1. Uvod u baze podataka

U ovom udžbeniku bavimo se problematikom trajnog pohranjivanja većih količina podataka u vanjskoj memoriji računala. To je iznimno važna problematika: naime unatoč svom nazivu, računala zapravo rijetko služe za *računanje*, a znatno češće za *spremanje i pretraživanje* podataka.

Mogućnost trajnog pohranjivanja podataka u računalima postoji gotovo jednako dugo koliko i sama računala te je podržana u svim programskim jezicima. Primjerice, program razvijen u jeziku COBOL ili C može stvoriti datoteku na disku i u nju upisati podatke ili otvoriti postojeću datoteku i iz nje pročitati podatke. Služeći se klasičnim programskim jezicima u principu je moguće stvoriti trajne kolekcije podataka na disku i izgraditi aplikacije koje rabe takve podatke.

Ipak, kad govorimo o bazama podataka, tada mislimo na višu razinu rada s podacima od one koju podržavaju klasični programski jezici. Ustvari mislimo na tehnologiju koja je nastala s namjerom da ukloni slabosti tradicionalne "automatske obrade podataka" iz 60-ih i 70-ih godina 20. stoljeća. Ta tehnologija osigurala je veću produktivnost, kvalitetu i pouzdanost u razvoju aplikacija koje se temelje na pohranjivanju i pretraživanju podataka u računalu.

1.1. Osnovni pojmovi

Osnovna je ideja tehnologije baza podataka u tome da pojedina aplikacija ne stvara vlastite datoteke na disku. Umjesto toga, sve aplikacije rabe zajedničku i povezanu kolekciju podataka. Također, aplikacija ne pristupa izravno podacima na disku. Umjesto toga, ona barata s podacima na posredan način, služeći se uslugama specijaliziranog softvera koji je zadužen da se brine za zajedničku kolekciju. Spomenuta zajednička kolekcija podataka naziva se baza podataka, a specijalizirani softver koji posreduje između aplikacija i podataka naziva se sustav za upravljanje bazom podataka. U nastavku ćemo najprije pokušati preciznije definirati ova dva ključna pojma, a zatim ćemo objasniti i druge pojmove koji su povezani s njima.

1.1.1. Baza podataka i sustav za upravljanje bazom podataka (DBMS)

Baza podataka je skup međusobno povezanih podataka, pohranjenih u vanjskoj memoriji računala. Podaci su istovremeno dostupni raznim korisnicima i aplikacijskim

programima. Upisivanje, promjena, brisanje i čitanje podataka obavlja se posredstvom posebnog softvera, tzv. *sustava za upravljanje bazom podataka* (DBMS-a). Korisnici i aplikacije pritom ne moraju poznavati detalje fizičkog prikaza podataka, već se referenciraju na neku idealiziranu logičku strukturu baze.

Sustav za upravljanje bazom podataka (*Data Base Management System* – DBMS) je poslužitelj (server) baze podataka. On oblikuje fizički prikaz baze u skladu s traženom logičkom strukturom. Također, on obavlja u ime klijenata sve operacije s podacima. Dalje, on je u stanju podržati razne baze, od kojih svaka može imati svoju logičku strukturu, no u skladu s istim *modelom*. Isto tako, brine se za sigurnost podataka, te automatizira administrativne poslove s bazom.

Slično kao i operacijski sustav, DBMS spada u temeljni softver koji većina korisnika i organizacija ne razvija samostalno, već ga kupuju zajedno s računalom. Danas postoji svega nekoliko važnih i široko zastupljenih DBMS-a.

- **DB2.** Proizvod tvrtke IBM, namijenjen prije svega velikim *mainframe*-računalima.
- **Oracle.** Proizvod istoimene tvrtke, pokriva gotovo sve računalne platforme, npr. UNIX, Linux i MS Windows.
- **MS SQL Server.** Microsoftov proizvod, namijenjen poslužiteljskim računalima s operacijskim sustavima MS Windows.
- **MySQL.** Besplatni proizvod tvrtke MySQL AB, popularan na raznim platformama, prije svega kao podrška *web*-aplikacijama.

Svi ovi proizvodi uz sam DBMS uključuju u sebi i dodatne alate za razvoj aplikacija, administriranje baze i slično.

1.1.2. Modeli za logičku strukturu baze podataka

Model podataka je skup pravila koja određuju kako sve može izgledati logička struktura baze podataka. Model čini osnovu za oblikovanje i implementiranje baze. Točnije rečeno, podaci u bazi moraju biti logički organizirani u skladu s onim modelom koji podržava odabrani DBMS.

Dosadašnji DBMS-i obično su podržavali neki od sljedećih modela:

- **Relacijski model.** Zasnovan je na matematičkom pojmu *relacije*. I podaci i veze među podacima prikazuju se tablicama koje se sastoje od redaka i stupaca.
- **Mrežni model.** Baza je predočena mrežom koja se sastoji od čvorova i usmjerenih *lukova*. Čvorovi predstavljaju tipove zapisa (slogova podataka), a lukovi definiraju veze među tipovima zapisa.
- **Hijerarhijski model.** Poseban slučaj mrežnog modela. Baza je predočena jednim *stablom* (hijerarhijom) ili skupom stabala. Svako stablo sastoji se od čvo-

rova i veza "nadređeni-podređeni" između čvorova. Čvorovi su tipovi zapisa, a odnos "nadređeni-podređeni" izražava hijerarhijske veze među tipovima zapisa.

- **Objektni model.** Inspiriran je objektno orijentiranim programskim jezicima. Baza je predočena kao skup trajno pohranjenih *objekata* koji se sastoje od svojih internih "atributa" (podataka) i "metoda" (operacija) za rukovanje tim podacima. Svaki objekt pripada nekoj klasi. Između klasa se uspostavljaju veze nasljeđivanja, agregacije te druge vrste veza.

Hijerarhijski i mrežni model bili su u uporabi u 60-im i 70-im godinama 20. stoljeća. Od 80-ih godina pa sve do danas prevladava relacijski model. Očekivani prijelaz na objektni model za sada se nije dogodio, tako da današnje baze podataka uglavnom i dalje možemo poistovjetiti s relacijskim bazama. Svi prije spomenuti poznati DBMS-i koji su danas u širokoj uporabi podržavaju isključivo relacijski model.

1.1.3. Ciljevi koji se nastoje postići uporabom baza podataka

Spomenuli smo da baze podataka predstavljaju višu razinu rada s podacima u odnosu na klasične programske jezike. Ta viša razina rada očituje se u tome što tehnologija baza podataka nastoji (i u velikoj mjeri uspijeva) ispuniti sljedeće ciljeve:

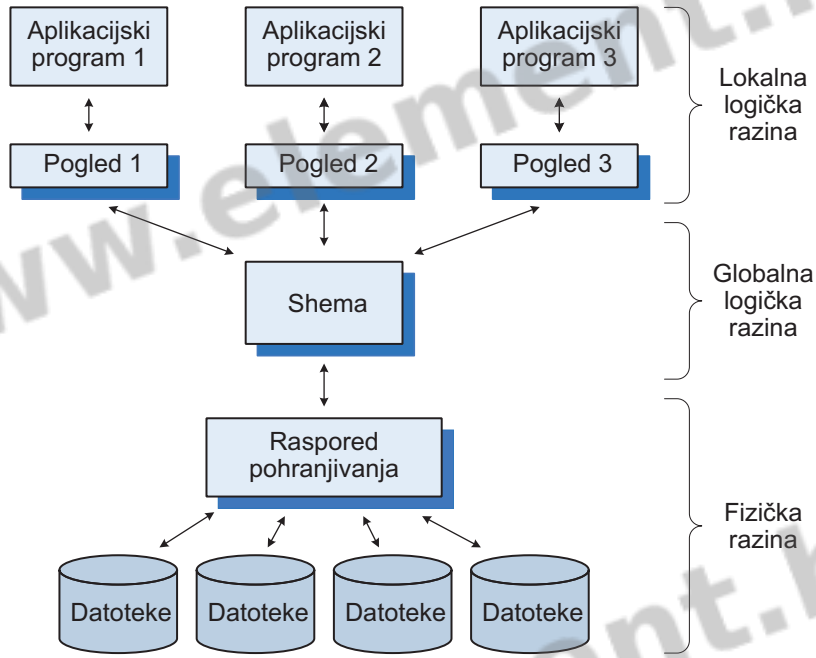
- **Fizička nezavisnost podataka.** Razdvaja se logička definicija baze od njezine stvarne fizičke građe. Znači, ako se fizička građa promijeni (npr., podaci se prepišu u druge datoteke na drugim diskovima), to neće zahtijevati promjene u postojećim aplikacijama.
- **Logička nezavisnost podataka.** Razdvaja se globalna logička definicija cijele baze podataka od lokalne logičke definicije za jednu aplikaciju. Znači, ako se globalna logička definicija promijeni (npr., uvede se novi zapis ili veza), to neće zahtijevati promjene u postojećim aplikacijama. Lokalna logička definicija obično se svodi na izdvajanje samo nekih elemenata iz globalne definicije, uz neke jednostavne transformacije tih elemenata.
- **Fleksibilnost pristupa podacima.** U starijim mrežnim i hijerarhijskim bazama načini pristupanja podacima bili su unaprijed definirani, dakle korisnik je mogao pretraživati podatke jedino onim redoslijedom koji je bio predviđen u vrijeme oblikovanja i implementiranja baze. Danas se podrazumijeva da korisnik može slobodno prebirati po podacima te po svom nahođenju uspostavljati veze među podacima. Ovom zahtjevu zaista udovoljavaju jedino relacijske baze.
- **Istovremeni pristup do podataka.** Baza mora omogućiti da veći broj korisnika istovremeno rabi iste podatke. Pritom ti korisnici ne smiju ometati jedan drugoga te svaki od njih treba imati dojam da sam radi s bazom.
- **Čuvanje integriteta.** Nastoji se automatski sačuvati korektnost i konzistencija podataka, i to u situaciji kad postoje pogreške u aplikacijama te konfliktne istovremene aktivnosti korisnika.

- **Mogućnost oporavka nakon kvara.** Mora postojati pouzdana zaštita baze u slučaju kvara hardvera ili pogrešaka u radu softvera.
- **Zaštita od neovlaštene uporabe.** Mora postojati mogućnost da se korisnicima ograniče prava uporabe baze, dakle da se svakom korisniku reguliraju ovlaštenja što on smije, a što ne smije raditi s podacima.
- **Zadovoljavajuća brzina pristupa.** Operacije s podacima moraju se odvijati dovoljno brzo, u skladu s potrebama određene aplikacije. Na brzinu pristupa može se utjecati odabirom pogodnih fizičkih struktura podataka te izborom pogodnih algoritama za pretraživanje.
- **Mogućnost podešavanja i kontrole.** Velika baza zahtijeva stalnu brigu: praćenje performansi, mijenjanje parametara u fizičkoj građi, rutinsko pohranjivanje rezervnih kopija podataka, reguliranje ovlaštenja korisnika. Također, svrha baze se s vremenom mijenja, pa povremeno treba podesiti i logičku strukturu. Ovakvi poslovi moraju se obavljati centralizirano. Odgovorna osoba zove se *administrator baze podataka*. Administratoru trebaju stajati na raspolaganju razni alati i pomagala.

1.1.4. Arhitektura baze podataka

Arhitektura baze podataka sastoji se od tri "sloja" i sučelja među slojevima, kao što je prikazano na slici 1.1. Riječ je o tri razine apstrakcije.

- **Fizička razina** odnosi se na fizički prikaz i raspored podataka na jedinicama vanjske memorije. To je aspekt koji vide samo sistemski programeri (oni koji su razvili DBMS). Sama fizička razina može se dalje podijeliti na više podrazina apstrakcije, od sasvim konkretnih staza i cilindara na disku, do već donekle apstraktnih pojmova datoteke i zapisa (sloga) kakve susrećemo u klasičnim programskim jezicima. *Raspored pohranjivanja* opisuje kako se elementi logičke definicije baze preslikavaju na fizičke uređaje.
- **Globalna logička razina** odnosi se na logičku strukturu cijele baze. To je aspekt koji vidi projektant baze odnosno njezin administrator. Opis globalne logičke definicije naziva se *shema* (također engl. *schema*). Shema je tekst ili dijagram koji definira logičku strukturu baze, i u skladu je sa zadanim modelom. Dakle, imenuju se i definiraju svi tipovi podataka i veze među tim tipovima, u skladu s pravilima rabljenog modela. Također, shema može uvesti i ograničenja kojima se čuva integritet podataka.
- **Lokalna logička razina** odnosi se na logičku predodžbu o dijelu baze koji rabi pojedina aplikacija. To je aspekt koji vidi korisnik ili aplikacijski programer. Opis jedne lokalne logičke definicije zove se *pogled* (*view*) ili *podshema*. To je tekst ili dijagram kojim se imenuju i definiraju svi lokalni tipovi podataka i veze među tim tipovima, opet u skladu s pravilima rabljenog modela. Također, pogled u svojoj konačnoj realizaciji zadaje i način na koji se iz globalnih podataka i veza izvode lokalni.



Slika 1.1. Arhitektura baze podataka

Primijetimo da se fizička neovisnost podataka spomenuta u odjeljku 1.1.3 ustvari postiže time što se pravi razlika između fizičke i globalne logičke razine, dok se logička neovisnost postiže razlikovanjem lokalne logičke razine i globalne logičke razine. Na taj način, opisana troslojna arhitektura omogućuje ispunjavanje dvaju najvažnijih ciljeva koji se nastoje postići uporabom baza podataka.

Za stvaranje baze podataka potrebno je zadati samo shemu i poglede. DBMS tada automatski generira potrebni raspored pohranjivanja i fizičku bazu. Projektant odnosno administrator može samo donekle utjecati na fizičku građu baze, podešavanjem njemu dostupnih parametara.

Programi i korisnici ne pristupaju izravno fizičkoj bazi, već dobivaju ili pohranjuju podatke posredstvom DBMS-a. Komunikacija programa odnosno korisnika s DBMS-om obavlja se na lokalnoj logičkoj razini. To znači da DBMS na transparentan način prevodi korisničke zahtjeve za podacima s lokalne logičke razine na globalnu logičku razinu, a zatim ih dalje realizira kao ekvivalentne operacije na fizičkoj razini.

1.1.5. Jezici za rad s bazama podataka

Komunikacija korisnika odnosno aplikacijskog programa i DBMS-a odvija se pomoću posebnih jezika. Ti jezici tradicionalno se dijele na sljedeće kategorije:

- **Jezik za opis podataka** (*Data Description Language* – DDL). Služi projektantu baze ili administratoru u svrhu zapisivanja sheme ili pogleda. Dakle, tim jezikom definiramo podatke i veze među podacima, i to na logičkoj razini. Naredbe DDL obično podsjećaju na naredbe za definiranje složenih tipova podataka u jezicima poput COBOL-a ili C-a.
- **Jezik za manipuliranje podacima** (*Data Manipulation Language* – DML). Služi programeru za uspostavljanje veze između aplikacijskog programa i baze. Naredbe DML omogućuju "manevriranje" po bazi te jednostavne operacije kao što su upis, promjena, brisanje ili čitanje zapisa. U nekim softverskim paketima, DML je zapravo biblioteka potprograma: "naredba" u DML-u svodi se na poziv potprograma. U drugim paketima zaista je riječ o posebnom jeziku: programer tada piše program u kojem su izmiješane naredbe dvaju jezika, pa takav program treba prevoditi s dva prevodioca (DML prekompajler, obični kompajler).
- **Jezik za postavljanje upita** (*Query Language* – QL). Služi neposrednom korisniku za interaktivno pretraživanje baze. To je jezik koji podsjeća na govorni (engleski) jezik. Naredbe su neproceduralne, dakle takve da samo specificiraju rezultat koji želimo dobiti, a ne i postupak za dobivanje rezultata.

Ovakva podjela na tri jezika danas je već prilično zastarjela. Naime, kod relacijskih baza postoji tendencija da se sva tri jezika povežu u jedan sveobuhvatni. Primjer takvog *integriranog jezika* za relacijske baze je SQL: on služi za definiranje podataka, manipuliranje i pretraživanje. Integrirani se jezik može rabiti interaktivno (preko *on-line* interpretera) ili se on može pojavljivati uklopljen u aplikacijske programe. Svi DBMS-i spomenuti u 1.1.1 koji su danas u širokoj uporabi rabe isključivo SQL za sve tri svrhe.

Naglasimo da gore spomenuti jezici DDL, DML i QL nisu programski jezici jer se u njima primjerice ne mogu zapisati algoritmi koji zahtijevaju petlju ili rekurziju. Dakle, DDL, DML i QL su nam nužni da bismo stvorili bazu i povezali se s njom, no oni nam nisu dovoljni za razvoj aplikacija koje će nešto raditi s podacima iz baze.

Tradicionalni je način razvoja aplikacija koje rade s bazom uporaba klasičnih programskih jezika (COBOL, C, …) s ugniježđenim DML naredbama. U 80-im godinama 20. stoljeća bili su dosta popularni i tzv. *jezici 4. generacije* (*4-th Generation Languages* – 4GL): riječ je o jezicima koji su bili namijenjeni isključivo za rad s bazama, te su zato u tom kontekstu bili produktivniji od programskih jezika opće namjene. Problem s jezicima 4. generacije bio je u njihovoj nestandardnosti: svaki je od njih u pravilu bio dio nekog određenog softverskog paketa za baze podataka, te se nije mogao rabiti izvan tog paketa ili baze.

U današnje vrijeme aplikacije se najčešće razvijaju u standardnim *objektno orijentiranim programskim jezicima* (Java, C++, C#, …). Za interakcije s bazom rabe se unaprijed pripremljene klase objekata. Ovakva tehnika dovoljno je produktivna zbog uporabe gotovih klasa, a razvijeni se program lako dotjeruje, uklapa u veće sustave ili prenosi s jedne baze na drugu.

1.2. Razvojni ciklus baze

Uvođenje baze podataka u neku ustanovu predstavlja složeni zadatak koji zahtijeva primjenu pogodnih metoda i alata te timski rad stručnjaka raznih profila. To je projekt koji se može podijeliti u pet aktivnosti: utvrđivanje i analiza zahtjeva, oblikovanje (projektiranje), implementacija, testiranje i održavanje. Riječ je o razvojnom ciklusu koji je dobro poznat u softverskom inženjerstvu i koji se u sličnom obliku pojavljuje kod razvoja bilo koje vrste softverskih produkata. No u slučaju baza podataka taj ciklus ima neke svoje specifičnosti. Od navedenih pet aktivnosti, nas u ovom nastavnom materijalu najviše zanima oblikovanje. Ipak, zbog cjelovitosti izlaganja, u ovom potpoglavlju ukratko ćemo opisati sve aktivnosti.

1.2.1. Utvrđivanje i analiza zahtjeva

Da bi se utvrdili zahtjevi, proučavaju se tokovi informacija u dotičnoj ustanovi. Dakle, gledaju se dokumenti koji su u optjecaju, prate se radni procesi, razgovara se s korisnicima, proučava se postojeći softver. Uočavaju se *podaci* koje treba pohranjivati i veze među njima.

U velikim organizacijama, gdje postoje razne grupe korisnika, pojavit će se razna tumačenja značenja i svrhe pojedinih podataka te razni načini njihove uporabe. Analiza zahtjeva treba pomiriti te razlike, tako da se eliminira redundancija i nekonzistentnost. Npr., u raznim nazivima podataka treba prepoznati sinonime i homonime te uskladiti terminologiju.

Analiza zahtjeva također mora obuhvatiti analizu *transakcija* (postupaka, operacija) koje će se obavljati s podacima, kako to redovito ima utjecaja na sadržaj i konačni oblik baze. Važno je procijeniti frekvenciju i opseg pojedinih transakcija te zahtjeve na performanse.

Rezultat utvrđivanja i analize zahtjeva jest dokument (obično pisan neformalno u prirodnom jeziku) koji se zove *specifikacija.* Taj dokument rijetko se odnosi samo na podatke. On ujedno definira i najvažnije transakcije s podacima, a često i cijele aplikacije.

1.2.2. Oblikovanje (konceptualno, logičko i fizičko)

Cilj je oblikovanja da se u skladu sa specifikacijom oblikuje građa baze. Dok je analiza zahtjeva otprilike odredila koje vrste podataka baza treba sadržavati i što se s njima treba moći raditi, oblikovanje predlaže način kako da se podaci na pogodan način grupiraju, strukturiraju i međusobno povežu. Glavni rezultat oblikovanja trebala bi biti shema cijele baze, građena u skladu s pravilima rabljenog modela podataka te

zapisana na način da je rabljeni DBMS može razumjeti i realizirati. Kod većih baza, rezultat oblikovanja mogu također biti i pogledi (podsheme) za potrebe pojedinih važnijih aplikacija.

Budući da je oblikovanje prilično složena aktivnost, ona se obično dijeli u tri faze koje slijede jedna iza druge i koje ćemo sad ukratko opisati:

- **Konceptualno oblikovanje.** Glavni rezultat prve faze oblikovanja jest tzv. *konceptualna shema* cijele baze, sastavljena od entiteta, atributa i veza. Ona zorno opisuje sadržaj baze i načine povezivanja podataka u njoj. Prikaz je jezgrovit, neformalan i pogodan ljudima za razumijevanje, no još je nedovoljno razrađen da bi omogućio izravnu implementaciju.
- **Logičko oblikovanje.** Kao glavni rezultat druge faze oblikovanja nastaje *logička shema* koja je u slučaju relacijskog modela sastavljena od relacija (tablica). Sastavni je dio logičkog oblikovanja i tzv. *normalizacija*, gdje se primjenom posebnih pravila nastoji popraviti logička struktura samih relacija, tako da se ona bolje prilagodi inherentnim osobinama samih podataka.
- **Fizičko oblikovanje.** Glavni je rezultat treće faze oblikovanja *fizička shema* cijele baze, dakle opis njezine fizičke građe. U slučaju uporabe DBMS-a zasnovanog na jeziku SQL, pojam fizičke razine treba shvatiti uvjetno. Fizička shema zapravo je niz SQL naredbi kojima se relacije iz logičke sheme realiziraju kao tablice. Pritom se dodaju pomoćne strukture i mehanizmi za postizavanje traženih performansi te čuvanje integriteta i sigurnosti podataka. Također se mogu uključiti i SQL naredbe kojima se pogledi (podsheme) za pojedine aplikacije realiziraju kao virtualne tablice izvedene iz stvarnih tablica.

Navedene tri faze oblikovanja detaljno ćemo obraditi u poglavlju 2, odnosno poglavljima 3 i 4 te poglavlju 6 ovog udžbenika. Svi rezultati oblikovanja opisuju se u odgovarajućim dokumentima koji zajedno čine *projektnu dokumentaciju* baze. O toj dokumentaciji opširnije ćemo govoriti u potpoglavlju 1.3.

1.2.3. Implementacija

Implementacija se svodi na fizičku realizaciju oblikovane baze na odgovarajućem poslužiteljskom računalu. U slučaju DBMS-a zasnovanog na SQL-u pokreću se SQL naredbe koje čine fizičku shemu baze, te se na disku stvaraju prazne tablice sa svim pratećim strukturama i mehanizmima.

Daljnji postupak sastoji se od punjenja praznih tablica s početnim podacima. Takvi podaci obično postoje u nekom obliku, npr. kao obične datoteke ili kao tekstualni dokumenti. Većina DBMS-a opskrbljena je alatima koji nam olakšavaju transfer podataka iz takvih izvora u bazu. No postupak je obično ipak mukotrpan i ne da se sasvim automatizirati, zbog potrebe čišćenja, ispravljanja i usklađivanja podataka.

Nakon što su početni podaci uneseni u bazu, razvijaju se aplikacije koje obavljaju najvažnije transakcije s podacima. Time je omogućeno testiranje.

1.2.4. Testiranje

Testiranje baze provodi se tako da korisnici pokusno rade s bazom i provjeravaju udovoljava li ona svim zahtjevima. Dakle, pokreću se najvažnije transakcije s podacima, prati se njihov učinak te se mjere performanse. Također se nastoji simulirati očekivana frekvencija pojedinih transakcija da bi se utvrdila stabilnost i pouzdanost rada pod opterećenjem.

Glavni je cilj testiranja otkrivanje i popravak pogrešaka koje su se mogle potkrasti u svakoj od prethodnih aktivnosti: dakle u analizi zahtjeva, oblikovanju, odnosno implementaciji. Pogreške u prijašnjim aktivnostima imaju teže posljedice jer se provlače i kroz sljedeće aktivnosti pa zahtijevaju više truda da se poprave. Npr., pogreška u analizi može uzrokovati da u specifikaciji nedostaje neki važni podatak, a to onda znači da tog podatka neće biti ni u projektnoj dokumentaciji ni u implementiranoj bazi, pa popravak treba izvršiti u svim dokumentima i shemama te u samoj bazi.

Mjerenjem performansi tijekom testiranja nastoji se utvrditi jesu li zadovoljeni zahtjevi povezani s performansama, npr. je li brzina odziva zadovoljavajuća. U slučaju da performanse nisu dovoljno dobre, administrator baze može to pokušati ispraviti podešavanjem određenih parametara fizičke organizacije, npr. dodavanjem novih pomoćnih struktura podataka (indeksa) ili raspoređivanjem podataka na više diskova. Ipak, loše performanse mogu biti i posljedica pogrešaka u oblikovanju, npr. posljedica neuočavanja važnih veza između određenih vrsta podataka. U takvom slučaju slijedi opet popravak pogrešaka, dakle nova revizija shema, dokumenata i same baze.

1.2.5. Održavanje

Održavanje se odvija u vrijeme kad je baza već ušla u redovitu uporabu. Riječ je o kontinuiranom procesu, gdje su baza i njezini prateći dokumenti podvrgnuti stalnim promjenama. Neki autori proces održavanja opisuju možda i primjerenijim pojmom *evolucije*.

Kao i općenito u softverskom inženjerstvu, tako i kod baza podataka možemo govoriti o nekoliko vrsta održavanja koje se razlikuju po sadržaju i svrsi traženih promjena. *Korekcijsko održavanje* svodi se na naknadni popravak pogrešaka koje nisu bile otkrivene tijekom testiranja. *Perfekcijsko održavanje* je mijenjanje sheme baze u svrhu prilagođavanja novim aplikacijama koje nisu postojale tijekom polaznog utvrđivanja i analize zahtjeva. *Adaptacijsko održavanje* potrebno je onda kad želimo bazu prilagoditi novom DBMS-u koji se nije rabio u vrijeme oblikovanja i početne implementacije.

Naglasimo da je održavanje baze nužnost na koju se treba pripremiti već tijekom njezina razvoja i uzimati je u obzir tijekom cijelog njezina života. Moramo biti svjesni činjenice da baza koja se ne mijenja vrlo brzo postaje neuporabljiva. Da bi promjene tekle što lakše i bezbolnije, iznimno je važno da baza od početka ima zdravu građu koja odražava inherentnu logiku i povezanost samih podataka. Kod relacijskih baza, ta zdrava građa znači normaliziranost. Ispravno normalizirana relacijska baza moći će se mijenjati bez većih problema, a promjene će se svoditi na povremeno ubacivanje novih podataka u već postojeće relacije ili dodavanje sasvim novih relacija. Pritom te promjene neće utjecati na ispravni rad već postojećih aplikacija jer su one zaštićene svojstvima fizičke i logičke nezavisnosti opisanim u odjeljku 1.1.3.

1.3. Dokumentacija baze

Svaki softverski produkt, pa tako i baza podataka, popraćen je odgovarajućom dokumentacijom. Općenito, razlikujemo *korisničku dokumentaciju* namijenjenu korisnicima i *razvojnu dokumentaciju* namijenjenu softverskim inženjerima. Razvojnu dokumentaciju dalje možemo podijeliti na dokumente koji prate pojedine aktivnosti iz razvojnog ciklusa. U ovom nastavnom materijalu mi ćemo se ograničiti isključivo na *projektnu dokumentaciju* za baze podataka, dakle na dokumente koji nastaju tijekom aktivnosti oblikovanja baze. Istaknut ćemo važnost takve dokumentacije, prikazati nekoliko predložaka za njezinu izradu te spomenuti alate koji se pritom obično rabe.

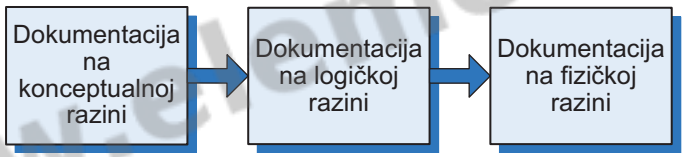
1.3.1. Važnost izrade dokumentacije

Projektna dokumentacija za bazu podataka važna je zato što ona omogućuje ispravan tijek samog oblikovanja i implementacije u skladu s pravilima struke te konzistentan prelazak iz pojedine razvojne faze ili aktivnosti u drugu. Također, dokumentacija je neophodna tijekom testiranja i kasnijeg održavanja baze jer ona predstavlja jedini relevantni izvor informacija o građi baze. Ne treba zaboraviti da u razvoju i održavanju baze obično sudjeluje veći broj ljudi, te da osobe koje će poslije mijenjati bazu nisu one iste osobe koje su je stvorile. Projektna dokumentacija predstavlja sponu između svih tih ljudi koji se možda nikada nisu sreli te njihovu kolektivnu memoriju.

Projektna dokumentacija za bazu podataka prati sve tri faze oblikovanja, i zato se dijeli na tri dijela.

- **Projektna dokumentacija na konceptualnoj razini.** Opisuje konceptualnu shemu baze.
- **Projektna dokumentacija na logičkoj razini.** Dokumentira logičku shemu baze.
- **Projektna dokumentacija na fizičkoj razini.** Sadržava fizičku shemu baze.

Odnos između ta tri dijela prikazan je na slici 1.2. Strelice na toj slici označuju da svaki prethodni dokument predstavlja polazište za izradu idućeg dokumenta.



Slika 1.2. Dijelovi projektne dokumentacije

Završni dio projektne dokumentacije jest dokumentacija na fizičkoj razini. Ona se jedina izravno rabi za implementaciju baze. No to ne znači da se dovršetkom fizičke sheme mogu pobrisati prethodni dokumenti, dakle konceptualna i logička shema. Sva tri dijela moraju se čuvati zato što svaki od njih daje korisnu i komplementarnu informaciju o građi baze. Npr., netko tko se tek želi upoznati s bazom, lakše će se snaći u konceptualnoj shemi nego u fizičkoj shemi. Također, u slučaju prebacivanja na novi DBMS logička shema može predstavljati bolje polazište nego fizička.

Nakon što se baza implementira i uđe u redovitu uporabu, pripadna projektna dokumentacija mora se čuvati zbog kasnijeg održavanja. Zaista, osoba koja namjerava izvršiti promjenu u bazi prisiljena je rabiti dokumentaciju da bi ustanovila gdje i što treba promijeniti. Kad god dođe do promjene u građi baze, ta promjena mora se unijeti i u dokumentaciju. Štoviše, sva tri dijela dokumentacije moraju se istovremeno mijenjati tako da ostanu međusobno konzistentni i konzistentni s realiziranom bazom. Jedino pod tim uvjetom dokumentacija će ostati relevantna za daljnje održavanje.

Kod manjih baza dovoljno je da projektna dokumentacija u svakom trenutku odražava ažurno stanje. No kod većih i složenijih baza korisno je da se osim ažurnog stanja dokumentira i povijest promjena. To se može realizirati tako da se sami dokumenti čuvaju u više verzija, ili tako da se u jednom dokumentu bilježi tko je i kada izvršio kakvu promjenu.

1.3.2. Predlošci za izradu dokumentacije

Projektna dokumentacija sastoji se od tekstualnih i grafičkih dijelova. Dokumentacija na konceptualnoj razini uglavnom je grafička, dakle prije svega se oslanja na dijagrame. Dokumentacija na logičkoj razini može dijelom biti grafička, no ipak se više oslanja na tekstualne dijelove koji se oblikuju uporabom bogate tipografije (raznoliki fontovi, podcrtavanje i slično). Dokumentacija na fizičkoj razini uvijek je goli ASCII tekst.

Rekli smo da dokumentacija na konceptualnoj razini ustvari opisuje konceptualnu shemu baze koja se sastoji od elemenata koji se zovu entiteti, atributi, odnosno veze.