Fizička organizacija baze podataka

Hrenić, Jakov

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:680941

Fizička organizacija baze podataka

Hrenić, Jakov

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Jakov Hrenić

FIZIČKA ORGANIZACIJA BAZE PODATAKA

Diplomski rad

Voditelj rada:
dr.sc. Robert Manger

Zagreb, 2021.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Mojoj obitelji i prijateljima

Sadržaj

Uvod

Baze podataka čine velik dio današnjeg svijeta. Predstavljaju višu razinu rada s podacima u odnosu na klasične programske jezike. Poduzeća, velika ili mala, jako se oslanjaju na baze podataka za analizu, predviđanja, recenzije, kvalitetu, bolju produktivnost, bolje rezultate i tako dalje. Baze podataka sadrže raznolike podatke, kao što su imena, e-mail adrese, zaporke i još mnogo toga. Teško je zamisliti da su se nekad važni podaci bilježili na papiru, ali bolje alternative nije bilo. Tehnologija baza podataka uklonila je slabosti tradicionalne automatske obrade podataka iz sredine 20. stoljeća.

Tema ovog rada su fizički aspekti organizacije baza podataka, a rad je podijeljen na četiri glavna poglavlja.

Prvo poglavlje započinjemo samim konceptom baza podataka. Definiramo što su zapravo baze podataka te njihova glavna obilježja.

U drugom poglavlju bavimo se fizičkom organizacijom baze podataka. Započinjemo metodama pristupa bazi podataka, zatim uvodimo nove pojmove koji će biti od velike važnosti kroz rad, kao što su prostor tablice i indeks. Da bismo objasnili kako se pretražuje tablica indeksima, definiramo podatkovnu strukturu B-stablo te navodimo glavna svojstva. Na kraju, objašnjava se prirodni spoj i glavne tehnike fizičke implementacije prirodnog spoja.

Treće poglavlje posvećeno je RAID tehnologijama te podsustavu za spremanje podataka. RAID sustavi su tehnologija za pohranjivanje podataka na disku, bez obzira radi li se o podacima iz baze podataka ili o bilo kakvim drugim podacima. To je za DBMS nevidljivo, no svejedno zanimljivo.

U četvrtom, završnom poglavlju nalazi se praktični dio ovog rada. Pomoću dostupnog softvera, realizira se vlastita baza te se na istoj testiraju razne varijante fizičke organizacije odnosno upravljanja transakcijama.

U svrhu ovog rada koriste se [11] ([7]), [10], [12] i [9] kao glavna literatura. Neke slike su preuzete sa [8]. Ostatak slika izradio je autor rada samostalno.

Poglavlje 1

Baze podataka i dizajn baza podataka

1.1 Koncept baza podataka

Baza podataka

Baza podataka je skupina podataka povezanih u nekom određenom procesu, pohranjenih u vanjskoj memoriji računala. Podaci su istovremeno dostupni raznim korisnicima i aplikacijama programa.

Sustav za upravljanje bazom podataka (Data Base Management System - DBMS) je softver koji definira, kreira, koristi i održava bazu podataka. On oblikuje fizički prikaz baze u skladu s traženom logičkom strukturom. Popularni DBMS-ovi koji se danas koriste su: Oracle, Microsoft SQL server, MySQL, IBM DB2.

Model baze podataka, odnosno *shema baze podataka*, sadrži podatke, njihove atribute, povezanosti i slično. *Stanje baze podataka* predstavlja trenutne podatke u bazi. Model baze podataka može biti:

- *konceptualni* – detaljni opisi podataka i veza
- *logički* – translacija konceptualnog prema specifičnim područjima; jasno opisuje gdje je koji podatak pohranjen
- *vanjski* – sadrži razne podskupove podataka, orijentiran prema specifičnim uporabama

Struktura baze podataka

Stuktura baze podataka sastoji se od tri sloja:

- *unutarnji* – opisuje kako je baza pohranjena ili organizirana fizički
- *konceptualni/logički* – fokus na podatke i veze
- *vanjski* – sadrži poglede[1]

Jezici baze podataka

Svaki DBMS sadrži jedan ili više jezika baze podataka.

- *Jezik za definiciju podataka (Data definition language - DDL)* služi administratoru baze podataka (Data base administrator - DBA) za model baze podataka.
- *Jezik za manipuliranje podacima (Data manipulation language - DML)* služi za hvatanje i modificiranje podataka.
- *Jezik za postavljanje upita (Query Language - QL)* služi korisnicima baze za pretraživanje podataka.

Pogodnosti baza podataka

Baze podataka imaju mnoge pogodnosti. Nezavisnost podataka osigurava da promjene u definiciji podataka imaju minimalni utjecaj na aplikacije koje koriste te podatke. Fizička nezavisnost podataka osigurava da sve promjene u pohrani podataka u unutarnjem sloju baze nemaju utjecaja na aplikacije, poglede ili logički model podataka. Slično, logička nezavisnost podataka osigurava minimalne promjene na softver aplikacije kad se dogode promjene u konceptualnom ili logičkom modelu podataka.

Fleksibilnost pristupa podacima je još jedna pogodnost baza podataka koja dozvoljava korisnicima da mogu slobodno pretraživati podatke. Baze podataka također imaju i pogodan način za upravljanje redundancijom podataka. DBMS se brine da duplicirani podaci budu jednako ažurirani kao i njihovi originali, što pridonosi točnosti podataka.

Omogućeno je da više korisnika istovremeno koristi iste podatke. Pravila integriteta podataka se mogu eksplicitno primjeniti. Ona osiguravaju da svaki podatak bude u točnom formatu. Primjerice, podaci koji su predstavljeni cijelim brojem ne mogu biti decimalni

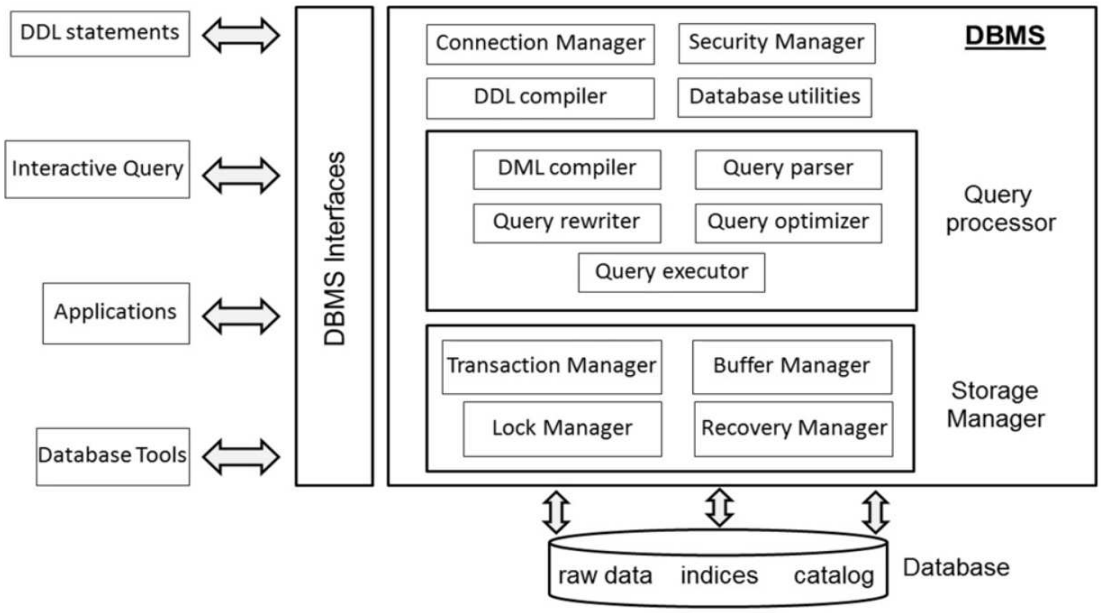
[1] Pogled (*view*) pokazuje dio baze podataka koji je od interesa, ignorirajući ostatak.

brojevi. Po pitanju pristupa datoteka, ta pravila moraju biti ugrađena u svakoj aplikaciji, a u smislu pristupa baza podataka, moraju biti dio konceptualnog ili logičkog modela podataka, a pohranjeni su u katalogu.[2]

Dobra rezerva i povrat (backup i recovery) podataka je još jedna ključna pogodnost baza podataka. Sigurnost baza podataka može se direktno primijeniti preko DMBS, što daje određenim korisnicima određena prava u radu s bazom podataka.

1.2 Arhitektura baza podataka

DBMS se sastoji od raznih modula koji zajedno čine *arhitekturu sustava za upravljanje bazom podataka (database management system architecture)*. Slika 1.1 prikazuje pregled ključnih komponenata arhitekture DBMS-a.



Slika 1.1: Arhitektura sustava za upravljanje bazom podataka

- *Connection manager* uspostavlja vezu baze podataka. Može biti postavljen lokalno ili kroz mrežu. Najčešće se postalja mrežno.

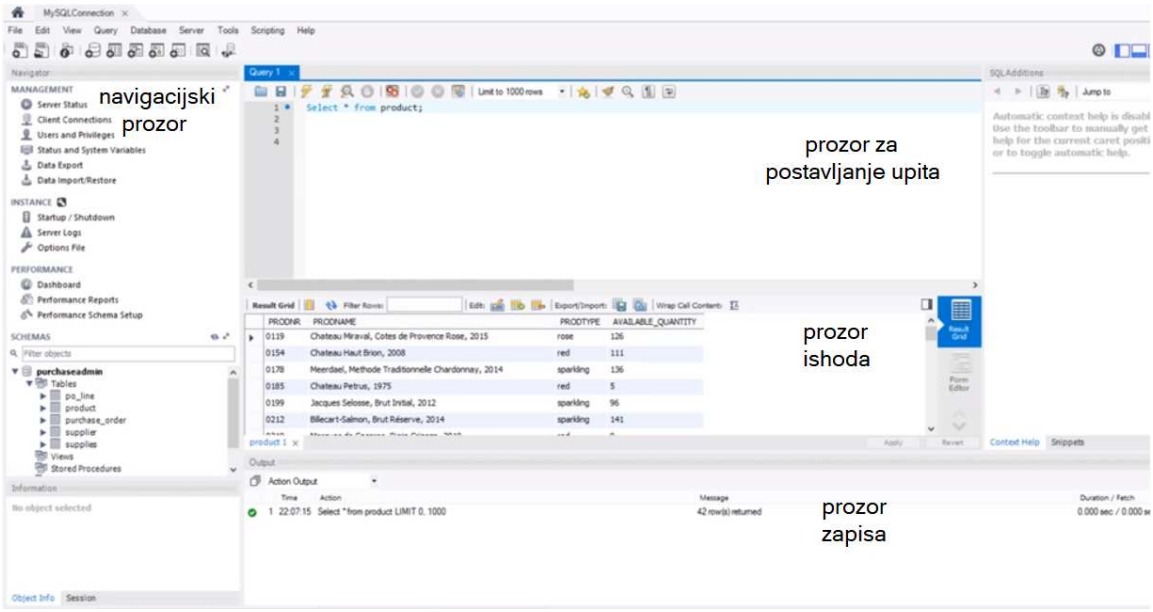
[2]Katalog je glavni dio DBMS-a, koji sadrži definicije podataka te pohranjuje i sinkronizira modele podataka da bi osigurao konzistentnost.

- *Security manager* verificira ima li korisnik pravo pristupu i vršenju određenih radnji nad bazom podataka. Neki korisnici, primjerice, imaju pravo samo čitanju, dok drugi mogu i pisati i mijenjati podatke unutar baze.
- *DDL compiler* kompajlira definicije podataka za svaki od modela.
- *DML compiler* kompajlira naredbe za manipuliranje podacima.
- *Query processor* je jedan od najvažnijih dijelova DBMS-a. On pomaže kod izvršavanja upita - dohvaćanje podataka te dodavanje, brisanje i ažuriranje podataka u bazi.
- *Query parser* raščlanjuje upit u unutarnji reprezentacijski format kojeg sistem može dalje evaluirati.
- *Query rewriter* optimizira upit, neovisno o trenutnom stanju baze podataka.
- *Query optimizer* optimizira upit na temelju trenutnog stanje baze podataka.
- Komponenta koja nastupa odmah nakon je *Query Executor*. Ona se pobrine da se upit izvrši.
- *Store manager* vlada nad fizičkim pristupima datoteka.
- *Transaction manager* nadzire izvršenja transakcija baze podataka.
- *Buffer manager* upravlja buferima. Iz bufera DBMS dohvaća podatke.
- *Lock manager* osigurava integritet podataka. Postoje dvije vrste ključa: read i write. Read ključ dopušta čitanje podataka, a write ključ dopušta ažuriranje podataka.
- *Recovery manager* osigurava točno izvođenje transakcija u bazi podataka.

Sučelje DMBS-a

DBMS mora komunicirati s dizajnerom baze podataka, administratorom baze podataka, aplikacijom, krajnjim korisnikom i ostalima. Da bi sva ta komunikacija bila moguća, DMBS pruža razna korisnička sučelja, kao što su: *mrežno sučelje*, *posebno sučelje za jezik za postavljanje upita*, *sučelje za naredbe* i mnoga druga.

Na slici 1.2 prikazan je primjer MySQL sučelja. Može se uočiti navigacijski prozor (ima više odjeljaka), prozor za postavljanje upita (ovdje se mogu pisati SQL upiti), prozor ishoda (prikazuje rezultate izvršenih upita) te prozor zapisa (prikazuje povijest radnji i greške).



Slika 1.2: Sučelje MySQL-a

Poglavlje 2

Fizička organizacija i metode pristupa bazi podataka

2.1 Metode pristupa bazi podataka

Prije svega, glavni fokus fizičke organizacije baze podataka je razumijevanje unutarnjeg modela podataka. Dizajn fizičke baze podataka se svodi na translaciju logičkog modela podataka u unutarnji model podataka, odnosno fizički model podataka.

U relacijskom smislu, govorimo o logičkoj bazi podataka kao skupu tablica ili relacija. Tablica se sastoji od redova ili n-torki koji sadrže neke vrijednosti. Unutarnji model podataka prikazuje kako su ti koncepti fizički realizirani te definira fizičku strukturu koja dozvoljava da ti koncepti daju pristup na efikasni način.

Primjerak podataka (data item) je skup bitova ili znakova koji predstavljaju specifičnu vrijednost u fizičkom mediju za pohranu. *Pohranjeni zapis (stored record)* je skup primjerka podataka koji se odnose na isti entitet te predstavljaju sve atribute vezane za taj entitet. *Fizička datoteka (physical file, dataset)* je skup pohranjenih zapisa koji predstavljaju slične entitete[1]. Obično svi zapisi u fizičkoj datoteci imaju sličnu strukturu te sadrže primjerke podataka koji predstavljaju isti skup atributa. Ipak, ako se radi o kombiniranju pohranjenih zapisa koji predstavljaju različite entitete, riječ je o *mješovitoj datoteci (mixed file)*.
Fizička baza podataka (physical database, stored database) je integrirani skup pohranjenih podataka. Sadrži primjerke podataka i pohranjene zapise koji se odnose na različite entitete te pokazuje odnose među njima[2]. Logičke strukture i veze također poprimaju fizički oblik.

[1] primjerice pacijenti, restorani, računi u restoranu...
[2] primjerice koji račun je iz kojeg restorana...

Razlikujemo dvije vrste baza podataka: korisničke i sistemske. Korisničke baze podataka imaju fizičku strukturu koju namješta dizajner baze, dok sistemske imaju takvu strukturu kakvu definiraju autori DBMS-a. *Prostor tablice (tablespace)* je fizički spremnik objekata baze podataka, koji sadrži jednu ili više fizičkih datoteka. Svaka logička tablica je dodijeljena prostoru tablice da bude fizički oblikovana. *Pohranjena tablica (stored table)* zauzima jedan ili više blokova u prostoru tablice. Prostor tablice može imati indekse[3], a mogu postojati i odvojeni *indeks prostori (index space)*. Neki indeksi su generirani automatski, obzirom na logički model podataka, a neke administrator baze eksplicitno generira. Primarni indeksi odlučuju fizički poredak zapisa podataka u datoteci, dok dodavanje ili ukidanje sekundarnih indeksa nema utjecaj na fizičke datoteke podataka.

2.2 Optimizator upita

Strukturirani jezik za postavljanje upita (Structured Query Language - SQL) je jezik koji se koristi za definiciju i manipulaciju podataka te za postavljanje upita. SQL-om razvitelj aplikacije specificira koje podaci su potrebni, ali ne kako su locirani i dohvaćeni iz fizičkih datoteka baza podataka. Obično može postojati više različitih puteva za dohvat istih podataka, sa mogućim znatnim varijacijama u vremenu dohvata podataka. Za svaki upit, optimizatorova je zadaća da za razne moguće načine razrješavanja upita pripremi najefikasniji plan pristupa podacima. Danas se optimizatori temelje na troškovima, te kalkuliraju optimalne planove pristupa obzirom na set ugrađenih formula troškova, tablice uključene u upit, dostupne indekse, statistička svojstva podataka i slično.

Procesor upita (query processor) je komponenta DBMS-a koja asistira pri dohvatu i ažuriranju upita za bazu podataka. Sastoji se od DML kompajlera, raščlanitelja upita, prepravljača upita, optimizatora upita i izvršitelja upita. *DML kompajler (DML compiler)* vadi ugnježđene DML naredbe iz programa pisanog u programskom jeziku. *Raščlanitelj upita (query parser)* raščlanjuje upit u interni reprezentacijski format te provjerava sintatičku i semantičku točnost upita. Tu reprezentaciju prepravlja *prepravljač upita (query rewritter)*, koji optimizira upit tako da ga pojednostavljuje obzirom na set predefiniranih pravila. Tada nastupa *optimizator upita (query optimizer)*. Uzima u obzir trenutno stanje baze podataka te informacije u katalogu i indekse. Rezultat te procedure je konačni plan pristupa koji se predaje *izvršitelju upita (query executor)* na izvršavanje.

[3] Indeksi će biti pobliže objašnjeni kasnije u zasebnom dijelu.

DBMS održava sljedeće statističke podatke u svom katalogu, koje optimizator koristi da optimizira plan pristupa:

- podaci obzirom na tablicu:
  - broj redaka
  - broj blokova diska koje tablica okupira
  - broj overflow zapisa udruženih s tablicom
- podaci obzirom na stupce:
  - broj različitih vrijednosti u stupcu
  - statistička distribucija vrijednosti u stupcu
- podaci obzirom na indeks:
  - broj različitih vrijednosti za indeksirane ključeve za pretraživanje i za individualne tipove atributa mješovitih ključeva za pretraživanje
  - broj blokova diska koje indeks zauzima
  - tip indeksa: primarni/grupirani ili sekundarni
- podaci obzirom na prostor tablica
  - broj i veličina tablica u prostoru tablica
  - I/O (input/output) svojstva uređaja u kojem tablica boravi

Faktor filtera (filter factor - FF) udružen je s *predikatom upita (query predicate)* koji specificira uvjet odabira obzirom na određeni atribut s tipom A_i unutar upita[4]. Predikatov FF_i predstavlja dio ukupnog broja n-torki koji bi trebao zadovoljiti predikat koji se odnosi na atribut s tipom A_i. Drugim riječima, FF označava šansu da određeni red bude odabran obzirom na predikat upita. Za upite preko jedne tablice[5], očekivani *kardinalitet upita (query cardinality - QC)* jednak je *kardinalitetu tablice (table cadinality - TC)* pomnoženim s produktom faktora filtera dotičnih predikata pretraživanja u upitu. Kardinalitet se definira kao broj n-torki: QC je broj n-torki odabranih za upit, a TC broj n-torki u tablici. Prethodnu tvrdnju možemo izaziti u obliku formule:

$$QC = TCxFF_1xFF_2x\cdots xFF_n$$

Ako nema drugih dostupnih statističkih informacija, onda je procjena za FF_i jednaka $\frac{1}{NV_i}$, gdje je NV_i broj različitih vrijednosti atributa s tipom A_i.

[4] npr. ”Spol=M” ili ”Dob>35”

[5] Join upiti, koji se više analiziraju kasnije u zasebnom dijelu.

2.3 B-stablo

Stablo je struktura podataka koja se sastoji od skupa čvorova i grana, povezanih određenim setom pravila.

- Postoji točno jedan *korijen* čvor.
- Svaki čvor, osim korijena, ima točno jedan *roditelj* čvor.
- Svaki čvor ima nula, jedno ili više *djece* čvorova.
- Čvorovi s istim roditeljem su *braća* čvorovi.
- Sva djeca, djeca od djece, itd. čvora su njegovi *potomci*.
- Čvor bez djece je *list*.
- Podstablo je dio stabla koji je sastavljen od jednog čvora koji nije korijen i svih njegovih potomaka.
- Čvorovi su raspodijeljeni u *razine* koje predstavljaju udaljenost od korijena. Razina korijena je 0; razina djeteta jednaka je razini roditelja plus 1. Sva braća su na istoj razini.
- Stablo u kojem su svi listovi na istoj razini zove se *balansirano* stablo. U suprotnome, stablo je *nebalansirano*.

Stuktura stabla pruža fizičku reprezentaciju logičke hijerarhije ili čak čistu fizičku strukturu indeksa da bi se ubrzalo pretraživanje i dohvaćanje podataka navigiranjem kroz međusobno povezane čvorove stabla. Potonji tip stabla zovemo *stablo pretraživanja*.

Binarno stablo pretraživanja je fizička struktura stabla u kojem svaki čvor ima najviše dvoje djece. Svaki čvor sadrži vrijednost ključa za pretraživanje i pokazivače na djecu. Svako dijete je korijen podstabla, gdje jedno podstablo sadrži samo vrijednosti ključeva koje su manje nego vrijednosti ključa u originalnom čvoru, a drugo podstablo sadrži samo one koje su veće. Navigiranje kroz binarno stablo je vrlo slično tehnici binarnog pretraživanja.

B-stablo služi kao fizička struktura za indeks. B-stablo je varijacija stabla pretraživanja eksplicitno dizajnirana za pohranu na tvrdom disku. Svaki čvor odgovara nekom bloku diska te su kapaciteti čvorova očuvani između pola i punog. Iz tog razloga postoji određena dinamičnost u indeksima. Svaki čvor sadrži skup *vrijednosti ključa za pretraživanje (search key values)*, skup *pokazivača stabla (tree pointers)* koji pokazuju na djecu čvorove i

skup *pokazivača podataka (data pointers)* koji pokazuju na zapise datoteka ili blokove sa zapisima datoteka koji odgovaraju vrijednostima ključa za pretraživanje. Zapisi datoteka su posebno pohranjeni i nisu dio B-stabla. *B-stablo reda k* ima sljedeća svojstva:

- Svaki čvor koji nije list je sljedećeg formata: $< P_0, < K_1, Pd_1 >, P_1, < K_2, Pd_2 > , \ldots < K_q, Pd_q >, P_q >$, gdje je $q \leq 2k$. Svaki P_i je pokazivač stabla - pokazuje na neki drugi čvor u stablu. Taj čvor je korijen podstabla na kojeg P_i pokazuje. Svaki Pd_i je pokazivač podatka - pokazuje na zapis s vrijednosti ključa K_i ili na blok diska koji sadrži taj zapis.
- B-stablo je balansirano. Svaki put od korijena do lista je jednake duljine, koja se zove *visina* B-stabla.
- Listovi imaju istu strukturu kao i čvorovi s djecom, samo što su svi njihovi pokazivači stabla P_i null.
- U svakom čvoru vrijedi: $K_1 < K_2 < \ldots < K_q$
- Za svaku vrijednost ključa X u podstablu na čiji korijen pokazuje P_i vrijedi:
  - $K_i < X < K_{i+1}$, za $i = 1, \ldots, q-1$
  - $X < K_{i+1}$, za $i = 0$
  - $K_i < X$, za $i = q$
- Korijen ima jednak broj vrijednosti ključa i pokazivača podataka, koji varira između 1 i $2k$. Broj pokazivača stabla i djece čvorova varira između 1 i $2k + 1$.
- Svi "normalni" čvorovi (koji nisu korijen ni list) imaju određen broj vrijednosti ključa i pokazivača podataka, koji je između k i $2k$. Broj pokazivača stabla i djece čvorova varira između $k + 1$ i $2k + 1$.
- Svaki list ima određen broj vrijednosti ključa i pokazivača podataka, koji je između k i $2k$, no nema pokazivača stabla.

Dakle, čvorovi koji nisu list s q vrijednosti ključa bi trebali imati q pokazivača podataka i q+1 pokazivača stabla na djecu. Pokazivači podataka Pd_i ne pokazuju na zapise direktno, već na blok koji sadrži pokazivače na sve zapise koji zadovoljavaju vrijednost ključa za pretraživanje K_i.

Pretraživanje B-stabla počinje od korijena. Ako je željena vrijednost ključa X pronađena u čvoru, recimo $K_i = X$, tada pridruženi zapisi podataka mogu biti pristupljeni u datoteci podataka prateći pokazivač podataka Pd_i. Ako tražena vrijednost nije nađena u čvoru,

pokazivač stabla navodi do podstabla koje zadovoljava određeni raspon vrijednosti ključa. Preciznije, pokazivač podstabla P_i kojeg se prati je onaj pridružen najmanjoj vrijednosti od i tako da vrijedi: $X < K_{i+1}$. Ako taj uvjet ne vrijedi za nijedan i, tada se prati pokazivač stabla Pd_{i+1}. Ista procedura se ponavlja i dalje sve dok nije nađena vrijednost ključa ili dok se ne dostigne list, što znači da tražena vrijednost ključa nije prisutna.

Kapacitet čvora jednak je veličini bloka diska. Svi čvorovi osim korijena su popunjeni barem do 50%. B-stablo, dakle, efikasno koristi kapacitet pohrane, ali i dalje ostavlja mjesta za dodatke datoteci podataka bez potrebe preslagivanja strukture indeksa, tako da se popuni prazan prostor. Ako se dogodi da su svi čvorovi u prikladnom podstablu za tu vrijednost ključa popunjeni, tada se čvor razdvaja u dva polupuna čvora. Oba postaju djeca originalnog čvora. Ako čvor postane manje nego polupun brisanjem vrijednosti ključa i zapisa, spaja se s jednim od svoje braće tako da nastane čvor koji je više nego polupun. Broj ovakvih promjena je ograničen te ovisi o visini B-stabla.

Većina implementacija DBMS-a koriste indekse bazirane na B^+-*stablima* rađe nego na B-stablima. Glavna razlika je da kod B^+-stabla samo listovi sadrže pokazivače podataka. Dodatno, sve vrijednosti ključa u čvorovima koji nisu listovi su ponovljene u listovima. Čvorovi na višim razinama sadrže samo podskupove vrijednosti ključeva u listovima. Konačno, svaki list B^+-stabla također ima jedan pokazivač stabla, koji pokazuje na svog sljedećeg brata. Na taj način potonje stablo ima povezanu listu listova; sekvencijalno uređeni listovi obzirom na vrijednosti ključa koje sadrže, što pruža dodatni način putovanja kroz stablo. Odnosno, stablo ne mora biti navigirano gore-dolje, već može biti navigirano lijevo-desno, preko listova, što je efikasnije za *intervalne upite (range query)*.
Obzirom da samo listovi sadrže pokazivače podataka, svako pretraživanje mora se nastaviti sve do listova, što nije bio slučaj kod B-stabla. Ipak, B^+-stabla su obično efikasnija, pošto čvorovi koji nisu listovi ne sadrže pokazivače podataka, stoga im s istom veličinom bloka red stabla može biti viši. Visina B^+-stabla je obično manja, što rezultira manjim pristupom blokovima za pretragu stabla.

2.4 Indeks

Indeksi su vrlo važni instrumenti dizajnerima i administratorima baza podataka. Ključni su za efikasno procesiranje upita. Većina DBMS-a koriste sljedeću sintaksu u SQL jeziku:

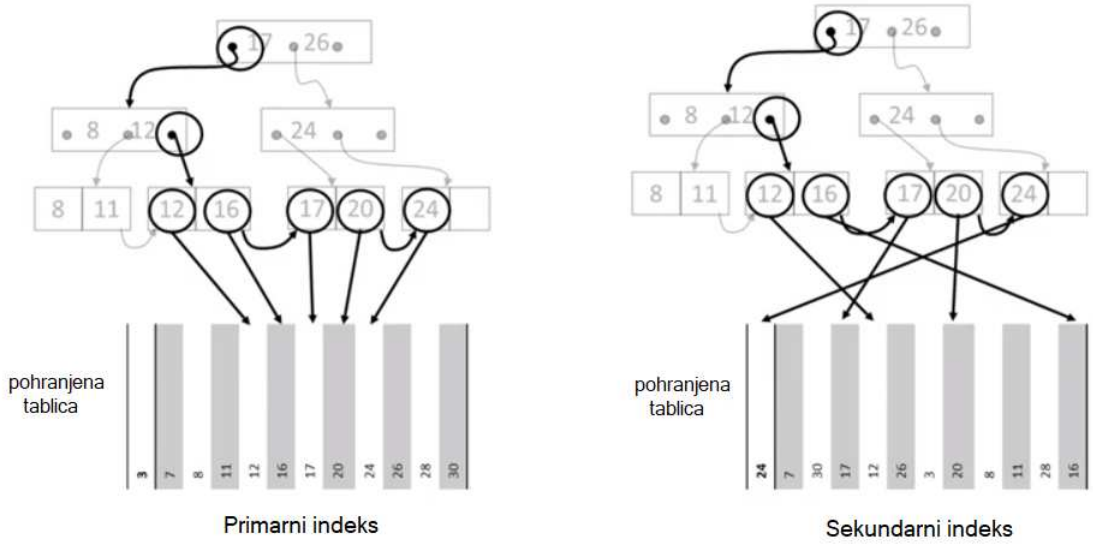
```
CREATE [UNIQUE] INDEX INDEX_NAME
ON TABLE_NAME (COLUMN_NAME [ORDER] {, COLUMN_NAME [ORDER] } )
[CLUSTER]
```

Pitanje je zašto su indeksi tako važni pa u nastavku navodimo najbitnije razloge. Indeksi mogu biti kreirani na svakom stupcu ili kombinaciji stupaca, a vađenje podataka je onda lakše za ključeve za pretraživanje koji su indeksirani. Indeksi se mogu koristiti za pretraživanje povezanih n-torki u dvije tablice (koje odgovaraju naredbi *join*), a takve pretrage obično obuhvaćaju i strane ključeve. Kad je indeks kreiran na odgovarajuće stupce ili kombinacije stupaca, DBMS mora pretraživati samo indeks da se uvjeri da nema dvostrukih vrijednosti. Takve indekse zovemo jedinstvenima (unique). Jedinstveni indeks je automatski generiran za svaki primarni ključ, no drugi jedinstveni indeksi mogu biti eksplicitno definirani. Ako jedinstveni tip atributa ili kombinacija tipa atributa određuje fizički slijed redova u tablici, tada je jedinstveni indeks primarni indeks, inače je sekundarni.
Svaki je indeks poredan na određen način, ovisno o tipovima atributa koje obuhvaća, a specificira se naredbama *ASC* (uzlazno) i *DESC* (silazno). Time indeksi specificiraju logički slijed na n-torkama u tablici, koje između ostalog sadrže iste tipova atributa. Sekundarnih indeksa može biti najviše onoliko koliko ima tipova atributa u tablici.
Redoslijed indeksa također može odrediti i fizički redoslijed redova u tablici. Ako je neka vrijednost ili kombinacija vrijednosti ključeva za pretraživanje jedinstvena, tada se radi o primarnom indeksu. Inače se radi o grupirajućem indeksu. U tom pogledu, dohvaćanje n-torki tim redoslijedom (naredbom *ORDER BY*) postaje vrlo efikasno. Pošto određuje fizički redoslijed n-torki u tablici, može postojati samo jedan takav indeks, ili primarni ili grupirani, za svaku tablicu.
Sekunardnih indeksa može postojati proizvoljno mnogo. Svaki takav indeks zauzima kapacitet pohrane pa stoga i usporava upite ažuriranja jer i sami indeksi tada moraju biti ažurirani. Adminitrator baze podataka, iz tog razloga, mora dobro procijeniti koji su mu indeksi potrebni. Dobra stvar je da se indeksi mogu kreirati i ukloniti bez da utječu na fizičke podatke, stvarne datoteke podataka ni logički model podataka, što nije slučaj kod primarnih i grupiranih indeksa.
Tipično su najbolji kandidati za indeksiranje upravo primarni ključevi i ostali koji povlače jedinstvenost, strani ključevi i ostali koji se često korsite kod pridruživanja (*join*) te ostali tipovi atributa koji se često koriste kod upita. Bolje je izbjegavati velike tipove atributa, odnosno one koji se sastoje od velikih stringova ili varchar. Drugi ključan faktor je veličina tablice. Naime, indeksi na manjim tablicama usporavaju ažuriranja, ali ubrzavaju odgovore na upit.

Promotrimo situaciju s jednim predikatom upita, gdje upit uzima ključ za pretraživanje sa samo jednim tipom atributa. Tada, ako pripadni indeks postoji, pretraživanje indeksa je obično najefikasniji način implementacije upita. Uzmimo za primjer B^+-stablo. Indeks dozvoljava filtriranje n-torki podataka koje bi trebale biti dohvaćene; dakle izbjegavaju se nepotrebna dohvaćanja i poboljšava se izvođenje izvršenja upita. Vrše se sljedeći koraci:

- Počevši od indeksa korijena, spušta se B^+-stablom preko pokazivača stabla, prema vrijednosti ključa za pretraživanje, na prvi list koji sadrži vrijednost ključa koji zadovoljava uvjet pretraživanja.
- Od tog lista, prate se pokazivači podataka da bi se dohvatili retci podataka koji zadovoljavaju vrijednost ključa za pretraživanje.
- Prati se "idući" pokazivač stabla[6] u listovima da bi se sekvencijalno pristupilo svim čvorovima koji sadrže vrijednosti ključa unutar traženog raspona ključa za pretraživanje. Za svaki čvor, prate se pokazivači podataka da bi dohvatili pripadne n-torke podataka.

Ako upit koristi samo jednu vrijednost ključa za pretraživanje, a ne raspon vrijednosti, tada bi efikasna alternativa bila koristiti datoteke s direktnim pristupom, mapirati vrijednosti ključeva za pretraživanje u fizičke lokacije "heširanjem". Kod intervalnih upita, primarni ili grupirajući indeks je više efikasan nego sekundarni. Ako su zapisi podataka pohranjeni istim redoslijedom kao i ključ za pretraživanje, tada listovi B^+-stabla mogu biti pristupljeni pomoći "idućih" pokazivača te n-torke podataka mogu biti dohvaćene istim redoslijedom kao pristup sekvencijalnih blokova. S druge strane, ista procedura sa sekundarnim indeksom bi rezultirala slučajnim pristupom blokovima, "skakajući amo-tamo" s bloka na blok jer nisu u prikladnom fizičkom redoslijedu. Situacija je prikazana slikom 2.1.



Slika 2.1: Prikaz intervalnih upita kod primarnih i sekundarnih indeksa

[6]Ovaj korak nije potreban ako upit koristi samo jednu vrijednost ključa za pretraživanje, npr. "AGE=35".

2.5 Višestruki indeks, multistupac i pristup samo preko indeksa

Ako je ključ za pretraživanje složen, sličan pristup je izvediv, ali postoje i druge opcije; ovisno o dostupnosti jednostupčanih ili multistupčanih indeksa, koji pokrivaju tipove atributa koji se pojavljuju u predikatu upita. Ako su indeksirani tipovi atributa jednaki tipovima atributa u ključu za pretraživanje, tada je pristup iz prošlog dijela najefikasniji. *Indeks multistupca (multicolumn index)* tada dozvoljava filtriranje i povrat samo onih n-torki podataka koje zadovoljavaju upit. U većini slučajeva, ne može se stvoriti indeks multistupca za svaku moguću kombinaciju ključeva za pretraživanje jer bi ti indeksi postali preveliki. Doista, pretpostavimo da postoji n mogućih atributa s tipom A_i koji se koriste u ključu za pretraživanje, $i = 1, \ldots, n$, tako da atribut s tipom A_i ima NV_i mogućih vrijednosti. Tada gust indeks kreiran po A_i ima $\prod_{i=1}^{n} NV_i$ unosa. Ako umjesto toga kreiramo odvojene guste indekse preko svakog zasebnog tipa atributa, tada postoji n indeksa s NV_i unosa, redom, $i = 1, \ldots, n$. Tada je ukupan broja unosa preko svih indeksa jednak samo $\sum_{i=1}^{n} NV_i$, što je znatno manje.
Situacija je još komliciranija ako se ne izvršavaju samo upiti koji se odnose na tipove atributa pretraživanja, već i upiti koji se odnose na proizvoljan podskup tih tipova atributa. Kako bi efikasno podržao ove vrste upita, indeks multistupca mora imati tipove atributa koji su uključeni u predikate upita u najljevijem stupcu. Da podrži sve moguće upite s n ili manje tipova atributa, višestruki redundantni indeksi multistupaca mogu biti kreirani pomoći istih n tipova atributa, ali s drugačijim redoslijedom stupaca. Može se dokazati da mu je za to potrebno $\binom{n}{\lceil \frac{n}{2} \rceil}$ indeksa.
Iz prethodnog jasno slijedi da broj mogućih kombinacija tipova atirbuta i broj potrebnih indeksa raste s brojem tipova atirbuta. Dakle, indeksi multistupca su prigodni samo za određene slučajeve, specifično s upitima koji se često izvršavaju ili kojima je kritično vrijeme izvršavanja. Također, koriste se za provođenje ograničenja jedinstvenosti, ali je tada broj tipova atributa ograničen i nema potrebe za redundantnim indeksima.

Alternativno, umjesto indeksa multistupca može se koristiti više jednostupčanih indeksa ili indeksa s manje stupaca. Na primjer, ako je upit oblika:

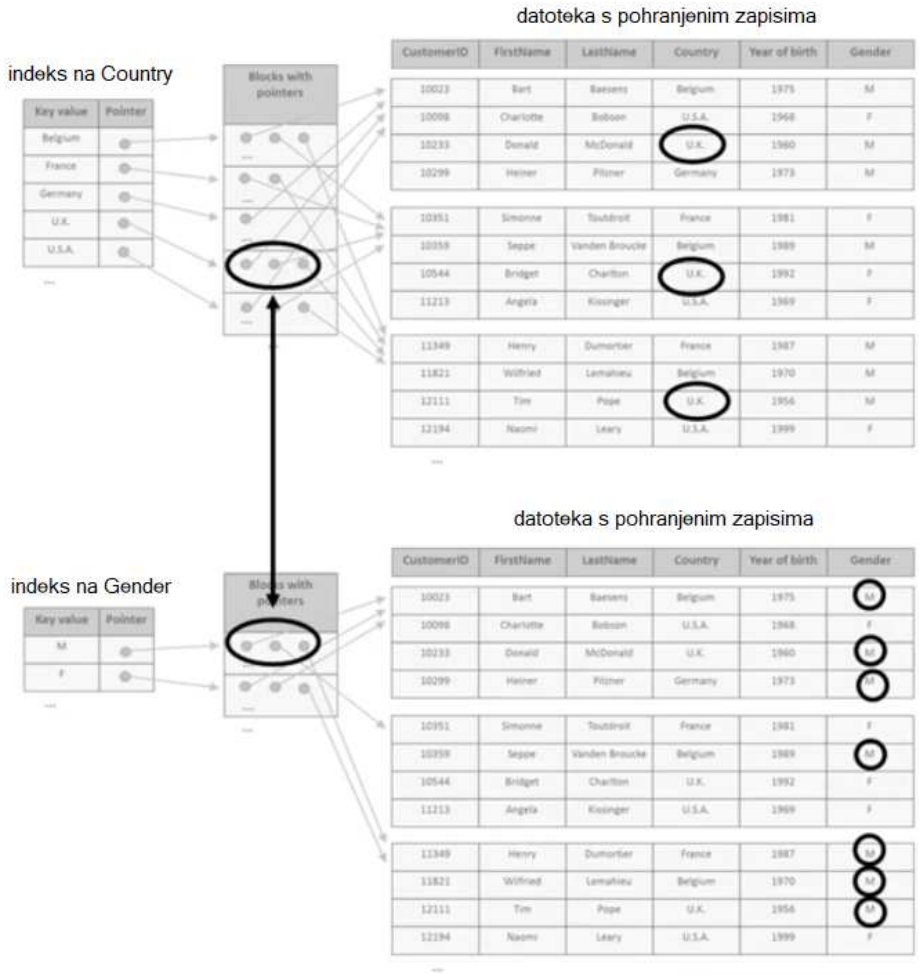
```
SELECT *
FROM MY_TABLE
WHERE A_{1}=VALUE_{1}
AND A_{2}=VALUE_{2}
AND ...
AND A_{n}=VALUE_{n}
```

i postoje jednostupčani indeksi za svaki atibut A_i, tada se koristi presjek skupova poka-

zivača u indeksima unosa koji odgovaraju traženim vrijednostima A_i. Takav presjek daje pokazivače na sve zapise koje zadovoljavaju upit. S druge strane, za upite oblika:

```
SELECT *
FROM MY_TABLE
WHERE A_{1}=VALUE_{1}
OR A_{2}=VALUE_{2}
OR ...
OR A_{n}=VALUE_{n}
```

može se koristiti unija skupova pokazivača da daje zapise koji zadovoljavaju upit.



Slika 2.2: Prikaz korištenja dva indeksa za izvršavanje upita s više predikata

Kao primjer, slika 2.2 prikazuje kako se koriste dva indeksa: Country (država) i Gender (spol), da bi se izvršio sljedeći upit:

```
SELECT CUSTOMER_ID
FROM CUSTOMER_TABLE
WHERE COUNTRY = 'UK'
AND GENDER = 'M'
```

Uzimanjem presjeka između skupa pokazivača koji pokazuju na ljude koji žive u UK i skupa pokazivača koji pokazuju na muškarce (M), mogu se identificirati svi pokazivači na muškarce koji žive u UK.

Konačno, može se koristiti *jednostruki indeks (single index)* ili nekoliko indeksa, koji zajedno pokrivaju samo podskup tipova atributa potrebnih za ključ za pretraživanje. U tom slučaju, indeks(i) se mogu koristiti za povrat samo onih zapisa koji odgovaraju pripadnim predikatima. Svaki od tih zapisa mora biti dohvaćen i testiran za usklađenost s predikatima povezanih s neindeksiranim tipovima atributa. Efikasnost ovakvog pristupa ovisi ponajviše o faktorima filtera predikata upita povezanih s indeksiranim tipovima atributa. Čim su više predikati selektivni, tim se manje n-torki podataka dohvaća uzalud jer se na kraju ne slažu s predikatima upita povezanim s neindeksiranim tipovima atributa. Najekstremniji slučaj je kad je jedan od indeksiranih tipova atributa primarni ključ ili neki drugi kandidat za ključ. U tom slučaju je ukupan broj zapisa koji se dohvaća jednak nula ili jedan, stoga je većina neučinkovitosti ublažena.

Vratimo se primjer sa slike 2.2 te ostanimo pri istom upitu. Pretpostavimo da postoji samo indeks na ”Country”. Taj indeks dohvaća sve kupce koji žive u UK, a za te n-torke se pregledava atribut tipa ”Gender”. Zadržana su dva reda, gdje je spol ”M”, a odbacuje se jedan red, gdje je spol ”F”. Analogno, kad bi postojao samo indeks ”Gender”, dohvatilo bi se sedam redova, od kojih bi se odbacilo pet n-torki. Kad bi se koristio indeks multistupca na (Country, Gender), dva reda bi se dohvatila i oba zadovoljavala kriterij. Ne bi bilo nepotrebnih dohvaćanja n-torki. Kad bi uzeli u obzir faktore filtera, uočljivo je da je efikasnost viša kod korištenja predikata vezanog za ”Country”.

Optimizator, kao što je pokazano, mora donijeti kompleksne odluke pri upitima s previše predikata. Najefikasnije alternativa je indeks multistupca preko svih ovih predikata: njegov FF jednak je FF-u upita pa su dohovaćene samo n-torke podataka koje zadovoljavaju upit. Ipak, ovakav pristup postaje neizvediv ako se puno tipova atributa koristi kod predikata upita jer bi indeksi postali preveliki i redundancija bi bila potrebna za upite s predikatima za samo podskup tipova atributa. Alternativno, više indeksa može se kombinirati, nebi li pokrili što više predikata upita. Ovo zahtjeva više indeksa, ali s puno manjim ukupnim

brojem indeksa unosa nego kod složenih indeksa. Drugačije rečeno, čim je FF predikata upita više selektivan, tim je više poželjno koristiti indeks pripadnog tipa atributa u planu pristupa. Ako su FF povezani s neindeksiranim tipovima atributa vrlo selektivni, mnogo zapisa dohvaća se uzalud, što, naravno, nije efikasno. Najvažnije od svega je da broj pristupa blokovima najviše utječe na performansu, a ne na broj dohvaćenih n-torki. Zbog toga, učinak grupiranja se uvijek treba uzeti u obzir jer bi korištenje primarnog ili grupirajućeg indeksa moglo biti efikasnije nego korištenje sekundarnog, pogotovo za intervalne upite. Može se dogoditi da se optimizatoru "posreći", u smislu da se upit izvrši bez pristupa datoteka podataka, bazirano samo na temelju informacija u indeksu. U tom slučaju, tipovi atributa koji se pojavljuju u SELECT-u upita moraju biti indeksirani, a ne samo oni koji se koriste u ključu za pretraživanje. Uzmimo sljedeći primjer:

```
SELECT LASTNAME
FROM TABLE
WHERE COUNTRY='CROATIA'
AND GENDER='M'
```

Ako indeks multistupca ili kombinacija jednostupčanih indeksa postoji preko tipova atributa LastName, Country i Gender, tada su podaci u indeksu (indeksima) dovoljni da daju rezultat upita. U tom slučaju se pristup stvarnim n-torkama podataka može izbjeći, što je pogotovo efikasno ako se one sastoje od puno tipova atributa i treba im puno više vremena za dohvat nego unosima indeksa. *Pristup samo preko indeksa (Index-only access)* je, dakle, dodatan razlog za kreiranje indeksa multistupca. Specifična aplikacija pristupa samo preko indeksa je *join indeks*, što je indeks multistupca koji kombinira tipove atributa iz dvije ili više tablica na takav način da sadrži prethodno kalkuliran rezultat spoja (join) između tih tablica; na taj način se join upiti izvršavaju vrlo efikasno. Rezultat join upita se, dakle, može izvući samo na temelju indeksa, bez da se pristupa pripadnim bazičnim tablicama. Naravno, više tipova atributa je uključeno u indeks, što daje više negativni utjecaj performanse na upite ažuriranja, pošto i unosi indeksa također moraju biti ažurirani. Budući da n-torke podataka nisu dohvaćene, ne vrijedi da primarni ili grupirajući indeksi prinose bolje performanse od sekundarnih indeksa za pristup samo preko indeksa.

2.6 Skeniranje cijele tablice

Kada nema dostupnih indeksa za tipove atributa i tablice povezane s upitom, jedina opcija je da se cijela tablica linearno pretraži, što znači da će se svi blokovi diska u prostoru tablice dohvatiti, a njihove n-torke će biti pregledane u glavnoj memoriji, čak i ako na kraju ispadne da ne sadrže nijednu n-torku koja zadovoljava upit. Logično, čim je tablica veća, tim je takvo skeniranje manje efikasno. No, za male tablice ili za upite koji ionako zahtjevaju većinu n-torki, ovakvo skeniranje može biti efikasnije od korištenja indeksa. U

specifičnim situacijama u kojima su zapisi podataka poredani prema tipovima atributa u ključu za pretraživanje, efikasna alternativna metoda linearnom pretraživanju je binarno pretraživanje stvarnih datoteka podataka. Dizajner ili administrator baze poadataka uzima takve stvari u obzir pri donošenju odluka za indekse.

Značajno je napomenuti da nekoliko organizacijskih tehnika koriste *overflow zapise (overflow records)*. Kad se te tehnike primjene u kontekstu fizičke organizacije, treba se pobrinuti za taj overflow (prelijevanje). Overflow se dogodi kad se pohranjeni zapis doda ili ažurira, a ne "stane" u poziciju kojoj pripada prema primjenjivoj organizacijskoj tehnici. Zapis je tada pohranjen na neku drugu lokaciju, obzirom na neku odabranu tehniku baratanja overflow-om. Naravno, overflow ima negativni utjecaj na efikasnost grupiranih indeksa; u smislu da čim je više n-torki krivo locirano, tim se više blokova dodatno mora pristupiti da bi se dohvatili podaci u pripadnom redoslijedu. Očito je da ako broj overflow zapisa postane prevelik, trebalo bi sortiranu tablicu reorganizirati.

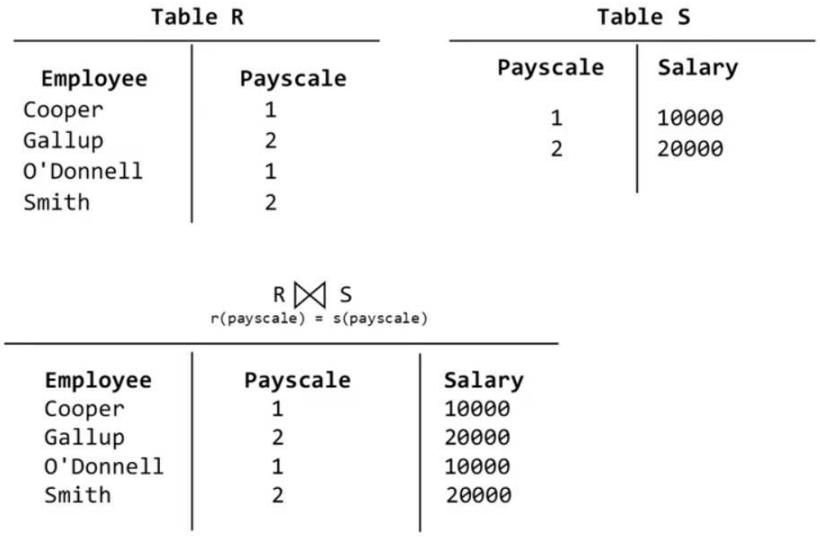
2.7 Implementacija prirodnog spoja (join)

Organizacija baze podataka trebala bi udovoljiti upitima koji koriste više tablica i to pomoću spoja (join) u SQL-u. *Join* upit dozvoljava korisniku da kombinira ili spoji podatke iz više tablica odjednom. Join upit između dvije tablice specificira selekcijski kriterij koji međusobno povezuje n-torke između tablica, prema takozvanom join operatoru. Join operacija je jedna od vremenski najzahtjevnijih operacija u *RDBMS-u (relacijski DBMS)*[7].
Uzmimo za svrhu ovog odjeljka *inner join (unutarnje spajanje)*. Generalna oznaka za inner join između tablica R i S je:

$$R \bowtie S$$
$$r(a)\ \theta\ s(b)$$

gdje θ operator specificira join uvjet, koji određuje koje n-torke tablice R se kombiniraju s kojim n-torkama tablice S. Skup atributa a-torke r u tablici R je uspoređen sa skupom atributa b-torke s tablice S. Operator usporedbe može sadržavati operatore jednakosti (=) i nejednakosti ($\leq, \geq$). Ovdje ćemo se ograničiti na spojeve bazirane na operatoru jednakosti.

[7] DBMS baziran na relacijskom modelu podataka



Slika 2.3: Primjer join naredbe

n-torke se dohvate i povezuju obzirom na join uvjet te se kombiniraju u objedinjeni rezultat. Primjer je pokazan slikom 2.3. Objasnimo tri glavna algoritma fizičke implementacije spoja: *nested-loop join, sort-merge join, hash-join*.

Algoritam ugnježđenih petlji (nested-loop join)

Kod *algoritma ugnježđenih petlji*, jedna je tablica označena kao *unutarnja (inner)*, dok druga postaje *vanjska (outer)*. Za svaki red vanjske tablice, sve n-torke unutarnje tablice su dohvaćene i uspoređene s trenutnom n-torkom vanjske. Ako je join uvjet zadovoljen, oba reda su spojena i stavljena u buffer (izlazni međuspremnik). Unutarnja tablica se obilazi toliko puta koliko ima n-torki u vanjskoj tablici. Algoritam je sljedeći:

$$R \bowtie S$$
$$r(a) = s(b)$$

```
Označi S kao vanjsku tablicu
Za svaku n-torku s u S radi:{
    za svaku n-torku r u R radi:{
        ako r(a)=s(b) tada {spoji r sa s
        i stavi ih u buffer;}
    }
}
```

Budući da se unutarnja tablica obilazi za svaki red vanjske, ovaj je pristup efikasan dokle god je unutarnja tablica vrlo mala ili ako unutarnji model podataka pruža alate za efikasni pristup unutarnjoj tablici, primjerice primarnim ili grupirajućim indeksom preko join stupaca unutarnje tablice. Ovaj pristup je također efikasan ukoliko je FF drugih predikata upita vrlo restriktivan, uzimajući u obzir n-torke unutarnje tablice koje se prate.

Algoritam zasnovan na sortiranju i sažimanju (sort-merge join)

Kod *algoritma zasnovanog na sortiranju i sažimanju*, n-torke u obje tablice su prvo sortirane obzirom na tipove atributa koji sudjeluju u join uvjetu. Obje tablice se obilaze tim redoslijedom, a n-torke koje zadovoljavaju join uvjet su kombinirane i stavljene u buffer. Algoritam je sljedeći:

$$R \bowtie S$$
$$r(a)\ \theta\ s(b)$$

```
1. razina:
    sortiraj R obzirom na r(a);
    sortiraj S obzirom na s(b);
2. razina:
    čitaj prvu skupinu n-torki r iz R;
    čitaj prvu skupinu n-torki s iz S;
    dok nismo prešli kraj ni od R ni od S:{
        ako s(b)<r(a) tada:
            pokušaj čitati sljedeću skupinu n-torki s iz S;
        inače ako r(a)<s(b) tada:
            pokušaj čitati sljedeću skupinu n-torki r iz R;
        inače:{
            spoji svaki zapis iz tekuće skupine n-torki r
            sa svakim zapisom iz tekuće skupine n-torki s
            i stavi ih u buffer;
            pokušaj čitati sljedeću skupinu n-torki r iz R;
            pokušaj čitati sljedeću skupinu n-torki s iz S;
            }
        }
    }
```

Za razliku od algoritma ugnježđenih petlji, ovdje se svaka tablica obilazi samo jednom. Takav pristup je prigodan ako obje tablice imaju puno n-torki koje zadovoljavaju predikati i-ili ako ne postoje indeksi preko join stupaca. Algoritam zasnovan na sortiranju i

sažimanju je više efikasan ako su n-torke u R i-ili u S već fizički poredane u sortiranim tablicama obzirom na tipove atributa korištenih u join uvjetu, zbog prve razine algoritma.

Algoritam zasnovan na hash funkciji i razvrstavanju (hash join)

S *algoritmom zasnovanim na hash funkciji i razvrstavanju*, *hashing* algoritam je primjenjen na vrijednosti tipova atributa uključenih u join uvjet za tablicu R. Obzirom na dane *hash* vrijednosti, pripadnim retcima su tada dodijeljeni pretinci (*buckets*) u hash datoteci. Isti je algoritam tada primjenjen na join tipove atributa druge tablice S. Ako hash vrijednost tablice S ukazuje na neprazni pretinac u hash datoteci, pripadne n-torke R i S tablica se uspoređuju obzirom na join uvjet. Ako je zadovoljen, n-torke R i S tablice se spoje i stave u buffer. Budući da postoji mogućnosti kolizije, odnosno da hash funkcija za različite vrijednosti atributa poprima istu vrijednost, n-torke s različitim vrijednostima stupaca još uvijek mogu biti dodijeljene istim pretincima, stoga neće nužno sve n-torke dodijeljene istim pretincima zadovoljiti join uvjet. Performansa ovog pristupa ovisi o veličini hash datoteke. Ukoliko se hash struktura može prikazati u unutarnjoj memoriji, a ne u fizičkoj datoteci, hash join je vrlo efikasan.

Poglavlje 3

Podsustav za spremanje podataka

3.1 Diskovni niz i RAID tehnologija

Iako se kapacitet *tvrdog diska (hard disc drive - HDD)* znatno povećao u poslijednja dva desetljeća, isto ne vrijedi i za performanse. Čak je efikasnije kombinirati više manjih fizičkih diskova u jedan veći. To rezultira dvjema pogodnostima, između ostalog. Prvo, raspodjela podataka preko više fizičkih diskova dozvoljava paralelno dohvaćanje podataka, što naravno daje puno bolju performansu. Drugo, svaki dodatni disk povećava rizik kvara, no ako je uvedena određena mjera redundancije, taj je rizik ublažen, stoga je više diskova zajedno pouzdanije od jednog. Oba su razloga temelj *RAID (Redundant Array of Independent Disks)* koncepta. RAID je tehnologija kod koje su standardni HDD-ovi upareni s pripadnim upravljačem tvrdog diska (*RAID upravljač)* da bi ih prikazao kao jedan logički disk. Sljedeće tehnike se odnose na RAID:

- **Trakiranje podataka**: Podvrsta datoteke podataka (zvani *trake*) distribuirani su preko više diskova tako da se čitaju i pišu paralelno. Mogu se sastojati od individualnih bitova ili čitavih blokova diska. Sa *n* diskova, bit ili blok *i* je zapisan u disku (i mod n) + 1.
  Bit-razina trakiranja podataka je proces kod kojeg je bajt razdvojen na osam individualnih bitova, koji se distribuiraju po slobodnim diskovima. Tako svaki disk sudjeluje u svakoj operaciji čitanja ili pisanja. Stoga je broj dohvaćenih bitova unutar jednog pristupa povećan za faktor jednak broju diskova, što rezultira većom brzinom prijenosa.
  Kod *blok-razina* trakiranja podataka, svaki je blok u cijelosti pohranjen unutar jednog diska, ali su pripadni blokovi iste datoteke raspodijeljeni preko diskova. Tako je pristup individualnom bloku neovisan od drugih blokova. Efikasnost pristupa bloku nije povećana, ali različiti procesi mogu paralelno čitati blokove iz istog logičkog

diska, što smanjuje vrijeme čekanja. Dodatno, jedan veliki proces može paralelno čitati više blokova odjednom, čime se smanjuje vrijeme prijenosa.

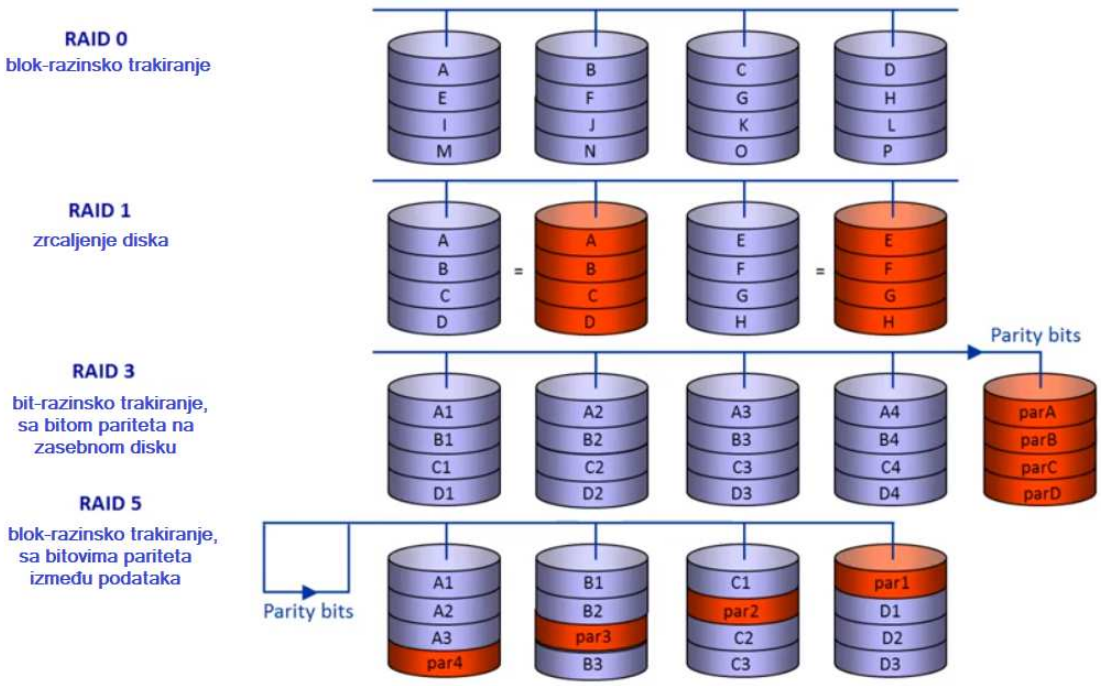
- **Redundantnost**: Redundantni podaci pohranjeni su zajedno sa originalnim podacima, da bi povećali pouzdanost. Redundantni podaci sadrže kodove ispravka i kodove detekcije grešaka. Bitovi pariteta su vrsta kodova detektiranja grešaka, koji dodaju jedan redundantni bit seriji bitova tako da ukupan broj 1-bitova u nizu bude uvijek paran ili uvijek neparan. Na taj način se mogu detektirati greške. Ako se bit ne može pročitati, njegova se vrijednost može deducirati na temelju ostalih bitova u bitovima pariteta.

- **Zrcaljenje diska**: Ovo je ekstremna vrsta redundancije, gdje za svaki disk postoji jednaka kopija, zvana *zrcalo*, koja sadrži iste podatke. Postoji vjerojatnost da se dogodi greška u disku, no u slučaju kopija, ta je vjerojatnost eksponencijalno manja za grešku kod obiju kopija. S druge strane, zrcaljenje zauzima puno više mjesta nego kodovi za detektiranje grešaka.

Postoje više RAID konfiguracija - zovemo ih *RAID razine*. Pojam ”razina” ovdje može zavarati jer u ovom kontekstu viša razina ne mora značiti ”bolja” razinu. Ovdje će razine predstavljati razne konfiguracije, sa različitim kombinacijama tehnika koje smo prije spomenuli, rezultirajući u različitim karakteristikama u smislu performanse i pouzdanosti. U tablici navodimo svojstva pripadnih originalnih RAID razina.

RAID razina	**Opis**	**Tolerancija kvara**	**Performansa**
0	blok-razinsko trakiranje	nema korekcija greški	poboljšana performansa čitanja i pisanja, zbog paralelizma
1	zrcaljenje diska	korekcija greški zbog kompletne duplikacije podataka	poboljšana performansa čitanja jer je omogućen pristup obiju diskovima različitim procesima paralelno; lošija performansa pisanja: podaci se moraju pisati dva puta
2	bit-razinsko trakiranje, sa posebnim checksum diskom	korekcija greški Hamming kodovima	poboljšana performansa čitanja kroz paralelizam; sporija performansa pisanja zbog kalkulacija checksuma
3	bit-razinsko trakiranje, sa bitom pariteta na zasebnom paritetnom disku	korekcija greški kroz bitove pariteta	poboljšana performansa čitanja kroz paralelizam; sporija performansa pisanja, no manje kalkulacija nego kod RAID 2
4	blok-razinsko trakiranje, sa posebnim checksum diskom	korekcija greški kroz bitove pariteta	nema poboljšanja performanse čitanja za individualne blokove; sporija performansa pisanja, kao kod RAID 3
5	blok-razinsko trakiranje, sa bitom pariteta na zasebnom paritetnom disku	korekcija greški kroz bitove pariteta	ista performansa čitanja kao kod RAID 4; bolja performansa pisanja nego kod RAID 4: diskovi pariteta su distribuirani po diskovima podataka

Na slici 3.1 prikazani su neke RAID razine. Slova predstavljaju blokove, a kombinacija slova i brojeva predstavlja bitove u bloku. Obojani diskovi označuju redundanciju.



Slika 3.1: Ilustracija RAID razina 0, 1, 3 i 5

RAID 0 se koristi ako se preferira performansa nad tolerancijom kvara. RAID 1 se ponajviše koristi za vrlo kritične podatke. Takva je konfiguracija najbolja za kontinuitet jer će uvijek postojati kopija u slučaju kvara diska. Ostalim RAID razinama koje uključuju redundanciju potrebno je čitanje drugih diskova da bi rekonstruirali podatke s pokvarenog diska. RAID 1 zato ima veći stupanj redundancije, stoga mu treba više kapaciteta. Jedan od popularnijih je RAID 5, koji balansira performansu čitanja i pisanja, efikasnost skladišta te toleranciju kvara. RAID razina 0+1 kombinira svojstva razina 0 i 1. Ima više razina; neke su zastarjele, neke su se dodale kasnije.

3.2 Podsustav za spremanje podataka

Tipični problem pohrane obično se rješava kombiniranjem uređaja za pohranu i to primjenom RAID-a, u veće entitete koje zovemo *podsustav za spremanje podataka*. *Podsustav*

za spremanje podataka možemo definirati kao posebni, vanjski[1] entitet sa određenom razinom "onboard" inteligencije koja se sastoji od barem dva uređaja za pohranu.

Podsustav za spremanje podataka proizlazi iz činjenice da trošak upravljanja pohranom zna premašiti trošak kupovine hardvera za pohranu. Također, javila se potreba za više fleksibilnom i distribuiranom arhitekturom pohrane koja nudi i mogućnosti upravljanja centraliziranjem. Moderni podsustavi za spremanje podataka često imaju i umreženu pohranu, spojenu na vrhunske servere preko brzih interveza. Mrežna topologija pruža transparentnu povezanost između servera i uređaja za pohranu[2] te time zapravo funkcionira kao "crna kutija"[3] pohrane. Ona kombinira velike kapacitete pohrane sa značajkama kao što su brzi transferi podataka, visoka dostupnost i sofisticirani alati upravljanja.
Glavne karakteristike kvalitetnog podsustava za spremanje podatka su sljedeće:

- prilagodba visokoj performansi pohrane podataka
- pružati opciju skalabilnosti, nezavisno od kapaciteta pohrane i servera
- podržavanje distribucije podataka i transparentnost lokacija[4]
- podržavanje interoperabilnosti te dijeljenja podataka između heterogenih sistema i korisnika
- pouzdanost i gotovo stalna dostupnost
- zaštita od gubitka podataka, neispravnosti hardvera i softvera te od "nasilnih" korisnika i hakera
- poboljšanje upravljanja te reduciranje troška upravljanja pomoću centraliziranih upravljanja objekitma pohrane koja mogu biti distribuirana

Tehnologije koje realiziraju podsustave za spremanje podataka možemo klasificirati obzirom na tri kriterija: *povezanost*, *medijum* te *I/O protokol*.

1. **Povezanost** se odnosi na način na koji su uređaji za pohranu povezani s procesorima i-ili serverima. Vezanje može biti direktno ili mrežno, što je ilustrirano na slici 3.2.
    - *Diretkno vezanje* odnosi se na vezanje uređaja za pohranu sa 1-na-1 vezom između servera i uređaja za pohranu.

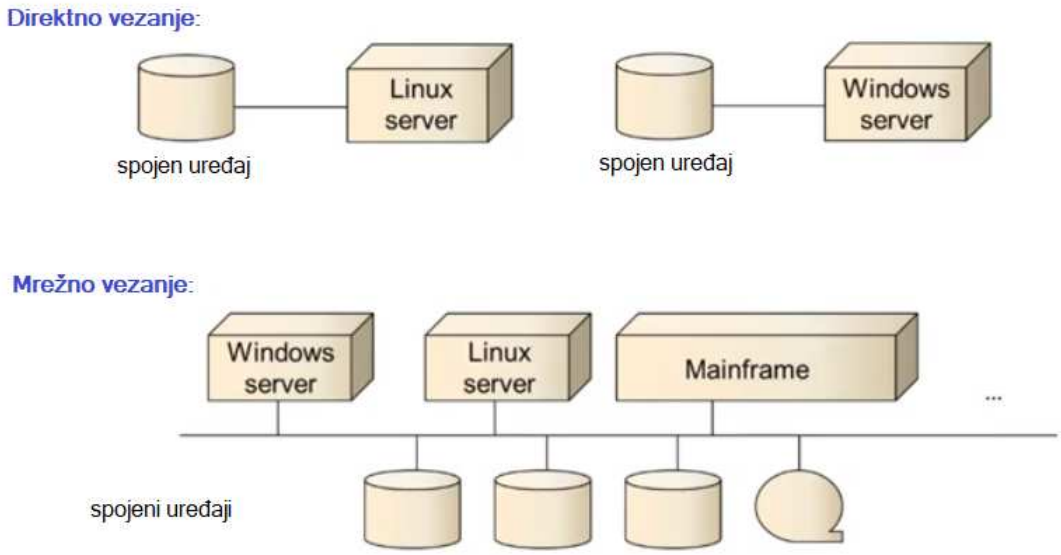
[1] vanjski obzirom na server

[2] bilo koji server s bilo kojim uređajem za pohranu

[3] Crna kutija može se definirati kao sistem kojeg se promatra u smislu njegovih ulaza i izlaza, gdje njegov unutarnji rad nije relevantan.[1]

[4] korisnici i aplikacije su izolirani od kompleksnosti distribucije podataka

- *Mrežno vezanje* odnosi se na vezanje uređaja za pohranu u više-na-više veza sa pripadnim serverima preko mrežne tehnologije.



Slika 3.2: Vrste povezanosti

2. **Medijum** se odnosi na fizičko "kabliranje" i pripadni protokol za prepoznavanje povezanosti, odnosno transfer podataka između medijuma pohrane i servera. Razlikujemo tri glavne vrste tehnologija:

   - *SCSI (Small Computer System Inferface)* je desetljećima bio standard za takvo povezivanje. Uglavnom se koristi za visoku performansu i veliki kapacitet radnih prostora i servera. SCSI koristi dva elementa: *set naredbi* za komunikaciju s uređajima za pohranu i specifikacije za *niskorazinski protokol i kabliranje* da bi se vršio transfer SCSI naredbi i podataka između servera i uređaja za pohranu. Potonji je bitniji kod medijuma.
   - *Ethernet* je standardni medijum za LAN-ove (local area network) i ponekad WAN-ove (wide area network).
   - *FC (Fibre Channel)* je "najsvježji" medijum, specifično konstruiran da bi povezao visokorazinske sisteme pohrane sa serverima.

3. **I/O protokol** označava skup naredbi za komunikaciju s uređajem za pohranu. Pomoću ovakvih visokorazinskih protokola, transportirani bitovi poprimaju značenje I/O zah-

tjeva izmjenjenih između servera i uređaja za pohranu. U tom kontekstu, razlikujemo sljedeće vrste protokola:

- *I/O protokol na razini bloka*: I/O naredbe definirane su na razini zahtjeva za individualnim blokovima na uređaju za pohranu. Skup SCSI naredbi koristi se u ovu svrhu.
- *I/O protokol na razini datoteke*: Naredbe su definirane na razini zahtjeva za cijele datoteke na uređaju za pohranu. Protokol je neovisan o uređaju jer je smješten na razini datoteke pa fizička pozicija bloka podataka na uređaju za pohranu nema utjecaj na njega. Rašireni I/O protokoli na razini datoteke su NFS (Network File System), porijekla u Linuxu, i CIFS (Common Internet File System), poznat kao SMB (Server Message Block), popularniji kod Windowsa. Neki protokoli internet aplikacije se mogu svesti pod ovaj zajednički nazivnik, kao što su HTTP (HyperText Transfer Protocol) i FTP (File Transfer Protocol).

Obzirom na te kriterije, možemo razlikovati između sljedećih arhitektura: Directly Attached Storage (DAS), Storage Area Network (SAN), Network Attached Storage (NAS), NAS prolaz (NAS gateway) i Skladište preko IP-a (Storage over IP - iSCSI).

DAS (Directly Attached Storage)

Kod DAS-a, uređaji za pohranu spojeni su na individualne servere direktno. Koristi se blok-razinski I/O protokol: serveri komuniciraju s uređajima za pohranu SCSI I/O naredbama. Za medijum moguće je koristiti standardni SCSI kabel, Fibre Channel ili čak Ethernet kabel. Svaki uređaj za pohranu spojen je na jedan ili dva[5] servera. Mreža se ne koristi u ovoj konfiguraciji, iako serveri i klijenti mogu biti spojeni na standardnu IP-mrežu. Postavljanje DAS-a je najjednostavnije i najjeftinije, no ne pruža neke mogućnosti za centralizirano upravljanje pohranom. Štoviše, pošto postoji jedinstven put između individualnih servera i jedinica pohrane, ovakav pristup se ne snalazi dobro s kvarovima.

SAN (Storage Area Network)

Sav prijenos podataka, vezano za pohranu, vrši se preko pripadne mreže. Ovdje se također koristi blok-razinski I/O protokol, ali ovaj put mreža dozvoljava da se bilo koji server povezuje s bilo kojim uređajem za pohranu. Najčešće se kao medijum koristi Fibre Channel. Moguće je i preko Etherneta, o čemu će biti riječ kasnije. Klijenti i serveri mogu komunicirati preko standardnog LAN-a ili WAN-a. Iz istog razloga, SAN je superiorniji od DAS-a u smislu dostupnosti. Jedna od SAN-ovih mogućnosti je dijeljenje podataka pa zbog toga

[5] u slučaju kvara; da se može prebaciti na drugi server

i ima znatno manje nepotrebnih redunantnih podataka; nekih ipak ima jer su redundantne kopije i dalje potrebne zbog dostupnosti. SAN obično daje najbolju performansu: pruža brza povezivanja te za to ima vrlo efikasni FC medijum koji je kompatabilan s uređajima više inteligencije pa zato mogu obavljati određene zadatke autonomno; dodatni razlog za bolju performansu je da oslobađa LAN ili WAN od prometa vezanih za pohranu, što rezultira većom mrežnom prolaznosti za zadatke vezane uz proces. Primjerice, podržava rezerve (*backup-ove*) za koje nije potreban LAN ili rezerve za koje nije potreban server. Dodatne pogodnosti su fleksibilnost i skalabilnost u smislu dodavanja servera i uređaja u SAN. SAN također može spajati šire udaljenosti nego direktne veze između uređaja za pohranu i servera. Povezanost među serverima dozvoljava centralizirano upravljanje. No, SAN tehnologija je poprilično kompleksna i skupa, stoga je obično koriste velike kompanije.

NAS (Network Attached Storage)

Za razliku od SAN-a, NAS je jednostavan te ima jeftin način organiziranja objekata pohrane u mrežu. NAS je specijalizirani uređaj za pohranu datoteka koji se može jednostavno ”uključiti u” TCP/IP LAN ili WAN, stoga mu je medijum Ethernet. Jednostavnije rečeno, NAS uređaj je neka vrsta servera datoteka[6] koja se sastoji od integrirane kombinacije procesora, operacijskog sustava te seta tvrdih diskova. No, operacijski sustav ima samo minimalne potrebne operacije: dijeljenje datoteka mreži. Otuda vidimo da je NAS i manje kompleksan od ”normalnog” servera. NAS uređaji se, za razliku od SAN-a, pristupaju I/O protokolom baziranim na datotekama, kao što su CIFS, NFS, HTTP ili FTP. NAS nudi sistem datoteka mreži, gdje se zahtjevi datoteka prevode NAS-ovim unutarnjim procesorom u SCSI blok I/O naredbi na tvrde diskove. Klijenti to neće vidjeti - njima NAS izgleda kao da se ponaša poput ”normalnog” servera datoteka. Vrijedi napomenuti da postoje puno različitih NAS uređaja, od onih s poprilično ograničenim funkcijama do onih s vrlo razvijenim funkcijama. Glavna prednost NAS tehnologije je to što pruža vrlo fleksibilne, jednostavne i jeftine usluge za dodavanje dodatne pohrane mreži. Naravno, performansa NAS-a nije tako dobra kao kod SAN-a, iz razloga što sav promet vezan uz pohranu prolazi kroz standardni LAN ili WAN, a ne određenom mrežom. Ipak, pristup datotekama olakšava stvari kad je glavni cilj dijeliti nestrukturirane datoteke na mreži, no translatiranje pristupa na razini datoteka u pristup na razini blokova je manje efikasno.

NAS prolaz (NAS gateway)

NAS prolaz sličan je NAS-u, samo što ne koristi tvrde diskove. Sastoji se samo od procesora i operacijskog sustava. Može se ”uključiti u” TCP/IP LAN ili WAN, a može se i povezati s vanjskim diskovima, što bi se omogućilo DAS ili SAN tehnologijom. Tako se

[6] Server datoteka pohranjuje te daje pristup cijelim datotekama podijeljenim unutar mreže.

primaju I/O naredbe na razini datoteka od servera spojenih na LAN ili WAN i prevode u SCSI blok I/O naredbi da bi se pristupilo vanjskim uređajima. NAS prolaz je više fleksibilan u vezi izbora diskova te je i više skalabilan od NAS uređaja. Štoviše, dozvoljava uključivanje postojećih diskova u LAN ili WAN, čime je čitav sadržaj dostupan cijeloj mreži. Diskovi su tako dostupni I/O-om na razini datoteka putem NAS prolaza i I/O-om na razini blokova putem DAS-a ili SAN-a. NAS prolaz možemo, dakle, shvatiti kao hibrid NAS-a i SAN-a.

Skladište preko IP-a (Storage over IP - iSCSI)

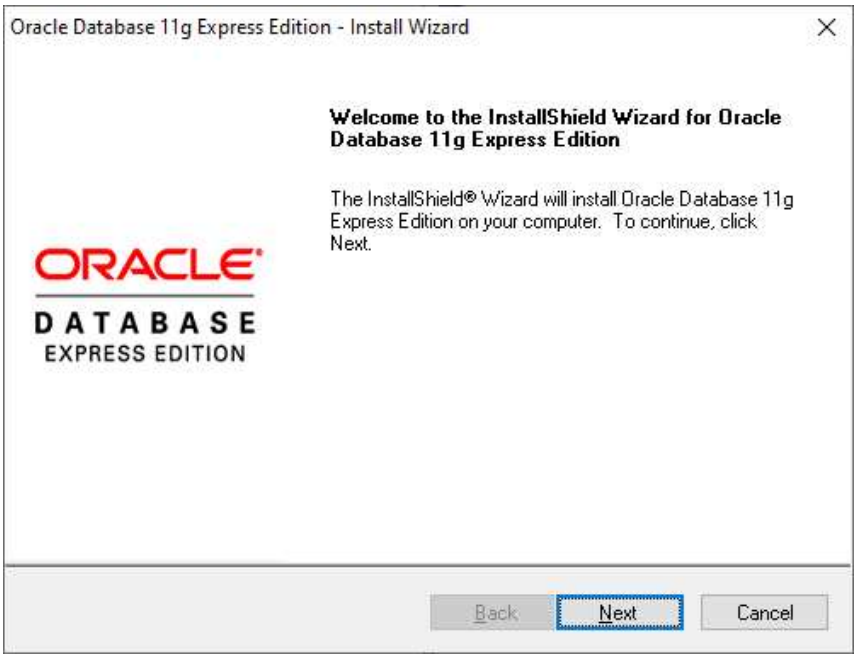
iSCSI, nazvan *internet SCSI* ili *Skladište preko IP-a (Storage over IP)*, slične je postave kao SAN, ali umjesto FC-a koristi puno poznatiji Ethernet kao medijum. SCSI I/O naredbe na razini blokova šalju se TCP/IP mreži koja može biti normalan LAN ili WAN, ili zasebna mreža posvećena pohrani, ali ipak bazirana na LAN tehnologiji. Iako sliči na SAN, iSCSI se može koristiti za direktne veze između servera i uređaja za pohranu, kao i DAS. Rješenja iSCSI-a su poput hibrida SAN-a i NAS-a: dozvoljavaju pristup diskovima na razini blokova, kao SAN, ali su bazirani na Ethernetu, kao NAS. Budući da Ehternet nije tako sklon problemima te je vrlo kompatabilan, iSCSI je praktički popularan u manjim i srednjim poduzećima. Također, iSCSI može pokriti veće udaljenosti nego SAN sa FC-om, ali je obično sporiji od FC-a.

Poglavlje 4

Studijski primjer

4.1 Instalacija baze podataka

U ovom dijelu rada realizira se vlastita baza na kojoj će se testirati razne varijante fizičke organizacije. Softver koji će se koristiti je ORACLE XE (verzija 11g) na platformi Windows 10. Baza se slobodno instalira u nekomercijalne svrhe bez naknade. Instalacija je jednostavna, vrši se pokretanjem programa instalacije, kao što prikazuje slika 4.1.



Slika 4.1: Instalacijski prozor Oracle baze podataka

Nakon završetka instalacije, kreirana je ORACLE baza podataka sa svim osnovnim postavkama, zajedno sa fizičkom i logičkom strukturom.

Fizička struktura baze podataka

Fizička struktura baze poadataka definirana je direktorijom 'C:/oraclexe'. Osim svih instaliranih aplikacija najvažniji je direktorij: 'C:/oraclexe/app/oracle/oradata/XE' u kojem su kreirane datoteke koje sadrže podatke baze podataka, što se vidi na slici 4.2.

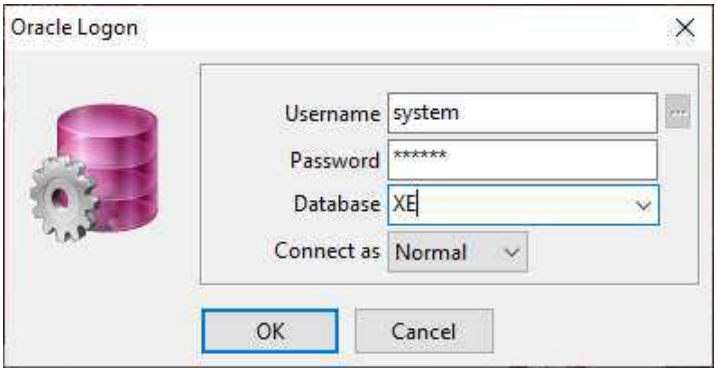
Slika 4.2: Mapa s datotekama potrebnim za bazu podataka

- *CONTROL.DBF* - kontrolna datoteka koja bilježi fizičku strukturu baze podataka; njom upravlja ORACLE i bez nje je nemoguće otvoriti bazu podataka.
- *SYSTEM.DBF* - datoteka koja sadrži podatke sistemskih tablica koje se kreiraju pri samoj inicijalnoj instalaciji ORACLE baze
- *SYSAUX.DBF* - sadrži podatke pomoćnog prostora tablica SYSAUX koji je pomoćni prostor prostoru tablica SYSTEM
- *TEMP.DBF* - datoteka u kojoj se smještaju privremene obrade podataka, kao što je, na primjer, sortiranje
- *UNDOTBS1.DBF* - datoteka u kojoj se smještaju tablice za "poništavanje" promjena na podacima, odnosno promjene do potvrde transakcija
- *USERS.DBF* - datoteka u kojoj se pohranjuju tablice konkretnih aplikacija koje nisu niti sistemske niti stvorene samim ORACLE-om

Logička struktura baze podataka

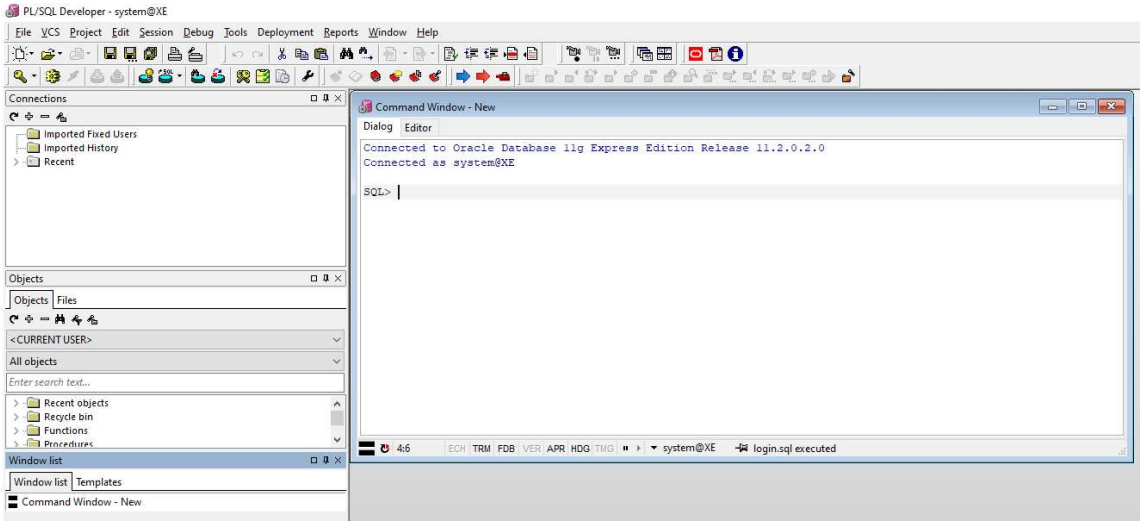
Nakon instalacije ORACLE baze, odmah se može koristiti SQL Plus za ulazak u ORACLE bazu podataka te upravljanje istom. U svrhu ovog rada, izabrana je dodatna ORACLE

aplikacija PL/SQL Developer jer pruža više mogućnosti za rad nad bazom podataka. Instalacija je također vrlo jednostavna; ulazi se nakon početnog LOGIN ekrana prikazanog na slici 4.3.



Slika 4.3: Oracle LOGIN ekran

Pri samoj instalaciji kreiraju se osnovni korisnici (users) baze, a najvažniji je SYSTEM kojem je pri instalaciji pridijeljena šifra (password) ORACLE. Nakon ulaska otvara se sljedeći prozor kao na slici 4.4.



Slika 4.4: PL/SQL sučelje

Kao i u SQL Plusu, SQL naredbe pišu se u naredbenom prozoru (Command Window-u). Lakše je raditi jer se s redovima naredbi može lakše upravljati (copy, paste...). Jednako kao što je ORACLE kreirao svoju osnovnu fizičku strukturu, tako je kreirana i logička

struktura baze. Pri samoj instalaciji kreirani su korisnici (users), prostori tablica (tablespace), tablice i indeksi.
Sljedećom naredbom može se utvrditi koliko još ima slobodnog prostora u kojem prostoru tablica:

```
SQL> select tablespace_name, sum(bytes)
from sys.dba_free_space
group by tablespace_name;

TABLESPACE_NAME SUM(BYTES)
--------------------------
SYSAUX          39124992
UNDOTBS1        22020096
USERS           102170624
SYSTEM          9306112
```

Takva informacija je korisna administratoru baze podataka da bi mogao povećavati prostor baze podataka kada se baza počne komercijalno koristiti te puniti podacima.
Kreirani su sljedeći korisnici:

```
SQL> select username, user_id, default_tablespace
from sys.dba_users;

USERNAME            USER_ID     DEFAULT_TABLESPACE
------------------------------------------------------------
SYS                 0           SYSTEM
SYSTEM              5           SYSTEM
ANONYMOUS           35          SYSAUX
APEX_PUBLIC_USER    45          SYSTEM
FLOWS_FILES         44          SYSAUX
APEX_040000         47          SYSAUX
XS$NULL             2147483638  SYSTEM
OUTLN               9           SYSTEM
MDSYS               42          SYSAUX
CTXSYS              32          SYSAUX
XDB                 34          SYSAUX
HR                  43          USERS
```

Kreirane su sistemske tablice kojima upravlja ORACLE, a u kojim se nalaze sve informacije logičke strukture baze podataka, uključujući i sve koje nastaju daljnjim kreiranjem

tablica i ostalog u svrhu izrade konkretne aplikacije. Po samoj instalaciji ukupno se kreiralo 1663 tablica, no prikazat ćemo samo neke od njih. Sistemska tablica ALL_TABLES sadrži sve podatke o svakoj tablici, a struktura joj je prikazana slikom 4.5.

```
SQL> DESC sys.all_tables
Name                          Type         Nullable Default Comments
----------------------------- ------------ -------- ------- ----------------------------------------------------------------------------------------------------
OWNER                         VARCHAR2(30)                  Owner of the table
TABLE_NAME                    VARCHAR2(30)                  Name of the table
TABLESPACE_NAME               VARCHAR2(30) Y                Name of the tablespace containing the table
CLUSTER_NAME                  VARCHAR2(30) Y                Name of the cluster, if any, to which the table belongs
IOT_NAME                      VARCHAR2(30) Y                Name of the index-only table, if any, to which the overflow or mapping table entry belongs
STATUS                        VARCHAR2(8)  Y                Status of the table will be UNUSABLE if a previous DROP TABLE operation failed,
VALID otherwise
PCT_FREE                      NUMBER       Y                Minimum percentage of free space in a block
PCT_USED                      NUMBER       Y                Minimum percentage of used space in a block
INI_TRANS                     NUMBER       Y                Initial number of transactions
MAX_TRANS                     NUMBER       Y                Maximum number of transactions
INITIAL_EXTENT                NUMBER       Y                Size of the initial extent in bytes
NEXT_EXTENT                   NUMBER       Y                Size of secondary extents in bytes
MIN_EXTENTS                   NUMBER       Y                Minimum number of extents allowed in the segment
MAX_EXTENTS                   NUMBER       Y                Maximum number of extents allowed in the segment
PCT_INCREASE                  NUMBER       Y                Percentage increase in extent size
FREELISTS                     NUMBER       Y                Number of process freelists allocated in this segment
FREELIST_GROUPS               NUMBER       Y                Number of freelist groups allocated in this segment
LOGGING                       VARCHAR2(3)  Y                Logging attribute
BACKED_UP                     VARCHAR2(1)  Y                Has table been backed up since last modification?
NUM_ROWS                      NUMBER       Y                The number of rows in the table
BLOCKS                        NUMBER       Y                The number of used blocks in the table
EMPTY_BLOCKS                  NUMBER       Y                The number of empty (never used) blocks in the table
AVG_SPACE                     NUMBER       Y                The average available free space in the table
CHAIN_CNT                     NUMBER       Y                The number of chained rows in the table
AVG_ROW_LEN                   NUMBER       Y                The average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS     NUMBER       Y                The average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS           NUMBER       Y                The number of blocks on the freelist
DEGREE                        VARCHAR2(40) Y                The number of threads per instance for scanning the table
INSTANCES                     VARCHAR2(40) Y                The number of instances across which the table is to be scanned
CACHE                         VARCHAR2(20) Y                Whether the table is to be cached in the buffer cache
TABLE_LOCK                    VARCHAR2(8)  Y                Whether table locking is enabled or disabled
SAMPLE_SIZE                   NUMBER       Y                The sample size used in analyzing this table
LAST_ANALYZED                 DATE         Y                The date of the most recent time this table was analyzed
PARTITIONED                   VARCHAR2(3)  Y                Is this table partitioned? YES or NO
IOT_TYPE                      VARCHAR2(12) Y                If index-only table, then IOT_TYPE is IOT or IOT_OVERFLOW or IOT_MAPPING else NULL
TEMPORARY                     VARCHAR2(1)  Y                Can the current session only see data that it place in this object itself?
SECONDARY                     VARCHAR2(1)  Y                Is this table object created as part of icreate for domain indexes?
NESTED                        VARCHAR2(3)  Y                Is the table a nested table?
BUFFER_POOL                   VARCHAR2(7)  Y                The default buffer pool to be used for table blocks
FLASH_CACHE                   VARCHAR2(7)  Y                The default flash cache hint to be used for table blocks
CELL_FLASH_CACHE              VARCHAR2(7)  Y                The default cell flash cache hint to be used for table blocks
ROW_MOVEMENT                  VARCHAR2(8)  Y                Whether partitioned row movement is enabled or disabled
GLOBAL_STATS                  VARCHAR2(3)  Y                Are the statistics calculated without merging underlying partitions?
USER_STATS                    VARCHAR2(3)  Y                Were the statistics entered directly by the user?
DURATION                      VARCHAR2(15) Y                If temporary table, then duration is sys$session or sys$transaction else NULL
SKIP_CORRUPT                  VARCHAR2(8)  Y                Whether skip corrupt blocks is enabled or disabled
MONITORING                    VARCHAR2(3)  Y                Should we keep track of the amount of modification?
CLUSTER_OWNER                 VARCHAR2(30) Y                Owner of the cluster, if any, to which the table belongs
DEPENDENCIES                  VARCHAR2(8)  Y                Should we keep track of row level dependencies?
COMPRESSION                   VARCHAR2(8)  Y                Whether table compression is enabled or not
COMPRESS_FOR                  VARCHAR2(12) Y                Compress what kind of operations
DROPPED                       VARCHAR2(3)  Y                Whether table is dropped and is in Recycle Bin
READ_ONLY                     VARCHAR2(3)  Y                Whether table is read only or not
SEGMENT_CREATED               VARCHAR2(3)  Y                Whether the table segment is created or not
RESULT_CACHE                  VARCHAR2(7)  Y                The result cache mode annotation for the table
```

Slika 4.5: Tablica ALL_TABLES

Kao što se vidi prema velikom broju stupaca, odnosno atributa, ORACLE sadrži mnogo podataka o svakoj tablici. Postoji mnogo načina upravljanja tablicom, ovisno o načinu njezinog kreiranja (CREATE TABLE. . . .) ili kasnijeg mijenjanja (ALTER TABLE. . .).

4.2 Upravljanje bazom podataka

Nakon što smo instalirali inicijalnu ORACLE bazu, može se krenuti s konkretnom primjenom stvaranja aplikacije prema potrebi krajnjeg korisnika. Osnova kreiranja neke aplikacije je naravno tablica podataka (table), no važno je dobro pripremiti sve za to, kako bi se bolje moglo upravljati performansama baze, brzinom izvođenja i ostalim. Kao što je opisano na početku ovog poglavlja, ORACLE je kreirao svoju logičku strukturu baze na koju se mogu početi kreirati tablice i ostalo. Kreiranje korisničkih tablica važno je ipak raditi u posebnom korisniku (user-u), a ne SYSTEM ili nekim od početnih korisnika. U daljnjim će se primjerima prikazati kreiranje korisnika, prostora tablica, tablica te će se prikazati što se događa u ORACLE-u.

Kreiranje usera

```
SQL> grant DBA to JAKOV identified by VOKAJ;
```

Tom se naredbom stvorio korisnik JAKOV sa zaporkom VOKAJ koji ima svoj prostor za stvaranje tablica i ostalog. Pritom mu se dodijelilo nivo DBA, što je najviši nivo, koji se koristi na malim bazama kada nad bazom uglavnom radi jedan korisnik. U većim bazama podataka, koriste se CONNECT i RESOURCE privilegije za ograničavanje prava novom korisniku za rad u bazi. CONNECT je privilegija za dozvoljavanje spajanja na bazu podataka, dok je RESOURCE privilegija za upravljanje nekim resursom kao što je tablica.
Kod većih baza, više korisnika i aplikacija, dobro je nakon toga kreirati i zasebni prostor tablica vezan ili za korisnika ili za aplikaciju, kako bi se moglo lakše pratiti performanse (popunjenost, brzina,...). Kod manjih baza dovoljno je da novi korisnik koristi već postojeći prostor tablica USERS.

Kreiranje prostora tablica

```
SQL> create tablespace TEST
datafile 'c:\oraclexe\app\oracle\oradata\XE\test1.dbf' size 1G;

Tablespace created.
```

Jedan prostor tablica može imati više datoteka. Pripadajuća datoteka je fizička struktura i uvijek je iste veličine. Ovim primjerom alociralo se 1 GB prostora za buduće tablice koje će kreirati korisnik JAKOV.
Promotrimo (trenutno) stanje popunjenosti prostora u bazi podataka:

```
SQL> select tablespace_name, sum(bytes)
from sys.dba_free_space
group by tablespace_name;

TABLESPACE_NAME     SUM(BYTES)
------------------------------
SYSAUX              38928384
UNDOTBS1            8257536
USERS               102170624
TEST                1072693248
SYSTEM              9306112
```

Vidi se da se smanjio i slobodni prostor u UNDOTBS1 radi mogućnosti povratka akcije jer se nije napravila potvrda kreiranja.

U slučajevima kada se napuni slobodni prostor nekog prostora tablica, a u takvim slučajevima ORACLE javlja grešku, postoji mogućnost njegovog povećanja tako da se kreira nova datoteka[1] ili da se postojećoj promijeni veličina. U sljedećem primjeru povećat će se prostor tablica TEST za novu datoteku veličine 1G:

```
SQL> alter tablespace TEST
add datafile 'c:\oraclexe\app\oracle\oradata\XE\test2.dbf' size
1G';
```

Istu stvar smo također mogli napraviti tako da postojeću datoteku TEST1.DBF[2] povećamo na 2G, naredbom:

```
SQL> alter tablespace TEST
RESIZE datafile 'c:\oraclexe\app\oracle\oradata\XE\test1.dbf'
size 2G';
```

Kod prostora tablica vrijedi sljedeće: korisnika, tablicu ili indeks možemo kod kreiranja pridijeliti tom prostoru tablica tako da usmjerimo spremanje pripadajućih podataka u točno određeni prostor. Primjer za korisnika JAKOV je:

```
SQL> alter user JAKOV default tablespace TEST;
```

[1] Prostor tablica može imati više datoteka, a datoteka ne može biti u više prostora tablica odjednom.

[2] Kod kreiranja datoteka, ne raspoznaje se razlika između velikih i malih slova. Naravno, kod sadržaja tablica, bitno ih je razlikovati.

Kreiranje tablice

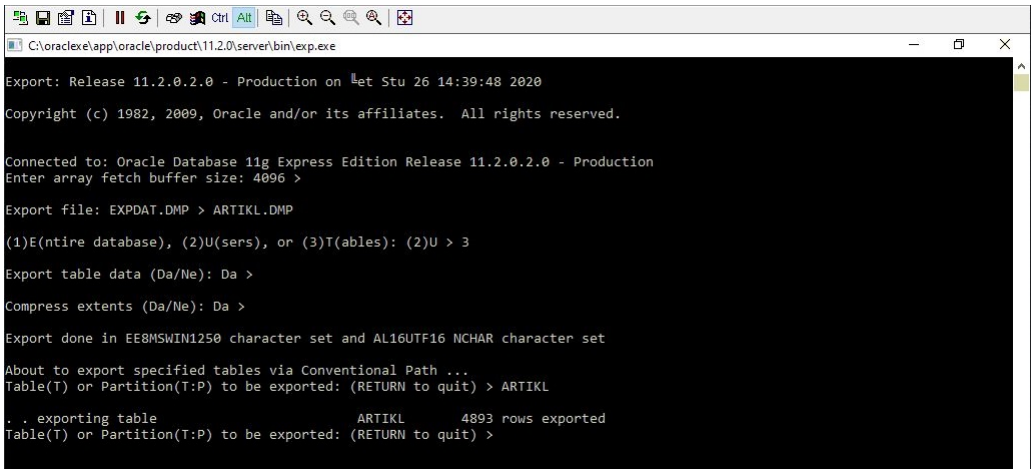
Kad su pripreme za aplikaciju gotove, može se krenuti sa kreiranjem tablica, procedura, trigera i ostalog potrebnog za aplikaciju. U ovom će se primjeru prikazati što se događa kada se kreira određena tablica. To se može napraviti sa CREATE TABLE ili čak učitati tablicu iz neke druge baze. U svrhu ovog poglavlja, kreirat će se tablica POREZ:

```
SQL> create table POREZ (sifra varchar2(3) primary key,
naziv varchar2(50) not null,
stopa number(4,2) default 25);
```

U ovom primjeru kreirala su se i određena ograničenja (CONSTRAINT). Tako je polje SIFRA primarni ključ[3], polje NAZIV mora imati neku upisanu vrijednost, a u polje STOPA, u slučaju nikakvog unosa od strane korisnika, ORACLE će automatski upisati vrijednost "25".
Druga tablica ARTIKL prenesena je iz druge baze sa postojećim sadržajem, zajedno sa svim ograničenjima i indeksima (CONSTRAINT-ima i INDEX-ima). Za to će se koristiti EXP i IMP programi. Iz druge baze tablica se eksportira jednostavnom naredbom EXP (export):

```
EXP druga/sifra@XE
```

gdje "druga" označava korisnika (username), "sifra" označava zaporku (password), a "XE" je oznaka baze na koju se spaja (connection string).
Nakon toga se otvara ekran kao na slici 4.6.



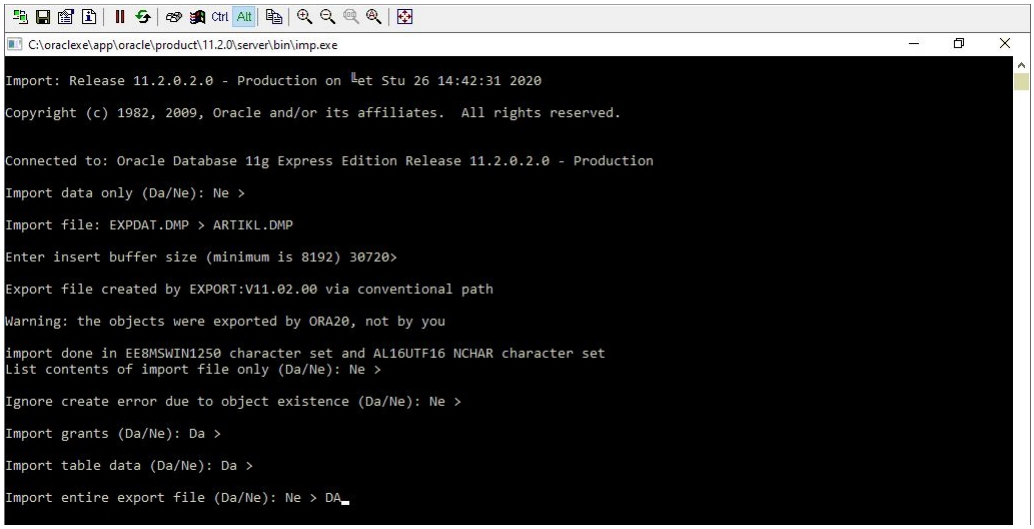
Slika 4.6: Prikaz ekrana: naredba EXP

[3] ne mogu biti 2 retka s istom vrijednosti tog polja

Programom EXP (Export baze), kao što je prikazano na slici, mogu se eksportirati pojedine tablice, sve tablice nekog korisnika ili cijela baza. U ovom slučaju eksportirala se samo tablica ARTIKL.
Sljedeći korak je učitati tu tablicu u bazu korisnika JAKOV/VOKAJ. To se provodi programom IMP (import):

```
IMP jakov/vokaj@XE
```

nakon čega se otvara ekran kao na slici 4.7.



Slika 4.7: Prikaz ekrana: naredba IMP

Programom IMP potrebno je samo navesti ime datoteke koja se prije eksportirala, no također može se odlučiti da li će se učitati tablica s podacima ili samo struktura tablice bez podataka, što se vidi prema upitima na ekranu.

Time je opisan proces kreiranja tablica potrebnih za određenu aplikaciju. U svakom trenutku mogu se pregledati sve tablice koje postoje za korisnika. ”tab” je sistemska tablica koja sadrži popis svih tablica koje postoje za korinsika. Pregled tablica korisnika JAKOV može se vidjeti sljedećom SQL naredbom:

```
SQL> select * from tab;
```

```
TNAME       TABTYPE    CLUSTERID
------------------------------
ARTIKL      TABLE
POREZ       TABLE

2 rows selected
```

Kako bi vidjeli strukturu pojedine tablice koristi se naredba DESC:

```
SQL> DESC porez

Name    Type            Nullable    Default Comments
-------------------------------------------------
SIFRA   VARCHAR2(3)
NAZIV   VARCHAR2(50)
STOPA   NUMBER(4,2)     Y           25
```

Kao što se i prije pokazalo, tablice imaju određena kreirana ograničenja (CONSTRAINTS) i indekse koji su važni za rad nad tablicama. Njih se može pregledati kroz tablice ALL_INDEXES i ALL_CONSTRAINTS:

```
SQL> select index_name, table_name, uniqueness from all_indexes
where owner='JAKOV';

INDEX_NAME          TABLE_NAME  UNIQUENESS
-----------------------------------------
IK_ARTIKL_VRSTA     ARTIKL      NONUNIQUE
PK_ARTIKL           ARTIKL      UNIQUE
SYS_C001538739      POREZ       UNIQUE
```

Kreiranje indeksa

Indekse nad tablicom kreira korisnik ili ORACLE. U navedenom slučaju je kreiran indeks IK_ARTIKL_VRSTA od strane korisnika, radi bržeg pretraživanja tablice po tom polju. Indeksi se uglavnom kreiraju nad poljima tablice koji se nalaze u upitima u WHERE uvjetu. Indeks PK_ARTIKL je kreiran automatski jer je posljedica definicije primarnog ključa. Jednako tako je indeks SYS_C001538739 kreiran zbog primarnog ključa u tablici POREZ za polje SIFRA.[4]

[4]U kreiranju se namjerno nije dao naziv primarnom ključu da se vidi kako tada ORACLE pridijeljuje naziv.

Također, vidi se da su se programom IMP, uz samu tablicu, učitali i njezini indeksi i ograničenja. Ograničenja (CONSTRAINT) vidimo sljedećim upitom:

```
SQL> select constraint_name, constraint_type, table_name,
search_condition, from all_constraints where owner='JAKOV';
CONSTRAINT_NAME CONSTRAINT_TYPE TABLE_NAME  SEARCH_CONDITION
------------------------------------------------------------
SYS_C001538738  C               POREZ       "NAZIV" IS NOT NULL
SYS_C001538740  C               ARTIKL      "SIFRA" IS NOT NULL
SYS_C001538741  C               ARTIKL      "NAZIV" IS NOT NULL
SYS_C001538742  C               ARTIKL      "OZNAKA" IS NOT NULL
SYS_C001538743  C               ARTIKL      "VRSTA" IS NOT NULL
SYS_C001538744  C               ARTIKL      "JM" IS NOT NULL
SYS_C001538754  C               ARTIKL      "CIJENA" IS NOT NULL
SYS_C001538739  P               POREZ
PK_ARTIKL       P               ARTIKL

9 rows selected
```

Vidi se da je naziv CONSTRAINT-a i pripadajućeg INDEX-a isti. Ukoliko se u samom kreiranju tablice ili učitavanju nisu predvidjeli svi indeksi i ograničenja, mogu se uvijek dodati naknadno. Indeksi i ograničenja se mogu kreirati, ali i brisati. Iz priloženog može se također primjetiti da primarni ključevi nemaju uvjet pretraživanja.

U sljedećem primjeru kreirat će se indeks I_POREZ_NAZ_STO nad poljima NAZIV i STOPA. Naziv indeksa daje se što prikladnije, a u ovom slučaju radi se takav indeks jer se pretpostavlja da će se u upitima na aplikaciji pretraživati po poljima STOPA i NAZIV zajedno:

```
SQL> create index I_POREZ_NAZ_STO on POREZ(naziv, stopa);

Index created
```

Često je teško znati samo prema nazivu indeksa nad kojim poljima je on stvoren. Za to se koristi tablica ALL_IND_COLUMNS:

```
SQL> select table_name, column_name from all_ind_columns where
index_name='I_POREZ_NAZ_STO' and table_owner='JAKOV';
```

```
TABLE_NAME  COLUMN_NAME
----------------------
POREZ       NAZIV
POREZ       STOPA
```

Indeksi su stvoreni radi bržeg pretraživanja baze. Svaki redak tablice ima ”skriveno” polje ROWID koji je adresa retka te se njima služi indeks radi ”pronalaženja” podataka:

```
SQL> select ROWID, sifra, naziv from ARTIKL where ROWNUM<5;

ROWID               SIFRA   NAZIV
------------------------------------------------------------
AACY4PAARAAA+aSAAA  183283  PREKIDAČ COLOR IZMJENIČNI 15291
AACY4PAARAAA+aSAAB  183284  PREKIDAČ COLOR KRIŽNI SE70CO 15293
AACY4PAARAAA+aSAAC  183285  UTIČNICA EKONOMIK CO VIJČANA
AACY4PAARAAA+aSAAD  183286  UTIČNICA COLOR SA POKLOPCEM VE11CT
```

U primjeru se također može vidjeti ”skriveno” polje ROWNUM koje se često koristi kada želimo vidjeti samo nekoliko stupaca tablice koja ima veliki broj stupaca. ROWNUM je redni broj retka u određenom SELECT-u.

Ograničenja nad tablicom su također važna jer se njihovim pravilnim kreiranjem, prema pravilima relacijskih baza podataka, olakšava kasniji pravilni rad aplikacije. Osim navedenih primarnih ključeva, koriste se i strani ključevi (FOREIGN CONSTRAINT). Strani ključ nad poljem ili poljima tablice kreira se u slučaju kada u neko polje može biti upisan samo podatak koji postoji u tablici nad kojom se definira strani ključ. U ovom primjeru kreirat će se strani ključ nad poljem POREZ u tablici ARTIKL koji će pratiti polje SIFRA u tablici POREZ:

```
SQL> alter table ARTIKL add constraint FK_ARTIKL_POREZ foreign
key(POREZ) references POREZ(sifra);

Table altered
```

Kreiranjem ovog ograničenja, nemoguće je u tablici ARTIKL unijeti u polje POREZ vrijednost koja ne postoji u tablici POREZ:

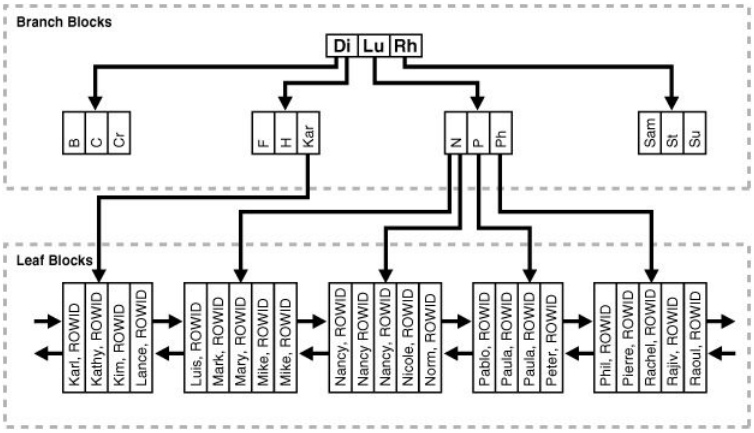
```
SQL> select * from POREZ;

SIFRA   NAZIV   STOPA
---------------------
25      Porez   25,00
```

```
SQL> insert into ARTIKL(sifra, naziv, porez)
values('123','Proba','26');
insert into ARTIKL(sifra, naziv, porez) values('123','Proba','26')

ORA-02291: integrity constraint (JAKOV.FK_ARTIKL_POREZ) violated
- parent key not found
```

4.3 Spremanje u bazu podataka – fizička realizacija

Kad se tablica kreira, ona se smješta u određeni prostor tablica koji je pridijeljen ili korisniku ili samoj tablici. Ukoliko se ništa ne navede, tablica ide u predodređeni (default) prostor tablica. Ukoliko sam prostor tablica ima više datoteka, tablica se može spremati kroz sve njih, ukoliko ona zahtijeva više prostora.
Za pristup tablici najvažniji je ROWID koji ima svaki pojedini redak, a koji je adresa koja sadrži točnu poziciju retka u datoteci. Pri kreiranju indeksa ORACLE koristi dva tipa indeksa. Radi se o indeks-sekvencijalnom pristupu. Specifično, to su B-stablo indeks i Bitmap indeks. Bitmap indeks nije podržan u verziji ORACLE XE 11.g pa će se samo spomenuti primjerom.

B-stablo je zadani način implementacije/izvođenja indeksa u ORACLE bazi; način kreiranja je prikazan prethodno (CREATE INDEX...).
Primjer pretraživanja tablice prema tom modelu prikazan je slikom 4.8[5].



Slika 4.8: Pretraživanje tablice u Oracle bazi pomoću B-stabla

[5]Slika preuzeta s [4].

Prvi nivo indeksa dijeli vrijednosti stupca na prvi nivo pa dalje slijedi sekvencijalno grananje.
Za razliku od B-stablo indeksa koji pozicionira jedan redak i nalazi njegovu adresu, Bitmap indeks se koristi nad stupcima koji nemaju puno različitih vrijednosti, a jednako tako može se kreirati i nad više redaka u više tablica.
Na primjer, u tablici ARTIKL malo je vrijednosti za stupac POREZ (0,10,13,25) pa se tako može kreirati Bitmap indeks na sljedeći način:

```
SQL> create bitmap index ART_POR_IDX
on ARTIKL(POREZ);
```

U slučaju kreiranja indeksa koji će brže davati rezultat u WHERE pretraživanjima koji imaju JOIN dvije tablice, indeks se može kreirati na sljedeći način:

```
SQL> create bitmap index ART_POR_IDX
on ARTIKL
from ARTIKL, POREZ
where ARTIKL.POREZ=POREZ.sifra;
```

Bitmap indekse dobro je koristiti kod ”read-only” tablica ili tablica gdje nema puno mijenjanja sadržaja.
Osim navedenih indeksa, ORACLE podržava i HASH indekse kada se zajedno s tablicom formiraju kroz CLUSTER[6].
Pohranjivanje tablice u hash klasteru neobavezan je način za poboljšanje performansi kod pretraživanja podataka. Hash klaster pruža alternativu tablici s ”običnim” indeksom ili indeksom klasterom. Pomoću indeksirane tablice ili klastera indeksa ORACLE baza locira retke u tablici koristeći vrijednosti ključa koje baza podataka pohranjuje u zasebni indeks. Da bi se koristilo raspršivanje, stvara se hash klaster i u njega se učitaju tablice. Baza podataka fizički pohranjuje retke tablice u hash klaster i dohvaća ih prema rezultatima hash funkcije.
Prvo će se kreirati klaster za koji se odmah može definirati gdje će se nalaziti; odnosno u kojem prostoru tablica. U primjeru se kreira klaster STUDENT_CLUSTER u koji će se smjestiti između ostalog tablica STUDENT.

```
SQL> create cluster STUDENT_CLUSTER(sifra number(10));

Cluster created
```

Nakon toga kreira se tablica STUDENT koja ima primarni ključ SIFRA i koja će se nalaziti u klasteru STUDENT_CLUSTER vezana na primarni ključ SIFRA:

[6]klaster kao skup nečega (tablica, indeksa, ...)

```
SQL> create table STUDENT(sifra number(10) primary key,
prezime varchar2(20),
ime varchar2(20))
cluster STUDENT_CLUSTER(SIFRA);

Table created
```

4.4 Transakcije

Transakcija je niz SQL naredbi (INSERT, UPDATE, DELETE...) koje izvršavaju promjene nad bazom podataka. Svako pisanje naredbi poput INSERT, UPDATE i DELETE je početak obrade nad tablicom i označuje početak transakcije. Transakcija završava naredbom COMMIT ili izlaskom iz baze.
Ključne riječi za upravljanje transakcijama su COMMIT, ROLLBACK i SAVEPOINT.

- Naredba COMMIT označava završetak transakcije i iza nje počinje nova transakcija.
- Naredba ROLLBACK označava vraćanje svih izvršenih promjena do početka transakcije.
- Naredba SAVEPOINT omogućuje definiranje kontrolnih točaka unutar velikih transakcija tako da se ROLLBACK može izvršiti unatrag do dijela transakcije obilježenog točkom spremanja SAVEPOINT.

Primjer 1

U ovom primjeru pokazuje se unos podataka u tablicu POREZ i mijenjanje podataka u tablici ARTIKL:

```
SQL> commit;

Commit complete

SQL> insert into POREZ(sifra, naziv, stopa)
values('02','Porez 10%','10');

1 row inserted

SQL> update ARTIKL set porez='02';

145314 rows updated
```

```
SQL> select distinct POREZ from ARTIKL;

POREZ
-----
02

SQL> select * from POREZ;

SIFRA   NAZIV           STOPA
-----------------------------
25      Porez           25,00
02      Porez 10%       10,00

SQL> ROLLBACK;

Rollback complete

SQL> select distinct POREZ from ARTIKL;

POREZ
-----
25
```

Može se primjetiti da se naredbom ROLLBACK transakcija ”poništila” i da su svi podaci vraćeni kao prije naredbe COMMIT.

Primjer 2

U ovom primjeru pokazuje se unos podataka u tablicu POREZ i mijenjanje podataka u tablici ARTIKL uz korištenje SAVEPOINT točke:

```
SQL> commit;

Commit complete

SQL> insert into POREZ(sifra, naziv, stopa)
values('02','Porez 10%','10');

1 row inserted
```

```
SQL> SAVEPOINT A;

Savepoint created

SQL> update ARTIKL set porez='02';

145314 rows updated

SQL> select * from POREZ;

SIFRA   NAZIV           STOPA
-----------------------------
25      Porez           25,00
02      Porez 10%       10,00

SQL> select distinct POREZ from ARTIKL;

POREZ
-----
02

SQL> ROLLBACK to SAVEPOINT A;

Rollback complete

SQL> select * from POREZ;

SIFRA   NAZIV           STOPA
-----------------------------
25      Porez           25,00
02      Porez 10%       10,00

SQL> select distinct POREZ from ARTIKL;

POREZ
-----
25
```

```
SQL> commit;

Commit complete
```

Može se primjetiti da su korištenjem SAVEPOINT točke vraćeni unatrag podaci mijenjani naredbama iza te točke, dok su promjene prije točke A prihvaćene. Korištenje takvog načina rada s transakcijama se koristi kod velikih transakcija ili distribuiranih transakcija koje se koriste u distribuiranim bazama podataka.
Za tu namjenu se koriste[7] prostor tablica UNDOTBS1 i rollback segmenti. Svakom transakcijom u bazi podataka, a koja nije završena (COMMIT), može se vidjeti da se prazan prostor u prostoru tablica UNDOTBS1 smanjuje. Pri samom kreiranju baze stvore se rollback segmenti koji omogućavaju vraćanje podataka. Ovo je prikazano na slici 4.9.[8]

```
SQL> select * from dba_rollback_segs;
SEGMENT_NAME                   OWNER  TABLESPACE_NAME                SEGMENT_ID    FILE_ID   BLOCK_ID INITIAL_EXTENT NEXT_EXTENT
------------------------------ ------ ------------------------------ ---------- ---------- ---------- -------------- -----------
SYSTEM                         SYS    SYSTEM                                  0          1        128         114688       57344

MIN_EXTENTS MAX_EXTENTS PCT_INCREASE STATUS           INSTANCE_NUM                              RELATIVE_FNO
----------- ----------- ------------ ---------------- ----------------------------------------- ------------
          1       32765              ONLINE                                                               1
```

Slika 4.9: Rollback segmenti kod kreiranja baze

Važna polja za administratora baze podataka su INITIAL_EXTENT, NEXT_EXTENT, MIN_EXTENTS i MAX_EXTENTS koja govore o načinu "povećavanja" samog segmenta. Naime, postoje velike transakcije koje mogu izazvati da postojeći ROLLBACK_SEGMENT koji prati transakcije nije dovoljno velik, odnosno da dostigne maksimum. ORACLE tada javlja grešku da nije moguće izvršiti zadanu promjenu jer se neće moći vratiti podaci te vraća transakciju na početak.
Za takve slučajeve treba kreirati novi ROLLBACK SEGMENT sa povećanim parametrima koji će se moći dovoljno proširiti. To naravno znači da i pripadajući prostor tablica UNDOTBS1 mora biti dovoljno velik, pa je moguće da paralelno treba i njega ukupno povećati dodavanjem nove datoteke (fizička struktura).

Kreiranje ROLLBACK segmenta

```
SQL> create rollback segment BIG storage (INITIAL 1310720);

Rollback segment created
```

[7] U slučaju ovog diplomskog rada: ORACLE XE
[8] Na slici je prikazan samo prvi red. Bitno je prikazati stupce za ovaj prikaz.

```
SQL> alter rollback segment BIG storage (NEXT 655360);

Rollback segment altered
```

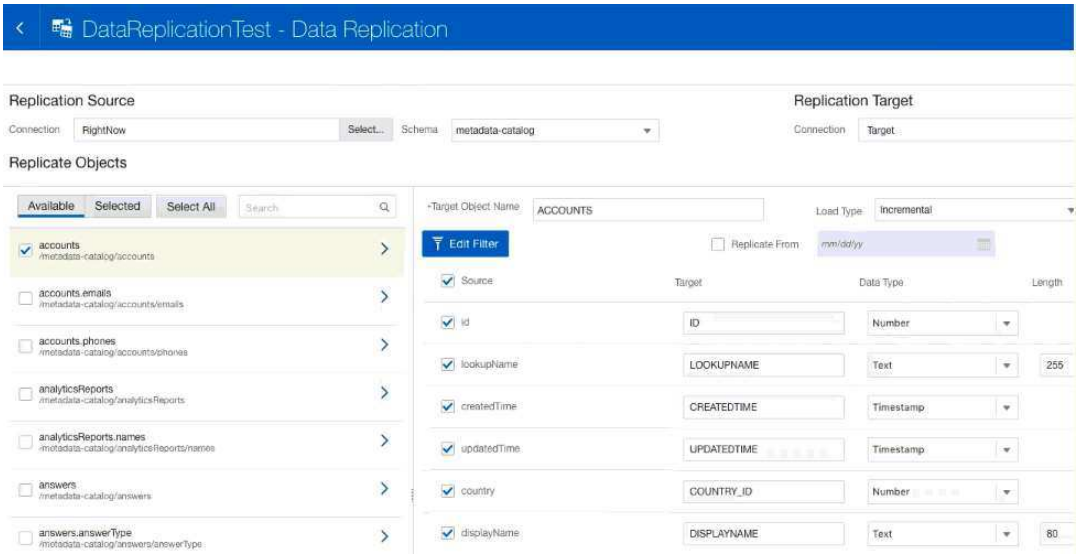
Kako bi neka transakcija koristila novi (veliki) segment potrebno je to eksplicitno definirati. U suprotnome, transakcija uzima prvi slobodni segment:

```
SQL> commit;

Commit complete

SQL> set transaction use rollback segment big;

Transaction set

......
SQL> commit;
```

4.5 Distribuirane baze podataka

Distribuirane baze podataka su baze podataka koje nisu ograničene na jedan sustav već su na različitim lokacijama. Mogu biti na različitim mrežnim (web) lokacijama, odnosno serverima. Ne mora biti nužno iste verzije RDBMS-a niti isti RDBMS-i. *Fragmentacija* i *replikacija* su načini na koje se baza podataka može podijeliti na više lokacija.

Fragmentacija, kako i riječ govori, omogućava da fragmenti budu raspoređeni svaki na svoju lokaciju, dok replikacija znači da su isti podaci kopirani na više lokacija. Replikacija omogućava da podaci budu stalno sinkronizirani i ažurirani. Novije verzije ORACLE-a imaju automatiziran princip kreiranja replikacije nekog objekta baze podataka prema drugom. Primjer je dan slikom 4.10.



Slika 4.10: Primjer

Kako bi ostvarili pristup nekom objektu, odnosno tablici, na drugoj lokaciji najčešće se koristi DATABASE_LINK. U sljedećem primjeru će se prikazati kreiranje poveznice (linka) na tablicu ARTIKL koji je na udaljenoj lokaciji B, ali je cilj da tablica ARTIKL na našoj lokaciji ima iste podatke kao i na lokaciji B.
Kreiranje linka:

```
SQL> create PUBLIC database link REMOTE1 connect to ORA20
identified by PROBA using 'lokacijab.orakon.net:1521/XE';
```

U navedenom slučaju naziv linka je REMOTE1, USER na udaljenoj lokaciji (vlasnik objekta) je ORA20 sa šifrom PROBA. Naziv udaljene baze, odnosno CONNECTION STRING, je 'lokacijab.orakon.net:1521/XE'. Definicija takvog stringa je dobivena mrežnim protokolima koje koristi ORACLE.
Nakon što se kreirao takav link, sljedećim primjerom će se napraviti jedna PL/SQL proceduru koja će "sinkronizirati" podatke o ARTIKL-ima iz lokacije B sa našom lokacijom A:

```
declare
br1 varchar2(10); ima number(2);
cursor c1 is select sifra from ARTIKL@REMOTE1;
begin
open c1;
loop
```

```
fetch c1 into br1;
exit when c1%notfound;
select count(*) into ima from ARTIKL where sifra=br1;
if ima=0 then
insert into artikl select * from ARTIKL@REMOTE1 where sifra=br1;
end if;
end loop;
close c1;
end;
/
```

Distribuirane baze podataka imaju svoje prednosti, no također su kompliciranije za održavanje. Također, u današnje vrijeme sve više se koriste Cloud serveri sa svojim bazama podataka pa se korištenje mijenja.

4.6 Oracle

U ovom diplomskom radu, proučavao sam ORACLE bazu podataka. ORACLE kao korporacija postoji još od 1977. godine te je jedan od vodećih proizvođača DBMS-a. Tvrtka razvija softver i tehnologiju baza podatka, sustave dizajnirane u oblaku (cloud) i softverske proizvode za poduzeća.
Poznati razvojni proizvodi koji su dosta korišteni u IT industriji su: Oracle Designer – CASE (alat za projektiranje informacijskih sustava), Oracle Developer (sadrži Oracle Forms, Oracle Discoverer i Oracle Reports, a služi za kreiranje aplikacija), Oracle JDeveloper, NetBeans (platforma za razvoj softvera zasnovana na Javi), Oracle Application Express (alat za web orijentirani razvoj), ORACLE SQL developer (alat za rad nad bazom podataka) i Oracle Developer Studio (sustav za generiranje softvera za razvoj C, C++, Fortran i Java softvera).
ORACLE je vlasnik i SUN Microsystems čime je osigurao i tržište u razvoju hardvera. Tako je postao vlasnik vrlo poznatog DBMS-a MySQL, a vlasnik je i programskog jezika Java. ORACLE ima podršku na svim platformama i serverima pa je njegova uporaba široko rasprostranjena. Mogući nedostatak je njegova cijena kod licenciranja, no i tu je napravljen napredak upravo sa verzijom ORACLE XE koja je korištena u ovom diplomskom radu jer je besplatna pa se često koristi kod manjih poduzeća i ustanova.[9]

[9] [6], [5], [2], [3]

Bibliografija

[1] *Black Box Model*, https://www.investopedia.com/terms/b/blackbox.asp, Investopedia, Will Kenton, (2020.).

[2] *Napredni modeli i baze podataka*, https://www.fer.unizg.hr/_download/repository/6._NoSQL_(1._od_3).pdf, FER, Zagreb, (2020.).

[3] *Oracle Corporation*, https://en.wikipedia.org/wiki/Oracle_Corporation, Wikipedia, (2020.).

[4] *Oracle Help Center*, https://docs.oracle.com/cd/B10500_01/server.920/a96524/c11schem.htm, Oracle, (2020.).

[5] *Oracle Tutorial*, https://www.oracletutorial.com/, Oracle, (2020.).

[6] *Oracle*, https://www.oracle.com/index.html, Oracle, (2020.).

[7] *Principles of Database Management - Lecture Materials*, https://www.pdbmbook.com/lecturers, Principles of Database Management, (2020.).

[8] *Principles of Database Management (book)*, https://www.youtube.com/playlist?list=PLdQddgMBv5zHcEN9RrhADq3CBColhY2hl, Youtube, Bart Baesens, (2020.).

[9] Darko Hrenić, *Oracle: principi, praksa, programiranje*, Znak, 1995.

[10] Thomas Kyte i Darl Kuhn, *Oracle Database Transactions and Locking Revealed*, Apress, 2014.

[11] Wilfried Lemahieu, Seppe vanden Broucke i Bart Baesens, *Principles of Database Management: The Practical Guide to Storing, Managing and Analyzing Big and Small Data*, Cambridge University Press, 2018.

[12] Robert Manger, *Baze podataka*, Element, 2012.

Sažetak

Cilj ovog diplomskog rada bio je proučiti i diskutirati fizičke aspekte organizacije baze podataka kao što su: fizička organizacija pojedinih datoteka i indeksa, fizička organizacija baze u cjelini, upravljanje transakcijama, distribuiranje podataka, pohrana podataka itd. Također, u praktičnom dijelu rada implementirana je baza podataka te je pokazano kako se njome upravlja, kako se spremaju podaci te kako se biraju fizičke realizacije određenih tablica.

Summary

The aim of this thesis was to study and discuss the physical aspects of database organization, such as: physical organization of individual files and indexes, physical organization of the database as a whole, transaction management, data distribution, data storage, etc. Furthermore, in the practical part of the paper the database was implemented and it was shown how it is managed, how data is stored and how the physical realizations of certain tables are selected.

Životopis

Jakov Hrenić rođen je 1.7.1996. godine u Zagrebu. Djetinjstvo je proveo u Novom Marofu, gdje je i pohađao osnovnu školu. Već je od malih nogu pokazivao interes za matematiku i programiranje.

Nakon završene osnovne škole upisao je prirodoslovno-matematički smjer Prve gimnazije Varaždin. Tijekom osnovnoškolskog i srednjoškolskog obrazovanja sudjelovao je na brojnim natjecanjima: matematika, informatika, logika. Između ostalog je 2014. i 2015. godine sudjelovao na državnim natjecanjima iz logike te ostvario dobre rezultate. 2015. godine položio je državnu maturu iz matematike sa visokim rezultatom, a imao je i najbolji rezultat državne mature iz logike u Hrvatskoj.

Nakon srednje škole, 2015. godine upisao je preddiplomski studij matematike na Prirodoslovno-matematičkom fakultetu u Zagrebu te je 2018. stekao titulu univ. bacc. math. Iste godine nastavio je obrazovanje pri istome fakultetu upisujući diplomski studij Financijske i poslovne matematike. Sve ispite je položio u roku, završavajući fakultet kao redovan student.

Kroz studij, inspiriran strasti prema financijama, bazama podataka i matematici, proširio je svoje ambicije studentskim poslovima. Prvo se zaposlio u poduzeću Photomath u Zagrebu, gdje primjenjuje znanja stečena tijekom studiranja. Zatim je, istovremeno, odradio praksu u PBZ kao SQL developer. Danas još uvijek radi u poduzeću Photomath. U budućem radu planira primjenjivati znanja stečena na petogodišnjem putovanju kroz matematički fakultet u dobru korist, prema boljoj budućnosti.

U slobodno vrijeme rekreativno se bavi trčanjem, biciklizmom i fitnesom ili ga provodi družeći se s prijateljima.