi podataka i/ili promenu podataka u njoj*. Transakcija predstavlja događaje iz realnog sveta, kao što su na primer prijem novog zaposlenog, izdavanje opreme, itd. Transakcija može da se sastoji iz nekoliko operacija, kao što je na primer transfer novca sa jednog računa na drugi. Transakcija prevodi bazu iz jednog konzistentnog stanja u drugo. DBMS mora da obezbedi konzistentnost čak i u slučaju greške. DBMS takođe mora da obezbedi da, po izvršenju transakcije, promene koje su učinjene ostanu trajno sačuvane. Ako transakcija iz bilo kog razloga ne može da se u potpunosti izvrši, DBMS mora da obezbedi da se učinjene promene ponište.

Kod kreiranja transakcija glavni zadatak je da se definišu i dokumentuju njihova najvažnija svojstva, kao što su:

- Podaci sa kojima će transakcija raditi
- Funkcionalne karakteristike transakcije
- Izlaz transakcije
- Očekivana frekventnost upotrebe.

Postoje tri glavna tipa transakcija:

1. **Transakcije za prezentaciju podataka**, koje pristupaju bazi i čitaju iz nje određene podatke koji se zatim prikazuju na ekranu ili u izveštajima.
2. **Transakcije za promenu podataka**, koje dodaju nove zapise u bazu, brišu nepotrebne zapise ili ažuriraju postojeće zapise.
3. **Mešovite transakcije**, koje služe i za prezentaciju i za promenu podataka. Primer za ovu vrstu transakcije bi bila situacija u kojoj se najpre vrši pretraga po nekom kriterijumu, zatim se podaci prikazuju korisniku koji ih onda ažurira i na kraju sačuva promene.

Korisnički interfejs

Prilikom dizajniranja korisničkog interfejsa (formi i izveštaja) korisno je imati na umu sledeće smernice:

- Dati smislen naziv koji nedvosmisleno identifikuje namenu forme ili izveštaja
- Ponuditi shvatljive instrukcije, zajedno sa sistemom pomoći, ako je potrebno
- Logički grupisati i rasporediti polja
- Kreirati vizuelno uredne forme i izveštaje
- Poljima dati lako razumljive nazive
- Omogućiti kretanje kroz formu/izveštaj ne samo mišem, već i tastaturom
- Skrenuti korisniku pažnju na svaku grešku koju učini
- U što je moguće većoj meri predvideti potencijalne greške korisnika i blagovremeno ih sprečiti da ih načine.

Implementacija

Implementacija baze se realizuje pomoću jezika DDL (**Data Definition Language**) iz same baze koja je izabrana kao razvojna platforma (npr. Oracle) ili pomoću vizuelnih alata za razvoj koji imaju sopstveno grafičko okruženje, a koji mogu biti deo same baze (npr. Access) ili posebna aplikacija (npr. Oracle Designer, Power Designer).

Aplikacije se takođe mogu razvijati alatima ugrađenim u bazu koja je izabrana kao razvojna platforma ili u nekim standardnim razvojnim okruženjima tipa Visual Basic, Visual C++, JBuilder. U slučaju upotrebe standardnih razvojnih okruženja radi se o primeni nekog od programskih jezika visokog nivoa (Basic, C++, Java) u koje se ugrađuju transakcije za rad sa bazom koje su implementirane pomoću jezika DML (**Data Manipulation Language**).

Punjenje baze podacima

U ovoj fazi nova baza se puni podacima, najbolje automatizovano pomoću nekog softverskog alata. U slučaju postojanja starije verzije baze, vrši se i migracija na novu verziju, što ponekad zahteva i konverziju podataka (ako je došlo do promene formata).

Testiranje

Pre konačnog puštanja u rad baza mora biti temeljito testirana. Testiranje se vrši sa dva cilja: da bi se otkrile eventualne greške i/ili nepravilnosti u radu, i da bi se demonstriralo da baza i prateće aplikacije, ako postoje, rade u skladu sa korisničkom specifikacijom zahteva. Testiranje treba obaviti u realnim uslovima, istim onim u kojima će se baza koristiti tokom eksploatacije, uz upotrebu pravih podataka. U testiranje treba uključiti korisnike koji će raditi sa bazom. Po mogućstvu, testiranje bi trebalo obaviti na posebnoj test platformi, a u nedostatku iste podatke treba zaštititi redovnim bekapovanjem.

Održavanje

Posle faza tokom kojih je implementirana i testirana, baza prelazi u fazu održavanja, koja podrazumeva sledeće aktivnosti:

- Nadgledanje performansi sistema. Ako performanse padnu ispod prihvatljivog nivoa, pristupa se podešavanju ili reorganizaciji baze.
- Održavanje i usavršavanje aplikacija. Novi korisnički zahtevi se ugrađuju u postojeće aplikacije.

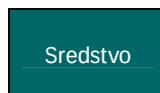
Veliki DBMS sistemi tipa Oracle ili SQL Server stavljaju korisniku na raspolaganje moćne i efikasne alate za nadgledanje i podešavanje performansi, kao na primer alat za analiziranje indeksa ili alat za analiziranje i optimizaciju SQL naredbi. Upotrebom ovakvih alata administrator ima mogućnost da bazu dovede u optimalno stanje i da je u takvom stanju održava tokom celog životnog veka.

ENTITY-RELATIONSHIP MODEL

Jedan od najvećih problema u procesu razvoja baze podataka je činjenica da projektanti, programeri i krajnji korisnici na potpuno različite načine shvataju podatke i načine njihove upotrebe, kao i procese iz posmatranog okruženja koje treba modelirati. Da bi se obezbedio precizan opis prirode podataka i načina na koji se oni koriste, potrebno je proizvesti jasan model koji nije striktno tehničke prirode. Jedan od najčešće korišćenih modela u praksi je *Entity-Relationship* (ER) model. Osnovne koncepte ER modela čine *entiteti*, *veze* i *atributi*.

Entiteti

Entiteti predstavljaju objekte iz realnog sveta. Entitet egzistira nezavisno i može predstavljati fizički, realni objekat (npr. *Sredstvo*) ili konceptualni, apstraktni objekat (npr. *Radno iskustvo*). Entitet se identifikuje nazivom i listom svojstava, a grafički se predstavlja kao pravougaonik u kome se ispisuje naziv entiteta, koji je najčešće imenica. U bazi **MAGACIN**, između ostalog, postoje entiteti kao što su *Sredstvo*, *Ulaz/Izlaz*, *Zaposleni*.



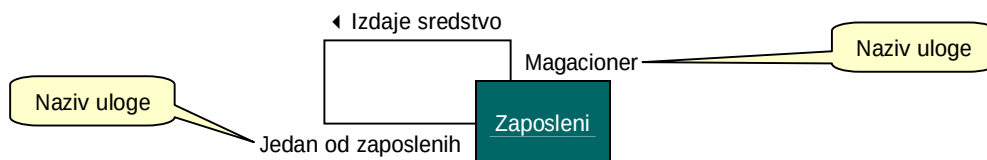
Veze

Veze označavaju da su određeni entiteti povezani, odnosno da su ti entiteti u određenom odnosu. Svakoj vezi se daje naziv koji opisuje njenu funkciju. Na primer, u bazi **MAGACIN** između entiteta *Ulaz/Izlaz-stavke* i *Sredstvo* postoji veza *Sadrži* koja označava da ulazno/izlazni dokument sadrži listu sredstava koja se unose u magazin (*Trebovanje*) ili izdaju nekome od zaposlenih na korišćenje (*Revers*). Veza se grafički predstavlja kao linija koja spaja entitete i uz koju se ispisuje naziv, koji je najčešće glagol. Treba napomenuti da naziv veze uglavnom ima samo jednosmerni smisao, pa je uobičajeno da se pored naziva veze iscrta i strelica koja označava pravilan smer, kao na slici.



Broj entiteta koji učestvuju u vezi se naziva *stepen veze*. Veza u kojoj učestvuju dva entiteta se naziva binarna, veza trećeg stepena (učestvuju tri entiteta) se naziva ternarna, itd.

Veza u kojoj jedan entitet učestvuje više puta u različitim ulogama naziva se *rekurzivna* ili *unarna* veza. Primer u bazi **MAGACIN** bila bi veza *Izdaje sredstvo*, koja stoji uz entitet *Zaposleni* i označavala vezu zaposlenih sa magacionerom, jer i on spada u kategoriju zaposlenih. Drugim rečima, entitet *Zaposleni* učestvuje dva puta u vezi *Izdaje sredstvo*: prvi put kao magacioner koji izdaje sredstva drugima, a drugi put kao jedan od zaposlenih kome takođe može da se izda sredstvo.



Atributi

Atributi predstavljaju određena svojstva entiteta koja ih opisuju. Atributi su na primer *Naziv sredstva*, *Fabrički broj*, *Ime i prezime*, *Broj telefona*, itd. Glavni deo podataka koji se čuvaju u bazi su upravo atributi.

Za svaki atribut može se definisati *domen* koji predstavlja set dozvoljenih vrednosti koje atribut može da primi. Na primer, polje *Kategorija sredstva* u tabeli *Sredstvo* baze **MAGACIN** ima tačno definisan set naziva kategorija koji mogu da se unesu u to polje. Set naziva kategorija se čuva u tabeli *Kategorija sredstva*.

Od svih vrsta atributa najvažniji su *ključevi*.

Ključevi

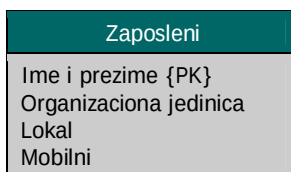
Ključ kandidat predstavlja minimalni set atributa čije vrednosti jedinstveno identifikuju entitet. Na primer, ključ kandidat za tabelu *Zaposleni* baze **MAGACIN** bio bi atribut *Ime i prezime*, jer on jedinstveno identifikuje svakog od zaposlenih. (Strogo uzevši atribut *JMBG* bi bio najbolji ključ kandidat, jer on jedinstveno identifikuje svakog građanina, ali s obzirom na to da u okruženju u kome se baza **MAGACIN** koristi ima samo oko 50 zaposlenih, mala je verovatnoća da će se pojaviti više osoba sa istim imenom i prezimenom). Ključ kandidat ne sme da ima vrednost *null*.

Primarni ključ je ključ kandidat koji je izabran da jedinstveno identifikuje entitet. Entitet može da ima više ključeva kandidata, pa se primarni ključ bira na osnovu kriterijuma kao što su dužina atributa, minimalni broj potrebnih atributa i procene da li će atribut i u budućnosti biti jedinstven. U prethodnom pasusu upoređeni su atributi *JMBG* i *Ime i prezime* i zaključeno je da, sa aspekta zahteva baze **MAGACIN**, atribut *Ime i prezime* zadovoljava potrebe jedinstvene identifikacije. Međutim, sa aspekta zahteva za dizajniranje baze koja sadrži podatke o svim građanima naše zemlje, atribut *Ime i prezime* ne bi bio dobar izbor, jer postoji mnogo građana sa istim imenom i prezimenom. Za takvu bazu jedini pravilan izbor za primarni ključ bio bi atribut *JMBG*. Ključ kandidat koji nije izabran za primarni ključ naziva se *alternativni ključ*.

Složeni ključ je ključ kandidat koji se sastoji od dva ili više atributa. U nekim slučajevima ključ entiteta se sastoji od nekoliko atributa čije su vrednosti jedinstvene ako se posmatraju zajedno, ali pojedinačne vrednosti tih atributa nisu jedinstvene. U tabeli *Ulaz/Izlaz-stavke* baze **MAGACIN** atributi *PKID_Ulaz/Izlaz* i *PKID_Sredstva* čine složeni ključ. Ovakvo rešenje je primenjeno zbog toga što je trebalo obezbediti da se jedno sredstvo može pojaviti samo jedanput u jednom dokumentu (*Revers*, *Trebovanje* ili *Povratnica*). Pošto će se atribut *PKID_Sredstva* sigurno pojaviti više puta u tabeli *Ulaz/Izlaz-stavke*, jedini način da se realizuje spomenuti zahtev je da se formira složeni ključ, pošto atributi *PKID_Ulaz/Izlaz* i *PKID_Sredstva* u kombinaciji formiraju jedinstvenu vrednost koja jednoznačno identifikuje

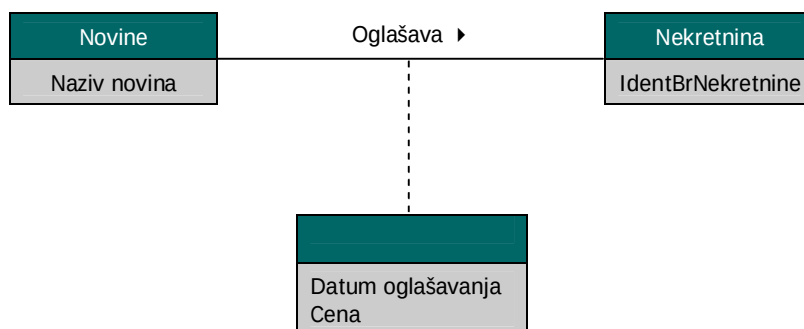
svaki zapis u tabeli, pod uslovom da se obezbedi da atribut *PKID_Ulaz/Izlaz* bude jedinstven za svaki dokument koga reprezentuje.

Atributi se grafički predstavljaju tako što se pravougaonik koji predstavlja entitet horizontalno podeli na dva dela, pa se u gornjem delu ispisuje naziv entiteta, a u donjem lista atributa. Prvi u listi se nalazi primarni ključ, ako postoji. Često se pored primarnog ključa ispisuje {PK} ili naziv primarnog ključa počinje sa PK.



Atributi veza

Veze takođe mogu da imaju attribute. Na primer, u bazi podataka neke agencije za prodaju i izdavanje nekretnina mogla bi da postoji veza *Oglašava* koja spaja entitete *Novine* i *Nekretnina*.



Da bi se dokumentovalo da je za nekretninu dat oglas, podaci datum oglašavanja i cena se pridružuju vezi *Oglašava*, u formi atributa *Datum oglašavanja* i *Cena*. Atributi pridruženi vezi se grafički prikazuju istim simbolom kao za entitet, s tom razlikom što se pravougaonik koji sadrži attribute veze sa samom vezom povezuje iscrtavanjem isprekidane linije.

Prisustvo jednog ili više atributa dodeljenih vezi može da ukazuje na to da veza skriva neidentifikovani entitet. U spomenutom primeru, prisustvo atributa *Datum oglašavanja* i *Cena* u vezi *Oglašava* ukazuje na prisustvo entiteta *Oglas*.

Strukturna ograničenja

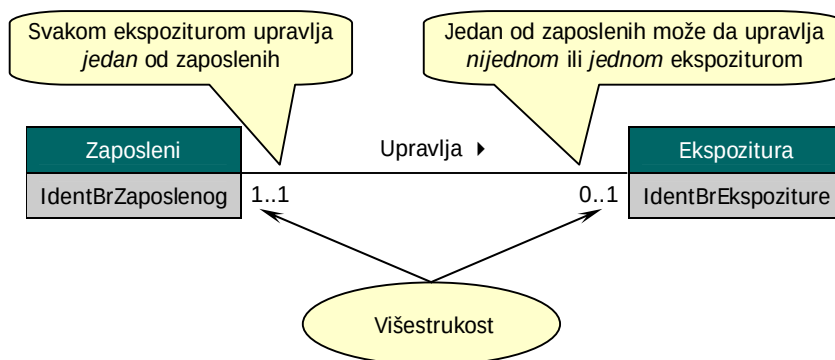
Entiteti koji učestvuju u vezi mogu imati izvesna ograničenja. Ograničenja treba da se reflektuju na veze tako da verno reprezentuju odnose iz realnog sveta, kao na primer da, u već spomenutoj u bazi podataka agencije za prodaju i izdavanje nekretnina, svaka nekretnina mora da ima vlasnika. Osnovni tip ograničenja se naziva *višestrukost*. Višestrukost ograničava način na koji se entiteti odnose jedan prema drugome i predstavlja poslovna pravila (poslovnu politiku) okruženja koje se modelira. Već je spomenuto da je najčešći tip veze binarni. Binarne veze mogu biti: jedan-prema-jedan (1:1), jedan-prema-više (1:*) ili

više-prema-više (*:*)). Sva tri tipa veza biće analizirana pomoću sledećih ograničenja (poslovnih pravila) koja važe u agenciji za prodaju i izdavanje nekretnina:

- Jedan od zaposlenih upravlja ekspoziturom (1:1)
- Jedan od zaposlenih nadgleda nekretnine (1:*)
- Novine oglašavaju nekretnine (*:*)).

Jedan-prema-jedan (1:1) veze

Razmotrimo vezu *Upravlja* koja spaja entitete *Zaposleni* i *Ekspozitura*.

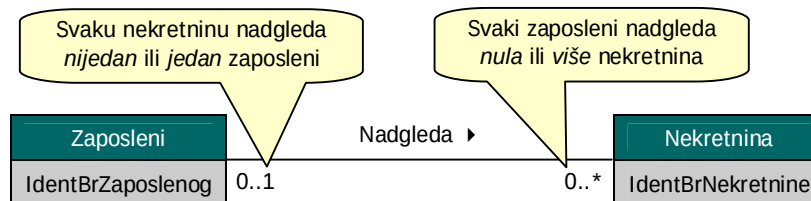


Određivanje višestrukosti se uglavnom vrši proučavanjem veza i odnosa između uzetih uzoraka podataka koji će biti u opticaju u posmatranoj sredini. Analizom navedenog primera može se zaključiti da jedan od zaposlenih može da upravlja *nijednom* ili *jednom* ekspoziturom. Pošto postoji maksimalno jedna ekspozitura za svakog zaposlenog koji učestvuje u ovoj vezi i maksimalno jedan zaposleni za svaku ekspozituru, ovaj tip veze se naziva *jedan-prema-jedan* (1:1).

Da bi se grafički prikazalo da jedan od zaposlenih može sa upravlja *nijednom* ili *jednom* ekspoziturom, pored entiteta *Ekspozitura* se ispisuje 0..1. Da bi se grafički prikazalo da ekspoziturom uvek upravlja *jedan* zaposleni, pored entiteta *Zaposleni* se ispisuje 1..1.

Jedan-prema-više (1:*) veze

Razmotrimo vezu *Nadgleda* koja spaja entitete *Zaposleni* i *Nekretnina*.

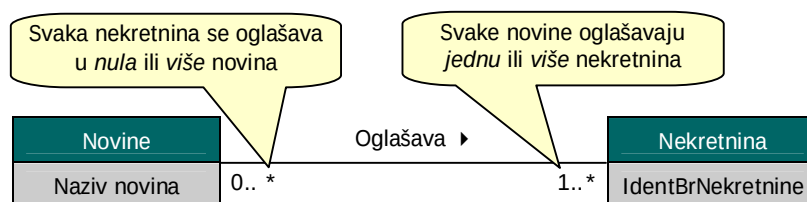


Zaposleni može da nadgleda *nula* ili *više* nekretnina, a nekretninu može da nadgleda *nijedan* ili *jedan* zaposleni. Dakle, za zaposlene koji učestvuju u ovoj vezi postoji više nekretnina, a za nekretnine koje učestvuju u ovoj vezi postoji maksimalno *jedan* zaposleni. Ovakav tip veze se naziva *jedan-prema-više* (1:*)).

Da bi se grafički prikazalo da zaposleni može da nadgleda *nula* ili *više* nekretnina, pored entiteta *Nekretnina* se ispisuje 0..*. Da bi se grafički prikazalo da svaku nekretninu nadgleda *nijedan* ili *jedan* zaposleni, pored entiteta *Zaposleni* se ispisuje 0..1. Ako se zna tačna vrednost za višestrukost, umesto * se može napisati taj broj, na primer (0..100).

Više-prema-više (*:*) veze

Razmotrimo vezu *Oglašava* koja spaja entitete *Novine* i *Nekretnina*.



Jedne novine oglašavaju *jednu* ili *više* nekretnina, a *jedna* nekretnina se oglašava u *jednim* ili *nijednim* novinama. Dakle, za novine postoji *više* nekretnina, a za svaku nekretninu koja učestvuje u ovoj vezi postoji *više* novina. Ovakav tip veze se naziva *više-prema-više* (*:*)).

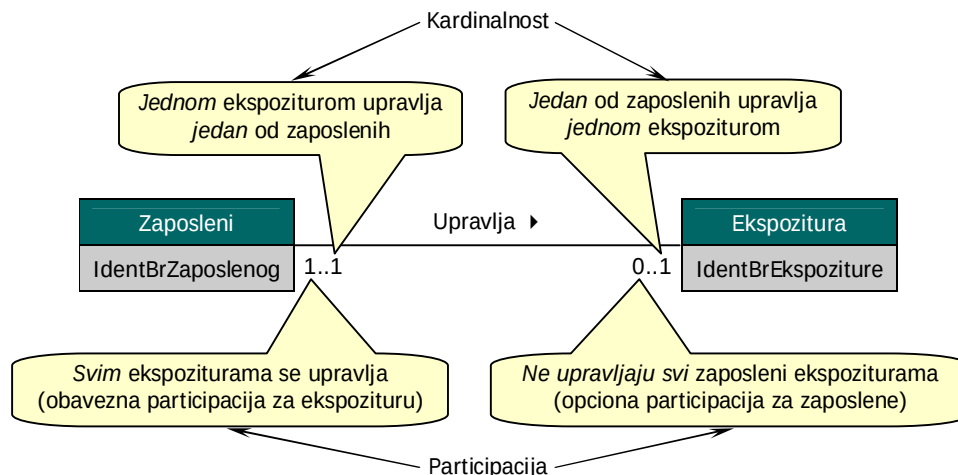
Da bi se grafički prikazalo da svake novine mogu da oglašavaju *jednu* ili *više* nekretnina, pored entiteta *Nekretnina* se ispisuje 1.. *. Da bi se grafički prikazalo da svaka nekretnina može da se oglašava u *nula* ili *više* novina, pored entiteta *Novine* se ispisuje 0.. *.

Kardinalnost i participacija (učešće)

Višestrukost se zapravo sastoji iz dva zasebna ograničenja poznata kao *kardinalnost* i *participacija* (učešće).

Kardinalnost određuje maksimalni broj mogućih primeraka entiteta za entitet koji učestvuje u datoj vezi.

Participacija (učešće) određuje da li u vezi učestvuju svi primerci entiteta ili samo neki od njih. U slučaju da u vezi učestvuju svi primerci entiteta, radi se o *obaveznoj* participaciji, dok se u slučaju da u vezi učestvuju samo neki od primeraka entiteta radi o *opcionoj* participaciji.



NORMALIZACIJA

Glavni cilj u procesu razvoja baze podataka je da se kreira sistem koji verno reprezentuje podatke, njihove veze i odnose, kao i ograničenja. Da bi se postigao ovaj cilj, moraju se pravilno uočiti odgovarajuće tabele, a metoda koja se za to koristi naziva se *normalizacija*. Normalizacija je tehnika za kreiranje tabela sa odgovarajućim svojstvima, uzimajući u obzir zahteve okruženja za čije potrebe se projektuje baza. Normalizacija se često realizuje tako što se vrši serija testova nad tabelom da bi se utvrdilo da li ona zadovoljava zahteve određene normalne forme ili ne. Inicijalno su promovisane tri normalne forme, *prva (1NF)*, *druga (2NF)* i *treća (3NF) normalna forma*. Kasnije su R.Boyce i E.F.Codd promovisali strožu definiciju treće normalne forme koja se naziva *Boyce-Codd normalna forma (BCNF)*. Sve normalne forme se baziraju na funkcionalnim zavisnostima između atributa tabele. Promovisane su i više normalne forme koje prevazilaze BCNF, i to su *četvrta (4NF)* i *peta (5NF) normalna forma*. Ipak, ove forme se bave situacijama koje su vrlo retke.

Normalizacija omogućava projektantu baze da izvrši optimalno grupisanje atributa po tabelama. Proces normalizacije identifikuje tabele na osnovu primarnih ključeva ili ključeva kandidata i na osnovu funkcionalnih zavisnosti koje postoje između atributa. Normalizacija sadrži pravila koja se mogu upotrebiti za testiranje tabela, tako da baza može da se normalizuje do bilo kog stepena. Tabela koja ne ispunjava zahteve normalizacije mora se rastaviti na dve ili više tabela od kojih svaka pojedinačno ispunjava zahteve normalizacije. Važno je napomenuti da je za kreiranje tabela u relacionom modelu podataka kritična samo 1NF. Sve ostale forme su opcione. Ipak, da bi se izbegle anomalije ažuriranja (objašnjene u sledećem pasusu), preporučuje se normalizacija najmanje do 3NF.

Redundantnost (ponavljanje) podataka i anomalije ažuriranja

Jedan od najvažnijih ciljeva prilikom projektovanja relacione baze podataka je pravilno grupisanje atributa po tabelama, što rezultuje minimalnim ponavljanjem podataka i smanjenjem prostora potrebnog za skladištenje fajlova u kojima se čuvaju podaci. Ponavljanje podataka je pojava da se isti podaci koji se odnose na neki entitet nepotrebno pojavljuju u dve ili više tabela. U tabelama koje sadrže podatke koji se ponavljaju mogu da se jave problemi poznati kao *anomalije ažuriranja*. Anomalije ažuriranja mogu biti *anomalije unosa*, *anomalije brisanja* ili *anomalije promene podataka*.

Anomalije unosa podataka

Anomalije unosa se manifestuju na dva načina:

1. Prilikom unosa novih podataka koji se odnose na jedan entitet, u tabelu koja sadrži podatke koji se ponavljaju moraju se uneti i podaci koji se odnose na druge entitete čiji se podaci nalaze u istoj tabeli, što može dovesti do nekonzistentnosti ako se načini greška prilikom unosa.
2. Prilikom unosa novih podataka koji se odnose na jedan entitet, u tabelu koja sadrži podatke koji se ponavljaju se u nekim situacijama mora uneti vrednost *null* u polja za koja ne postoji vrednost. Ako se desi da je baš to polje primarni ključ za tu tabelu, unos takvog zapisa mora biti sprečen, jer bi inače narušio integritet entiteta.

Anomalije brisanja podataka

Brisanje poslednjeg zapisa koji se odnosi na jedan entitet iz tabele koja sadrži podatke koji se ponavljaju može dovesti do potpunog gubitka podataka o drugom entitetu čiji se podaci nalaze u istoj tabeli, u istom zapisu.

Anomalije promene podataka

Prilikom promene vrednosti atributa koji se odnosi na jedan entitet, u tabeli koja sadrži podatke koji se ponavljaju moraju se promeniti i svi zapisi koji sadrže taj promenjeni atribut, kao i podaci koji se odnose na druge entitete, a koji stoje u direktnoj vezi sa promenjenim atributom. Ako se ne izvrše sve ove promene, baza postaje nekonzistentna.

Funkcionalna zavisnost

Funkcionalna zavisnost opisuje odnose između atributa u tabeli i predstavlja jedan od glavnih koncepata vezanih za normalizaciju. Osnovni razlog zbog koga se pristupa definisanju funkcionalnih zavisnosti za tabelu je potreba određivanja *ograničenja za očuvanje integriteta* (*integrity constraints*). Verovatno najvažnije ograničenje za očuvanje integriteta je uočavanje ključeva kandidata, od kojih se jedan bira za primarni ključ tabele. U procesu njihovog definisanja, naročito je značajno pronaći one funkcionalne zavisnosti koje važe za sve moguće vrednosti atributa jedne tabele, jer one predstavljaju tipove ograničenja za očuvanje integriteta. Tipovi ograničenja za očuvanje integriteta definišu vrednosti koje tabela može legitimno da primi. Takođe je značajno ignorisati tzv. *trivijalne funkcionalne zavisnosti*, jer za njih važi da su *uvek zadovoljene*, pa stoga nisu od značaja.

Prva normalna forma (1NF)

Pre diskusije o 1NF, najpre treba definisati stanje pre 1NF. Tabela koja sadrži podatke koji se ponavljaju nalazi se u *nenormalizovanoj formi (NNF)* i naziva se *nenormalizovana tabela*.

Tabela je u 1NF ako presek svakog njenog reda i kolone sadrži jednu i samo jednu vrednost.

Da bi se nenormalizovana tabela transformisala u 1NF, moraju se identifikovati i ukloniti podaci koji se ponavljaju. Postoje dva uobičajena pristupa za uklanjanje podataka koji se ponavljaju iz nenormalizovanih tabela:

1. Po prvom pristupu, odgovarajući podaci se ubacuju u prazne kolone redova koji sadrže podatke koji se ponavljaju. Drugim rečima, tamo gde je to potrebno, prazna polja se popunjavaju dupliranim podacima koji se ne ponavljaju. Rezultujuća tabela sadrži atomične (pojedinačne) vrednosti u preseku svakog reda i kolone, pa je stoga u 1NF. Dakle, u ovom postupku se u tabelu *namerno unose podaci koji se ponavljaju*, a oni se tokom sledećih, viših faza procesa normalizacije uklanjaju iz tabele.
2. Po drugom pristupu, podaci koji se ponavljaju se, zajedno sa kopijom originalne vrednosti atributa ključa (ili originalne vrednosti više atributa ključeva), izmeštaju u posebnu tabelu. Za novu tabelu se definiše primarni ključ. Ponekad nenormalizovana tabela sadrži više od jedne grupe podataka koji se ponavljaju ili čak grupe unutar grupe. U takvim slučajevima postupak izmeštanja se ponavlja sve dok se ne ukloni i

poslednja grupa podataka koji se ponavljaju. Za grupu tabela se kaže da je u 1NF ako ne sadrže podatke koji se ponavljaju.

Oba pristupa su ispravna, mada drugi pristup direktno daje tabele koje su u najmanje 1NF. I prvi pristup daje tabele koje su u 1NF, ali koje se tek u kasnijim fazama normalizacije razlažu na iste one tabele koje nastaju kao rezultat primene drugog pristupa. Dakle, može se zaključiti da je drugi pristup direktniji i praktičniji.

Druga normalna forma (2NF)

Druga normalna forma se bazira na konceptu *potpune funkcionalne zavisnosti*, koja će najpre biti definisana.

Potpuna funkcionalna zavisnost

Funkcionalna zavisnost $A \rightarrow B$ (čita se *B je funkcionalno zavisno od A*) je *potpuna funkcionalna zavisnost* ako uklanjanje bilo kog atributa iz A rezultuje time što zavisnost prestaje da važi, pri čemu su A i B atributi tabele i A je *determinanta* od B. Funkcionalna zavisnost $A \rightarrow B$ je *delimično zavisna* ako postoje neki atributi koji se mogu ukloniti iz A, a zavisnost i dalje važi. Determinanta je *atribut ili grupa atributa od kojih je neki drugi atribut potpuno funkcionalno zavisan*.

Definicija druge normalne forme

Druga normalna forma se odnosi na tabele sa složenim ključevima, tj. tabele čiji se primarni ključevi sastoje iz dva ili više atributa. Tabela čiji primarni ključ sadrži samo jedan atribut je automatski u najmanje 2NF. U tabeli koja nije u 2NF mogu da se pojave anomalije ažuriranja.

Tabela je u 2NF ako je u 1NF i ako je svaki atribut koji nije primarni ključ potpuno funkcionalno zavisan od primarnog ključa.

Normalizacija tabele iz 1NF u 2NF se vrši uklanjanjem delimičnih zavisnosti, tj. funkcionalno zavisni atributi se izmeštaju u novu tabelu zajedno sa kopijom njihovih determinanti (ključeva).

Treća normalna forma (3NF)

Čak iako je u 2NF, u tabeli mogu da se pojave anomalije ažuriranja zbog *tranzitivnih zavisnosti*, koje se moraju ukloniti iz tabele postupkom normalizacije do 3NF.

Tranzitivna zavisnost

Tranzitivna zavisnost je stanje u kome su A, B i C atributi tabele takvi da, ako $A \rightarrow B$ (*B je funkcionalno zavisno od A*) i $B \rightarrow C$ (*C je funkcionalno zavisno od B*), onda je C *tranzitivno zavisno od A preko B* (pod uslovom da A nije funkcionalno zavisno od B ili C).

Definicija treće normalne forme

Tabela je u 3NF ako je u 1NF i u 2NF i ako u njoj ne postoje atributi ne-primarni-ključevi koji su tranzitivno zavisni od primarnog ključa.

Normalizacija tabela iz 2NF u 3NF se vrši uklanjanjem tranzitivnih zavisnosti, tj. tranzitivno zavisni atributi se izmeštaju u novu tabelu zajedno sa kopijom njihovih determinanti (ključeva).

Boyce-Codd normalna forma (BCNF)

Druga i treća normalna forma eliminišu delimične i tranzitivne zavisnosti za primarne ključeve, ali one ne razmatraju da li takve zavisnosti postoje i za ključeve kandidate, ako ih ima u tabeli. Dakle, i u 3NF mogu da postoje delimične i tranzitivne zavisnosti za ključeve kandidate, pa je stoga promovisana stroža normalna forma poznata kao *Boyce-Codd normalna forma (BCNF)*.

BCNF se bazira na funkcionalnim zavisnostima koje uzimaju u obzir sve ključeve kandidate u tabeli, a sadrži i još neka ograničenja u poređenju sa 3NF.

Tabela je u BCNF ako i samo ako je svaka determinanta ključ kandidat.

Da bi se odredilo da li je tabela u BCNF, potrebno je uočiti determinante i proveriti da li su sve one ključevi kandidati. Potsetićemo se da je determinanta *atribut ili grupa atributa od kojih je neki drugi atribut potpuno funkcionalno zavisan*.

Razlika između 3NF i BCNF je u tome što 3NF dozvoljava funkcionalnu zavisnost $A \rightarrow B$ (B je funkcionalno zavisno od A) ako je B primarni ključ a A nije ključ kandidat, dok BCNF nalaže da A mora biti ključ kandidat. Dakle, BCNF je jača forma od 3NF, pa je svaka tabela koja je u BCNF automatski i u 3NF, a obratno ne važi.

Tabela se transformiše u BCNF tako što se vrši njena dekompozicija na dve ili više novih tabela. Međutim, ponekad nije poželjno normalizovati tabelu do BCNF. Naime, može se desiti da se prilikom dekompozicije tabele izgubi jedna ili više funkcionalnih zavisnosti zbog toga što se determinanta i od nje zavisni atributi izmeste u različite tabele. U ovakvoj situaciji treba proceniti da li je bolje stati kod 3NF, koja uvek čuva sve funkcionalne zavisnosti. Odluku da li normalizovati tabelu do BCNF ili stati kod 3NF treba doneti na osnovu sledeće analize:

- Da li će značajni podaci biti izgubljeni u slučaju dekompozicije tabele i gubitka jedne ili više funkcionalnih zavisnosti
- Kolika će biti količina redundantnih podataka (podataka koji se ponavljaju) u slučaju da se dekompozicija ne izvrši, tj. da se ostane na 3NF.

Četvrta normalna forma (4NF)

Iako BCNF eliminiše sve anomalije koje proističu iz funkcionalnih zavisnosti, dalja istraživanja su dovela do uočavanja još jednog tipa zavisnosti koji se naziva *zavisnost višestruke vrednosti* koji takođe može da prouzrokuje redundantnost podataka.

Zavisnost višestruke vrednosti

Pojava zavisnosti višestruke vrednosti u tabeli može da se desi zbog 1NF koja ne dozvoljava da atribut ima više vrednosti, tj. da se sastoji iz više podataka. Zavisnost višestruke vrednosti biće objašnjana na primeru tabele *EkspozituraZaposleniVlasnik* iz baze podataka agencije za prodaju i izdavanje nekretnina.

IdentBrEkspoziture	ImeZaposlenog	ImeVlasnikaNekretnine
B003	Zoran Petrović	Jelena Janković
B003	Miodrag Aleksić	Jelena Janković
B003	Zoran Petrović	Predrag Stefanović
B003	Miodrag Aleksić	Predrag Stefanović

Tabela EkspozituraZaposleniVlasnik

U ovom primeru zaposleni Zoran Petrović i Miodrag Aleksić rade u ekspozituri B003, a vlasnici nekretnina Jelena Janković i Predrag Stefanović su registrovani u istoj ekspozituri, dakle B003. Pošto ne postoji direktna veza između zaposlenih i vlasnika u datoj ekspozituri, mora se kreirati zapis za sve kombinacije zaposlenih i vlasnika da bi se obezbedila konzistentnost. Ova situacija predstavlja zavisnost višestruke vrednosti u tabeli *EkspozituraZaposleniVlasnik*. Drugim rečima, zavisnost postoji zato što su u tabeli predstavljene dve nezavisne 1:* veze.

Zavisnost višestruke vrednosti predstavlja zavisnost između atributa A, B i C u tabeli, takvu da za svaku vrednost A postoji više vrednosti B i više vrednosti C, pri čemu su vrednosti B i C nezavisne jedna od druge.

Definicija četvrte normalne forme

Tabela je u 4NF ako je u BCNF i ako ne sadrži zavisnosti višestruke vrednosti.

4NF je jača normalna forma od BCNF, jer ne dozvoljava da tabele imaju zavisnost višestruke vrednosti, a samim tim i redundantne podatke (podatke koji se ponavljaju). Normalizacija iz BCNF u 4NF se vrši dekompozicijom tabele i izmeštanjem atributa zajedno sa njihovim determinantama u novu tabelu. Tabela *EkspozituraZaposleniVlasnik* iz prethodnog pasusa se dekomponuje na tabele *EkspozituraZaposleni* i *EkspozituraVlasnik*.

IdentBrEkspoziture	ImeZaposlenog
B003	Zoran Petrović
B003	Miodrag Aleksić

Tabela EkspozituraZaposleni

IdentBrEkspoziture	ImeVlasnikaNekretnine
B003	Jelena Janković
B003	Predrag Stefanović

Tabela EkspozituraVlasnik

4NF eliminiše redundantnost podataka (podatke koji se ponavljaju) i mogućnost pojave anomalija ažuriranja.

Peta normalna forma (5NF)

Uvek kada se vrši dekompozicija tabele na *dve* nove, rezultujuće tabele imaju svojstvo poznato kao *postojanost spajanja (lossless-join)*. Ovo svojstvo se odnosi na činjenicu da se tabele nastale dekompozicijom mogu ponovo spojiti u originalnu tabelu. Međutim, postoje slučajevi kada je potrebno izvršiti dekompoziciju originalne tabele na *više od dve* nove tabele. Iako se u praksi prilično retko sreću, ovakvi slučajevi se rešavaju primenom *pete normalne forme (5NF)*.

Postojanost spajanja (lossless-join)

Postojanost spajanja je svojstvo dekompozicije koje omogućava da se, prilikom ponovnog spajanja tabela, generišu samo originalni podaci.

Dakle, postojanost spajanja osigurava da se, prilikom ponovnog spajanja dve tabele u onu čijom su dekompozicijom nastale, generišu samo oni podaci koji su postojali u originalnoj tabeli pre dekompozicije, što znači da je zagarantovana potpuna rekonstrukcija originalne tabele.

Definicija pete normalne forme

Tabela je u 5NF ako nema zavisnosti spajanja.

Normalizacija do 5NF se vrši dekompozicijom tabele u kojoj se uoče zavisnosti spajanja na *više od dve* nove tabele. Važno je napomenuti da će ponovno spajanje bilo koje *dve* od novonastalih tabela generisati "lažne" podatke, ali će zato ponovno spajanje *svih* novonastalih tabela verno rekonstruisati originalnu tabelu.

METODOLOGIJA DIZAJNIRANJA BAZE PODATAKA

Metodologija dizajniranja je strukturirani pristup koji primenjuje procedure, tehnike, alate i dokumentaciju u procesu razvoja baze podataka.

U procesu razvoja baze podataka definisane su tri osnovne faze, od kojih svaka sadrži niz koraka.

Konceptualni dizajn

Konceptualni dizajn predstavlja prvu fazu tokom koje se kreira konceptualni model podataka na osnovu korisničke specifikacije zahteva. Konceptualni dizajn baze je potpuno nezavisan od detalja implementacije kao što su na primer softverska platforma, programski jezici, hardverska platforma ili bilo koji drugi fizički aspekti. Konceptualni dizajn se sastoji iz sledećih koraka (po hronološkom redosledu):

1. Identifikovati entitete
2. Identifikovati veze
3. Identifikovati atribut i pridružiti ih entitetima i vezama
4. Odrediti domene atributa
5. Odrediti ključeve kandidate i primarne ključeve
6. Proveriti da li u modelu postoji redundansa
7. Proveriti da li model zadovoljava potrebe korisničkih transakcija
8. Po potrebi revidirati model zajedno sa korisnicima.

Logički dizajn

Logički dizajn predstavlja drugu fazu u dizajniranju baze podataka i podrazumeva kreiranje modela koji se bazira na konceptualnom modelu iz prethodne faze. Model iz prethodne faze se doraduje i prilagođava konačnom modelu koji je usvojen za bazu (npr. relacioni model). Dakle, logički model se kreira namenski, za tačno određeni model podataka koji koristi izabrana DBMS platforma. Kao rezultat se dobija model koji je nezavisan od fizičkih detalja DBMS, kao na primer strukture podataka i indeksa. Za testiranje korektnosti logičkog modela koristi se tehnika normalizacije koja obezbeđuje da podaci budu pravilno raspoređeni po tabelama. Logički dizajn se sastoji iz sledećih koraka (po hronološkom redosledu):

1. Kreirati tabele
2. Izvršiti normalizaciju tabela
3. Proveriti da li tabele zadovoljavaju potrebe korisničkih transakcija
4. Definisati ograničenja za očuvanje integriteta (*integrity constraints*)
5. Razmotriti ekspanziju sistema u budućnosti
6. Po potrebi revidirati model zajedno sa korisnicima.

Fizički dizajn

Fizički dizajn je treća i poslednja faza u procesu dizajniranja baze podataka. Ova faza podrazumeva kreiranje elemenata baze na tačno određenoj platformi sa svim detaljima i specifičnostima vezanim za nju. Treba spomenuti i to, da ova faza može izazvati povratni efekat na prethodnu, jer neki specifični detalji vezani za fizički dizajn na izabranoj platformi mogu da izazovu promene u logičkom dizajnu.

Dakle, fizički dizajn se odnosi na konkretnu implementaciju na izabranoj DBMS platformi i sastoji se iz sledećih koraka (po hronološkom redosledu):

1. Kreirati tabele
2. Kreirati ograničenja (*constraints*) koja reprezentuju poslovna pravila (*business rules*)
3. Analizirati transakcije i njihov uticaj na performanse sistema
4. Izabrati organizaciju fajlova (npr. heap, hash, B⁺-drvo, klasteri)
5. Kreirati indekse
6. Izvršiti procenu potrebnog prostora za skladištenje podataka
7. Kreirati "korisničke preglede" (*user views*)
8. Realizovati bezbednosne mehanizme
9. Razmotriti uvođenje kontrolisane redundanse u cilju poboljšanja performansi sistema.

2.

Baza podataka Magacin

Baza podataka **MAGACIN** predstavlja efikasno softversko rešenje za vođenje evidencije o sredstvima za rad, opremi, potrošnom materijalu i još nekim vrstama imovine jedne ustanove ili firme. Baza je originalno razvijena za potrebe ustanove u kojoj se oprema i imovina izdaju zaposlenima na korišćenje, a oni ih upotrebljavaju u svakodnevnom radu, kao osnovno ili pomoćno sredstvo.

Razvoj baze **MAGACIN** biće predstavljen hronološki, kroz faze.

I faza: PLANIRANJE

Zadatak: Glavni zadatak ove faze je definisanje ciljeva koje baza podataka treba da ispuni.

Kao što je u uvodu u ovo poglavlje već spomenuto, baza treba da omogući vođenje evidencije o sredstvima za rad, opremi i potrošnom materijalu u jednoj ustanovi ili firmi. U ustanovi za čije potrebe je baza razvijena javila se potreba za vođenjem precizne i ažurne evidencije o tome koja su sredstva, oprema, nameštaj i druge vrste imovine izdati na korišćenje svakom od zaposlenih. Takođe je bilo potrebno omogućiti magacioneru vođenje evidencije o potrošnom materijalu u smislu količina koje su izdate i preostalih količina na lageru, kao i štampanje odgovarajućih vrsta dokumenata i izveštaja.

II faza: DEFINICIJA SISTEMA

Zadatak: U okviru druge faze potrebno je identifikovati opseg i granice baze, način na koji će se ona uklopiti u informacioni sistem posmatranog okruženja, ako on već postoji, kao i definisati glavne korisnike baze.

Pod pojmom opsega i granica baze podrazumeva se definisanje korisnika i vrste podataka sa kojima će baza raditi. Korisnici mogu biti pojedinačne osobe ili manje organizacione jedinice okruženja za čije potrebe se razvija baza (oddeljenja, odseci, itd.).

U slučaju baze **MAGACIN** realizacija ove faze je bila jednostavna, jer se kao potencijalni korisnik javlja samo magacioner, a vrsta podataka kojima on barata je praktično definisana tokom prve faze, dakle radi se o podacima vazanim za sredstva, opremu i imovinu.

III faza: PRIKUPLJANJE I ANALIZA ZAHTEVA

Zadatak: Prikupljanje i analiza informacija o okruženju za čije potrebe se baza razvija i izrada specifikacije zahteva iz koje će proistići funkcionalnost baze podataka.

Za prikupljanje i analizu zahteva tokom ove faze korišćene su tehnike intervjuisanje, posmatranje rada u okruženju za čije potrebe se razvija baza i proučavanje dokumentacije.

Intervjuisanje je obavljeno sa magacionerom, od koga su prikupljene informacije o metodologiji rada, najvažnijim aktivnostima koje obavlja tokom rada (konkretno o vrstama aktivnosti i njihovom hronološkom redosledu), vrsti podataka kojima barata, željenom načinu obrade i prezentacije podataka.

Posmatranje rada u okruženju za čije potrebe se razvija baza je tehnika koja je primenjena u kombinaciji sa intervjuisanjem i koja je pomogla da se stekne konkretniji uvid u proces rada koga je potrebno modelirati.

Proučavanje dokumentacije je tehnika koja je u slučaju baze **MAGACIN** primenjena kroz proučavanje dokumenata koji se pojavljuju u posmatranom procesu rada. Predmet proučavanja je bila struktura i forma dokumenata koji služe za evidentiranje unosa novih stavki u magacin (*Trebovanje*), za izdavanje sredstava (*Revers*) i za povraćaj sredstava u magacin (*Povratnica*). To je pomoglo u uočavanju i definisanju potencijalnih atributa za entitete koji će se naći u bazi.

Primenom spomenutih tehnika izrađena je studija na osnovu koje se mogu jasno definisati biznis model posmatranog okruženja i specifikacija zahteva.

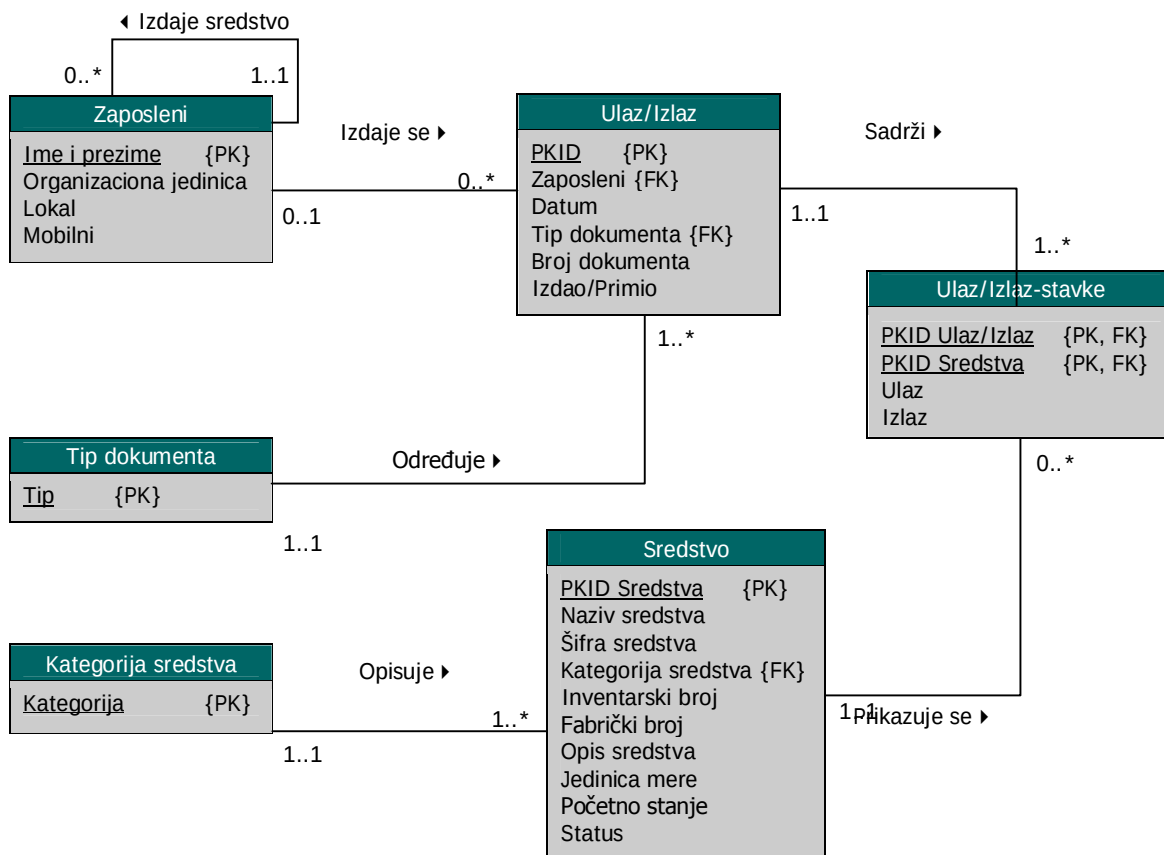
Studija

Potrebno je razviti bazu podataka za potrebe ustanove u kojoj se oprema i imovina izdaju zaposlenima na korišćenje, a oni ih upotrebljavaju u svakodnevnom radu, kao osnovno ili pomoćno sredstvo. Ustanova ima pedesetak zaposlenih, a ukupan broj komada sredstava, opreme i ostale raspoložive imovine ne prelazi hiljadu. Imovina je podeljena na nekoliko jasno definisanih kategorija, kao što su na primer nameštaj, informatička oprema, mobilni telefoni, itd. Postoji samo jedan potencijalni korisnik baze – magacioner, koji obavlja i unos sredstava u magacin i izdavanje sredstava i opreme zaposlenima. Pored evidencije u lokalnom magacinu ustanove za čije potrebe se razvija baza, sva sredstva moraju biti evidentirana i u centralnom magacinu, koji je hijerarhijski nadređen lokalnom. Svako sredstvo (osim potrošnog i kancelarijskog materijala) mora da ima inventarski broj, a za sva sredstva koja imaju fabrički broj i on se evidentira. U lokalni magacin sredstva stižu direktno od dobavljača, a unos se vrši odmah po prijemu, stavku po stavku. U tom trenutku ni jedno sredstvo nema inventarski broj koji se dodeljuje od strane centralnog magacina. Zato se za svaku nabavku formira pisani dokument *Trebovanje*, koji ima višestruku ulogu: na osnovu njega centralni magacin vrši dodelu inventarskog broja svakom sredstvu, za centralni magacin ono predstavlja dokument na osnovu koga se sredstva izdaju lokalnom magacinu, a za lokalni magacin trebovanje predstavlja dokument na osnovu koga sredstva ulaze u magacin. Izdavanje sredstava i opreme zaposlenima od strane lokalnog magacina se vrši putem pisanog dokumenta *Revers*, koji sadrži jednu ili više stavki, za svako sredstvo ili komad opreme koji se izdaje. Svakom zaposlenom može da se izda neograničen broj reversa. Povraćaj sredstava i opreme od strane zaposlenih u magacin se vrši pomoću pisanog dokumenta *Povratnica*, koji sadrži stavke koje zaposleni razdužuje. Moguće situacije su da zaposleni vraća deo zaduženih sredstava ili da se kompletno razdužuje. Magacioner je u obavezi da povremeno vrši pregled stanja imovine u smislu kod kog zaposlenog se nalazi određeno sredstvo ili oprema, a jedanput godišnje vrši popis kompletne imovine sa stanjem. O potrošnom i kancelarijskom materijalu nije potrebno voditi posebno preciznu evidenciju, osim orijentacionog stanja količina na lageru.

IV faza: DIZAJNIRANJE BAZE

Zadatak: Izrada konceptualnog, logičkog i fizičkog modela baze podataka.

Kao rezultat projektovanja konceptualnog i logičkog modela nastao je relacioni dijagram baze prikazan na slici.



Relacioni dijagram baze Magacin

Entiteti

U okruženju ustanove za čije potrebe se razvija baza uočeni su sledeći entiteti:

1. **Zaposleni**, koji reprezentuje sve radnike zaposlene u posmatranoj ustanovi.
2. **Ulaz / Izlaz**, koji reprezentuje pisana dokumenta *Trebovanje*, *Revers* i *Povratnica* koja se izdaju prilikom unosa sredstava u magacin, prilikom izdavanja sredstava i opreme zaposlenima i prilikom povraćaja sredstava u magacin, respektivno.
3. **Ulaz / Izlaz-stavke**, koji reprezentuje stavke dokumenta Ulaz/Izlaz. Entiteti *Ulaz/Izlaz* i *Ulaz/Izlaz-stavke* bi, generalno gledano, mogli da budu objedinjeni u jedan entitet, npr. *Dokument*, ali bi se u tom entitetu pojavila velika redundansa podataka, tj. entitet bi sadržao mnogo podataka koji se ponavljaju iz razloga što bi bio normalizovan samo do 1NF.

4. **Sredstvo**, koji reprezentuje sredstva za rad, opremu i svu ostalu imovinu ustanove.
5. **Tip dokumenta**, koji određuje da li se radi o *Trebovanju*, *Reversu* ili *Povratnici*. U teoriji baza podataka ovaj entitet bi se svrstao u grupu *slabih entiteta*. Entiteti čije postojanje zavisi od postojanja drugih entiteta nazivaju se slabim entitetima. Entitet *Tip dokumenta* zavisi od entiteta *Ulaz/Izlaz*, jer ako ne postoji ulazno/izlazni dokument, ne postoji ni tip dokumenta.
6. **Kategorija sredstva**, koji bliže opisuje komad imovine tako što ga svrstava u neku od postojećih kategorija. I *Kategorija sredstva* spada u grupu slabih entiteta, jer je zavisn od entiteta *Sredstvo*.

Za svaki od navedenih entiteta definisani su odgovarajući atributi i domeni atributa, gde je to bilo potrebno (npr. *Tip dokumenta* i *Kategorija sredstva*). Zatim je izvršen izbor primarnih ključeva, pri čemu posebno objašnjenje zaslužuju samo ključevi entiteta *Zaposleni*, *Sredstvo* i *Ulaz/Izlaz-stavke*.

Primarni ključevi

Primarni ključ entiteta *Zaposleni*. Iako, generalno gledano, *Ime i prezime* nije idealan atribut za primarni ključ zbog toga što se mogu pojaviti osobe sa istim imenom i prezimenom, on je ipak određen za primarni ključ zbog toga što u posmatranom okruženju ima samo pedesetak zaposlenih i, u trenutku projektovanja baze, nisu postojale osobe sa istim imenom i prezimenom. Uz malu verovatnoću za značajnije povećanje broja zaposlenih u skorijoj budućnosti, izgledi za pojavu osoba sa istim imenom i prezimenom su mali. Čak iako se takav problem javi, ne bi bilo isuviše komplikovano dodati srednje slovo i na taj način zadovoljiti kriterijum jedinstvenosti.

Primarni ključ entiteta *Sredstvo*. Za entitet *Sredstvo* uočeni su ključevi kandidati *Naziv sredstva*, *Fabrički broj* i *Inventarski broj*. U studiji je spomenuto da se inventarski broj dodeljuje sredstvu naknadno, često i posle unošenja u bazu. Zbog toga on nije izabran za primarni ključ. Intervjuisanjem magacionera došlo se do saznanja da mnoga sredstva nemaju fabrički broj, pa je i on proglašen neodgovarajućim kandidatom. *Naziv sredstva* takođe nije u potpunosti zadovoljio, jer se može javiti više sredstava sa istim nazivom. Taj problem bi se mogao rešiti formiranjem složenog primarnog ključa (npr. *Naziv sredstva*, *Inventarski broj* i *Fabrički broj*) ali se, zbog spomenutih problema sa inventarskim i fabričkim brojem, i od toga odustalo. Konačno, kao primarni ključ je određen automatski generisani broj koji obezbeđuje jedinstvenost zapisa, bez obzira na to da li u trenutku unošenja u bazu sredstvo ima inventarski i/ili fabrički broj i bez obzira na naziv sredstva.

Primarni ključ entiteta *Ulaz/Izlaz-stavke*. Entitet *Ulaz/Izlaz-stavke* ima složeni primarni ključ koji se sastoji iz dva atributa: *PKID_Ulaz/Izlaz* i *PKID_Sredstva*. Ovakvo rešenje je primenjeno zbog toga što je trebalo obezbediti da se jedno sredstvo može pojaviti samo jedanput u jednom dokumentu (*Revers*, *Trebovanje* ili *Povratnica*). Pošto će se atribut *PKID_Sredstva* sigurno pojaviti više puta u tabeli *Ulaz/Izlaz-stavke*, a atribut *PKID_Ulaz/Izlaz* je isti za svaku stavku jednog dokumenta, jedini način da se realizuje jedinstvenost svakog zapisa u tabeli *Ulaz/Izlaz-stavke* je da se formira složeni ključ, pošto atributi *PKID_Ulaz/Izlaz* i *PKID_Sredstva* u kombinaciji formiraju jedinstvenu vrednost koja jednoznačno identifikuje svaki zapis u tabeli, pod uslovom da se obezbedi da atribut *PKID_Ulaz/Izlaz* bude jedinstven za svaki dokument koga reprezentuje.

Veze

U modelu baze **MAGACIN** definisane su sledeće veze:

1. **Izdaje sredstvo**, koja spada u grupu *rekurzivnih* ili *unarnih* veza i predstavlja odnos entiteta *Zaposleni* sa samim sobom. Veza je tipa 1:*, i u njoj entitet *Zaposleni* učestvuje u dve uloge: u ulozi magacionera koji izdaje sredstva radnicima ustanove, i u ulozi "običnog" zaposlenog kome se izdaje sredstvo. Sa aspekta magacionera, on *izdaje sredstvo nijednom zaposlenom ili više zaposlenih* (0..*), a sa aspekta "običnog" zaposlenog sredstvo se *uvek izdaje od strane jednog magacionera* (1..1). Dakle, u ulozi magacionera participacija entiteta *Zaposleni* je opciona (0), jer nije obavezno da se svakom zaposlenom izda neko sredstvo, a kardinalnost je više (*), jer se sredstva mogu izdati neograničenom broju zaposlenih. Preciznije rečeno, umesto * mogao bi se upisati broj zaposlenih (npr. 50), jer je to maksimalni broj osoba kojima se mogu izdavati sredstva. U ulozi "običnog" zaposlenog participacija entiteta *Zaposleni* je obavezna (1), jer se izdavanje sredstva ne može obaviti bez magacionera. Kardinalnost u ovom slučaju iznosi 1, jer u ustanovi postoji samo jedan magacioner.
2. **Izdaje se**, koja predstavlja odnos između entiteta *Zaposleni* i *Ulaz/Izlaz*. Veza je tipa 1:*, i njome se realizuje poslovno pravilo ustanove da se, prilikom izdavanja ili prijema sredstava, opreme ili neke druge vrste imovine, zaposlenom izdaje odgovarajući ulazno/izlazni dokument (predstavljen entitetom *Ulaz/Izlaz*) koji opisuje odgovarajuću radnju. Sa aspekta zaposlenog, njemu se *izdaje nijedan ili više dokumenata* (0..*), a sa aspekta ulazno/izlaznog dokumenta, on se *izdaje nijednom ili jednom zaposlenom, tj. dokument može da glasi na samo jednog zaposlenog* (0..1). Dakle, participacija entiteta *Zaposleni* je opciona (0), jer nije obavezno da se svakom zaposlenom izda neko sredstvo, a samim tim i dokument. Kardinalnost entiteta *Zaposleni* je više (*), jer se zaposlenom može izdati neograničeno mnogo dokumenata. Participacija entiteta *Ulaz/Izlaz* je takođe opciona (0), jer se ne mora svakom zaposlenom izdati dokument, a kardinalnost je 1 zato što dokument može da glasi na samo jednog zaposlenog.
3. **Sadrži**, koja predstavlja odnos između entiteta *Ulaz/Izlaz* i *Ulaz/Izlaz-stavke*. Veza je tipa 1:*, i govori o tome da se na jednom ulazno/izlaznom dokumentu može naći jedna ili više stavki. Sa aspekta dokumenta, on *može da sadrži jednu ili više stavki* (1..*), a sa aspekta stavke, ona se *može pojaviti samo jedanput u jednom dokumentu* (1..1). Dakle, participacija entiteta *Ulaz/Izlaz* je obavezna (1), jer mora postojati dokument koji će sadržati stavke, dok je kardinalnost više (*), jer jedan dokument može sadržati više stavki. Participacija entiteta *Ulaz/Izlaz-stavke* je obavezna (1), jer na dokumentu mora postojati bar jedna stavka, a kardinalnost je 1, jer se jedna stavka sme pojaviti samo jedanput u jednom dokumentu.
4. **Određuje**, koja predstavlja odnos između entiteta *Tip dokumenta* i *Ulaz/Izlaz*. Veza je tipa 1:*, i određuje tip ulazno/izlaznog dokumenta. Sa aspekta tipa dokumenta, on *određuje jedan ili više dokumenata* (1..*), a sa aspekta ulazno/izlaznog dokumenta, on je *uvek određen samo jednim tipom dokumenta* (1..1). Dakle, participacija entiteta *Tip dokumenta* je obavezna (1), jer za svaki dokument mora da se navede tip, dok je kardinalnost više (*), jer jedan tip može da se navede za neograničeno mnogo dokumenata. Participacija entiteta *Ulaz/Izlaz* je obavezna (1), jer bez ulazno/izlaznog dokumenta tip dokumenta nema nikakav smisao, a kardinalnost je 1, jer jednom dokumentu može da se dodeli samo jedan tip.

5. **Opisuje**, koja predstavlja odnos između entiteta *Kategorija sredstva* i *Sredstvo*. Veza je tipa 1:*, i određuje kojoj od postojećih kategorija pripada sredstvo. Sa aspekta kategorije sredstva, ona *opisuje jedno ili više sredstava* (1..*), a sa aspekta sredstva, ono je *uvek opisano samo jednom kategorijom*, tj. *pripada samo jednoj kategoriji* (1..1). Dakle, participacija entiteta *Kategorija sredstva* je obavezna (1), jer svako sredstvo mora da pripada jednoj kategoriji, dok je kardinalnost više (*), jer jedna kategorija može da se navede za neograničeno mnogo sredstava. Participacija entiteta *Sredstvo* je obavezna (1), jer bez sredstva kategorija sredstva nema nikakav smisao, a kardinalnost je 1, jer jedno sredstvo može da pripada samo jednoj kategoriji.
6. **Prikazuje se**, koja predstavlja odnos između entiteta *Sredstvo* i *Ulaz/Izlaz-stavke*. Veza je tipa 1:*, i prikazuje koja sredstva (navedena kao stavke) se nalaze na datom ulazno/izlaznom dokumentu. Sa aspekta sredstva, ono *može da se prikaže na nijednom ili na više dokumenata*, tj. *ne mora nikada da se pojavi kao stavka ili može da se pojavi kao stavka na više različitih ulazno/izlaznih dokumenata* (0..*), a sa aspekta stavke, ona *uvek prikazuje samo jedno sredstvo* (1..1). Dakle, participacija entiteta *Sredstvo* je opcionalna (0), jer sredstvo ne mora obavezno da bude izdato nekome na korišćenje, dok je kardinalnost više (*), jer sredstvo može da se nađe kao stavka na neograničeno mnogo ulazno/izlaznih dokumenata. Participacija entiteta *Ulaz/Izlaz-stavke* je obavezna (1), jer se sredstvo uvek prikazuje kao stavka dokumenta, a kardinalnost je 1 zato što jedna stavka uvek prikazuje samo jedno sredstvo.

V faza: IZBOR DBMS PLATFORME

Zadatak: Izbor DBMS platforme na osnovu sistemskih zahteva kao što su performanse, bezbednost i integritet.

Uzimajući u obzir činjenicu da okruženje za čije potrebe je baza razvijena ima pedesetak zaposlenih i da ukupan broj komada sredstava, opreme i ostale raspoložive imovine ne prelazi hiljadu, kao DBMS platforma je izabran *Microsoft Access*. Procenjeno je da, zbog relativno malog opterećenja u smislu transakcija i količine podataka koje će čuvati, ova platforma u potpunosti zadovoljava potrebe po pitanju performansi. Sledeći kriterijum za izbor bio je kvalitet grafičkog okruženja. *Access* dizajneru stavlja na raspolaganje raznovrsne elemente grafičkog interfejsa od kojih se može kreirati vrlo kvalitetno radno okruženje koje je integrisano u samu bazu, pa ne postoji potreba za izradom posebne aplikacije. Izabrana platforma zadovoljava i kriterijum bezbednosti, naročito ako se uzme u obzir činjenica da je broj korisnika baze samo jedan (magacioner) i da baza ne radi u mrežnom okruženju.

VI faza: DIZAJNIRANJE APLIKACIJE

Zadatak: Kreiranje korisničkog interfejsa i programa (aplikacija) koji će raditi sa bazom, kao i transakcija koje pristupaju bazi i rade sa podacima.

Korisnički interfejs je realizovan kao sastavni deo baze, i kreiran je tako da pre svega bude lak i intuitivan za upotrebu, da blagovremeno detektuje greške u radu korisnika i da ga upozori na postojanje istih, da poništi efekte načinjene greške i na taj način bazu konstantno održava u konzistentnom stanju. Prozori aplikacije su strogo namenski kreirani za tačno određenu operaciju ili nekoliko logički povezanih operacija i realizovani su kao estetski čiste i pregledne forme. Isto važi i za štampane izveštaje.

Korisničke transakcije zajedno sa pratećim funkcijama i procedurama implementirane su delom kroz ugrađene mehanizme programa Access, a delom kao sastavni deo prozora aplikacije upotrebom programskog jezika Visual Basic. U bazi **MAGACIN** su zastupljene sve tri vrste transakcija, dakle transakcije za prezentaciju podataka, transakcije za promenu podataka i mešovite transakcije.

VII faza: IMPLEMENTACIJA

Zadatak: Konkretna implementacija, tj. kreiranje elemenata baze podataka na izabranoj platformi.

Kompletna baza **MAGACIN** je implementirana primenom alata integrisanih u samo razvojno okruženje. Naime, izabrana DBMS platforma (Microsoft Access) raspolaže kvalitetnim grafički orijentisanim alatima za kreiranje svih potrebnih elemenata baze, pa je ta pogodnost i iskorišćena u procesu implementacije.

VIII faza: PUNJENJE BAZE PODACIMA

Zadatak: Punjenje novonastale baze podacima i migracija sa prethodne verzije, ako postoji, uz eventualnu konverziju podataka.

Baza **MAGACIN** je manualno napunjena podacima koji su se do tada nalazili u raznim dokumentima u papirnoj formi a, pošto prethodna verzija baze nije postojala, nije bilo potrebno vršiti ni migraciju ni konverziju podataka.

IX faza: TESTIRANJE

Zadatak: Otkrivanje eventualnih grešaka i/ili nepravilnosti u radu i demonstriranje da baza i prateće aplikacije, ako postoje, rade u skladu sa korisničkom specifikacijom zahteva.

Testiranje baze **MAGACIN** je izvršeno posle punjenja, dakle sa realnim podacima, ali pre konačnog puštanja u rad i nad kopijom originalnog fajla, tako da bezbednost podataka nije bila ugrožena. U testiranju su aktivno učestvovali krajnji korisnik i projektant, pa su svi uočeni nedostaci i nepravilnosti u radu brzo i efikasno otklonjeni.

X faza: ODRŽAVANJE

Zadatak: Nadgledanje performansi sistema (baze), održavanje i usavršavanje aplikacija.

S obzirom na jednostavnost baze **MAGACIN**, ova faza se praktično svela na naknadnu realizaciju još nekoliko manjih korisničkih zahteva u domenu funkcionalnosti i grafičkog interfejsa.

3.

Klijent/Server arhitektura

Termin klijent/server se po prvi put pojavio tokom 80-tih godina prošlog veka i odnosio se na personalne računare (PC) na mreži. Pravi klijent/server model je počeo da se primenjuje u kasnim 80-tim godinama. Klijent/server softverska arhitektura je fleksibilna, modularna i bazirana na porukama, a u poređenju sa centralizovanom mejnfrejm arhitekturom koja joj je prethodila nudi poboljšanu upotrebljivost, prilagodljivost, međuoperativnost i skalabilnost.

Klijent se definiše kao zahtevalac usluge, a server kao pružalac usluge. Jedan računar može da bude i klijent i server u zavisnosti od softverske konfiguracije.

Klijent/server arhitektura je nastala iz potrebe za prevazilaženjem ograničenja dvaju arhitektura koje su joj prethodile, mejnfrejm i arhitekture bazirane na deljenju fajlova (*file sharing*). Ograničenja su se ogledala u slaboj ili nikakvoj podršci za grafičko korisničko okruženje, limitiranom broju korisnika koje je sistem u stanju da podrži, kao i u otežanom pristupu fizički razmeštenim izvorima podataka.

Klijent/server arhitektura uvodi u upotrebu relacione baze podataka i servere baze podataka koji zamenjuju fajl servere, tako da korisnički upiti mogu direktno da se procesiraju. Mrežni saobraćaj je osetno smanjen upravo zbog toga što se preko mreže više ne prenose celi fajlovi, već samo korisnički upiti i odgovori na njih. Koncept grafičkog korisničkog interfejsa i deljene baze podataka omogućava jednostavan unos i promenu podataka od strane više korisnika. U klijent/server okruženju, za komunikaciju između klijenta i servera se najčešće koriste RPC (*Remote Procedure Calls*) i SQL (*Structured Query Language*) upiti i odgovori na njih.

Prednosti klijent/server arhitekture se ogledaju u sledećem:

- Procesiranje se raspoređuje na više računara, nekritični podaci i funkcije se obrađuju na klijentu, kritične funkcije se obrađuju na serveru.
- Klijentske radne stanice se optimiziraju za unos i prezentaciju podataka. Radne stanice treba da imaju kvalitetne grafičke podsisteme.
- Server se optimizira za obradu i skladištenje podataka. Serveri treba da imaju mnogo RAM memorije i velike hard-diskove.
- Horizontalno skaliranje – opterećenje između servera može ravnomernije da se rasporedi jednostavnim dodavanjem više servera u sistem.
- Vertikalno skaliranje – ukoliko je procesorska snaga postojećih računara nedovoljna, može se preći na snažnije računare ili računarske platforme.

- Smanjuje se redundansa i potreba za replikacijom podataka – podaci se čuvaju samo na serverima, a ne na svakom klijentu.

4.

Višeslojna klijent/server arhitektura

Zadržavajući sve karakteristike klijent/server arhitekture, višeslojna klijent/server arhitektura predstavlja njenu praktičnu realizaciju koja je tokom vremena evoluirala u skladu sa zahtevima koji su postavljani od strane korisnika, ali i od strane inženjera koji su u stalnoj potrazi za boljim performansama razvijali sisteme čije su arhitekture najbolje odgovarale novonastalim potrebama. Višeslojne klijent/server arhitekture se najčešće realizuju kao 2-slojne, 3-slojne i n-slojne.

2-slojna arhitektura

2-slojna softverska arhitektura je razvijena tokom 80-tih godina prošlog veka iz fajl server arhitekture. Ova arhitektura poboljšava upotrebljivost primenom grafičkog korisničkog interfejsa, skalabilnost (u stanju je da uspešno podrži oko 100 korisnika) i fleksibilnost uvođenjem koncepta deljenih podataka. 2-slojna arhitektura zahteva minimalnu intervenciju operatera, a uglavnom se koristi za izgradnju jednostavnijih sistema za obradu informacija kod kojih vreme obrade i odziva nisu od ključnog značaja.

2-slojna arhitektura se sastoji iz tri komponente koje su raspoređene u dva sloja – klijentski i serverski. Ova arhitektura obuhvata sledeće komponente:

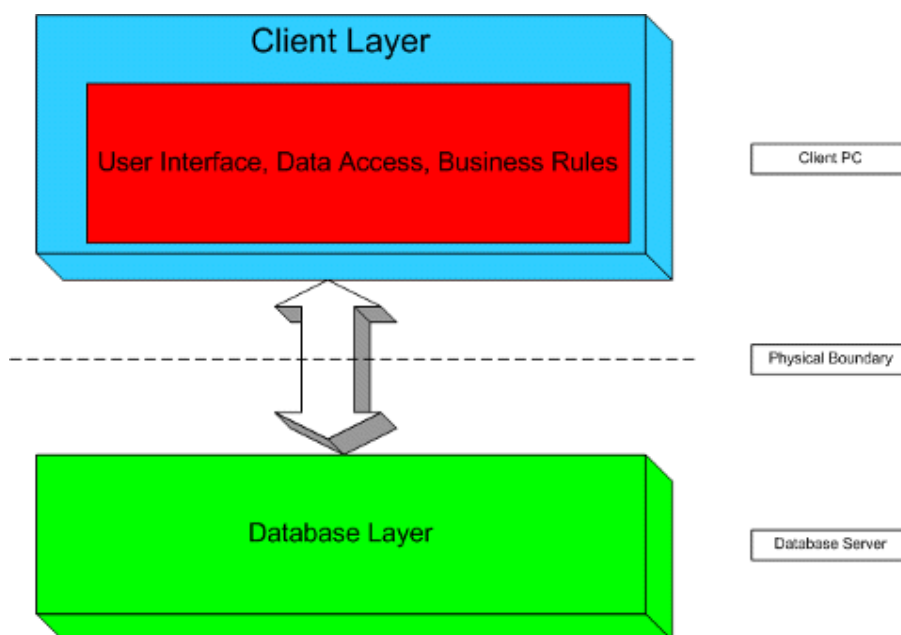
- Korisnički interfejs (unos i prezentacija podataka)
- Procesiranje/obrada
- Baza podataka.

2-slojni dizajn izdvaja korisnički interfejs isključivo na klijenta. Baza podataka se nalazi na serveru, a obrada se deli između klijenta i servera (slika 4.1).

Kod 2-slojnih aplikacija, najveći deo procesiranja se obavlja na strani klijenta. Server baze podataka uglavnom obrađuje zahteve za pristup podacima, ponekad u formi uskladištenih procedura (*stored procedures*). Klijenti komuniciraju sa serverom preko SQL upita.

U poređenju sa fajl server arhitekturom, 2-slojna arhitektura poboljšava fleksibilnost i skalabilnost upravo zbog postojanja dva sloja koje je lakše održavati, menjati i precizno prilagoditi postojećim potrebama. Upotrebljivost 2-slojne arhitekture je poboljšana zato što je moguće razviti korisnički interfejs koji je do detalja prilagođen korisničkim zahtevima, a da se pri tome ne remeti rad ostatka sistema.

2-slojne aplikacije se često koriste za razvoj jednostavnijih sistema kod kojih je broj korisnika relativno mali (do 100), kod kojih je transakciono opterećenje relativno malo i kod kojih se poslovna pravila (*business rules*) ne menjaju često.



Slika 4.1 – Dvoslojna klijent/server arhitektura

Skalabilnost 2-slojnih sistema se, kao što je već spomenuto, kreće oko granice od 100 korisnika. Veći broj korisnika prevazilazi kapacitete ovakvih sistema, pre svega zbog činjenice da klijenti i serveri konstantno razmenjuju *keep-alive* poruke, dakle poruke za proveru da li je "sagovornik sa druge strane konekcije i dalje živ". Ove poruke se razmenjuju bez obzira na to da li se neki posao obavlja ili ne, pa stoga u znatnoj meri opterećuju mrežu. Skalabilnost se takođe smanjuje i u slučajevima kada se poslovna logika implementira u formi uskladištenih procedura (*stored procedures*), dakle na serveru baze podataka, jer u toj situaciji raste potreba za procesorskom snagom servera. Pošto svi klijenti koriste isti server za izvršavanje aplikacije, to smanjuje broj korisnika koji se mogu opslužiti.

Konekcije između klijenata i servera se uspostavljaju dugo, a serveri moraju da raspolazu sa dosta RAM memorije. Pošto je povezivanje sporo, većina klijentskih aplikacija uspostavlja konekciju prilikom startovanja, a raskida je prilikom prestanka rada. Ako aplikacija nekontrolisano prestane sa radom (sruši se), konekcija na serveru ostaje otvorena i resursi se nepotrebno troše. Pored toga, najveći broj korisnika stvarno rade sa aplikacijom samo 2-3% od ukupnog vremena konekcije na server, što 2-slojne aplikacije čini prilično neekonomičnim po pitanju potrošnje resursa.

Međuoperabilnost 2-slojnih sistema je limitirana zbog upotrebe uskladištenih procedura (*stored procedures*) za implementaciju poslovne logike. Uskladištene procedure se implementiraju na programskom jeziku konkretne baze podataka koja se koristi. To znači da, ako je potrebna međuoperabilnost sa bazom podataka nekog drugog tipa, aplikacija mora da bude ponovo napisana ili modifikovana. Pored toga, programski jezici baza podataka generalno nisu toliko moćni kao standardni programski jezici, a isto važi i za razvojna okruženja koja ne raspolazu alatima za testiranje, debugovanje, itd.

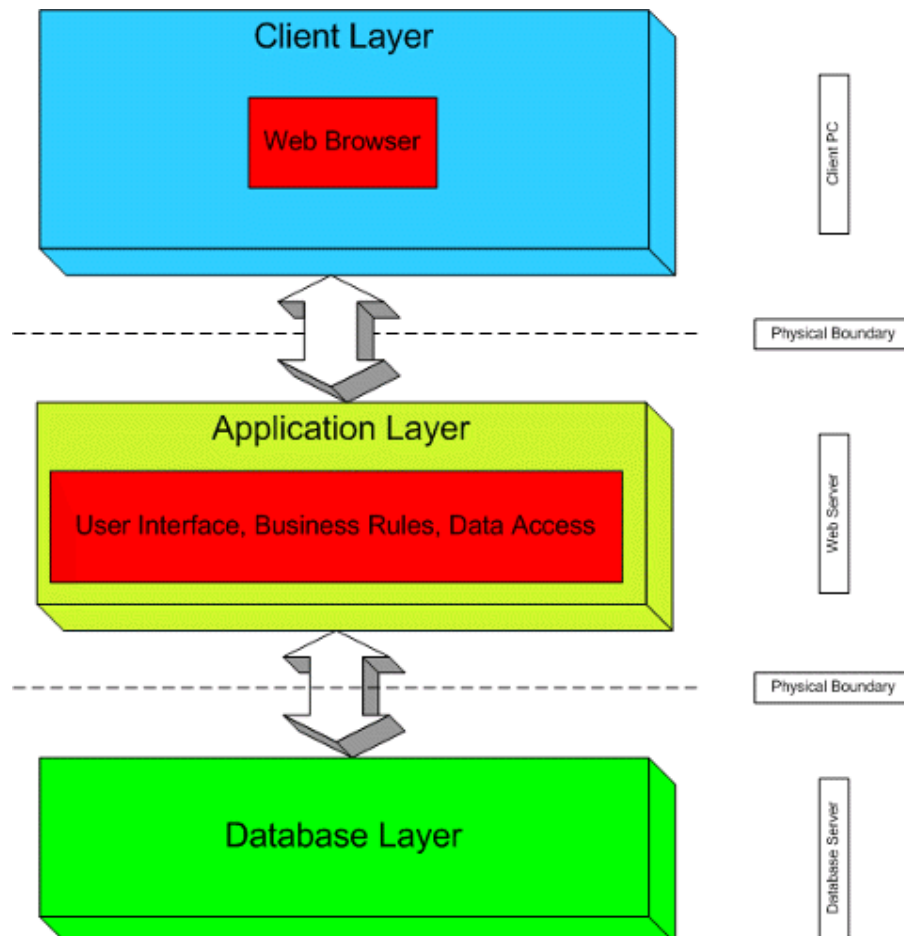
Administracija, konfiguracija i održavanje 2-slojnih arhitektura može biti otežano zbog činjenice da, pošto se aplikacija nalazi i izvršava na klijentu, svaka nova verzija mora da se

distribuirati i instalirati na sve klijente, koji često nisu konfigurirani na isti način ili im se konfiguracije i podešavanja menjaju bez znanja i uticaja administratora.

2-slojni klijent/server sistemi su intenzivno bili u upotrebi tokom 80-tih godina prošlog veka.

3-slojna arhitektura

3-slojna softverska arhitektura je nastala tokom 90-tih godina prošlog veka sa ciljem prevazilaženja ograničenja 2-slojne arhitekture. Treći sloj, koji se naziva srednji (*middle tier*), nalazi se između klijenta (korisničkog interfejsa) i sloja za čuvanje podataka (baze podataka). Ovaj srednji sloj sadrži procese koji implementiraju poslovnu logiku i poslovna pravila. Primenom naprednih tehnika kao što su, na primer, redovi za čekanje, 3-slojna arhitektura je u stanju da podrži više stotina korisnika. 3-slojna arhitektura se primenjuje za izgradnju sistema od kojih se očekuju vrhunske performanse, fleksibilnost, pojednostavljeno održavanje, mogućnost ponovnog korišćenja komponenta i skalabilnost. Složenost sistema se "krije" od korisnika. 3-slojna arhitektura je prikazana na slici 4.2.



Slika 4.2 – Troslojna arhitektura

Na vrhu 3-slojne distribuirane klijent/server arhitekture nalazi se klijentski sloj koji implemetira korisničke servise (unos i prezentacija podataka).

Srednji sloj obezbeđuje servise upravljanja procesima koji se dele između više aplikacija. Srednji sloj centralizuje procesnu logiku i čini administraciju i održavanje jednostavnijim zbog toga što se sve promene moraju izvršiti samo jedanput i samo na jednom mestu, i odmah postaju dostupne u celom sistemu.

Srednji sloj takođe kontroliše transakcije i asinhrono redove za čekanje, što omogućava pouzdano izvršavanje transakcija. U ovom sloju se čuva integritet distribuiranih baza podataka primenom dvofaznog procesa potvrde izvršenja (*two phase commit process*). Ovaj proces se primenjuje u distribuiranim okruženjima za koordinaciju distribuiranih transakcija i bazira se na principu da ili sve baze koje učestvuju u procesu potvrde izvršenja (*commit*) ili ga ponište (*rollback*). Proces se odvija u dve faze, *pripremne faze* i *faze izvršenja*, po čemu je i dobio naziv.

Postoji nekoliko naprednih tehnika u implementaciji 3-slojne arhitekture, i to su: nadzor obrade transakcija, serveri poruka, aplikacioni serveri i ORB (*Object Request Broker*).

Nadzor obrade transakcija. Ova tehnika implementira redove za čekanje, raspoređivanje obrade transakcija i servise za dodelu prioriteta. Klijent uspostavlja konekciju sa modulom za nadzor obrade transakcija koji se nalazi u srednjem sloju, a ne direktno sa bazom podataka. Modul stavlja transakciju u red za čekanje i od tog trenutka preuzima brigu o njoj sve dok se ne izvrši, oslobađajući na taj način klijenta od te obaveze. Postoje dve varijante ovakve arhitekture. U prvoj se primenjuju *middleware* proizvodi specijalizovanih proizvođača koji su u stanju da opsluže hiljade korisnika. U drugoj varijanti je logika za nadzor obrade transakcija ugrađena u bazu podataka, i ovom slučaju broj korisnika koji se može opslužiti iznosi oko sto. Primenom tehnike nadzora obrade transakcija sistem će biti u stanju da:

- Ažurira više različitih baza podataka izvršavanjem samo jedne transakcije
- Uspostavi konekciju sa više različitih izvora podataka, kao npr. fajlovima, nerelacionim bazama podataka ili mejnfrejmskim računarima
- Transakcijama dodeljuje prioritete
- Implementira robustniju bezbednost podataka.

Serveri poruka. Kod ove tehnike, u 3-slojnim arhitekturama porukama se dodeljuju prioriteti i one se obrađuju asinhrono. Svaka poruka ima zaglavlje koje sadrži podatak o prioritetu, adresu i identifikacioni broj. Server poruka uspostavlja konekciju sa relacionim bazama podataka i drugim izvorima podataka. Razlika između tehnike nadzora obrade transakcija i tehnike servera poruka je u tome što se tehnika servera poruka fokusira na "inteligentne poruke", dok se kod nadzora obrade transakcija sva "inteligencija" nalazi u samoj logici za nadzor, a transakcije se tretiraju kao "glupi" paketi sa podacima. Tehnika servera poruka je dobro rešenje za bežične infrastrukture.

Aplikacioni serveri. Ova tehnika podrazumeva alokaciju glavnog tela aplikacije na posvećeni i deljeni server. Na aplikacionom serveru se, dakle, ne izvršava kod koji implementira elemente grafičkog okruženja, već samo elementi poslovne logike, proračuna i pristupa podacima. Prednosti ove tehnike se ogledaju u tome što se na klijentu izvršava samo deo aplikacije koji se odnosi na prezentaciju podataka, što je po pitanju bezbednosti znatno povoljnije, zatim aplikacije su skalabilnije, a troškovi instaliranja i održavanja za jedan server su niži nego za više klijenata. Tehniku aplikacionih servera treba primenjivati kada su glavni zahtevi u sistemu bezbednost, skalabilnost i troškovi.

ORB (Object Request Broker). Ova tehnika podržava distribuirane objekte i međuoperativnost između različitih programskih jezika i platformi. Glavne tehnologije koje se primenjuju za izgradnju ovakvih arhitektura su:

- CORBA (*Common Object Request Broker Architecture*) predstavlja specifikaciju standardne ORB arhitekture, a može se smatrati i generalizacijom RPC tehnologije (*Remote Procedure Call*), u odnosu na koju donosi dosta poboljšanja. Klijent transparentno poziva metod serverskog objekta koji može da se nalazi na istom računaru ili bilo gde u mreži. ORB presreće taj poziv i njegova je obaveza da locira objekat koji može da obradi upućeni zahtev, da mu prosledi parametre, da pozove odgovarajući metod i da vrati rezultat obrade. Klijent ne mora da zna gde se objekat nalazi, u kom programskom jeziku je implementiran, na kom operativnom sistemu se izvršava, kao ni jedan drugi aspekt koji nije deo interfejsa objekta. U CORBA tehnologiji, sistemi se sastoje iz distribuiranih objekata, koji izlažu interfejse koji su opisani jezikom visokog nivoa, neutralnim u odnosu na arhitekturu i koji podržava abstrakciju svojstvenu objektno-orjentisanom dizajnu.

CORBA specifikacija je razvijena od strane OMG (*Object Management Group*), industrijske grupe koju čini više od 600 kompanija članica i kao takva predstavlja industrijski standard.

- COM/DCOM (*Component Object Model/Distributed Component Object Model*). Ovo su istovremeno i specifikacije i implementacije razvijene od strane Microsoft-a, a predstavljaju okvir za integrisanje komponentata. Podržavaju međuoperativnost i mogućnost ponovne upotrebe (*reusability*) distribuiranih objekata. Kreatori sistema u procesu razvoja mogu da "sklapaju" sistem od komponentata istog ili različitih proizvođača koje komuniciraju preko COM modela. Izgradnjom od već postojećih komponenti sistemi postaju lakši za održavanje i prilagodljiviji.

COM definiše Aplikacioni Programski Interfejs (*API*) pomoću koga se mogu kreirati komponente koje će se koristiti za integraciju specifičnih aplikacija ili za interakciju raznih komponenti. Komponente mogu biti razvijene u različitim programskim jezicima ali, da bi bile u stanju da ostvare interakciju, moraju poštovati binarnu strukturu koju je propisao Microsoft.

DCOM je proširenje COM modela koji omogućava interakciju komponentata u mreži (a ne samo u domenu jednog računara). U COM modelu, procesi se izvršavaju na istom računaru, ali u različitim adresnim (memorijskim) prostorima. DCOM omogućava da se procesi rasporede na više lokacija u mreži, na više različitih platformi.

- COM+ je takođe Microsoft-ova tehnologija koja inkorporira servise za obradu transakcija i redove za čekanje u COM model.

3-slojne arhitekture se koriste u komercijalnim i vojnim distribuiranim klijent/server okruženjima u kojima se zahtevaju deljeni resursi kao što su heterogene baze podataka i poslovna pravila.

3-slojne arhitekture olakšavaju razvoj softvera zato što svaki sloj može da se razvija, testira i izvršava na posebnoj platformi. Slojevi mogu da se razvijaju u različitim programskim jezicima, što takođe predstavlja olakšavajuću okolnost, jer je moguće odabrati programski jezik koji najbolje odgovara postavljenom razvojnom zadatku.

Migracija postojećih (*legacy*) sistema na 3-slojnu arhitekturu može se izvršiti bez visokih troškova i bez mnogo rizika.

Razvoj 3-slojnih arhitektura je složen posao. Programski alati i razvojna okruženja koja omogućavaju dizajniranje ovakvih sistema još uvek ne raspolažu svim servisima potrebnim za podršku distribuiranih okruženja.

Potencijalni problem pri dizajniranju 3-slojnih arhitektura leži u činjenici da način razdvajanja korisničkog interfejsa, logike upravljanja procesima i logike koja se bavi podacima nije uvek očigledan. Neki elementi logike upravljanja procesima mogu da se pojave u sva tri sloja. Raspoređivanje funkcija po slojevima treba da se vrši po kriterijumima kao što su:

- Lakoća razvoja i testiranja
- Sposobnost administriranja
- Skalabilnost servera
- Performanse (odnosi se na procesiranje i mrežno opterećenje) .

3-slojne arhitekture se uspešno primenjuju od ranih 90-tih godina prošlog veka za izgradnju najrazličitijih sistema u komercijalnom i vojnom domenu, npr. u Ministarstvu odbrane SAD. U američkom ratnom vazduhoplovstvu razvijen je automatizovani sistem za planiranje i izvršavanje vazdušnog ratnog plana (*Theater Battle Management Core System*), koji se u potpunosti bazira na 3-slojnoj arhitekturi.

N-slojna arhitektura

Ponekad se srednji sloj 3-slojne aplikacije deli na dve ili više zasebnih celina sa različitim funkcijama koje obavljaju. U tom slučaju radi se o *višeslojnoj* ili *n-slojnoj* arhitekturi (slika 4.3). Ovakva arhitektura se često primenjuje za razvoj Internet aplikacija zbog toga što se u njima klijenti najčešće implementiraju u običnom HTML-u, a serverske aplikacije u programskim jezicima C++ ili Java, pa je konceptualna razlika između klijenta i servera tolika da se oni teško mogu direktno povezati. Zato se uvodi jedan ili više međuslojeva (*business layer*). Ovaj međusloj prima zahteve od Internet klijenata i generiše HTML stranice na osnovu rezultata njihove obrade. On pruža i dodatnu izolaciju između grafičkih i logičkih elemenata aplikacije.

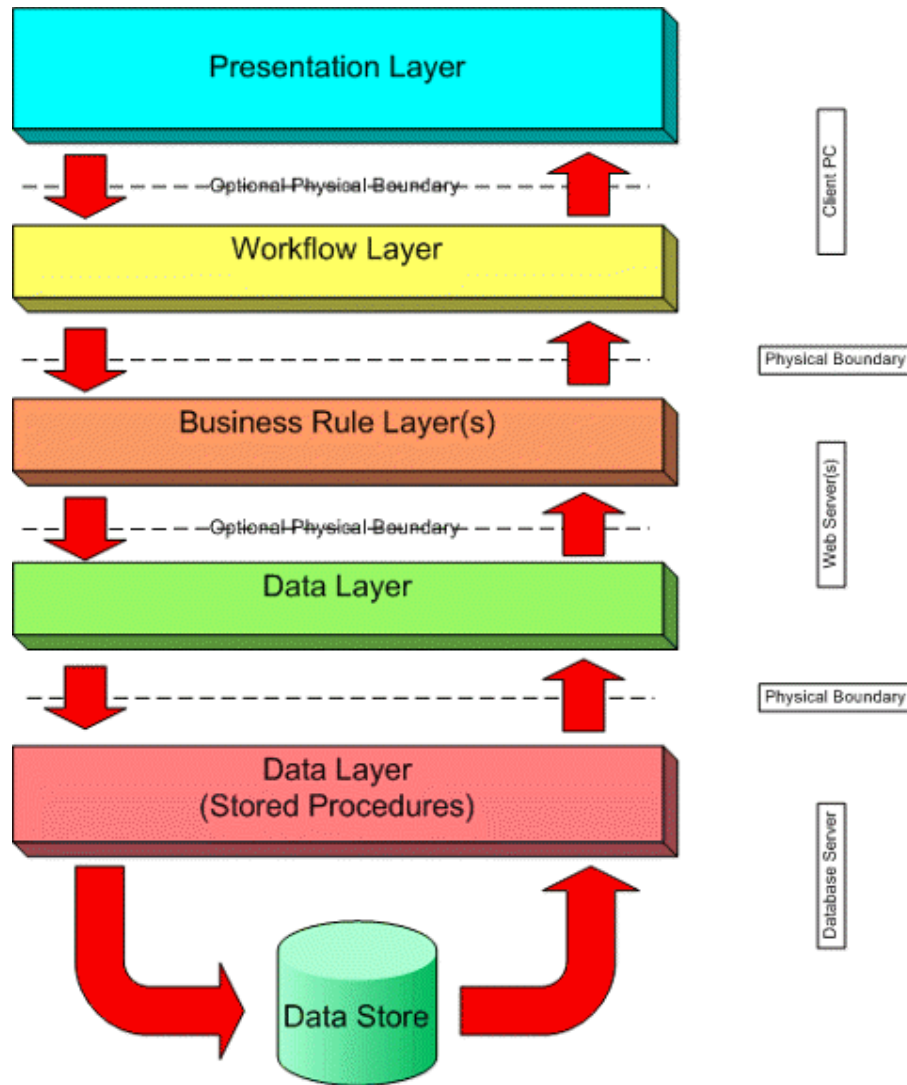
Treba spomenuti i to, da se u poslednje vreme u distribuirana okruženja uvode i mejnfrejm računari koji rade kao serveri i obezbeđuju potreban prostor za čuvanje podataka uz visok nivo bezbednosti.

Sledi pregled svakog od pojedinačnih slojeva.

Sloj podataka

Sloj podataka se obično može podeliti na dva posebna sloja.

Prvi sloj se sastoji iz seta uskladištenih procedura (*stored procedures*) koje su implementirane direktno u bazi podataka. Ove procedure se izvršavaju na serveru baze podataka i samo obezbeđuju osnovne podatke. One ne samo da su prekompajlirane i optimizovane, već se mogu zasebno testirati i analizirati pomoću specijalizovanih alata.



Slika 4.3 – N-slojna arhitektura

Drugi sloj se sastoji iz seta klasa koje pozivaju uskladištene procedure. Najčešće se jednoj grupi uskladištenih procedura pridružuje jedna klasa koja obrađuje sve SELECT, INSERT, UPDATE i DELETE operacije nad bazom podataka. Kласe se dizajniraju u objektno-orijentisanom maniru i pridružuju se jednoj tabeli ili grupi povezanih tabela. Ove kласe predstavljaju "štit" za aplikacione podatke. Svi zahtevi za podacima moraju da prođu kroz ovaj sloj i sva pitanja koja se tiču konkurentnosti se moraju rešavati upravo ovde. Na taj način se obezbeđuje integritet podataka, zato što se njihova promena ne može vršiti ni iz jednog drugog izvora, ni na koji način. Ako se baza podataka promeni na bilo koji način, te promene će se reflektovati samo na ovaj sloj, dok ostale komponente aplikacije ostaju nepromenjene. Ovo značajno pojednostavljuje održavanje.

Sloj poslovnih pravila (business rules)

Ovaj sloj, kao što sam naziv govori, implementira poslovna pravila. Razvija se na osnovu dokumenata koji opisuju entitete, njihove veze, aktivnosti, način na koji kolaboriraju, podatke i sve druge elemente poslovnog okruženja koje se modelira. Na osnovu tih

dokumenata nastaju slučajevi korišćenja (*use-cases*), koji detaljno opisuju poslovno okruženje i na osnovu kojih nastaju klase i sve potrebne vrste dijagrama klasa. To je gradivni materijal od koga nastaje sloj poslovnih pravila.

Klase iz sloja poslovnih pravila ne pristupaju direktno podacima, niti se bave prezentacijom podataka. U ovom sloju se brine samo o kompleksnosti samog poslovnog okruženja. Ako se to okruženje promeni, te promene će se projektovati samo na ovaj sloj i samo on će pretrpeti potrebne modifikacije.

Sloj toka podataka (workflow layer)

Ovo je jedan od opcionih slojeva i bavi se tokom podataka ka sistemu i od sistema. Može i ne mora da bude u direktnoj interakciji sa korisničkim interfejsom, ali uvek radi sa eksternim izvorima podataka.

Na primer, ako se šalju ili primaju poruke iz reda za čekanje, ako se koriste web servisi za prikupljanje dodatnih informacija, ako se razmenjuju informacije sa drugim sistemima, programski kod koji to radi bi se nalazio u ovom sloju. U ovom sloju bi se, recimo, svi podaci za prezentovanje najpre mogli konvertovati u XML, što bi umnogome proširilo izbor kandidata u smislu izbora tehnologije ili programskog jezika za implementaciju prezentacionog sloja.

Prezentacioni sloj

Ovaj sloj se bavi svim aktivnostima vezanim za prezentaciju podataka. U opštem slučaju, ovaj sloj ne sadrži samo elemente grafičkog okruženja, već i pomoćne klase za prezentaciju podataka.

Idealno, web ili Windows forme koji se koriste za vizuelnu reprezentaciju samo uspostavljaju transparentnu interakciju sa klasama prezentacionog nivoa koje vrše sve potrebne translacije između sloja poslovnih pravila/sloja toka podataka i samih formi. To znači da se promene u vidljivom delu aplikacije mogu izvršiti brzo i lako.

5.

Komparativna analiza klasične desktop aplikacije i višeslojne klijent/server arhitekture

U razvojnim okruženjima kao što su *Microsoft Visual Basic*, *Microsoft Visual C++*, *Microsoft Access*, *Borland JBuilder* i sl. mogu se razviti klijentske desktop aplikacije koje rade sa bazama podataka, kao *front-end*. Da bi korisnici mogli da ih koriste, takve klijentske aplikacije se najpre moraju distribuirati i instalirati na sve računare u lokalnoj mreži. Tokom rada, *front-end* aplikacije kroz mrežu pristupaju serveru baze podataka. U programu *Microsoft Access* moguće je razviti aplikaciju koja sadrži i klijentsku i serversku komponentu i koja obe čuva u jednom fajlu. Ovakva aplikacija je najčešće predviđena za izvršavanje na pojedinačnom i izolovanom računaru. U ovom poglavlju pod pojmom desktop aplikacije podrazumeva se jedna od upravo spomenutih vrsta aplikacija, u zavisnosti od konteksta u kome se analizira.

Komparativna analiza klasične desktop aplikacije i višeslojne klijent/server web aplikacije (web je najčešća varijanta ove vrste aplikacija) mora se izvršiti po određenim kriterijuma u odnosu na koje se mogu porediti obe vrste aplikacija koje se inače suštinski razlikuju, pre svega po arhitekturi. Kriterijumi po kojima će poređenje biti izvršeno su sledeći: *bezbednost i integritet podataka*, *stepen složenosti razvoja*, *raspoloživost*, *jasnoća/preglednost*, *upotrebljivost* i *brzina reagovanja*.

Bezbednost i integritet podataka

Ako desktop aplikaciju baze podataka koristi nekoliko korisnika na lokalnoj mreži i baza je relativno mala, aplikacija ima dobre performanse, zahteva minimalno održavanje koje se vrši jednostavno i relativno je stabilna.

Uporedo sa porastom broja korisnika i količine podataka u bazi, raste i obim transakcija. Iznenada, performanse osetno opadaju. Saobraćaj na mreži se uvećava, a sve češće se javljaju problemi sa integritetom podataka i indeksa. Ako radne stanice ili mreža otkažu, postoji mogućnost da baza ostane u nekonzistentnom stanju. Dodavanjem novih korisnika može se pojaviti i problem bezbednosti. Vreme pristupa bazi podataka postaje duže, pa korisnici ne mogu da obavljaju svoje poslove na vreme.

Kada aplikacija pristupa bazi, podaci kojima se pristupa moraju da se zaključaju, upravo zbog zaštite integriteta i održavanja konzistentnosti. U situaciji u kojoj bazi pristupa samo nekolicina korisnika, verovatnoća da će biti zahtevan istovremeni pristup od strane više njih je mala, ali sa povećanjem broja korisnika raste i mogućnost da do takve situacije ipak dođe, što bi na kraju rezultovalo usporavanjem pristupa podacima, možda i do nivoa neupotrebljivosti aplikacije. Saobraćaj na mreži bi se povećao zbog toga što bi korisnici bili prinuđeni da više puta pokušaju da izvrše neku operaciju.

U situaciji opisanoj u prethodnim paragrafima mora se razmotriti prelazak na višeslojnu klijent/server arhitekturu koja će biti u stanju da zadovolji novonastale potrebe.

Bezbednost podataka se mora sagledati i sa stanovišta okruženja u kome se aplikacija izvršava. Ako se radi o klasičnoj desktop aplikaciji kod koje se i baza podataka i sama aplikacija izvršavaju na istom računaru (kao što je slučaj sa aplikacijama kompletno razvijenim u razvojnom okruženju *Microsoft Access*), onda se bezbednost podataka ostvaruje kroz mehanizme za kontrolu pristupa resursima tog računara. Ukoliko se primenjuju kvalitetni mehanizmi zaštite, može se reći da je bezbednost u tom slučaju na prilično visokom nivou zbog same činjenice da postoji relativno malo potencijalnih izvora opasnosti. Ako se pak radi o klijent/server aplikaciji, situacija postaje znatno složenija zbog toga što se menja okruženje u kome se aplikacija izvršava. Sada se klijentski i serverski deo aplikacije izvršavaju na različitim računarima koji za međusobnu komunikaciju koriste Intranet, a ponekad i Internet. O bilo kom od ova dva okruženja da se radi, broj potencijalnih izvora opasnosti rapidno raste, pa se i koncepcija zaštite bezbednosti mora učiniti znatno složenijom i sofisticiranijom. Primenom kvalitetnih softverskih i hardverskih metoda zaštite moguće je izgraditi skoro potpuno bezbedan sistem, ali to znatno usložnjava razvoj i povećava troškove.

Web aplikacije koje logiku za interakciju sa korisnikom implementiraju u formi Java apleta imaju krajnje ograničen pristup sistemskim resursima na klijentskom računaru. Naime, iz bezbednosnih razloga Java apleti se izvršavaju u tzv. *sandbox-u*, režimu u kome se apletu ograničava pristup svim onim resursima računara preko kojih bi mogla da se naruši bezbednost sistema, privatnost korisnika i slično. U poređenju sa desktop aplikacijom, ovo ponekad znatno degradira funkcionalnost, pa i upotrebljivost web aplikacije.

Stepen složenosti razvoja

Za razvoj klasičnih desktop aplikacija postoji mnogo kvalitetnih razvojnih okruženja koja su praktično dovedena do perfekcije po pitanju funkcionalnosti, komfora u radu, izbora ponuđenih opcija, itd. Sva raspoložu kvalitetnim grafički orijentisanim alatima za kreiranje svih elemenata aplikacije koji će korisniku omogućiti intuitivan i jednostavan rad. Debugovanje i testiranje se vrše u samom okruženju, uz pomoć za to predviđenih alata.

Web aplikacije često moraju da se razvijaju u više verzija, od kojih je svaka prilagođena određenom web browseru, a ponekad skripting jezici ne mogu da se izvršavaju na klijentu ako je ta opcija isključena u web browseru. To zahteva angažovanje korisnika koji mora da dozvoli izvršavanje skripting jezika, pa neke opcije aplikacije nisu odmah dostupne ili aktivne.

Kompleksnost višeslojnih klijent/server aplikacija čini njihov razvoj težim. Činjenica da se softver razvija za više slojeva produžava vreme razvoja i usložnjava testiranje. Ove otežavajuće okolnosti se donekle mogu kompenzovati primenom standardizovanih razvojnih alata i modernih razvojnih okruženja.

Raspoloživost

Na prvi pogled izgleda kao da je desktop aplikacija u prednosti po pitanju raspoloživosti, jer se može koristiti čak i kada je web server nedostupan ili ne radi, odnosno za njeno funkcionisanje uopšte nije potreban Internet/Intranet. Ali, desktop aplikacija se ne može koristiti bez računara na kome je instalirana. Čak i kada je aplikacija instalirana na

prenosivom računaru, to je samo delimično rešenje zato što se raspoloživost aplikacije dovodi u direktnu vezu sa fizičkim prisustvom računara.

Web aplikacija je raspoloživa na svim mestima gde je dostupan web browser, što je ipak za nijansu stavlja ispred desktop aplikacije po pitanju raspoloživosti.

Jasnoća/preglednost

Korisnički interfejs u klijentskim aplikacijama, pogotovo ako se one implementiraju u formi web stranica (HTML, DHTML ili u nekoj sličnoj tehnologiji), po svojoj prirodi ne može biti tako bogat i raznovrstan kao GUI kod klasičnih desktop Windows aplikacija. Ako se poredе po tom kriterijumu, web aplikacije su inferiorne u odnosu na klasične desktop Windows aplikacije.

Upotrebljivost

U stručnoj literaturi se često polemise da li je važnije da je aplikacija upotrebljiva ili raspoloživa i jasna. I pored podeljenih mišljenja, prevladava stav da je i najfunkcionalnija aplikacija potpuno beskorisna ako nije raspoloživa ili ako je komplikovana za korišćenje. Dakle, opšti zaključak bi bio da, od svih kriterijuma, po važnosti upotrebljivost zauzima poslednje mesto.

Postavlja se pitanje da li je upotrebljivost faktor koji odvaja desktop od web aplikacija. Odgovor je uglavnom da, zato što još uvek ne postoji web aplikacija koja radi sve što su u stanju da urade, na primer, Microsoft Word ili Excel. Ali, to je krajnje jednostran odgovor zato što postoje web aplikacije koje jednostavno ne mogu da se repliciraju u desktop varijantu.

Na primer, pretraživač kao što je Google ne može da se implementira u desktop varijanti. Desktop računari nemaju dovoljno snažne resurse za pretragu web-a, indeksiranje i izvršavanje kompleksnih upita.

Ipak, možda bi bilo bolje porediti Internet aplikacije kojima se pristupa preko web servera i Internet aplikacije kojima se pristupa preko desktop aplikacije, kao što su Google Earth u desktop varijanti i Google Maps kome se pristupa preko web browsera. Obe aplikacije prikazuju mape, satelitske fotografije i slično, ali izgleda kao da je desktop aplikacija snažnija i elegantnija.

Bez obzira na sve spomenuto, u praksi se mnogo više koriste aplikacije kojima se pristupa preko web browsera.

Brzina reagovanja

Ovo je veoma važan kriterijum čiji je uticaj možda od presudnog značaja za donošenje odluke desktop ili klijent/server aplikacija. Microsoft, na primer, savetuje programere da izbegavaju "reakciju sa zadržkom" u svojim programima zato što to može dovesti korisnika u iskušenje da prekine sa radom ili čak da resetuje računar.

U ovom trenutku, većina desktop aplikacija ima veću brzinu reagovanja od svojih web ekvivalenata. To je potpuno logično ako se u obzir uzme činjenica da je brzina protoka podataka unutar računara za nekoliko redova veličine veća od brzine protoka podataka na mreži između klijenta i servera. Takođe, desktop aplikacije se razvijaju pomoću efikasnih programskih jezika i isporučuju u kompajliranoj formi, dok se većina web aplikacija razvija pomoću interpretiranih skripting jezika (ovo se odnosi na prezentacioni sloj i, ponekad, na srednji sloj), što ih čini znatno inertnijim na korisničke akcije.

Možda je pravo pitanje koje treba postaviti da li je zaista važna apsolutna brzina reagovanja ili da li se brzina reagovanja može predvideti. Korisnik će se radije odlučiti za aplikaciju koja uvek reaguje za, recimo, 1/10 s (desktop) nego za aplikaciju koja ponekad odreaguje za 1/10 s, a ponekad za 2 s (web). Dakle, varijacija, nepredvidivost u brzini reagovanja web aplikacija je ono što daje prednost desktop aplikacijama. Ipak, nepredvidiva brzina reagovanja je nešto što je neizbežno i sa čime se mora računati kada su web aplikacije u pitanju zbog toga što je mreža za razliku od interne arhitekture računara krajnje nepredvidivo okruženje u kome se često dešavaju nepredviđeni događaji.

Od značaja za ovu analizu može biti i činjenica da su moderne web aplikacije mešavina najmanje dve tehnologije, web-a u smislu hiperteksta za prezentaciju i web-a u smislu linka između klijenta i servera. Ovaj koncept, iako izuzetno moćan, ipak može prouzrokovati degradaciju performansi i funkcionalnosti web aplikacije.

Može se zaključiti da su, zbog predvidivosti brzine njihovog reagovanja, desktop aplikacije u prednosti.

Zaključak

I klasične desktop aplikacije i moderne višeslojne klijent/server arhitekture imaju svoje prednosti i mane. Možda ih i ne treba direktno porediti zbog toga što u poslovnim okruženjima u kojima se koriste postoji mnogo različitih poslova od kojih neke bolje obavlja aplikacija sa klasičnom arhitekturom, a neke moderna klijent/server aplikacija. Činjenica da nam danas na raspolaganju stoji tako bogat izbor daje nam komfor da izaberemo vrstu aplikacije koja najbolje odgovara postavljenim zahtevima.

Klasične desktop aplikacije i moderne klijent/server arhitekture ne treba posmatrati kao suprotstavljene strane koje teže ka tome da eliminišu jedna drugu jer, kao što je prikazano u poglavlju o komparativnoj analizi, za obe vrste aplikacija se mogu naći primene u kojima su superiorne i u tim situacijama ne treba da postoji dilema desktop ili klijent/server.

Dakle, desktop i klijent/server aplikacije će nastaviti da koegzistiraju i u budućnosti. I jedne i druge će se uporedo razvijati i svaka od njih će imati oblasti svoje primene u kojoj će biti superiorna u odnosu na onu drugu. Možda će početi da se razvijaju i hibridne desktop-web aplikacije koje će nuditi približno istu funkcionalnost i korisnicima koji ih izvršavaju na svom desktop/laptop računaru i korisnicima koji im pristupaju kroz web browser.

Literatura

1. Thomas M. Connolly, Carolyn E. Begg: **DATABASE SYSTEMS, Third Edition**, ADDISON-WESLEY, 2002.
2. **N-Tier Application Development with Microsoft.NET**,
<http://www.microsoft.com/belux/msdn/nl/community/columns/hyatt/ntier1.msp>
3. **Client/Server Software Architectures - An Overview**,
<http://www.sei.cmu.edu/str/descriptions/clientserver.html>
4. **Two Tier Software Architectures**,
<http://www.sei.cmu.edu/str/descriptions/twotier.html>
5. **Three Tier Software Architectures**,
http://www.sei.cmu.edu/str/descriptions/threetier_body.html
6. **3-Tier Architecture**,
<http://members.tripod.com/ChannuKambalyal/NTierArchitecture.pdf>
7. **TEA client/server whitepaper**,
http://www.commongoals.com/docs/TEAClientServer_Whitepaper.pdf?CFID=7622087&CFTOKEN=74530658
8. **Rich Internet application**,
http://en.wikipedia.org/wiki/Rich_Internet_application
9. **Desktop vs. Web applications**,
<http://answers.google.com/answers/threadview?id=747326>