PBP TEORIJA

NACINI KORISCENJA BAZA PODATAKA U VISIM PROGRAMSKIM JEZICIMA:

Dva vida izvrsavanja SQL naredbi:

1.interaktivni koji podrazumeva izvrsavanje samostalnih SQL naredbi preko online terminala

2.aplikativni koji podrazumeva izvrsavanje SQL naredbi umetnutih u program na visem programskom jeziku opste namene, naredbe se izvrsavaju naizmenicno

Aplikativni SQL moze da ukljucuje dve vrste SQL naredbi:

1. staticke SQL naredbe koje vec u fazi pisanja programa imaju precizan oblik(strukturu) i cija analiza i priprema za izvrsenje mogu u potpunosti obaviti u fazi pretprocesiranja; npr. U fazi pretprocesiranja se moraju znati nazivi kolona i tabela na koje se referise

2.dinamicke SQL naredbe koje se zadaju u obliku niske karaktera koja se dodeljuje promenljivoj u programu; analiza i priprema moze se obaviti tek u fazi izvrsavanja kada je poznat njen precizan oblik; npr. Aplikacija ocekuje od korisnika da unese SQL naredbe nazive tabela i kolona nad kojima se upit izvrsava

Pored ovakvog nacina koriscenja baze podataka , postoje I druge mogucnosti ugradnje blokova. U slucajy sistema DB2 to su:

1. Upotreba ODBC standarda podrazumeva model komunikacije : korisnik komunicira sa ODBC medjusistemo, koji dalje komunicira sa konkretnim SUBP-om, potpuno nezavisno od korisnika. Ovaj pristup omogucava koriscenje razlicitih SUBP-a, cak I u okviru jednog programa. Imamo zajednicki interfejs, potrebno je ukljuciti drajver za odredjeni tip baze podataka. Korisniku je lakse jer pise program na isti nacin bez obzira nad kojim SUBP-om se radi, ali nije moguce iskoristiti specificnost pojedinog sistema(DB2).

2. Direktni pozivi DB2 funkcija su specificni za sistem DB2 I koriste njegove mogucnosti. Postoji najveci moguci zajednicki skup funkcija I principa sa ODBC pristupom, korisnik moze da prilagodjava programe jednom ili drugom obliku. Prednosti: portabilnost aplikacija, eliminisana zavisnost od preprocesora, aplikacije se distribuiraju kao prevedena aplikacija ili izvrsna biblioteka.

Osnovna prednost aplikativnog SQL-A u odnosu na direktne pozive DB2 funkcija je u tome sto moze da koristi staticki SQL I sto je u velikoj meri standardizovan pa se moze koristiti I medju razlicitim SUBP platformama.

Zapamcena procedura se koristi u slucaju kada nam trebaju predbosti oba mehanizma (direktnih poziva DB2 funkcija I aplikativnog SQL-a). Ona se poziva iz DB2 CLI aplikacije I

izvrsava se na serveru. Jednom napisanu zapamcenu proceduru moze da pozove bilo koja DB2 CLI ili ODBC aplikacija. Zapamcena procedura je deo kojis e izvrsava na serveru.

APLIKATIVNI(UGRADJENI) SQL

Potreba za umetanjem SQL-A u visi programski jezik javlja se zbog odsustva kontrolnih struktura u upitnom jeziku, cesto neophodnih pri obradi podataka iz baze podataka. Umetanjem u visi programski jezik tzv. Maticni jezik kao sto su C, C++, Java ,fortran,Cobol, dobija se aplikativni SQL na kome se mogu pisati kompleksni programi za najraznovrsnije obrade. Osnovni princip svih aplikativnih SQL jezika je princip dualnostni, prema kome se svaka interaktivna SQL naredba moze izraziti I u aplikativnom SQL -u ali obratno ne vazi.

POVEZIVANJE SQL SA MATICNIM JEZIKOM

Kada zelimo da koristimo SQL naredbu u programu na maticnom jeziku, upozoravamo pretprocesor da nailazi SQL kod navodjenjem kljucne reci EXEC SQL ispred naredbe.

Razmena info sizmedju baze I programa na maticnom jeziku se vrsi kroz host(maticne) promenljive kojima je dozvoljeno da se pojave I u naredbama maticnog jezika I u SQL naredbama. Host promenljive imaju kao prefiks dvotacku u naredbama SQK-A , dok se u naredbama maticnog jezika javljaju bez dvotacke.

Program koji sadrzi izvrsne SQL naredbe komunicira sa sistemom DB2 preko memorijskog prostora koji se zove SQL prostor za komunikaciju. SQLCA je struktura podataka koja seazurira posle izvrsavanja svake SQL naredbe. U ovu strukturu upisuje se informacija o izvrsenoj SQL naredbi.

EXEC SQL INCLUDE SQLCA;

Najcesce koriscena promenljiva strukture SQLCA je promenljiva SQLCODE. Ona predstavlja indikator uspesnosti izvrsenja SQL naredbe I moze da ima sledece vrednosti:

0 ako je izvrsavanje proslo uspesno, pozitivna vrednost ako se pri uspesnom izvrsavaanju dogodilo nesto izuzetno (+100 tabela prazna) , negativna vrednost naredba se nije izvrsila uspesno zbog neke nastale greske

Kada zelimo da se proveri status u promenljivoj SQLCODE , zadajemo direktivu pretprocesoru :

EXEC SQL WHENEVER < USLOV> <AKCIJA>

USLOV: NOT FOUND - NIJE NADJEN ILI SQLCODE=100, SQLERROR - INDIKATOR GRESKE ILI SQLCODE < 0, SQLWARNING - INDIKATOR UPOZORENJA

AKCIJA: CONTINUE - PROGRAM NASTAVLJA SA IZVRSAVANJEM, GOTO - SKOK NA DEO PROGRAMA U KOME SE NASTAVLJA OBRADA

Dakle, deklarativni iskaz WHENEVER omogucuje programeru da zada nacin na koji ce proveravatai vrednosti promenljive SQLCODE nakon svake izvrsene SQL naredbe.Bitan je poredak naredbi WHENEVER, dok se ne navede naredna odnosi se na sve SQL naredbe.

Medju ostalim zanimljivim promenljivama SQLCA strukture je skup promenljivih SQLWARN0 - SQLWARN9 koje su tipa char I sadrze info o uzorku upozorenja. Npr. Znacenje nekih ovih elemenata SQLCA strukture:

SQLWARN0 ima vrednost 'W' ako bar jedna od preostalih SQLWARNi promenljivih sadrzi upozorenje ('W')

SQLWARN1 sadrzi 'W' ako je pri dodeli promenljivoj maticnog jezika odsecena vrednost kolone koja je tipa niska karaktera

SQLWARN2 sadrzi 'W' ako je doslo do eliminacije NULL vrednosti pri primeni agregatne funkcije

KOMENTARI U SQL: -- traje do kraja reda , ne sme da se prekine par kljucnih reci EXEC SQL

POVEZIVANJE NA BAZU PODATAKA - SQL CONNECT I CONNECT RESET

FAZE PREVODJENJA:

Da bi se program mogao izvrsavati potrebno je ugraditi paket u bazu, pri cemu se vrsi optimizacija svih pristupa bazi od strane programa.

Pretrprocesiranje se vrsi DB2 naredbom PREP (ili PRECOMPILE) :

DB2 PREP datoteka.sqc BINDFILE

Kao rezultat dobijaju se dve datoteke sa istim nazivom ali sa razlicitim ekstenzijama .c ili .bnd. Opcijom BINDFILE zadaje se da ce ugradnja paketa biti odlozena za kasnije. Nakon faze pretprocesiranja moguce je vrsiti ugradnju paketa u bazu nardebom BIND:

DB2 BIND datoteka.bnd

Nakon ovoga potrebno je jos prevesti dobijenu C datoteku, a zatim I izvrsiti linkovanje sa odgovarajucom bibliotekom db2 funkcija.

SEKCIJA ZA DEKLARACIJU HOST PROMENLJIVIH:

Host promenljive se deklarisu tako sto se njihove deklaracije postavljaju izmedju dve ugradjene SQL naredbe:

EXEC SQL BEGIN DECLARE SECTION;

…

EXEC SQL END DECLARE SECTION;

Ova sekcija se naziva sekcijom za deklaraciju host promenljivih.format deklaracije promenljive treba da odgovara formatu maticnog jezika.Ima smisla deklarisati promenljive samo onih tipova sa kojima se moze raditi I u maticnom jeziky I u SQL-u .Host promenljiva I atribut mogu imati isti naziv.

SQL TIP	C TIP
SMALLINT	short
INTEGER	long
DOUBLE	double
CHAR	char
CHAR(n)	char[n+1]
VARCHAR(n)	char[n+1]
DATE	char[11]
TAME	char[9]

KORISCENJE HOST PROMENLJIVIH:

```
EXEC SQL INSERT INTO Knjiga(k_sifra,naziv,god_izdavanja) VALUES
(:uneta_sifra, :naziv, :god_izdavanja);
```

Poslednje dve linije koda sadrze ugradjenu SQL naredbu INSERT. Ovoj naredbi prethodi par kljucnih reci EXEC SQL da bi se naznacilo da je ovo ugradjena SQL naredba. Vrednosti koje se unose u ovim linijama nisu eksplicitne konstante, vec host promenljive cije tekuce vrednosti postaju komponente n-torke koja se dodaje u tabelu.

Neke naredbe se mogu ugraditi u program na maticnom jeziku dodavanjem kljucne reci EXEC SQL . Npr INSERT, UPDATE, DELETE

Upiti tipa SELECT-FROM-WHERE se ne mogu direktno ugraditi u maticni jezik zbog razlike u modelu podataka.Maticni jezik ne podrzava direktno skupovni tip podataka.

Stoga, ugradjeni SQL mora da koristi jedan od dva mehanizma za povezivanje rezultata upita sa programom u maticnom jeziku, a to su:

1. SELECT naredba koja vraca jedan red: ovakav upit moze svoj rezultat da sacuva u host promenljivama, za svaku komponentu n-torke

2. Kursori: upiti koji proizvode vise od jedne n-torke u rezultatu se mogu izvrsavati ako se deklarise kursor za taj upit; opseg kursora cine sve n-torke rezultujuce relacije I svaka pojedinacna n-torka se moze prihvatiti u host promenljivama I obradjivati programom u maticnom jeziku

UPITI KOJI VRACAJU TACNO JEDAN RED:

Forma naredbe SELECT koja vraca tacno jedan red je ekvivalentna formi SELECT -FROM-WHERE naredbe, osim sto stavku SELECT prati kljucna rec INTO i lista host promenljivih. Ove host promenljive se pisu sa dvotackom ispred naziva, kao sto je slucaj i sa svim host promenljivama u okviru SQL naredbe. Ako je rezultat upita tacno jedna n-torka, onda komponente ove n-torke postaju vrednosti datih promenljivih. Ako je rezultat prazan skup ili vise od jedne ntorke, onda se ne vrsi nikakva dodela host promenljivama i odgovaraju ci kod za gresku se upisuje u promenljivu SQLSTATE.

EXEC SQL SELECT naziv, god_izdavanja

INTO :naziv, :godina_izdavanja

FROM Knjiga WHERE k_sifra = :uneta_sifra;

S obzirom na to da je atribut k_sifra primarni kljuc tabele Knjiga, rezultat prethodnog upita je najvise jedna n-torka, tj. tabela sa jednom vrstom cije se vrednosti atributa naziv i god_izdavanja upisuju, redom, u host promenljive naziv i godina_izdavanja. Ovde je dat primer kada su isto imenovani atribut tabele i odgovarajuca host promenljiva. U slucaju da je u navedenom primeru vrednost atributa god_izdavanja bila NULL, doslo bi do greske pri izvrsenju SELECT naredbe, jer host promenljiva godina_izdavanja ne moze da sacuva NULL vrednost. Iz ovog razloga se uz host promenljive koje mogu imati nedefinisanu (NULL) vrednost uvode i indikatorske promenljive koje na neki nacin mogu da sacuvaju informaciju da li je vrednost odgovarajuceg atributa NULL.Host promenjliva sada ima uz sebe oznaku INDICATOR kao npr :promenljiva:ind_promenljiva

KURSORI:

Najcesci nacin za povezivanje upita iz SQL-a sa maticnim jezikom je posredstvom kursora koji prolazi redom kroz sve n-torke relacije.Ova relacija moze biti neka od postojecih tabela ili moze biti dobijena kao rezultat upita. Jedan vid pokazivaca koji prolazi kroz rezultat upita ili tabele.

Da bismo napravili I koristili kursorn potrebni su nam naredni koraci:

1. deklaracija kursora,koja ima formu:

EXEC SQL DECLARE <naziv_kursora> CURSOR FOR <upit>

Opseg kursora cine ntorke relacije dobijene upitom.

2. Otvaranje kursora,koje ima formu:

EXEC SQL OPEN <naziv_kursora>

Ovim se izvrsava upit kojim je kursor definisan I dovodi se u stanje u kojem je spreman da prihvati prvu ntorku relacije nad kojom je deklarisan.

3. Jedna ili vise naredbi citanja podataka, kojom se dobija naredna ntorka relacije nad kojom je kursor definisan. Ova naredba ima formu:

EXEC SQL FETCH <naziv_kursora> INTO <lista_promenljivih>

U listi promenljivih mora da postoji po jedna promenljiva odgovarajuceg tipa z svaki atribut relacije. Ako u relaciji vise nema ntorki onda se ne vraca nijedna ntorka, vrednost promenljive SQLCODE postaje 100, a vrednost SQLSTATE se postavlja na 02000 sto znaci da nije pronadjena nijedna ntorka.

4. zatvaranje kursora,koje ima formu:

EXEC SQL CLOSE <naziv_kursora>

Kada se krsor zatvori onda on vise nema vrednost neke ntorke relacije nad kojom je definisa. Naravno moze se opet otvoriti.

FETCH naredba je jedina naredba kojom se kursor moze pomerati.Pretrazivanje se vrsi u petlji I koristi se CHECKERR za proveru.

Kursor moze biti deklarisan samo za citanje , za citanje I brisanje, ili za citanje,brisanje I menjanje podataka.ako kursor deklarisemo sa opcijom FOR READ ONLY(FOR FETCH ONLY)ova opcija se navodi na kraj deklaracije kurosra,onda SUBP moze biti siguran da relacija nad kojom je kursor deklarisan nece biti izmenjena uako joj dati kursor pristupa.Npr.

EXEC SQL DECLARE kursor_o_knjigama CURSOR FOR

select k_sifra, naziv, izdavac

from Knjiga where god_izdavanja = 2016

FOR READ ONLY;

IZMENE POSREDSTVOM KURSORA:

Moguce je u progaramu na maticnom jeziku za datu ntorku ispitivati da li zadovoljava sve zeljene uslove pre nego sto odlucimo da je obrisemo ili izmenimo.Pritom, ako zelimo da podatke menjamo putem krsora,potrebno je u deklaraciji kursora nakon upita navesti stavku: FOR UPDATE OD <lista_atributa> gde lista atributa oznacava listu naziva atributa,medjusobno razdvojenih zarezima, koje je dozvoljeno menajti koriscenjem kursora. Ukoliko se navede stavka FOR UPDATE bez navodjenja naziva kolona,ond aje putem kursora moguce menjati sve kolone tabele ili pogleda navedenih u stavci FROM spoljasnje naredbe SELECT. Za brisanje tekuceg reda putem kursora ne zahteva se navodjenje stavke FROM UPDATE. Posredstvom kursora moze se jednom anredbom izmeniti, odnosno obrisati samo jedan red I to onaje koji je poslednji procitan. Neposredno nakon brisanja poslednjeg reda,kursor se moze upotrebljavati samo za citanje narednog reda.

Kursor se implicitno ogranicava samo za citanje ako SELECT naredba u deklaraciji kursora ukljucuje neku od sledecih konstrukcija:

-Dve ili vise tabela u FROM liniji ili pogled koji se ne moze azurirati

-GROUP BY ili HAVING stavku u spoljasnjoj SELECT naredbi

-kolonsku funkciju u spoljasnoj SELECT liniji

-skupovnu (UNION, INTERSECT, EXCEPT) operaciju osim UNION ALL

-DISTINCT opciju u spoljasnjoj SELECT liniji

-agregatnu funkciju u spoljasnjoj SELECT liniji

-ORDER BY stavku.

DINAMICKI SQL:

Za razilku od ovog pristupa , u dinamickom SQL-U naredbe nisu poznate u vreme pisanja programa I preciziraju se tek u fazi izvrsavanja programa. Takve SQL naredbe, umesto da se eksplicitno navode u programu, zadaju se u obliku niske karaktera koja se dodeljuje host promenljivoj. Vrednost odgovarajuce host promenljive moze se uneti sa standardnog ulaza. Svaki put prlikom izvrsavanja programi se moraju iznova analizirati I optimizovati od strane sistema DB2.

Najcesci razlozi za koriscenje dinamickog SQL-A su sledeci:

- deo ili kompletna SQL naredba ce iti generisana tek u vreme izvrsavanja

- objekti nad kojima je SQL naredba formulisana ne postoje u fazi preprocesiranja

- zelimo da se za izvrsavanje uvek koristi optimalan plan pristupa podacima, zasnovan na tekucim statistikama baze podataka

Da bi dinamicka SQL naredba mogla da se izvrsi, potrebno je pripremiti je za izvrsenje ugnjezdenom (statickom) naredbom PREPARE. Nakon toga se dinamicka naredba moze izvrsiti ugnjezdenom (statickom) naredbom EXECUTE.

Aplikacija koja koristi dimanicki SQL se najcesce sastoji iz narednnih koraka:

* sql naredba se formira na osnovu ulaznih podataka

*sql naredba se priprema za izvrsavanje I zahteva se opisivanje rezultujuce relacije(ako postoji

*ako je u pitanju naredba select obezbedjuje se dovoljno prostora za smestanje podataka koje dohvatamo

*naredba se izvrsava(ako nije u pitanju anredba select) ili se hvata jedan red podatak (ako je u pitanju naredba select)

*vrsi se obrada dobijenih informacija

TIPOVI DINAMICKIH NAREDBI:

Dakle,razlikujemo sledece tipove naredbi:

*naredba koja nije SELECT:

-potpuna

-parametrizovana

*SELECT naredba

-sa fiksnom listom kolona koje se izdvajaju

-sa promenljivom listom kolona koje se izdvajaju

*nepoznata naredba

Naredba PREPARE:

Naredba PREPARE prevodi tekstualnu formu SQL naredbe u izvrsni oblik, dodeljuje pripremljenoj naredbi naziv I eventualno upisuje potrebne informacije u SQLDA strukturu

U okviru naredbe koja se priprema ne sme se pojavljivati niz kljucnih reci EXEC SQL ,znak za kraj naredbe ‘;’, host promenljive, niti komentari

Ukoliko naziv naredbe referise na vec postojecu pripremljenu naredbu, onda se ta prethodno pripremljena naredba unistava.

Neka je npr. V1 host promenljiva koja je u programu dobila vrednot - nisku karaktera neke izvrsne SQL naredbe . Tada naredba:

EXEC SQL PREPARE S1 FROM :V1

Na osnovu niske karaktera iz host promenljive V1 kreira izvrsnu SQL nardbu cije je ime S1. Ovim se u stvari analizira SQL naredba I utvrdjuje se najefikasniji nacin da se izdvoje trazeni podaci.

Metod na osnovu koga sistem DB2 bira kako pristupiti podacima se naziva plan pristupa.

Pripremljenu naredbu moguce je koristiti u narednim naredbama:

DESCRIBE(proizvoljna naredba),

EXECUTE(ne sme biti naredba SELECT)

DECLARE CURSOR(samo naredba SELECT)

Dinamicka SQL naredba ne sme da sadrzi obracanja host promenljivama. Umesto toga mogu se koristiti oznake parametara koje mogu biti neimenovane I imenovane. Neimenovana oznaka parametra se najcesce oznacava znako pitanja ‘?’ I navodi se na mestu na kome bi se navela host promenljiva kada bi naredba bila staticka. Postoje dve vrste neimenovanih oznaka parametara : sa opisom tipa I bez oisa tipa. Oznake parametara bez opisa tipa mogu se koristiti na mestma gde se u fazi pripreme moze na nedvosmislen nacin na osnovu konteksta oderediti njihov tip. Imenovane oznake parametara se navode dvotackom iz koje sledi identifikator oznake parametara npr. :parametar. Postoji slicnost sa host promenljivom, ali host promenljiva sadrzi vrednost u memoriji I koristi se direktno u statickoj SQL naredbi, imenovana oznaka predstavlja zamenu za vrednost u dinamickoj SQL naredbi I njena vrednost se navodi prilikom izvrsavanja pripremljene naredbe.

Naredba EXECUTE se moze upotrebljavati u dva oblika:

EXECUTE <naziv_naredbe> [USING <lista_host_prom>]

Ili

EXECUTE <naziv_naredbe> [USING DESCRIPTOR <lista_host_prom>]

Ako pripremljena naredba sadrzi oznake parametara, onda se prilikom navodjenja EXECUTE naredbe mora navesti neka od USING stavki. U slucaju da se navodi lista host promenljivih, njihov broj mora odgovarati broju oznaka parametara u naredbi, a tipovi host promenljivih redom tipovima parametara.

Ako se upotrebljava druga vrsta USING stavke, njen argument je SQLDA struktura koja mora biti popunjena podacima o parametrima na odgovarajuci nacin. Naredna polja SQLDA strukture moraju biti popunjena na sledeci nacin:

• polje SQLN sadrzi broj alociranih SQLVAR struktura,

• polje SQLDABC sadrzi ukupan broj bajtova alociranih za SQLDA strukturu,

• polje SQLD sadrzi broj promenljivih koji se upotrebljava pri obradi naredbe,

• polja SQLVAR sadrze opise pojedinacnih promenljivih.

Najcesce se za izracunavanje broja bajtova koji je otreban za smestanje svih podataka koristi se makro SQLDASIZE(n) , gde je n broj potrebnih SQLVAR struktura, odnosno vrednost polja SQLD.

Neke od naredbi koje se mogu izvrsiti (date u abecednom redosledu) su: ALTER, CREATE, DELETE, DROP, INSERT, SELECT, UPDATE. Medu izvrsnim SQL naredbama koje ne mogu da se izvrsavaju dinamicki nalaze se naredbe: CLOSE, DECLARE, EXECUTE, EXECUTE IMMEDIATE, FETCH, OPEN, PREPARE.

Naredba EXECUTE IMMEDIATE:

Ona je ugnjezdena(staticka) naredba pomocu koje se u jednom koraku pripremi I izvrsi dinamicka SQL naredba. Sintaksa ove naredbe je:

EXECUTE IMMEDIATE <host_prom>

gde je host_prom host promenljiva koja sadrzi nisku karaktera sa tekstom SQL naredbe.U okviru naredbe se ne sme pojavljivati niz kljucnih reci EXEC SQL, znak za kraj naredbe ‘;’, host promenljive, oznake parametara,niti komentari.Na ovaj nacins e moze izvrsiti svaka naredba koja nije SELECT, ako nije parametrizovana tj. Ako ne sadrzi oznake parametara.Info o greski o ovoj naredbi se nalaze u SQLCA strukturi. Cena pripreme se plaa svaki put kada se ona izvrsava pa se ova naredba primenjuje kada je SQL naredbu potrebno izvrsiti samo jednom.

Naredba DESCRIBE :

Naredba DESCRIBE upisuje potrebne info o naredbi u SGLDA strukturu,pod predpostavkom da je vec izvrsena njena priprema. Ptrebno je ukljuciti EXEC SQL INCLUDE SQLDA

Naredba DESCRIBE se najcesce upotrebljava u formi:

DESCRIBE [OUTPUT] <naziv_naredbe> INTO <ime_deskriptora>

Pri cemu mora postojati dinamicka SQL naredba pripremljena pod datim nazivom I mora biti alocirana odgovarajuca SQLDA deskriptorska struktura. Bitno je da je unapred popunjeno SQLN polje iz SQLDA strukture koje sadrzi maksimalan broj promenljivih.

Nakon izvrsavanja naredbe DESCRIBE, SQLDA struktura se popunjava na sledeci nacin:

• polje SQLDABC sadrzi velicinu SQLDA strukture u bajtovima;

• polje SQLD sadrzi broj kolona u rezultujucoj relaciji, ako je u pitanju naredba SELECT, ili broj oznaka parametara;

• polje SQLVAR sadrzi podatke o pojedinacnim parametrima: ako je SQLD nula, ili je vece od SQLN, ne dodeljuje se nista SQLVAR elementima. Svaka SQLVAR struktura sadrzi sledece komponente:

• polje SQLTYPE koje cuva oznaku tipa parametra i oznaku da li moze ili ne sadrzati NULL vrednost;

• polje SQLLEN koje sadrzi velicinu parametra u bajtovima;

• polje SQLNAME cija je vrednost naziv kolone ili tekstom zapisan broj koji opisuje originalnu poziciju izvedene kolone.

Koriscenje SQLDA strukture pruza vecu fleksibilnost od koriscenja liste host promenljivih. Npr. Koriscenjem SQLDA strukture moguce je preneti podatke koji nemaju nativni ekvivalent u maticnom jeziku (npr DECIMAL u programskom jeziku C)

Korisnik je duzan da nakon izvrsavanja naredbe DESCRIBE odgovarajucim podacima popuni preostal polja SQLVAR strukture :

* polje SQLDATA mora sadrzati pokazivac na prostor predvidjen za parametar (bilo za citanje ili pisanje parametra)

* polje SQLIND mora sadrzati pokazivac na prostor predvidjen za indikator , ako parametar moze imati NULL vrednost

SQLN - KOLIKO PROSTORA NEOPHODNO

SQLD - SADRZACE BROJ KOLONA U REZULTATU

Dinamicki kursori:

Dinamicki pripremljena SELECT naredba se moze izvrsiti koriscenjem kursora. Umesto statickog upita potrebno je navesti naziv pripremljenog dinamickog upita.Prilikom rada sa dinamickim kursorima potrebno je da se izvrse naredni koraci:

*priprema naredbe,

*deklarisanje kursora pod datim nazivom,

*otvaranje kursora,

*dohvatanje vrsta iz rezultujuce relacije,

*zatvaranje kursora

DIREKTNI POZIVI FUNKCIJA SUBP (DB2 CLI) :

DB2 CLI je IBM-ov C I C++ aplikacioni programski interfejs (API) za pristup relacionim bazama podataka. On koristi pozive funkcija ciji su argumenti SQL naredbe I dinamicki ih izvrsava.

DB2 CLI aplikaciju nema potrebe ponovo pretprocesirati ili iznova vezivati da bi se pristupilo drugom proizvodu baza podataka, vec se ona povezuje da odgovarajucom bazom podataka u vreme izvrsavanja programa.

DB2 CLI aplikacije se mogu povezati na veci broj baza podataka, ukljucujuci I veci broj konekcija na istu bazu podataka. Svaka od konekcija ima svoj zasebni prostor.

DB2 CLI sam alocira I kontrolise posebne strukture podataka I obezbedjuje mehanizme kojima aplikacija moze na njih da referise.

SLOGOVI:

U programu se mogu kreirati I moze se raditi sa cetiri tipa slogova (struktura u C-u):

*okruzenjima - slog ovg tipa se pravi od strane aplikacije kao priprema za jednu ili vise konekcija sa bzaom podataka

*konekcijama - slog ovog tipa se pravi da bi se apliakcija povezala sa bazom podataka; svaka konekcija postoji unutar nekog okruzenja;

*naredbama - u aplikaciji se moze kreirati jedan ili vise slogova naredbi. Svaka od njih cuva info o pojedinacnoj SQL naredbi, koja ukljucuje I podrazumevani kursor ako je naredba upit.

*opisima/deskriptorima - slogovi ovg tipa cuvaju info o kolonama relacije koaj predstavlja rezultat upita ili o dinamickim parametrima u SQL naredbama. Svaka naredba ima nekoliko opisnih slogova koji se implicitno prave(naziv atributa I njihovi tipovi) , s korisnik ih moze napraviti jos, ukoliko je potrebno.

Navedenim slogovima se u programu rukuje putem pokazivaca na slog. Koriscenjem slogova se izbegava potreba da se alociraju I da se upravlja globalnim promenljivama I strukturama podataka.

Slogovi se prave koriscenjem funkcije:

SQLRETURN SQLAllocHandle(SQLSMALLINT hType, SQLHANDLE hIn, SQLHANDLE *hOut)

Argumenti ove funkcije su:

1. hType je tip zeljenog sloga I ima vrednost kada zelimo da kreiramo slog okruzenja, nove konekcije, nove naredbe, slog opisa

2. hIn je slog elementa viseg nivoa u koji se novoalocirani element smesta. Ako pravimo slog konekcije onda je hIn slog okruzenja u okviru kojeg se kreira konekcija a ako pravimo slog anredbe ili slog opisa onda je hIn slog konekcije u okviru koje se kreira anredba.

3. hOut je adresa sloga

SQLALLOCHANDLE VRACA VREDNOST TIPA SQLRETURN (celobrojnu vrednost). Ova vrednost je jednaka SQL_SUCCESS ako je izvrsavanje naredbe bilo uspesno, a SQL_ERROR ako je izvrsavanje dovelo do greske.

Sa funkcijom SQLFreeHandle oslobadjamo sve slogove.moramo prvo osloboditi slogove konekcija da bi se oslobodili slogovi okruzenja da ne bi doslo do SQL_ERROR

POVEZIVANJE NA BAZU:

U okviru koda potrebno je povezati se na bazu podataka i to se postize naredbom:
SQLRETURN SQLConnect(SQLHDBC hdbc, SQLCHAR db_name, SQLSMALLINT db_name_len, SQLCHAR user_name, SQLSMALLINT user_name_len, SQLCHAR password, SQLSMALLINT password_len);

pri cemu su argumenti ove funkcije redom:

1. hdbc – slog konekcije; funkcija SQLAllocHandle(SQL_HANDLE_DBC,...,&hdbc) mora biti pozvana pre ove funkcije

2. db_name – naziv baze podataka

3. db_name_len – duzina naziva baze podataka

4. user_name – identifikator korisnika

5. user_name_len – duzina identifikatora korisnika

6. password – sifra

7. password_len – duzina sifre

TIP SQLRETURN VRACA VREDNOST UKOLIKO JE USPESNA ONDA SQL_SUCCESS

SQLRETURN SQLDisconnect(SQLHDBC hdbc); RASKIDA SE KONEKCIJA

OBRADA NAREDBI:

Proces pridruzivanja I izvrsavanja SQL naredbe je analogan pristupu koji se koristi u dinamickom SQL-u : tamos mo pridruzivali tekst SQL naredbe host promenljivoj koriscenjem naredbe PREPARE I onda je izvrsavali koriscenjem nardebe EXECUTE. Ako na slog naredbe gledamo kao na host promenljivu, situacija sa CLI pristupom je prilicno slicna. Postoji fja:

SQLRETURN SQLPrepare(SQLHSTMT sh, SQLCHAR st, SQLINTEGER si)

ciji su argumenti:

1. sh – slog naredbe

2. st – niska karaktera koja sadrzi SQL naredbu

3. si – duzina niske karaktera st. Ukoliko duzina niske nije poznata, a niska je terminisana, moze se proslediti konstanta SQL_NTS koja govori funkciji SQLPrepare da sama izracuna duzinu niske.

Funkcijom:

SQLRETURN SQLExecute(SQLHSTMT sh) izvrsava se naredba na koju referise slog naredbe sh. I naredba SQLPrepare i naredba SQLExecute vracaju vrednost tipa ´ SQLRETURN

SQLRETURN SQLRowCount(SQLHSTMT sh, SQLLEN *vr) kojom se vraca broj redova tabele na koje je naredba imala uticaj.

CITANJE PODATAKA IZ REZULTATA UPITA:

Pre nego sto krenemo da citamo podatke, komponenta n-torke se moze povezati sa promenljivom u maticnom jeziku, pozivom funkcije:

SQLRETURN SQLBindCol(SQLHSTMT sh, SQLUSMALLINT colNo, SQLSMALLINT colType, SQLPOINTER pVar, SQLLEN varSize, SQLLEN *varInfo)

sa narednom listom argumenata:

1. sh je slog naredbe,

2. colNo je redni broj komponente (u okviru n-torke) cije vrednosti citamo

3. colType je kod tipa promenljive u koju se smesta vrednost komponente (kodovi su definisani u zaglavlju sqlcli.h i imaju vrednosti npr. SQL_CHAR za niske karaktera ili SQL_INTEGER za cele brojeve),

4. pVar je pokazivac na promenljivu u koju se smesta vrednost,

5. varSize je velicina u bajtovima promenljive na koju pokazuje pVar,

6. varInfo je pokazivac na celobrojnu vrednost koja se moze koristiti da bi se obezbedile neke dodatne informacije; na primer poziv SQLFetch vraca vrednost SQL_NULL_DATA na ovom mestu argumenta, ako je vrednost kolone NULL

PROSLEDJIVANJE PARAMETARA UPITU:

Ugradeni SQL daje mogu cnost izvrsavanja SQL naredbe, pri cemu se deo naredbe sastoji od vrednosti koje su odredene tekucim vrednostima host promenljivih. Slican mehanizam postoji i u CLI pristupu, ali je malo komplikovaniji. Koraci koji su potrebni da bi se postigao isti efekat su:

1. iskoristiti funkciju SQLPrepare za pripremu naredbe u kojoj su neki delovi koje nazivamo parametrima zamenjeni znakom pitanja. i-ti znak pitanja predstavlja i-ti parametar;

2. iskoristiti funkciju SQLBindParameter za vezivanje vrednosti za mesta na kojima se nalaze znakovi pitanja;

3. izvrsiti upit koriscenjem ovih veza, pozivom funkcije SQLExecute. Primetimo da ako promenimo vrednost jednog ili vise parametara, moramo ponovo da pozovemo ovu funkciju sa novim vrednostima parametara.

Za vezivanje vrednosti za oznake parametra koristimo funkciju

SQLRETURN SQLBindParameter(SQLHSTMT sh, SQLUSMALLINT parNo, SQLSMALLINT inputOutputType, SQLSMALLINT valueType, SQLSMALLINT parType, SQLULEN colSize, SQLSMALLINT decimalDigits, SQLPOINTER parValPtr, SQLLEN bufferLen, SQLLEN *indPtr)

koja ima narednih 10 argumenata:

1. sh je slog naredbe

2. parNo je redni broj parametra (pocevsi od 1)

3. 3. inputOutputType je tip parametra: SQL_PARAM_INPUT – za SQL naredbe koje nisu zapamcene procedure, SQL_PARAM_OUTPUT – izlazni parametar zapamcene procedure, SQL_PARAM_INPUT_OUTPUT – ulazno-izlazni parametar zapamcene procedure,

4. valueType je C tip parametra, npr. SQL_C_LONG,

5. parType je SQL tip parametra, npr. SQL_INTEGER,

6. colSize je preciznost odgovarajuceg parametra, npr. maksimalna duzina niske

7. decimalDigits je broj decimala, ako je tip parametra SQL_DECIMAL

8. parValPtr je pokazivac na bafer za smestanje parametra

9. bufferLen je velicina bafera za smestanje parametra

10. indPtr je pokazivac na lokaciju koja sadrzi duzinu parametra koja se cuva u parValPtr; sluzi npr. za zadavanje NULL vrednosti parametra tako sto se vrednost ovog polja postavi na vrednost SQL_NULL_DATA

SUMIRANJE NEKO ZA DB2 CLI 39. STR

UPOTREBA ODBC STANDARDA: BLA BLA UOPSTENO O NJEMU NAS ZANIMA **JDBC**

JDBC API obezbedjuje Java programerima standardizovan nacin za pristup I komunikaciju sa bazom podataka, nezavisno od drajvera I proizvoda baza podata. Java kod prosledjuje SQL naredbe kao argumente DB2 funkcija JDBC drajveru I drajver ih dalje obradjuje.

UVOD U JDBC :

Imprt java.sql.* ; - na ovaj nacin su na raspolaganju JDBC klase

Class.forName(<ime_drajvera>); - ukljucujemo drajver zavisno od SUBP-A

Ucitava se odgovarajuca jlasa I onda DriverManager pomocu kojeg se moze ucitati I registrovati proizvoljan broj drajvera

Koristimo getConnection na objekat klase DriverManager pomocu kojeg uspostavljamo konekciju sa bazom podataka. Razlicit URL za svaki SUBP. Konekcija se raskida pozivom metoda : void close() na objekat klase Connection.Pozeljno je eksplicitno raskinuti sve konekcije sa bazom kada vise nisu potrebne.

Izuzeci try-catch blokovi, DB2 baca izuzetak tipa SQLException uvek kada dodje do greske prilikom izvrsavanja SQL naredbe,takodje I SQLWarning. Objekat klase SQLException sadrzi informacije o nastaloj gresci koje je moguce dobiti narednim metodama:

* getMessage() - vraca tekstualni opis koda greske,

* getSQLState() - vraca nisku SQLSTATE,

* getErrorCode() - vraca celobrojnu vrednost koja ukazije na tip greske

PRAVLJENJE NAREDBI U JDBC-U :

Naredbu je moguce napraviti koriscenjem dva razlicita metode od kojih se svaka primenjuje na objekat klase Connection:

1. Metod createStatement() - vraca objekat klase Statement. Moze se koristiti za naredbe SQL-a koje ne sadrze parametre. Analogan funkciji SQLAllocHandle.

2. Metod prepareStatement(Q), gde je Q niska koja sadrzi naredbu SQL-a I vraca objekat klase PreparedStatement. Analogan SQLAllocHandle dobija se slog I upit Q primenjuje fja SQLPrepare Koristi se nad SQL naredbom koja sadrzi parametre.

Cetiri metode za izvrsavanje nardebe su:

1. executeQuery(Q) koja kao argument prima nisku Q koja sadezi upit u SQL-u I primenjuje je na objekat klase Statement. Ovaj metod vraca objekat klase ResultSet, koji je skup n-torki u rezultatu upita Q.

2. executeQuery() koja se primenjuje na objekat klase PreparedStatement. S obzirom na to da vec pripremljena naredba ima pridruzen upit, ova funkcija nema argumenata. I ona vraca objekat klase ResultSet.

3. executeUpdate(U) koja prima kao argument nisku U koja sadrzi SQL naredbu koja nije upit I kada se primeni na objekata klase Statement izvrsava SQK naredbu U . Efekat ove operacije se moze uociti samo na samoj bazi podataka; ne braca se objekat klase ResultSet vec vraca broj redova na koje je ta naredba uticala , odnosno 0 ako je u pitanju naredba DDL-a.

4. executeUpdate() koja nema argument I primenjuje se na vec pripremljenu naredbu.U ovom slucaju izvrsava se SQL naredba koja je pridruzena naredbi. SQL naredba ne sme biti upit.

Kada anredba nije potrebna dobra praksa je zatvoriti je pozivom metoda close(). Isto vazi I za objekte klase ResultSet. Ovim se oslobadju resutrsi koje su ovi objekti zahtevali.

OPERACIJE NAD KURSORIMA U JDBC-U :

Kada izvrsavamo upit I dobijemo skup redova u rezultatu,mozemo pokrenuti kursor kroz n-torke u skupun rezultata. Klasa ResultSet raspolaze narednim metodama:

* next() - implicitni kursor pomeri na sledecu n-torku,ako nema anredne n-torke ova metoda vraca FALSE

* getString(i), getInt(I), getFloat(I) -vraca I-tu komponentu n -torke na koju kursor trenutno ukazuje

* close() - zatvara se objekat klase ResultSet; oslobadjaju se resursi baze podataka I JDBC resursi koje je ovaj objekat zauzimao

* wasNull() - proverava da li je poslednja procitana vrednost iz rezultujuceg skupa bila NULL, na ovaj nacin se u JDBC pristupu rukuje nedostajucim rednostima

* first() I last() - kursor se pozicionira na prvi odnosno oslednji red u rezultatu; vraca FALSE ako nema redova u rezultujucem skupu.

Moguce je definisati kursor kojim se moze kretati I unapred I unazad ,navodjenjem odgovarajucih vrednosti kao argumenata funkcija preparedStatement() ili createStatement().

Naredne tri konstante se koriste za eksplicitno zadavanej dozvoljenih smerogva kojim je moguce kretati se po skupu redova u rezultatu:

*TYPE_FORWARD_ONLY: moguce je kretati se samo unapred po skupu redova u rezultatu

*TYPE_SCROLL_SENSITIVE: moguce je kretati se I unapred I unazad I izmene su odamh vidljive

*TYPE_SCROLL_INSENSITIVE: moguce je kretati se I unapred I unazad I izmene nisu odmah vidljive

Prosledjivanje parametara: Potrebno je napraviti pripremljenu naredbu, a zatim pre izvrsavanja naredbe za parametre vezati vrednosti. To se moze psotici primenom metoda kao sto je recimo setString(I,v),setInt(I,v) koji vezuje vrednost v za I-ti parametar upita; Takodje se na ovaj nacin I NULL vrednost moze proslediti.

ZAPAMCENE PROCEDURE:

One omogucavaju pisanje procedura u jednostavnom jeziku opste namene I njihovo pamcenje u bazi podataka , kao dela sheme. Tako definisane procedure mogu se koristiti u SQL upitima I drugim naredbama za izraunavanja kojas e ne mogu uraditi pomocu SQL-a.

Zapamcena procedura je programski blok koji se poziva iz aplikacije na klijentu a izvrsava se na serveru baza paodataka. Pise se u odgovarajucim prosirenjima SQL, kompilira I pamti u biblioteci odgovarajuceg SUBP. Ovime se povecava funkcionalnost servera. Najcesci razlog za koriscenje zapamcenih procedura je intenzivna obrada podataka iz baze podataka koja proizvodi malu kolicinu rezultujucih podataka, ili cinjenica da je skup operacija (koje se izdvajaju u zapamcenu proceduru) zajednicki za vise aplikacija.

Prednosti :

1. koriste prednost mocnih servera
2. Donose poboljsanja performansi staticom SQL-u
3. Smanjuju mrezni saobracaj
4. Poboljsavaju integritet podataka dopustanjem raznim aplikacijama da pristupe istom programskom kodu

DEFINISANJE FUNKCIJA I PROCEDURA:

Definisu se moduli koji su kolekcije definicija procedura I funkcija, deklaracija privremenih relacija I jos nekih opcionih deklaracija.

Procedura se definise na sledeci nacin:

CREATE PROCEDURE <naziv>(<parametri>)

<lokalne_deklaracije>

<telo_procedure>;

Funkcija se definise na slican nacin, osim sto se koristi kljucna rec FUNCTION i sto postoji povratna vrednost koja se mora zadati.

CREATE FUNCTION <naziv>(<parametri>) RETURNS <tip>

<lokalne_deklaracije>

<telo_funkcije>;

Parametri procedure se ne zadaju samo svojim nazivom I tipom , vec im prethodi I naziv moda, koji moze biti IN za parametre koji su samo ulazni, OUT za parametre koji su samo izlazni, odnosno INOUT za parametre koji su ulazni-izlazni. Podrazumevano IN .

NEKE JEDNOSTAVNE FORME NAREDBI:

1. naredba poziva: poziv procedure je

CALL < naziv_procedure> (<lista_argumenata>)

-moze se naci u maticnom jeziku, ili u okviru neke druge funkcije ili procedure

2. Naredba vracanja vrednosti procedure:

RETURN <izraz>

-moze se javitit samo u funkciji

3. Deklaracija lokalnih promenljivih

DECLARE <naziv> <tip>

Deklarise promenljivu datog tipa sa datim nazivom,moraju da prethode izvrsnim naredbama u telu funkcije ili procedure

4. Naredbe dodele:

SET <promenljiva> = <izraz>

Vrednost izraza sa desne strane dodeljuje se promenljivoj sa leve strane, moze I NULL da se dodeli

5. Grupne naredbe:

Mozemo formirati listu naredbi koje se zavrsavaju tackom I zarezom I koje su ogradjene kljucnim recima BEGIN I END

6. Imenovanje naredbi: naredbu mozemo imenovati tako sto joj kao prefiks navedemo ime I stavimo dvotacku

NAREDBE GRANANJA:

IF <uslov> THEN

<lista_naredbi>

ELSEIF <uslov> THEN

<lista_naredbi>

ELSEIF

...

[ELSE ---------- OVO JE OPCIONA STVAR

<lista_naredbi>]

END IF;

UPITI:

Postoji nekoliko nacina na koje se u zapamcenim procedurama mogu koristiti upiti:

1. Upiti se mogu koristiti u uslovima, odnosno opstem slucajy na svakom mestu gde se u SQL-u moze naci podupit

2. Upiti koji vracaju jednu vrednost mogu se koristiti sa desne strane naredbe dodele

3. Upit koji vraca jedan red je legalna naredba u zapamcenoj proceduri

4. Nad upitom mozemo deklarisati I koristiti kursor , isto kao kod ugradjenog SQL-a. Nema prefiksa EXEC SQL na pocetku naredbe.

PETLJE:

Osnovna konstrukcija za petlju je oblika:

LOOP

<lista_naredbi>

END LOOP;

Ovim se definise beskonacna petlja iz koje se moze izaci samo nekom naredbom za transfer kontrole toka. Ova vrsta petlje se cesto imenuje da bi se iz nje moglo izaci koriscenjem naredbe:

LEAVE <naziv_petlje> ;

Najcesce se u petlji putem kursora citaju n-torke u rezultatu upita I iz petlje je potrebno izaci kada vise nema n-torki u rezultatu upita. Korisno je uvesti naziv za uslov na vrednost promenljive SQLSTATE koji ukazuje da nijedna n-torka nije procitana (za vrednost '02000')

FOR PETLJA:

Ona se koristi samo za iteriranje kroz kursor. Naredba FOR petlje ima sledeci oblik:

FOR [<naziv_petlje> AS] [<naziv_kursora> CURSOR FOR]

<upit>

DO

<lista_naredbi>

END FOR;

Ova naredba ne samo da deklarise kursor vec nas oslobadja velikog broja tehnickih detalja: otvaranja I zatvaranja kursora, citanja podataka I provere da li vise nema n-torki za citanje. Na kursor koji je deklarisan u FOR petlji se ne moze referisati van petlje: dakle poziv naredbe OPEN, FETCH ili CLOSE rezultovace greskom. U okviru FOR petlje moze se javiti naredba LEAVE , ali se onda kompletna petlja mora imenovati.

WHILE PETLJA:

- Forma:

WHILE <uslov> DO

<lista_naredbi>

END WHILE

REPEAT-UNTIL PETLJA:

REPEAT

<lista_naredbi>

UNTIL <uslov>

END REPEAT

IZUZECI:

SQL sistem nam ukazuje na potencijalne greske postavljanjem nenula niza vrednosti u nisku SQLSTATE. Npr. kod: ‘02000’ - nije pronadjena naredna n-torka, ‘21000’ - SELECT koji vraca jedan red vraca vise redova

Kod zapamcene procedure imamo HVATAC IZUZETKA koji se poziva uvek kada promenljiva SQLSTATE dobije vrednost jedan od kodova iz date liste kodova. Hvatac izuzetaka se nalazi u okviru bloka BEGIN END I primenjuje se samo na naredbe unutar tog bloka. Njegove komponente su:

1. Lista kodova izuzetaka za koje se poziva hvatac,

2. Kod koji se izvrsava kada se uhvati neki od pridruzenih izuzetaka

3. Naznaka gde treba ici nakon sto je hvatac zavrsio posao

DECLARE <gde_ici_nakon> HANDLER FOR <lista_uslova>

<naredba>

Gde ici nakon obrade izuzetaka:

1. CONTINUE- izvrsava se naredba sledeca

2. EXIT - izlazi se iz bloka BEGIN END I izvrsava se naredna naredba

3. UNDO - ima isto znacenje kao EXIT samo se ponistavaju sve izmene na bazi podataka

TRANSAKCIJE

TRANSAKCIJA I INTEGRITET:

Transakcija je logicka jedinica posla pri radu sa podacima. Ona predstavlja niz radnji koje ne narusava uslove integriteta. Sa stanovista korisnika, izvrsavanje transakcija je atomicno.

Po zavrsetku kompletne transakcije stanje baze treba da bude konzistentno, tj. Takvo da su ispunjeni uslovi integriteta. Ona prevodi jedno konzistentno stanje baze podataka u drugo takvo stanje baze, dok u medjukoracima transakcije konzistentnost podataka moze biti narusena.Transakcija na taj nacin predstavlja I bezbedno sredstvo za inetrakciju korisnika sa bazom. Sistem za upravljanje transakcijama obezbedjuje da se, bez greske, upisuju u bazu podataka ili efekti izvrsenja svih radnji od kojih se transakcija sastoji ili nijedne radnje.

RSUBP DB2 koristi dve radnje upravljaca transakcija: to su COMMIT I ROLLBACK. COMMIT oznacava uspesan kraj transakcije u bazi podataka, dok ROLLBACK oznacava neuspesan kraj transakcije I ponistenje svih efekata koje su proizvele radnje te transakcije.

Ugnjezdene transakcije u DB2 sistemu nisu moguce.

KONKURENTNOST:

Prednosti konkurentnog (istovremenog) rada transakcija jeste krace vreme odziva ,kao I veca propusnost. Pod izvrsenjem skupa transakcija ovde se podrazumeva niz radnji od kojih se te transakcije sastoje. Kao osnovne komponente transakcije posmatracemo objekte(npr.slogove) I dve radnje citanje I upis tj. Azuriranje. Dve radnje u ovom modelu su konfliktne ako se obavljaju nad istim objektom a jedna od njih je radnja upisa. To znaci da parovi konfliktnih radnji mogu biti samo parovi (citanje, upis) I (upis, upis)

Problemi pri konkurentnom radu:

Pri konkurentnom izvrsavanju skupa transakcija mogu nastati sledeci problemi:

1. problem izgubljenih azuriranja
2. Probklem zavisnosti od nepotvrdjenih podataka
3. Problem nekonzistentne analize

Problem izgubljenih azuriranja:

Neka se svaka od transakcija A I B sastoji od citanja I upisa istog sloga datoteke , pri cemu transakcije A I B najpre citaju posatke, a zatim vrse upis u istom redosledu. Npr A procita podatke I B procita podatke , A izmeni podatke , onda B izmeni podatke , na kraju ostaju samo podaci koje je B izmenio I nista vise , a ovi od A kao da ne postoje.

Problem zavisnosti od nepotvrdjenih podataka:

Neka transakcija a ukljucuje citanje nekog sloga R, a da rad transkcije B ukljucuje azuriranje istog sloga. Prvo B azurira slog R , pa A procita slog R I zavrsi izvrsavanje. Kasnije B ponisti efekte U tom slucaju transakcija A je procitala azuriranu vrednost sloga R koja nije ni trebalo da bude upisana u bazu podataka.

Slicna prica I da transakcija A ima azuriranje.

Problem nekonzistentne analize:

Ovaj problem se desava kada jedna transakcija cita nekoliko vrednosti,a druga transakcija azurira neke od tih vrednosti tokom izvrsavanja prve transakcije. Npr transkacija A sabira sve vrednosti sa tri racuna, a transakcija B prenosi 1000 din sa treceg racuna na prvi. Tada moze da se desi sityuacija gde suma svih racuna moze da bude manja od realne jer B prenosi 1000 dinara u nekom trenu.

Do ponistenja transakcije moze doci, zbog greske u transakciji ili zbog pada sistema.

Sva tri problema bice resavana koriscenjem mehanizma zakljucavanja. Mehanizam zakljucavanja obezbedjuje konkurentno izvrsenje skupa transakcija koj je ekvivalentno serijskom izvrsenju transakcija - kada se transakcije izvrsavaju jedna za drugom, bez preplitanja sa radnjama druge transakcije.Kazemo da je izvrsavanje datog skupa transakcija korektno ako je serijalizabilno tj. Ako proizvodi isti rezultat kao I serijsko izvrsavanje istog skupa transakcija.

ZAKLJUCAVANJE:

Zakljucavanje podrazumeva postavljanje katanaca na objekat. Ideja zakljucavanja je sledeca: kada transakcija zeli da radi sa nekim resursom ona zahteva zakljucavanje tog objekta (postavlja katanac nad tim objektom(, a kada se transakcija zavrsi taj resurs se oslobadja.

Scojstva katanca:

* vrsta, mod, rezim - odredjuje nacine pristupa koji su dozvoljeni vlasniku katanaca , kao I nacine pristupa koji su dozvoljeni konkurentnim korisnicima zakljucanog objekta

* objekat - resurs koji se zakljucava objekat odredjuje I opseg tj granularnost katanca

* trajanje - vremenski interval tokom koga se katanac drzi , on zavisi od nivoa izolovanosti

Sistem podrzava dve vrste katanaca:

* deljivi katanac - S

* privatni ili eksluzivni katanac - X

S katanac je potreban za citanje, a X katanac za azuriranje

Dvofazni protokol zakljucavanja - podrazumeva da se izvrsavanje svake transakcije sastoji iz dve faze: faze zakljucavanja objekta I faze otkljucavanja objekta I te dve faze se izvrsavaju jedna za drugom(serijski) . U fazi zakljucavanja nema nijedne radnje otkljucavanja objekta, dok u fazi otkljucavanja vise nema radnji zakljucavanja (ali naravno obe ove faze u sebi ukljucuju I druge operacije kaos to su citanje, upis, itd) . Faza otkljucavanja objekata zapocinje prvom radnjom otkljucavanja. Dakle, kada transakcija izvrsi zeljenu radnju(citanja, pisanja) ona moze da oslobodi katanac na tom slogu, ali nakon sto oslobodi neki katanac ne moze da zahteva nove katance.

Striktni dvofazni protokol zakljucavanja se pridrzava dvofaznog protokola zakljucavanja I dodatno u ovom protokolu se ekskluzivni katanaci oslobadjaju tek nakon kraja transakcije. Sastoji se od narednih koraka:

1. transakcija koja zeli da procita neku vrstu mora prvo fa nad njom postavi deljivi (S) katanac

2. Transakcija koja zeli da azurira neku vrstu mora prvo da nad njom postavi eksluzivni (X) katanac. Ako vec postoji (S) katanac od neke transakcije onda mora I (X) da postavi ista transakcija.

3. Ako imamo B transakciju koja dobije odbijanje jer je transakcija A vec postavila katanac onda B ide u stanje cekanja dok A ne oslobodi katanac I sistem mora da garantuje da B nece zauvek da ostane u stanju cekanja

4. Ekskluzivni katanac se zadrzava do kraja transakcije (naredbe COMMIT ili ROLLBACK). Deljivi katanac se uobicajeno zadrzava najduze do kraja transakcije.

Efekat zakljucavanja jeste da isforsira serijalizabilnost transakcija.

RESENJE NAVEDENIH PROBLEMA KONKURENTNOSTI:

Mrtva petlja:

Kod problema izgubljenih azuriranja I nekonzistentne analize, resavanje samog problema dovelo je do novog problema - u\zajamnog blokiranja transakcija. Mrtva petlja je situacija kada dve ili vise transakcija imaju postavljen katanac nad resursom koji je potreban drugoj transakciji tako da nijedna transakcija ne moze da nastavi sa radom. Mrtva petlja se otkriva koriscenjem grafa cekanja. To je usmereni graf koji ima po jedan cvor za svaku transakciju , dok (Ti ,Tj) oznacava da transakcija Tj drzi katanac nad resursom koji je potreban transakciji Ti, te transakcija Ti ceka da transakcija Tj oslobodi katanac na tom resursu. U ovako zadatom grafu mrtva petlja odgovara usmerenom ciklusu u grafu. Sledi slika: 67.str

Nakon sto se mrtva petlja pronadje neka od transakcija koja ucestvuje u mrtvoj petlji bira se za zrtvu, njeni efekti se ponistavaju I na taj ancin se oslobadjaju svi katanci koje je ona drzala.

Zakljucavanje sa namerom:

Prilikom zakljucavanja reda tabele , tabela koja sadrzi taj red se takodje zakljucava. Ovo zakljucavanaje sa namerom ukazuje na plan koji transakcija ima u principu podataka: namera da se vrsi zakljucavanje sa finijom glanularnoscu. U ovom slucaju koriste se naredni katanci I oni su izlistani u rastucem redosledu kontrole nad resursima:

* IS katanac: transakcija koja drzi ovu vrstu katanaca namerava da postavi S katanac nad pojedinacinm n-torkama tabele, svaki red mora da ima S katanac.

* IX katanac: transakcija koja drzi ovu vrstu katanaca moze da cita I da menja pojedinacne n-torke tabele, ali pre toga mora da dobije odgovarajuci katanac na pojedinacnim redovima koje zeli da cita/menja.

* S katanac : transakcija tolerise konkurentno citanje, ali ne I konkurentno azuriranje. Ne traze katanci na svakom redu

* SIX katanac: ovo je kombinacija S I IX katanca ,transakcija tolerise konkurentno citanje ,ali ako planira da vrsi azuriranje onda postavlja X katanac

* X katanac: ne tolerise nikakav konkurentan pristup tabeli, ne postavlja pojedinacke katance na svaku n-torku I moze a ne mora da azurira n-torku.

IS katanac je u konfliktu sa X katancem , dok je IX katanac konfliktan sa S I X katancem.

U KATANAC:

Update (U) katanac predstavlja hibrid deljenog I ekskluzivnog katanca. On se zahteva u situaciji kada transakcija zeli da cita ali ce naknadno zeleti I da menja dati red (pre nego sto krene da menja trazice ekskluzivni katanac nad tim redom). Koriscenje U katanca onemogucava da dva poziva iste transakcije naprave mrtvu petlju na istom redu. Namera da se menja onemogucava da dva poziva izvode operacije na istom redu u isto vreme, s obzirom na to da su oba poziva kroz definiciju kursora naglasila mogucu zelju da dobiju X katanac.

ZAKLJUCAVANJE U SISTEMU DB2:

DB2 UVEK ZAHREVA KATANAC NA TABELI PRE NEGO STO SE DOZVOLI PRISTUP PODACIMA. Tabela se u ovom pristupu zakljucava na nizem nivou restrikcije u odnosu na strategiju koja zakljucava samo tabelu. U sistemu DB2 moze se postaviti da se pri svakom pristupu tabeli zahtevaju katanci na sledeci nacin:

ALTER [TABLESPACE|TABLE] naziv LOCKSIZE TABLE

Vrste katanaca koje sistem DB2 koristi na nivou tabela:

IN - vlasnik katanaca moze da cita sve podatke u tabeli ukljucujici I nepotvrdjene podatke,ali ne mzoe da ih menja. Ostale aplikacije mogu da citaju ili da azuriraju tabelu, nikakvi katanci na redovima

IS - vlasnik katanca moze da cita proizvoljan podatak u tabeli ako se S katanac moze dobiti na redovima ili stranicama od interesa

IX - vlasnik katanca moze da cita ili menja proizvoljni podatak u tabeli ako je ili X katanac na redovima ili stranicama koje zelimo da menjamo ili S ili U katanac na redovima koje zelimo da citamo

SIX - vlasnik katanca moze da cita sve podatke iz tabele I da menja redove ako moze da dobije X katanac na tim redovima. Ostale aplikacije mogu da citaju.

S - vlasnik katanca moze da cita proizvoljan poredak u tabeli I nece dobiti katance na redovima ili stranicama

U - vlasnik katanca moze da cita prizvoljan podatak u tabeli I moze da menja podatke ako se na tabeli moze dobiti X katanac. Nema nikakvih katanaca na redovima ili stranicama

X - vlasnik katanca moze da cita I da menja proizvoljan podatak iz tabele. Pritom se ne dobijaju nikakvi kantanci na redovima ili stranicama

Z - ovaj katanac se zahteva na tabeli u posebnim prilikama, kao sto sub recimo menjanje strukture tabele ili njeno brisanje. Nijedna druga aplikacija ne moze da cita niti da azurira podatke u tabeli.

S,U,X se koriste na nivou tabela da nametnu striktnu strategiju zakljucavanja tabela.

Striktni katanac na tabelu se moze zahtevati naredbom:

EXEC SQL LOCK TABLE <naziv_tabele> IN SHARE|EXCLUSIVE MODE;

SHARE - S, EXCLUSIVE - X

EFEKTI TRAZENJA KATANACA:

U nekim situacijama katanci koji su potrebni da bi se izvrsila neka naredba jedne aplikacije su u konfliktu sa katancima koje vec imaju druge konkurentne aplikacije. Da bi se usluzio sto veci broj aplikacija, menadzer baza podataka omogucava funkciju eskalacije katanaca. Ovaj proces objedinjuje proces dobijanja katanca na tabeli I oslobadjanja katanaca na redovima. Cilj je smanjiti ukupnu kolicinu skladistenja za katance. Dva konfiguraciona parametra baze podataka imaju direktan uticaj na proces eskalacije katanaca:

* locklist: to je prostor u globalnoj memoriji baze podataka koji se koristi za skladistenje katanaca

* maxlocks: to je procenat ukupne liste katanaca koji je dozvoljeno da drzi jedna aplikacija

Oba parametra su konfigurabilna.

Eskalacija katanaca se desava u dva slucaja:

* aplikacija zahteva katanac koji bi proizveo da ona prevazidje procenat ukupne velicine liste katanaca, koji je definisan parametrom maxlocks. U ovoj situaciji menadzer baze podataka ce pokusati da oslobodi memorijski prostor.

* aplikacija ne moze da dobije katanac jer je lista katanaca puna. Menadzer baze podataka ce pokusati da oslobodi memorijski prostor.

Eskalacija katanaca moze da ne uspe. Ako se desi greska, aplikacija koja je pokrenula eskalaciju dobice vrednost SQLCODE -912.

Detekcija prekoracenog vremena citanja na katanac je svojstvo menadzera baza podataka kojim se sprecava da aplikacija beskonacno dugo ceka na katanac.

Mrtva petlja se obradjuje procesom u pozadini koji se naziva detektor mrtve petlje. Ako se uoci mrtva petlja, odredjuje se zrtva, zatim se za nju automatski poziva ROLLBACK I vraca se SQLCODE -911 with reason code 2. Ponistavanjem radnji zrtve oslobadjaju se katanci I to bi trebalo da omoguci drugim procesima da nastave sa radom.

Parametar dlcktime definise frekvenciju sa kojom se proverava da li je doslo do mrtve petlje izmedju aplikacija koje su povezane na bazu podataka. Ovaj parametar je konfigurabilan I vrednost mu je izrazena u milisekundama. Podrazumevana vrednost ovog parametra je 10s.

SVOJSTVA TRANSAKCIJA I NIVOI IZLOVANOSTI TRANSAKCIJA

Operacije COMMIT I ROLLBACK zavrsavaju transakciju, ali ne prekidaju izvrsavanje programa.

Izvrsavanje jedne transakcije moze da se zavrsi planirano ili neplanirano. Do planiranog zavrsetka dolazi izvrsavanjem operacije COMMIT kojom se transakcija uspesno zavrsava ili eksplicitne ROLLBACK operacije, koja se izvrsava kada dodje do greske za koju postoji programska provera. Neplanirani zavrsetak izvrsenja transakcije dogadja se kada dodje do greske za koju ne postoji programska provera; tada se izvrsava implicitna (sistemska) ROLLBACK operacija, radnje te transakcije se ponistavaju a program prekida sa radom.

Izvrsavanjem operacije COMMIT , efekti svih azurirana te transakcije postaju trajni, odnosno vise se ne mogu ponistiti procedurom oporavka. U log datoteku se upisuje odgovarajuci slog o komplementiranju transakcije (COMMIT slog) a svi katanci koje je transakcija drzala nad

objektima se oslobadjaju. Izvrsenje COMMIT operacije ne podrazumeva nuzno fizicki upis svih azuriranih podataka u bazu.

Pad transakcije nastaje kad transakcija ne zavrsi svoje izvrsenje planirano. Tada sistem izvrsava implicitnu - prinudnu ROLLBACK operaciju tj. Sprovodi aktivnost oporavka od pada transakcije.

Izvrsavanjem operacije BEGIN TRANSACTION u log datoteku se upisuje slog pocetka transakcije. Operacija ROLLBACK , bilo eksplicitna ili implicitna , sastoji se od ponistavanja ucinjenih promena nad bazom. Izvrsava se citanjem unazad svih slogova iz log datoteke koji pripadaju toj transakciji, do sloga BEGIN TRANSACTION.

ACID SVOJSTVA TRANSAKCIJE:

Transakcija se karakterise sledecim vaznim svojstvima (poznatim kao ACID

svojstva):

• atomicnost (eng. Atomicity): transakcija se izvrsava u celosti ili se ne

izrsava ni jedna njena radnja;

• konzistentnost (eng. Consistency): transakcija prevodi jedno konzistentno

stanje baze podataka u drugo konzistentno stanje;

• izolacija (eng. Isolation): efekti izvrsenja jedne transakcije su nepoznati

drugim transakcijama sve dok ona ne zavrsi uspesno svoj rad;

• trajnost (eng. Durability): svi efekti uspesno zavrsene transakcije su

trajni, odnosno mogu se ponistiti samo drugom transakcijom. Dakle,

nakon potvrde transakcije, promene ostaju upisane u bazi podataka, cak i u slucaju pada sistema.

NIVOI IZLOVANOSTI:

Na ovaj nacin se opisuje stepen ometanja koji tekuca transakcija moze da podnese pri konkurentnom izvrsavanju. Ove nivoe nazivamo nivoima izlovanosti transakcije. Ako su transakcije serijalizabilne stepen ometanja ne postoji, tj. Nivo izolovanosti je maksimalan. Realni siostemi iz razlicitih razloga dopustaju rad sa nivoima izolovanosti koji su manji od maksimalnog. Vazi sa sto je veci nivo izlovanosti manje su dopustene smetnje I obratno.

Nivou izolovanosti (od najjaceg do najslabijeg) :

• SERIALIZABLE

• REPEATABLE READ

• READ COMMITED

• READ UNCOMMITED

Prvi nivo izolovanosti SERIALIZABLE je najvisi nivo izolovanosti I garantuje serijalizabilnost izvrsenja. Slede tri nacina na koja serijalizabilnost moze da bude narusena:

* prljavo citanje odgovara pojmu zavisnosti od nepotvrdjenih podataka.

A - azurura neki slog

B - cita taj slog

A - ponisti promene

B - procitalo slog koji ne postoji : (

* neponovljivo citanje

A - cita slog

B - azurira taj slog

A - pokusava da procita “ Isti ” slog

* fantomsko citanje

A - cita skup slogovo koji zadovovljavaju neki uslov

B - unese novi slog koji zadovoljava isti uslov

A - sada ponovi zahtev, tj. Procita ponovo podatke po istom uslovu, videce slog koji ranije nije postojao tzv. Fantomski slog

NIVOI IZLOVANOSTI U SISTEMU DB2 :

U sistemu DB2 postoje naredni nivoi izolovanosti (koji su navedeni u redosledu

od najjaceg do najslabijeg):

• REPEATABLE READ (RR)

• READ STABILITY (RS)

• CURSOR STABILITY (CS)

– CURRENTLY COMMITED (CC)

• UNCOMMITED READ (UR)

Oni odgovaraju nivoima izlovanosti definisanim SQL standardom, ali su drugacije imenovani.

Najvisi nivo izolovanosti aplikacije u sistemu DB2 je nivo ponovljivog citanja - RR. Ovaj nivo obezbedjuje zakljucavanje svih vrsta kojima se transakcija obraca, a ne samo onih vrsta koje zadovoljavaju uslov upita. Tako se rezultat citanja transakcijom T ne moze promeniti od strane druge transakcije pre nego sto se transakcija T zavrsi. Katanci za citanje reda se drze do naredne naredbe COMMIT ili ROLLBACK. S druge strane, ovaj nivo izlovanosti obezvedjuje I da transakcija nema uvid u nepotvrdjene promene drugih transakcija.

Sledeci novo izlovanosti jeste nivo stabilnosti citanja - RS . Za razliku od prethodnog ovaj nivo izolovanosti obezbedjuje zakljucavanje samo onih vrsta koje zadovoljavaju uslov upita tj. Ne dopusta promenu (od strane drugih transakcija) vrste koju je procitala transakcija T , sve do zavrsetka transakcije T. Mada ce, pri ovom nivou izolovanosti, vrsta koju je procitala transakcija T ostati nepromenjena od strane drugih transakcija sve do zavrsetka transakcije T , rezultati ponovljenih izvrsenja istog upita od strane transakcije T mogu da se razlikuju. Katanci za citanje reda se drze do naredne naredbe COMMIT ili ROLLBACK.

Nivo stabilnosti kursora - CS obezbedjuje zakljucavanje samo one vrste koju transakcija t trenutno cita. Sve vrste koje je transakcija t prethodno procitala(ali ne I menjala) otkljucane su I druge transakcije ih mogu menjati I pre okoncanja transakcije T. rad sa ovim nivoom moze dovesti do fantomskog citanja I neponovljivog citanja jer neka druga transakcija moze azurirati neki slog koji je transakcija T ranije procitala. Transakcija T sa ovim nivoom izlovanosti I dalje nema uvid u promene drugih transakcija pre okoncanja tih transakcija. Ovaj nivo izlovanosti je podrazomevani u sistemu DB2

U novim verzijama DB2 podrazumeva se semantika trenutno potvrdjenih izmena - CC. Transakcijama sa nivoom izlovanosti CS koje implementiraju ovu semantiku se citaju samo potvrdjeni podaci.

Najslabiji nivo izlovanosti transakcije jeste nivo neponovljivog citanja - UR. Transakcija T sa ovim nivoom izlovonasti cita podatke bez toga da zahteva (I dobije) deljivi katanac na tom objektu. Ovaj nivo omogucuje transakciji T da procita nepotvrdjene promene drugih transakcija , kao I drugim transakcijama da pristupe podacima koje transakcija T upravo cita. Ovaj nivo kao READ ONLY za tabele.

X katanci se ne oslobadjjau sve do naredbe COMMIT ili ROLLBACK , bez obzira na nivo izlovanosti transakcije.

Izabrani nivo izlovanosti je vazeci tokom date jedinice posla.

Kako utvrditi nivo izlovanosti za neku SQL naredbu:

* za staticki SQL:

Koristi se vrednost tog nivoa ukoliko je postavljen, ako nije eksplicitno postavljen koristi se nivo koji je zadat prilikom ugradnje paketa u bazu

* za dinamicki SQL:

Koristi se vrednost tog nivoa ukoliko je postavljen, ako nije eksplicitno postavljen koristi se nivo koji je zadat naredbom SET CURRENT ISOLATION, a ako nije ni tom naredbom postavljen onda je nivo koji je zadat prilikom ugradnje paketa u bazu

Moze se I sa WITH <nivo_izolovanosti> postaviti .

KURSORI DEKLARISANI SA OPCIJOM WITH HOLD:

Nekada je zgodno da nakon uspesnog kraja transakcije otvoreni kursor I dalje ostane otvoren, moguce je navesti opciju WITH HOLD u deklaraciji kursora cime se postize da on ostane otvoren I nakon uspesnog kraja transakcije. Kursori u ugradjenom SQL - u mogu biti deklarisani koriscenjem opcije WITH HOLD na sledeci nacin:

DECLARE <naziv> CURSOR WITH HOLD FOR

    <upit>

Na ovaj nacin kursor ostaje otvoren sve dok se eksplicitno ne zatvori, ili se ne izvrsi naredba ROLLBACK.

Izvrsavanje naredbe COMMIT:

*zatvara sve kursore koji nisu definisani sa opcijom WITH HOLD

*oslobadja sve katance osim na tekucem redu kursora koji su definisani sa opcijom WITH HOLD

*transakcija se potvrdjuje I sve promene se trajno zapisuju u bazi podataka

Izvrsavanje naredbe ROLLBACK:

*zatvara sve kursore

*oslobadja sve katance

*ponistava sve promene nastale tokom transakcije

OPORAVAK:

Oporavak podrazumeva aktivnost koju sistem preduzima u slucaju da se u toku izvrsenja jedne ili vise transakcija otkrije neki razlog koji onemogucava njihov uspesan zavrsetak. Taj razlog moze biti:

*u samoj transakciji kao sto je prekoracenje neke od dozvoljenih vrednosti - pad transakcije

*u sistemu npr. Prestanak elektricnog napajanja - pad sistema ili

*u disku na kome je baza podataka - pad medijuma

Log datoteka se obicno koristi kao drugo mesto u sistemu , na koje se (osim u bazu) upisuju informacije o izvrsenim radnjama, naredbom GET DATABASE CONFIGURATION mozemo dobiti info o lokaciji log datoteke u sistemu, kao I o njenoj maksimalnoj velicini

Aktivnosti koje upravljac oporavka radi kada je neophodno oporaviti bazu podataka:

*periodicno prepisuje celu bazu podataka na medijum za arhiviranje

*pri svakoj promeni baze podataka, upisuje slog promene u log datoteku

*za svaku naredbu DDL-a upisuje odgovarajuci slog u log datoteku

*upravljac opravka koristi informacije iz log datoteke da ponisti dejstva parcijalno izvrsenih transkcija odnosno da ponovo izvrsi neke kompletirane transakcije

*kada padne medijum , koristi se info iz log datoteke za ponovno izvrsenje transakcija kompletiranih posle psolednjeg arhiviranja a pre pada medijuma

SUBP poseduje osim tzv. DO logike (“uradi”), I tzv. UNDO (“ponisti”) I REDO (“ponovo uradi”) . Pad sistema ili medijuma moze se dogoditi I u fazi oporavka od prethodnog pada. Zato moze doci do ponovnog ponistavanja vec ponistenih radnji, odnosno do ponovnog izvrsavanja vec izvrsenih radnji. Ova mogucnost zahteva da UNDO I REDO logika imaju svojstvo idempotentnosti tj. da je:

UNDO(UNDO(x)) ≡ UNDO(x)

REDO(REDO(x)) ≡ REDO(x)

Za svaku radnju x. U log datoteci pokazivacima su povezani slogovi koji se odnose na jednu transakciju I log datoteka nikada ne pada. KEKEKEKE

Oporavak od pada transakcije:

Izvrsavanjem operacije BEGIN TRANSACTION u log datoteku se upisuje slog pocetka transakcije . ROLLBACK se sastoji od ponistavanja ucinjenih promena nad bazom podataka. Citaju se unazad svi slogovi iz log datoteke koji pripadaju toj transakciji , do sloga pocetka

transakcije. Za svaki slog promena se ponistava sa UNDO, I aktivnost oporavka od pada ne ukljucuje REDO logiku.

Opravak od pada sistema:

U slucaju pada sistema , sadrzaj unutrasnje memorije je izgubljen. Zato se ponovnim startovanjem sistema za oporavak koriste podaci iz log datoteke da bi se ponistili efekti transakcija koje su bile u toku u trenutku pada sistema.

Procedura opravka od pada sistema kod ranijih SUBP:

Uvode se tacke preseka stanja , u momentu koji odredjueje tacku preseka stanja, fizicki se upisuju podaci I informacije iz log bafera I bafera podataka u log datoteku I u bazu podataka , redom I u log datoteku se upisuje slog tacke preseka stanja. Ovaj slog sadrzi informaciju o svim aktivnim transakcijama u momentu tacke preseka stanja, adrese poslednjih slogova tih transakcija u log datoteci, a moze sadrzati I niz drugih informacija o stanju baze podataka u tom trenutku. Adresa sloga tacke preseka stanja upisuje se u datoteku ponovnog startovanja. Prema protokolu upisivanja u log unapred obezbedjeno je da su zavrsetak transakcije (upis COMMIT sloga u log datoteku) I definitivni upis svih azuriranja te transakcije u bazu podataka - dve odvojene radnje, za koje se ne sme dogoditi da se jedna izvrsi a druga ne. Prema ovom protokolu pri izvrsenju operacije COMMIT najpre se odgovarajuci slog fizicki upisuje u log datoteku, pa se zatim podaci iz bafera podataka prepisuju u bazi podataka. Sa REDO logikom se restaurira sadrzaj iz log datoteke.

Poboljsavanje procedure oporavka od pada sistema:

Jedno evidentno poljsanje protokola oporavka od pada sistema odnosi se na aktivnosti vezane za tacku preseka stanja.

Moguce je eliminisati fizicko upisivanje bafera podataka u bazu podataka u tacki preseka stanja, a u fazi oporavka od pada sistema ponistavati samo one radnje neuspelih transakcija ciji su efekti zaista upisani u bazu, odnosno ponovo izvrsavati samo one radnje uspelih transakcija ciji efekti nisu upisani u bazu. Za tp slize serijski brojevi I oni su u rastucem poretku. Ako je serijski broj upisan u stranicu P veci ili jednak od serijskog broja sloga log datoteke, efekat azuriranja kome odgovara taj slog fizicki je upisan u bazu: ako je manji, efekat nije upisan.

Da bi se slog baze podataka azurirao, potrebno je procitati stranicu na kojoj se slog nalazi. Posle azuriranja sloga, stranica je spremna za upis u bazu, pri cemu I dalje nosi serijski broj svog prethodnog upisa u bazu.

Tacnije, za slog log datoteke uspele transakcije gde stranica baze podataka ima LSN vrednost vecu ili jednaku od LSN vrednosti sloga log datoteke , onda ne treba raditi nista. Efekat se cuva trajno na disku. Za slog log datoteke uspele transakcije u situaciji kada stranica baze

podtaka ima LSN vrednost manju od LSN vrednosti sloga log datoteke, potrebno je ponovo izvrsiti da bi se osiguralo da efekti transakcije budu trajno sacuvani.

Za slog log datoteke neuspele transakcije gde stranica baze posataka ima LSN vrednost vecu ili jednaku od LSN vrednosti tog sloga log datoteke, potrebno je ponistiti efekte tog sloga da bi se obezbedilo da efekti transakcije ne budu trajno sacuvani. Za slog log datoteke neuspele transakcije u situaciji kada stranica baze podataka ima LSN vrednost manju od LSN vrednosti sloga log datoteke,nije potrebno raditi nista. Efekat nije trajno na disku.

Novi SUBP imaju podrsku za tacke pamcenja koje omogucavaju da ako tokom izvrsavanja dodje do greske da se ne ponistavaju izmene kompletne transakcije,vec samo izmene koje su se desile tokom trajanja transakcije od trenutka kada je tacka pamcenja pocela do trenutka u kom je zahtevano ponistavanje efekata. Na ovaj nacin moguceno je da se jedna velika transakcija podeli u nekoliko delova tako da svaki od njih ima svoju definisanu tacku pamcenja.

SQL podrska za tacke pamcenja

Tacka pamcenja unutar neke transakcije se pravi naredbom:

SAVEPOINT <naziv>

Pozeljno je dati informativna imena tackama.

Dodatne opcije: UNIQUE, ON ROLLBACK RETAIN CURSORS, ON ROLLBACK RETAIN LOCKS.

Oslobadjanje tacke pamcenja vrsi se naredbom:

RELEASE SAVEPOINT <naziv>

Ako se tacka pamcenja eksplicitno ne oslobodi, ona ce biti oslobodjena na kraju transakcije.

Da bi se izvrsio povratak na neku tacku pamcenja potrebno je pozvati naredbu:

POLLBACK TO SAVEPOINT [<naziv>]

Ovom naredbom se sve izmene nad bazom podataka koje su vrsene nakon te tacke pamcenja ponistavaju.Ukoliko se ne navede naziv , vracamo se na poslednju aktivnu tacku pamcenja. Ukoliko je tacka pamcenja vec oslobodjena, nije moguce izvrsiti povratak na nju naredbom ROLLBACK TO SAVEPOINT.

OBJEKTNO - RELACIONO PRESLIKAVANJE I ALAT HIBERNATE

Alati za objektno - relaciono preslikavanje koji vrse automatsko preslikavanje relacionog modela baze podtaaka na objektni model aplikacije I obrnuto, oni omogucavaju da objektno - orijentisane aplikacije ne rade direktno sa tabelarnom reprezentacijom podtaka vec da imaju svoj objektno - orjentisani model. Jednom kada se napravi odgovarjauce preslikavanje za datu klasu, instance ove klase se mogu koristiti svuda u samoj aplikaciji. Alat Hibernate najpogodniji za prikaz resavanja ovog problema.

OBJEKTNO - ORIJENTISANO PRESLIKAVANJE

Trajnost ili perzistentnost podataka jedan je od osnovnih koncepata koji se podrazumevaju prilikom razvoja aplikacije. Trajnost objekta podrazumeva da pojedinacni objekti mogu da nadzive izvrsavanje aplikacije koja ih je kreirala - objekti se mogu sacuvati u skladistu podataka I mozemo im iznova pristupiti u nekom kasnijem trenutku. Nije neophodno trajno sacuvati sve objekte , ali je potrebno trajno sacuvati objekte koji su od kljucne vaznosti za poslovnu logiku aplikacije.

Racionalna tehnologija je danas najzastupljenija zbog svojih dobrih ososbina, izmedju ostalog zbog svog fleksibilnog I robusnog pristupa upravljanja podacima.

Nezavisnost podataka je princip koji je poznat tako da podaci zive duze od svake aplikacije, a relaciona tehnologija daje mogucnost deljenja podtaaka izmedju razlicitih aplikacija ili medju razlicitim delovima istog sistema.

KORISCENJE SQL-A U PROGRAMSKOM JEZIKU JAVA:

Tehnicka resenja ovih problema se dizajniraju imajuci na umu model domena. Model domena definise objekte koji predstavljaju probleme iz realnog sveta I kojima se objedinjuju ponasanje I podaci. Rezervacija leta, bankovnih racuna ili potraga za knjigom su primeri objekata iz domena, rezervacija leta, transfer novca sa jednog na drugi racun, kupovina knjiga…. Za ove domenske objekte se ocekuje da budu sacuvani u bazi podatak. Umesto da se direktno radi sa redovima I kolonama rezultijuceg skupa java.sql.ResultSet, poslovna logika aplikacije rukuje objektno - orijentisanim modelom domena koji je specifican za aplikaciju. Npr. , ako SQL shema baze podataka sadrzi tabelu Knjiga , Java aplikacija definise klasu Knjiga. Umesto da cita I pise vrednosti pojedinacnog reda I kolone, aplikacija ucitava I cuva instance klase Knjiga.

OBJEKTNO - RELACIONO NESLAGANJE:

Osnovna razlika je ta da model podataka prikazuje podatke (kroz kolone vrednosti) , dok objektni model skriva podatke (enkapsulirajuci ih iza javnih interfejsa) . Kombinacija objektne I relacione tehnologije moze da dovede do konceptualnih I tehnickih problema. Ti problemi su poznati pod nazivom objektno - relaciono neslaganje. Medju najvaznije probelem spadaju:

* neslaganje izmdju nivoa granularnosti u objektno - orijentisanom modelu I odgovarajucem relacionom modelu

* neslaganje hijerarhijskog uredjenja podataka - ne postoji mogucnost izgradnje hjerarhije izmdeju tabela u relacionoj bazi podataka, dok se klase mogu organizovati hijerarhijski

* neslaganje pojma istovetnosti - npr dva reda tabele koji sadrze ispodatke smatraju se istim, te iz tog razloga nije dozvoljeno unosenje dva ista reda u tabelu, dok objekti koji sadrze iste podatke mogu biti razliciti jer imaju razlicite memorijske adrese

* neslaganje u nacinima na koji se realizuju veze

* neslaganje u nacinu na koji se pristupa podacima u programskom jeizu Java I u relacionim bazama podataka

PROBLEM GRANULARNOSTI:

Granularnost se odnosi na relativne velicine tipova sa kojima radimo. Nekad ce objektni model imati veci broj klasa neo sto je broj odgovarajucih tabela u bazi podataka (kazemo da je objektni model granularniji od relacionog modela). S druge strane, u SQL bazma podataka postoje samo dva nivoa granularnosti : nivo relacija koje mi kreiramo, kao sto su recimo relacije Korisnici I Racun I ugradjeni tipovi podataka kao sto su recimo VARCHAR, BIGINT, …

PROBLEM PODTIPOVA:

Takodje, cim se uvede pojam nasledjivanja , postoji mogucnost polimorfizma. Klasa Korisnik ima definisanu vezu ka natklasi Racun. Ovo je polimorfna veza sto znaci da u vreme izvrsavanja instanca klase Korisnik moze da referise na instancu bilo koje potklase klase Racun. Slicno, zelimo da omogucimo da pisemo polimorfne upite koji bi referisali na klasu Racun, a kojim bi se vratile I instance svih potklasa te klase.

PROBLEM ISTOVETNOSTI:

Na problem istovetnosti se nailazi u situaciji kada je potrebno proveriti da li su dve instance identicne. Naime, postoje tri nacina na koja se ovaaj problem moze adresirati: dva u kontekstu programskog jezika Java I jedan u kontekstu SQL baze podtaka. U programskom jeziku Java definisu se dva razlicita pojma istovetnosti:

* identicnost instanci, koja se odnosi na ekvivalentnost memorijskih lokacija koaj se proverava uslovom a==b

* jednakost instanci, koja se utvrdjuje implementacijom metoda equals() - na ovu vrstu jednakosti se cesto referise kao na jednakost prema vrednosti.

Surogat kljucem podrazumeva kolonu koja cini primarni kljuc I koja nema nikakvo znacenje samom korisniku aplikacije - drugim recima kljuc koji se ne prikazuje korisniku aplikacije. Njegova jedina svrha jeste identifikacija podataka unutar aplikacije.

PROBLEM ASOCIJACIJA (VEZA):

U modelu domena, asocijacije (odnosno veze) predstavljaju odnose izmedju entiteta. Ako zelimo da u SQL bazma podataka predstavimo vezu tipa “vise ka vise “ potrebno je uvesti novu tabelu, koju najcesce nazivamo veznom tabelom. U vecini slucajeva ova tabela se ne javllja nigde u modelu domena.

PROBLEM NAVIGACIJE PODATAKA:

Lenjo ucitavanje je efekat poznat kao preuzimanje podataka (tj. materijalizacija slogova baze podtaak u objekte programa) vrsi se samo na zahtev, u trenutku kada su oni potrebni (kada im se pristupi u programskom kodu).

OBJEKTNO-RELACIONO PRESLIKAVANJE I JAVA PERSISTENCE API:

U sustini, objektno - relaciono presliakvanje (ORP) je automatsko (I transparentno) trajno cuvanje objekata iz Java aplikacije u tabelama SQL baze podataka, koriscenjem metapodataka koji opisuju preslikavanja izmedju klasa aplikacije I scheme SQL baze podataka. Ovo preslikavanje funkcionise tako sto transformise podatke iz jedne reprezentacije u drugu.

Hibernate je jedan od alata kojima se moze izvesti objektno - relaciono preslikavanje u programskom jeziku Java. Razmotrimo neke prednosti koriscenja alata Hibernate:

* produktivnost - koriscenjem alata Hibernate programer se oslobadja vecine zamornog posla niskog nivoa I omogucava mu se da se skoncentrise na poslovnu logiku problema.

* pogodnost odrzavanja - automatsko izvrsavanje objektno - relacionog preslikavanja smanjuje broj linija koda, cineci na taj nacin sistem razumljivim I jednostavnim za refaktorisanje.Alat Hilbernate predstavlja bafer izmedju modela domena I SQL sheme.

* performanse - automatska resenja kao sto je Hibernate omogucavaju koriscenje raznih vidova optimizacije.

* nezavisnost od proizvodjaca - alat Hibernate moze da doprinese smanjenu rizika koji su povezani sa situacijom da korisnik postane zavistan od proizvodjaca nekog proizvoda ili usluge, bez mogucnosti da koristi usluge drugog proizvodjaca , bez znacajnijih troskova.

Hibernate pristup obezbedjivanju trajnosti podataka je dobro primljen od strane Java programera I napravljen je standardizovani Java Persistence API (JPA). JPA specifikacijom se definise:

* mogucnost zaavanja metapodtaka preslikavanja - kako se trajne klase I njihova svojstva odnose na shemu baze podataka.

* API za izvodjenje osnovnih CRUD operacija (Create,Read,Update,Delete)

* jezik I API za zadavanje upita koji se odnose na klase I svojstva klasa.

* kako mehanizam trajnosti interaguje sa mehanizmom transakcija.

Dakle, objektno - relaciono preslikavanje je nastalo kao trenutno najbolje resenje problema objektno - relacionog neslaganja. Na primer, mreza objekata ne moze biti sacuvana u tabeli baze podataka: ona mora biti rastavljena na kolone portabilnih SQL tipova podataka. Zadatak objektno - relacionog preslikavanja je da programera oslobodi od 95% posla obezbedjivanja trajnosti objekata, kao sto su pisanje slozenih SQL upita koji ukljucuju moga spjanja tabela I kopiranje vrednosti iz JDBC rezultujuceg skupa u objekte ili grafove objekata.

HIBERNATE:

KORISCENJE HIBERNATE OKRUZENJA:

Standardni koraci koje izvrsavamo prilikom pravljenja Hibernate aplikacije su:

1. konfigurisanje konekcije na bazu podataka
2. Kreiranje definicija preslikavanja
3. Trajno cuvanje klasa

U nastavku je dat spisak uobicajenih koraka koje treba primeniti u razvoju Hibernate verzije Java aplikacije SkladisteKnjiga:

1. Kreirati objekat sa domenom Knjige,

2. Konfigurisati okruzenje Hibernate,

3. Kreirati klijentsku aplikaciju koja izvrsava razne operacije nad knjigama (dodaje/azurira/brise/trazi knjigu)

Glavni zadatak Hibernate aplikacije jeste njeno konfgurisanje. Postoje dva dela konfiguracije koja se zahtevaju u svakoj Hibernate aplikaciji: jedan deo je zaduzen za konfigurisanje konekcije na bazu podataka,a drugim se zadaju preslikavanja objekata na tabele: njima se definise koja se polja objekata preslikavaju u koje kolone tabele.

KONFIGURISANJE KONEKCIJE NA BAZU PODATAKA:

Da bi se povezali na bazu podtaka, Hibernate mora da zna detalje o bazi podataka, tabelama, klasama. Ove info se mogu proslediti posredstvom XML datoteke ili u vidu jednostavne

tekstualne datoteke koja sadrzi parove oblika ime / vrednost. Ukoliko se koriste podrazumevani nazivi datoteka, okruzenje ce automatski ucitati ove datoteke.

Element <property> sluzi za zadavanje vrednosti pojedinacnih svojstava, kao sto su SUBP koji se koristi, naziv baze podtaka I slicno. Ove info je moguce zadati I putem tekstualne datoteke koja sadrzi parove oblika ime = vrednost.

hibernate.dialect = org.hibernate.dialect.DB2Dialect

hibernate.connection.driver_class = com.ibm.db2.jcc.DB2Driver

hibernate.connection.url = jdbc:db2://localhost:50001/test

Kao sto mozemo da primetimo ako info o bazi podataka zadajemo na ovaj nacin sva navedena svojstva imaju prefiks “hibernate”.

Connection.url ukazije na URL adresu baze podataka na koju se treba povezati

Driver_class na relevantnu klasu drajvera potrebnu za pravljenje konekcije

Dialect na dijalekt baze podataka koji koristimo cime se automatski postavljaju neka svojstva tog dijalekta

Pored konfiguracionih svojstava potrebno je ukljuciti I datoteke koje sadrze preslikavanja I njihove lokacije. Ova preslikavanja se zadaju u posebnim datotekama sa sufiksom .hbm.xml. Njih je u konfiguracionoj datoteci potrebno navesti kao vrednosti atributa resource elementa mapping ,ali vazno je napomenuti da ako se konfiguracija prosledjuje putem datoteke hibernate.properties, tada nije moguce direktno zadati preslikavanja klasa(za sta se ovde koristi element <mapping>), vecje ovaj pristup moguce koristiti kada programski konfigurisemo okruzenje.

SESIJA I TRANSAKCIJA:

Hibernate okruzenje na osnovu prosledjenih info o konfiguraciji pravi keator sesija(fabriku sesija) SessionFactory. To je klasa koja omogucava kreiranje sesija I nju moze na beznedan nacin koristiti vise niti u isto vreme. U idealnom slucaju potrebno je kreirati jednog kreatora sesija I deliti ga medju aplikacijama. Kao sto sam nazib kaze, cilj kreatora sesije je kreiranje objekta sesija. Sesija predstavlja “ prolaz” do baze podtaka. Zadatak sesije jeste da vodi racuna o svim operacijama nad bazom podataka kao sto su cuvanje, ucitavanje I izdvajanje podtaka iz relevantnih tabela. Ona predsavlja I posrednika u rokovodjenju transakcijama u okviru aplikacije. Operacije su upakovane u jedinsvenu celinu pola - jednu transakciju.

Objekat tipa Session nije bezbedno koristiti od strane vise niti u isto vrme (kazemo da nije thread safe) . Stoga objekat sesije ne bi trebalo da bude otvoren neki duzi vrmenski period.

openSession() - otvori novu sesiju

Close() - zatvori sesiju

Kada imamo objekat sesije mozemo izvoditi operacije nad bazom podtaka u okviru transakcije. Sesija I transakcija idu “ ruku pod ruku”.

beginTransaction() - zapocinjemo transakciju nad objektom sesijei vraca se referenca na njega

Transakcija traje sve dok se ona ne potvrdi ili ponisti. Ako se potvrsi objekti se trajno pamte u bazi podataka, ako dodje do greske bice izbacen izuzetak koji treba hvatati I potrebno je ponistiti transakciju

Hibernate aplikacija se moze izvrsavati u kontrolisanom I u nekontrolisanom okruzenju. Na osnovu ovog razlikujemo dva pristupa rada sa transakcijama:

* kontrolisano okruzenje: CMT transakcije kod kojih kontejner moze da kreira I upravlja radom aplikacije u svojim transakcijama I u tom slucaju nema potrebe da razmisljamo o semantici transakcija kao sto je potvrdjivanje ili ponistavanje transakcije jer sve to radi kontejner

* nekontrolisano okruzenje: JDBC transakcije kod kojih mi mozemo da uprvljamo mehanizmom transakcija I potrebno je kodirati njen pocetak transakcije I njen uspesan zavrsetak (commit) ili neuspesan zavrsetak (rollback)

PRESLIKAVANJE KLASA NA TABELE:

Postoje dva nacina na koja je moguce zadati preslikavanje trajnih klasa na tabele baze podataka:

1. Koriscenjem XML datoteka preslikavanja

2. Koriscenjem anotacija

XML DATOTEKE PRESLIKAVANJA:

- 110 str kod nije hteo da se kopira ali ovde je najvaznije spomenuti jedinstveni indentifikator

Svaki objekat mora imati jedinstveni identifikator - nalik primarnom kljucu tabele I on se preslikava u primarni kljuc odgovarajuce tabele. Postavlja se koriscenjem etikete <id> I potrebno je postaviti name - polje objekta I column -kolona tabele koja predstavlja primarni kljuc. U okviru elemneta <id> potrebno je navesti element <generator> kojim se zadaje strategija kojom se generisu ovi indentifikatori. Atribut class elementa < generator> odredjuje strategju generisanja identifikatora. Neke od mogucih vrednosti su:

* increment - vrednost kolone celobrojnog tipa se automatski inkrementira

* identity - vrednost kolone celobrojnog tipa se automatski inkrementira na osnovu vrednosti koju odredi baza podataka

* sequence - vrednost kolone celobrojnog tipa se postavlja tako sto se konsultuje baza podataka za narednu vrendost odgovarajuceg niza

* native - nira se identity ili sequence strategija u zavisnosti od toga sta baza podtaka podrzava

* assigned - ne vrsi se aitomatska dodela vrednosti, vec je potrebno da aplikacija postavi jedinstveni indentifikator, ovo je podrazumevana strategija ukoliko se ne navede generator

ANOTACIJE:

Anotacije predstavljaju metapodatke o preslikavanju objekata na tabele I one se dodaju klsi na nivou izvornog koda. Pisu se ispred elemnata klase na koji se odnose. Svaka anotacija pocinje karakterom @ I ima definisan svoj skup atributa koji je moguce koristiti. Atributi se anotaciji dodaju nakon navodjenja naziva anotacije unutar male zagrade u obliku atributa = vrednost I ako ih ima vise, medjusobno su razdvojene zapetama. Anotacije ne menjaju,niti uticu na to kako radi izvorni kod. Za razliku od XML datoteka, one su prilicno koncizne.

Sve klase koje hocemo da budu trajno zapamcene je potrebno definisati kao entitet navodjenjem anotacije @Entity ispred definicije klase. Anotacijom @Table zadaje se naziv tabele baze podataka u kojoj ce se cuvati ovi entiteti. Ako je ime klase isto kao I naziv odgovarajuce tabele, onda nije neophodno navoditi anotaciju @Table. Ako se ime polja razlikuje od imena odgovarajuce kolone tabele potrebno je anotaciju @Column zadati ime kolone u koju se polje slika. Putem atributa anotacije @Column moguce je koloni postaviti I neka dodatna svojstva. Npr, ako kolona ne moze da sadrzi BULL vrednost, treba postaviti nullable = false. Slicno, ako se kolina treba generisati sa ogranicenjem jedinstvenosti treba postaviti unique = true.

Ukoliko neko polje klase koje nije konstanta niti staticka clanica ne treba trajno pamtiti u tabeli, to je moguce uraditi koriscenjem anotacije @Transient. Naredni korak jeste definisanje jedinstvenog identifikatora instance objekta - to postizemo dodavanjem anotacije @Id ispred polja koja predstavljaju identifikator.

Napomenimo I to da je moguce ne anotirati polja objekta vec odgovarajuce get metode. Ukoliko anotacijom @Id anotirapolje klase koje odgovara priamrnom kljuvu, onda ce Hibernate pristupati svojstvima objekta direkno kroz polja, dok ako anotiramo get metodu anotacijom @Id , onda se omogucava pristup sojstvima objekta kroz get I set metode. Uobicajeno anotiraju polja objekta.

Razlicite strategije se mogu zadati postavljanjem anotacije @GeneratedValue na polje koje predstavlja identifikator I postavljanjem atributa strategy na jednu od narednih vrednosti:

*GenerationType.AUTO - strategija generisanja identifikatora zavisi od konkretnog dijalekta baze podataka koji se koristi (SQUENCE)

*GenerationType.IDENTITY - startegija se oslanja na automatsko inkrementiranje kolone baze podataka, ostavlja se bazi podataka da genrise novu vrednost za svaku novu operaciju umetanja

*GenerationType.SEQUENCE - naredna vrednost identifikatora se izdvaja na osnovu datog niza vrednosti definisanog u okviru baze podataka

*GenerationType.TABLE - ova strategija je slicna prethodnoj I ne koristi se cesto. BLA BLA

OVE SADA DALJE STAVRI UKOLIKO JE POTREBNO OD 116. STR POGLEDATI KODOVE U SKRIPTI JER BAS SMARA ,A OVO SU NEKE BITNE RECENICE I DELOVI KOJE TREBA ZAPAMTITI!!!

TRAJNO CUVANJE OBJEKATA:

Ukoliko kao argument funkcije configure nije naveden naziv konfiguracione datoteke , podrazumeva se da se u putanji nalze datoteke sa podrazumevanim nazivima kao sto su : hibernate.cfg.xml ili hibernate.properties. Ako imena ovih datoteka nisu podrazumevana, onda se moraju navesti kaoa rgumenti odgovarajucih metoda.

SLOZENI IDNTIFIKATORI:

Ne vazi uvek da jedna kolona na jedinstven nacin identifikuje red tabele. Nekada se kombinacija kolona uzima za primarni kljuc tabele I takav kljuc naziva se slozeni primarni kljuc. U situaciji kada tabela ima slozeni primarni kljuc, jedinstveni identifikator objekta moramo zadati na drugaciji nacin.

Klasa mora biti oznacena anotacijom @Embeddable cime se postize da se slozeni primarni kljuc posmatra kao jedinstveno polje, potrebno je da implementira interfejs Serializable I da ima definisan podrazumevani konstruktor. Takodje, neophodno je da ima implementirane metode hashCode I equals koji pomazu da Hibernate proveri da li je doslo do kolizije na vrenostima primarnog kljuca. Ove dve metode su oznacene anotacijom @Override koja nije neophosna, ali je pozeljno navesti je zbog citljivosti koda I zbog jednostavnijeg testiranja(ako napravimo neku gresku u nazivu metoda, tipovima argumenata I slicno, dobicemo gresku u vreme kompilacije).

TRAJNO CUVANJE KOLEKCIJA:

U kontekstu programiranja u programskom jeziku Java, rad sa kolekcijama je neizbezan. Hibernate podrzava Java kolekcije kaos to su: List, Set,Array,Map…

U programskom jeziku Java postoje naredni interfejsi kolekcija:

• java.util.Collection (koji je roditelj svih interfejsa kolekcija osim mapa),

• java.util.List (za rad sa listama),

• java.util.Set (za rad sa skupovima),

• java.util.Map (za rad sa mapama, tj. preslikavanjima oblika kljuc/vrednost),

. . .

Iako mi koristimo ArrayList ili LinkedList, za Hiberate nam je potrebo dizejnirati tako da implementira interfejs List, a ne njegovu konkretnu implementaciju. Tako da se koristi:

List<String> knjige = new ArrayList<String>();

TRAJNO CUVANJE KOLEKCIJA KORISCENJEM XML SATOTEKA PRESLIKAVANJA:

Trajno cuvanje liste. Liste predstavljaju jdenostavnu strukturu podataka za cuvanje objekata u urejdenom poretku. One , takdoje, cuvaju informaciju o poziciji elemenata, posredstvom indeksa. Element mozemo umetnuti na proizvoljno mesto u listi, a mozemo I izdvojiti proizvoljan element iz liste na osnovu njegovog indeksa Npr. Pretpostvaimo da smo trajno sacuvali listu automobila koriscenjem programa u programskom jeziku Java. Ocekujemo da iz tabele izdvojimo podatke o automobilima u onom poretku u kom su bili umetnuti (pri cemu to naravno ne mora da odgovara redosledu u kome su redovi zapamceni u tabeli) . Dakle, prilikom pokretanja upita lista.get(n), ocekujemo da bude vracen n-ti elemnet liste. Da bi ovaj zahtev bio zadovoljen, Hibernate odrzava jos jednu tabelusa indeksima automobila. Prilikom izdvajanja automobila iz glavne tabele, izdvaja se indeksirani poredak elemenata iz dodatne tabele. Nakon toga vrsi se povezivanje odgovarajucih elemenata da bi se oderdio njohv ispravni poredak. Na ovaj nacin se omogucava izdvajanje automobila u zadatom poretku.

Atribut cascade lementa <list> se koristi da bi se postavilo da li je prilikom trajnog cuvanja/brisanja/menjanja tog objekta potrebno da spamte/brisu/izmene I svi njemu pridruzeni objekti. Ukoliko je ona postavljena na cascade = “all” sve pomenute operacije se kaskadno izvrsavaju na svim pridruzenim entitetima. Podrazumevana vrednost ovog atributa je cascade = “none” cime se ignorisu sve asocijacije. Cascade = “save-update” trajno pamcenje I azuriranje, cascade = “delete” brisanje.

Element <list> sadrzi elemente:

* key - njime se zadaje kolona tabele koja predstavlja strani kljuc ka tabeli drugoj

* list-index - njime se zadaje u koju kolonu tabele se preslikava pozicija elemneta u uredjenoj kolekcviji,podrazumevana na 0

* one-to-many - ovde se postavlja tip veze izmedju klasa, moguce je zadati I druge tipove veza elemntima <one-to-one>, <many-to-one>, <many-to-many>

Trajno cuvanje skupova java.util.Set predstavlja neuredjenu strukturu podataka u kojoj nije dozvoljeno pojavljivanje duplikata.

Bla bla …U prikazanom primeru kreirali smo objekat tipa Salon I dodali mu tri automobila. Pokusaj dodavanja istog automobila ne uspeva jer se utvrdjuje da su ova dva automobila identicna I duplikat se ne pamti u kolekciji. Prilikom rada sa skupovima zbog potrebe da se ne cuvaju duplikati, potrebno je u klasi Automobil definisati metode equals I hasCode u klasi Automobil. Kao sto zznamo skup ne moze da sadrzi dva ista elemneta I ova dva metoda pomazu da se ispuni ovaj zahtev.

Trajno cuvanje mapa Kada postoji zahtev za predstavljanjem parova oblika kljuc/vrendost, pogodno je koristiti mape. Struktura podataka Map je kao recnik u kome postoji kljuc (rec) I odgovarajuca vrednost (znacenje).

TRAJNO CUVANJE KOLEKCIJA KORISCENJEM ANOTACIJA:

Videli smo kako je kolekcije moguce trajno sacuvati koriscenejme XML datoteka za zadavanje preslikavanja. Odgovarajuca preslikavanja je moguce zasati I koriscenjem anotacija. Pre negos to kodu koji sadrzi kolekcije dodamo anotacije, potrebno ga je pripremiti na jedan od dva nacina:

* koriscenjem stranog kljuca,

* koriscenjem spojne tabele

Koriscenjem stranog kljuca

Salon sadrzi potencijalno vise automobila, a svaki automobil pripada tacno jednom salonu. Imamo polje sa anotacijom @OneToMany .Kolekcija utomobila treba da ima svoju tabelu sa stranim kljucem koji referise na primarni kljuc tabele Salon (u ovom slucaju to je ID_salona).

Hibernate daje polju automobili anotaciju @JoinColumn kojom se definise strani kljuc. Anotacijom @Cascade(CascadeType.ALL) zadaje se da Hibernate prilikom cuvanja objekta/brisanja/azuriranja trajno sacuva/obrise/azurira I kolekcije povezane sa glavnom instancom. Ukoliko je potrebno ovo ponasanje omoguciti samo za cuvanje objekata, potrebno je postaviti opciju @Cascade(CascadeType.PERSIST), a ako zelimo da samo obezbedimo kaskadno brisanje svih pridruzenih objekata potrebno je postaviti opciju @Cascade(CascadeType.REMOVE).

Koriscenjem spojne tabele

Prilikom koriscenja ove strategije, potrebno je da postoji spojna tabela preslikavanja koja sadrzi primarne kljuceve obe tabele. Sa anotacijom @JoinTable kreiramo spojnutableu, sa atributom joinColumns postavljaju se kolone stranog kljuca spojne tabele koje referisu na primarni kljuc entiteta koji je odgovoran za odrzavanje asocijacije. Sa @OrderColumn postizemo uredjenost.

U slucaju mapa, potrebno je zadati gde se cuva kljuc mape: kljuc moze biti neko od postojecih polja samog entiteta ili moze postojati namenska kolona za cuvanje vrednosti kljuca. Druga mogucnost jeste da se koristi spojna tabela I tada je pored anotacije @JoinTable potrebno navesti I anotaciju @MapKeyColumn kojom se zadaje kljuc te mape.Dakle spojna tabela ce sadrzati primarne kljuceve obe tabele I ovu kolonu kljuca.

ANOTACIJE (VEZE):

Pod asocijacijom (vezom) podrazumevacemo veze izmedju tabela relacione baze podatak. One se najcesce realizuju koriscenjem mehanizma primarnih I stranih kljuceva. Veza karakterise dva svojstva: visestrukost I usmerenost. Termin visestrukost se odnosi na to koliki broj ovjekata je povezan sa koliko ciljnih objekata(odnosno, koliki broj objekata je sa svake strane veze). U prethodnom primeru mogli bismo zakljuciti da svaki automobil ima jedan motor, a svaki motor pripada jednom automobilu. Ovo je tip veze “jedan prema jedna”. Pored ovog tipa veze postoje I tipovi “vise prema vise” I “jedan prema vise”.

Drugo bitno svojstvo veze je njena usmerenost. Asocijacije mogu biti jednosmerne ili dvosmerne. U primeru automobila I motora , ispitivanjem atributa automobila mozemo da ustanovimo koji je motor u pitanju, dok na osnovu mototra ne mozemo dobiti detalje o automobilu. Veza izmedju automobila I motora je jednosmerna.

S druge strane ako se mozemo kretati od polaznog do ciljnog objekta, kao I u suprotnom smeru, za vezu se kaze da je dvosmerna. Za dvosmernu vezu se uobicajeno definise jedna strana koja je odgovorna za odrzavanje te veze (kazemo I da ova strana vlasnik veze) I druga -pridruzena ili inverzna strana veze koja nije odgovorna za odrzavanje te veze.

```
public class Vlasnik {

private int id = 0;

// mogli bismo imati listu automobila koje vlasnik poseduje

// ali jednostavnosti radi neka bude samo jedan automobil

private Automobil automobil = null;

...

}

public class Automobil {

private int id = 0;

private Vlasnik vlasnik = null;

...
```

}

U slucaju klasa Automobil I Vlasnik vazi:

* za svaki automobil mozemo odrediti vlasnika, I

* za svakog vlasnika mozemo dobiti infor o njegovom automobilu.

Ovo je primer dvosmerne veze tipa “jedan prema jedan”.

Veza tipa “vise prema vise” mozemo uociti u odnosu studenta I kursa. Naime, mozemo za datog studenta utvrditi koje kursve on pohadja, a ,takodje, za dati kurs zakljuciti koji studenti pohadjaju taj kurs.

```
public class Student {

private int id = 0;

private List<Kurs> kursevi = null;

…

}

public class Kurs {

private int id = 0;

private List<Student> studenti = null;

...

}
```

Na ovaj nacin se ostvaruje veza tipa vise prema vise.

PRESLIKAVANJE PUTEM XML DATOTEKA PRESLIKAVANJA:

Modelovanje veze tipa “jedan prema jedan”:

Postoje dva nacina za uspostavljanje veze tipa “jedan prema jedan”:

* koriscenjem deljenog primarnog kljuca

* koriscenjem stranog kljuca

Svaka od klasa sadrzi referencu na drugu klasu kroz polja motor I automobil. Ovim je zadata dvosmetna “jedan prema jedan” veza u kojoj je moguce kretati se od jednog entiteta ka rugom I obratno.

Koriscenjem deljenog primarnog kljuca, element <one-to-one> se koristi na obe strane veze da bi njihovu uzajamnu vezu tipa “jedan prema jedan”. Potrebno je navesti atribute name I cascade uz element <one-to-one>. Postoji jos jedan novi atribut , a to je constrained = “true”. Na ovaj nacin se za objekat klase Motor postavlja uslov da automobil mora da postoji, tj. Motor ne moze da postoji bez automobila. Znamo da je primarni kljuc (id) motora isti kao id automobila, za to se koristi klasa generatora foreign.

Koriscenjem ogranicenja stranog kljuca. Za modelovanje odnosa “jedan prema jedan” putem ogranicenja stranog kljuca potrebno je iskoristiti element <many-to-one>, ali se stavkom unique = ‘true’ postavlja da je tacan odnos izmedju njih u stvari “jedan prema jedan”. Modelovanje veze tipa “jedan prema vise”. Odnos izmedju automobila I salona automobila jeste odnos tipa “jedan prema vise”: svaki automobil pripada tacno jednom salonu automobila,dok jedan salon automobila sadrzi vise automobila. S obzirm na to da za neusmerene “jedan prema vise” veze vazi da vrednost stranog kljuca moze biti NULL, a da je kolina deklarisana kao NOT NULL, potrebno je eksplicitno zadati opciju not-null = “true”. Potrebno je dodati referencu na objekat Salon u klasi Automobil, druga izmena bi se odnosila na preslikacanje klase Automobil ; sada bi I ovo preslikavanje sadrzalo element preslikavanja tipa “jedan prema vise”. Element <many-to-one> ukazuje na stranu “vise” veze izmedju automobila I salona - odnosno ukazuje na to da vise automobila moze iamti istu vrednost polja salon.Sada, ne samo da mozemo da dobijemo sve automobile jednog salona automobila, vec je moguce dobiti I slon automobila u kome se automobil izlaze,pozzivom metoda:automobil.getSalon(). Ovo je dvosmerna veza. Modelovanje veze tipa “vise prema vise” se uspostavlja odnos izmedju dve klase na taj nacin svaka od klasa sadrzi potencijalno vise instanci druge klase. U slucaju klasa Student I Kurs, odnos je tipa “vise prema vise” jer student moze da upise vise kurseva, a takodje jedan isti kurs moze da upise vise studenta. Potrebno je da obe strane imaju atribut koji je tipa kolekcija objekata druge klase. Kao sto smo vec vise puta pomenuli kod hilberta je samo jedna strana odgovorna za odrzavanje dvosmerne veze. Potrebno je zadati kljucnu rec inverse = “false” - to znaci da je ova strana odgovorna za vezu, odnosno da ce dodavanje, brisanje ili azuriranje kurseva za nekog studenta trajno sacuvati info o vezama studenat I kurseva u veznoj tabeli, ali ako je inverse = “true” ova strana nije odgovorna za vezu. Drugim recima ovo znaci da necemo mi to raditi jer Hilbert nece trajno sacuvati info. Podrazumevano inverse = “false”.

Preslikavanje putem anotacija:

Jedan prema jedan preslikavanje deklarise se koriscenjem anotacije @OneToOne. Anotacija @JoinColumn u kombinaciji sa anotacijom @OneToOne ukazije na to da data kolona u entitetu koji je vlasnik veze referise na primarni kljuc u pridruzenoj strani veze. S obzirom da je veza dvosmerna I da se moze obici u oba smera, potrebno je postaviti koja strana veze ce biti odgovorna za tu vezu. Atribut mappedBy anotacije @OneToOne (ili analogno @OneToMany ili @MantToMany) se koristi za definisanje polja entitea koje ce biti odgovrno za odrazvanje veze. Ovaj atribut je neophodno navesti za dvosmerne veze I to u okviru

pridruzenog entiteta. On predstavlja analogan atribut inverse u XML preslikavanjima. mappedBy ima sledece znacenje : izmene na ovoj strani veze su vec mapiranje na drugoj strani veze, te ih nema potrebe pratiti u vidu zasebne tabele.

Vise prema vise .Potrebno je dodati anotaciju @JoinTable koja ukazuje na to da ce biti kreirana spoljna tabela. Ova anotacija se takodje koristi za jenosmerne veze “jedan prema vise” veze. Ova anotacija se primenjuje na entitet koji je zaduzen za odrzavanje ove asocijacije. Atributom joinColumns se postavlja kolina stranog kljuca koja referise primarni kljuc tabele entiteta koji je odgovoran za osrzavanje asocijacije. Moguce je atributom inverseJoinColumns zadati kolone stranog kljuca entiteta koje referisu na primarni kljuc entiteta koji je pridruzen asocijaciji

Modelovanje veze tipa “jedan prema vise” isto kao kao “vise prema vise” samo koristimo anotacije @OneToMany I @ManyToOne

BLA BLA SVE OVO IZNAD STRANA 139,140 MOZDA MALO RTD NAPISANO!!!!!!!!!

HIBERNATE QUERY LANGUAGE(HQL):

To je upitini jezik specijalno dizajniran za zadavanje upita nad Java objekitima, nastao sa idejom da poveze koncepte Java klasa I tabela baza podtaka.Nalik SQL I predstavlja deo Hibernate biblioteke. On podrzava sve operacije relacionog modela ali za razliku od SQL-a HQL podrzava koncepte asocijacija, nasledjivanja I polimorfizma I rezultat upita vraca u vidu objekata. S obzirom na to da u HQL-u radimo sa objektima, potrebno je umesto naziva tabele navesti ime klasnog entiteta kojim je predstavljena tabela. Ime klasnog entiteta je podrazumevano jednako imenu klase, a ako zelimo da drugacije imenujemo entitet koji ce se koristiti u HQL upitima , to mozemo postici postavljanjem vrednosti atributa name na zeljeno ime u anotaciji @Entity.

U Hibernate okruzenju postoji Query API za rad sa objektno-relacionim upitima. Klasa Query, koja ima jednostavan iterfejs, predstavlja centralni deo API -a. Ocigledno, prva stvar koja nam je potrebna jeste instanca klase Query. Nju dobijamo pozivom metoda createQuery na tekucem objektu sesije. Metod createQuery prihvata stringovnu reprezentaciju upita koji zelimo da izvrsimo nad bazom podataka I pravi instancu klase Query. Hibernate nakon toga transformise prosledjeni HQL upit u odgovarajuci SQL upit. Za razliku od SQL-a , u HQL -u se kljcuna rec SELECT ne pise (tj. Opciona je) ako izdvajamo celu tabelu, dok se u SQL naredbama javljaju relacione tabele, HQL koristi nazive klasa.

Sa izuzetkom imena Java klasa I njihovih polja, u HQL upitima se ne pravi razlika izmedju malih I velikih slova (nisu “case - sensitive”).

Dakle, kada kreiramo instancu upita sa ugradjenim odgovarajucim HQL upitom, pozivamo metod list kojim se upit izvrsava I njem se izdvajaju svi redovi u rezultatu upita. Hibernate iza scene transformise sve redove tabele u instance klase I izdvaja ij koriscenjem kolekcije

java.util.List. Najpre pravimo naredbu I izvrsavamo upit nad njom, a kao rezultat dobijamo instancu klase ResultSet. Ona se satoji od redova , ali oni nisu u objektnom formatu, vec u sirovoj formi, te je potrebno proci kroz svaku instancu da bismo izdvojili svaki pojedinacni red u rezultatu upita. U Hibernate-u smo sve ove korake apstrahovali u par linija koda ali cak I jedna linija koda:

```
List<Knjiga> knjige = sesija.createQuery("FROM Knjiga").list();
```

Sa setMaxResults() izdvajamo odredjenibroj redova

Sa setFirstResult postavlja se pocetna pozicija rezultujuceg skupa

Da bismo dobili iterator potrebno je pozvati metod iterator nad listom.

Sa stavkom WHERE mozemo precizirati listu instanci koja se izdvaja

HQL podrzava I predikat IS NULL

unigueResult -metod koji radi samo sa jednim redom, ukoliko ne postoji odgovarajuci red vraca null, ukoliko postoji vise redova u rezultatu bice izbacen izuzetak

Imenovani parametri:

Direktno kodiranje ulaznih kriterijuma nije najbolji izbor jer je pozeljno podrzati mogucnost da se ovaj kriterijum moze izmeniti. Stoga je zgodno parametrizovati ovakve stvari I to HQL podrzava. Vrednosti vezujemo za parametre putem njihovog naziva,koriscenjem metoda setParameter koji kao prvi argument prima naziv imenovanog parametra, a kao drugi njegovu vrednost. Nekada je potrebno izdvojiti podatke koji zadovoljavaju kriterijum da neka vrednost pripada datoj listi vrednosti. Za to koristimo IN I na kolekciju ArrayList primenjujemo metod setParameterList. Dakle za parametre mozemo vezati Java kolekcije,a ne samo promenljive osnovnih tipova.

Kljucna rec SELECT:

Ovde nema nsita spec da se kaze sem da se u HQL ne koristi obicno SELECT vec moze odmah FROM .

Spajanje tabela:

U HQL-U je dozvoljeno uvoditi nova imena za tabele - tzv. Alijase. Oni su posebno korisni prilikom spajanja tabela ili koriscenja podupita.

Kljucna rec AS je opciona.

HQL podrzava operatore spjanja I to”

* INNER JOIN,

* LEFT OUTER JOIN

* RIGHT OUTER JOIN

Agregatne funkcije:

HQL podrzava agregatne funkcije poput AVG, MIN, MAX, COUNT(*), COUNT(…), COUNT(DISTINCT …)

Polimorfni upiti:

Sa FROM java.klang.Object o se izdvajaju svi trajni objekti

Azuriranje, brisanje I umetanje:

Podatke je moguce azurirati I obrisati pozivom metoda ececuteUpdate. Ovaj metod kao argument ocekuje string koji sadrzi odgovarajuci upit. Sto se operacije umetanja podtaka tice, nije moguceumetnuti entitet koji ima proizvoljne vrednosti (kao sto je to bilo moguce klauzom VALUES u SQL-u) , vec je jdeino moguce umetnuti entitete formirane na osnovu informacija dobijenih naredbom SELECT

Imenovani upiti:

Pomocu njih se upiti generisu na jednom mestu a koriste kasnije u kodu gde god je potrebno. Na ovaj nacin dobija se cistiji kod, upiti se mogu koristiti veci broj puta sa razlicitih mesta u kodu I njihova ispravnost se proverava prilikom pravljenaj fabrike sesija. Postoje dva naziva na koja mozemo kodirati imenovane upite: mozemo koristiti anotaciju @NamedQuery da vezemo upite za netitete na nivou klasa ili ih mozemo deklarisati u daotetaka presliakvanja.

U pristupu kada koristimo anotacije, entitetu Knjiga potrebno je dodati anotaciju @NamedQuery. Ova anotacija prihvata naziv upita I sam upit kao u narednom primeru:

```
@Entity

@NamedQuery(name = "Izdvoji_knjige",

query = "FROM Knjiga")

public class Knjiga{

...

}
```

Jednom kada smo definisali imenovani upit preko anotacije, njemu se moze pristupiti iz sesije u vreme izvrsavanja pozivom metoda getNamedQuery koji kao argument prima nzziv imenovanog upita.

Za jedan isti entitet moguce je vezati vise razlicitih upita, tako sto definisemo roditeljsku anotaciju @NamedQueries koja kao vrednost atributa value sadrzi niz definicija bolika @NamedQuery.

Koriscenje cistog SQL-a:

Hibernate takodje podrzava mogucnost izvrsavanja cistih SQL upita. Metod createNativeQuery vraca objekat klase NativeQuery, slicno kao sto createQuery vraca Query objekat.Na primer, imenovane SQL upite mozemo deklarisati dodavanjem anotacija @NamedNativeQuery I @NamedNativeQueries na samom entitetu, ili deklarisanjem elementa <sql-query> u datoteci preslikavanja.

ADMINISTRACIJA BAZA PODATAKA:

Baze podataka se kreiraju unutar instance menadzera baze podtaka. Menadzer baze podataka je aplikacija koja je deo SUBP-a I koja obezbedjuje osnovne funkcionalnosti rada sa bazama podataka, kao sto su kreiranje, preimenovanje, brisanje I odrzavanje baze podataka, mogucnost pravljenja rezervnih kopija I povratak podataka.

Instanca je logicko okruzenje menadzera baza podataka koje omogucava upravljanje bazama podataka. Moguce je kreirati vise od jedne instance na istom fizickom serveru (jednu instancu, npr mozemo koristiti kao razvojno okruzenje, a drugu kao okruzenje za produkciju).

Sa db2list listaju se sve instance, sa get instance prikazuju se detalji o instanci, sa set DB2INSTANCE = <naziv_instance> SE MOZE IZMENITI TEKUCA INSTANCA, sa db2start se pokrece, a sa db2stop tekuca instanca se moze zaustaviti.

Prostori tabela su skladisne strukture koje sadrze tabele, indekse I velike objekte. Oni predstavljaju logicki sloj izmedju baze podataka I objekata koje se cuvaju u okviru baza podataka. Dakle, prostori tabela se prave unutar baze podatak, dok se tabele prave unutar prostora tabela. Prostori tabela se cuvaju u grupama particionisanja baze podataka koje predstavljaju skupove od jedne ili vise particijja koje pripadaju bazi podatak. U okviru svake instance moguce je definisati jednu ili vise baza podataka, a svaka baza podataka sadrzi veci broj prostora tabela. Svaka baza podtaka pravi se sa tri podrazumevana sistemska prostora tabela:

* SYSCATSPACE: tabele sistemskog kataloga DB2

* TEMPSPACE1: privremen sistemske tabele

*USERSPACE1: inicijalni prostor tabela za definisanje korisnickih tabela I indeksa

Prostori tabela se sastoje iz jednog ili veceg broja kontejnera koji mogu biti naziv direktorijuma, naziv uredjaja ili naziv datoteke. Kreiranjem prostora tabela unutar baze

podataka se prostoru tabela dodeljuju kontejneri I pamte se njihove definicije I atributi u sistemskom katalogu baze podataka.

Za svaku bazu podataka potrebno je obezbediti skup kontrolnih datoteka baze podataka, kao sto su datoteka konfiguracije baze podataka (koja sadrzi informacije o kodnoj strani , uparivanju I slicno) , datoteka istorije oporavka (koja cubva info o svim izmenama nad bazom podtaka I azurira se nakon svakog kopiranja I oporavka baze podataka, nakon kreiranja/brisanja /menjanja prostora tabela I slicno), skup kontrolnih log datoteka.Takodje za svaku bazu podataka potreban je skup log datoteka baze podataka.

Imenovani skup putanja za skladistenje tabela i drugih objekata baze podataka naziva se grupom skladista. Prostori tabela se mogu dodeliti grupi skladista. Prilikom kreiranja baze podataka, svi prostori tabela se prave u podrazumevanoj grupi koja je IBMSTOGROUP. Sve grupe skladista mogu ce je izlistati naredbom:

SELECT * FROM SYSCAT.STOGROUPS

Prostori tabela se mogu konfigurisati na razlicite nacine, u zavisnosti od toga kako zelimo da ih koristimo. Moze se zadati da operativni sistem upravlja alokacijom prostora tabela, moze se ostaviti menadzeru baze podataka da alocira prostor ili se moze odabrati automatska alokacija prostora tabela za podatke. DB2 podrzava naredna tri tipa upravljanja prostorima:

• SMS (eng. System Managed Space) – menadzer fajl sistema operativnog sistema alocira i upravlja prostorom u kom je sacuvana tabela,

• DMS (eng. Database Managed Space) – menadzer baza podataka upravlja prostorom za skladistenje,

• automatsko skladistenje.

Tip upravljanja skladistem se postavlja prilikom pravljenja prostora tabela naredbom CREATE TABLESPACE

U bazama podataka za koje nije omoguceno automatsko skladistenje, prilikom pravljenja prostora tabela neophodno je navesti stavku MANAGED BY DATABASE ili MANAGED BY SYSTEM. Koriscenje ovih stavki rezultuje u pravljenju DMS prostora tabela, odnosno SMS prostora tabela. Moguce je i eksplicitno navesti stavku MANAGED BY AUTOMATIC STORAGE cime se obezbeduje da se vrsi automatsko upravljanje prostorom tabela. Ako se stavka MANAGED BY ... u potpunosti izostavi onda se podrazumeva automatsko skladistenje, tako da je navodenje stavke MANAGED BY AUTOMATIC STORAGE opciono.

Za svaku bazu podataka pravi se i odrzava skup tabela sistemskog kataloga. Ove tabele sadrze informacije o definicijama objekata baze podataka (kao sto su tabele, pogledi, indeksi itd.), kao i informacije o vrstama pristupa koje korisnici imaju na ovim objektima. One se cuvaju u prostoru tabela SYSCATSPACE. Ove tabele se azuriraju prilikom izvodenja

operacija nad bazom podataka; na primer, kada se kreira tabela. Menadzer baze podataka je zaduzen za pravljenje i odzavanje skupa pogleda kataloga. Razlikujemo dva skupa pogleda. Prvi skup pogleda se nalazi u okviru sheme SYSCAT, ovi pogledi se mogu samo citati i pravo SELECT nad ovim pogledima se podrazumevano daje svima (PUBLIC). Drugi skup pogleda se nalazi u okviru sheme SYSSTAT i on je formiran na osnovu podskupa pogleda koji se nalaze u okviru sheme SYSCAT. Ovaj pogled sadrzi statisticke informacije koje koristi SQL optimizator.

Prilikom kreiranja nove baze podataka, ona se automatski katalogizira u sistemskoj datoteci. Moguce je i eksplicitno katalogizirati bazu podataka pod drugim nazivom - aliasom naredbom CATALOG DATABASE ili ako je prethodno bila obrisana iz sistema naredbom UNCATALOG DATABASE. Okruzenje baza podataka moze sadrzati jednu ili veci broj particija. Za svaku instancu kreira se jedna konfiguraciona datoteka pod nazivom db2nodes.cfg. Ona sadrzi po jedan red za svaku particiju baze podataka.

Naredba CREATE DATABASE:

Naredba CREATE DATABASE inicijalizuje novu bazu podataka, pravi tri inicijalna prostora tabela, kreira sistemske tabele i alocira log datoteku oporavka.

Uparivanje (eng. collation) se odnosi na skup pravila kojima se zadaje nacin sortiranja i poredenja podataka. Podrazumevana vrednost je SYSTEM i ona se postavlja u zavinsnosti od identifikatora zemlje date baze podataka. Prilikom kreiranja baze podataka moguce je definisati i alias te baze u sistemskom direktorijumu baza podataka, navodenjem stavke ALIAS . Ukoliko se ne navede alias baze podataka, koristi se naziv baze podataka.

U okviru baze podataka moguce je praviti pojedinacne objekte baze podataka: sheme, tabele, poglede, alijase, indekse, trigere, itd

Dakle, da rezimirimo, prilikom pravljenja baze podataka izvrsavaju se naredne radnje:

• pravi se baza podataka na navedenoj putanji (ili podrazumevanoj putanji);

• ako je omoguceno automatsko skladistenje, pravi se podrazumevana grupa skladista pod imenom IBMSTOGROUP;

• za sve particije navedene u datoteci db2nodes.cfg prave se particije baze podataka;

• prave se prostori tabela SYSCATSPACE za tabele sistemskog kataloga, TEMPSPACE1 za privremene tabele koje se kreiraju tokom obrada nad bazom podataka, USERSPACE1 za korisnicki definisane tabele i indekse;

• prave se tabele sistemskog kataloga i logovi oporavka;

• baza podataka se katalogizuje u direktorijumu lokalnih baza podataka i direktorijumu sistemskih baza podataka;

• cuvaju se navedene vrednosti za kodni skup, teritoriju i uparivanje;

• prave se sheme SYSCAT, SYSFUN, SYSIBM, SYSSTAT;

• ugraduju se prethodno definisane vezne datoteke menadzera baze podataka u bazu podataka;

• daju se prava: npr. svim korisnicima se daje pravo SELECT na pogledima sistemskog kataloga, kao i prava CREATETAB i CONNECT; informacije o tome ko ima koja prava nad kojim objektom nalaze se u pogledu sistemskog kataloga SYSCAT.DBAUTH.

Premestanej podataka:

Naredbom INSERT moguce je dodati jedan ili vise redova podataka u tabelu ˇ ili pogled. INSERT ne predstavlja najbolji i najbrzi nacin za ucitavanje velike kolicine podataka

Naredbe IMPORT i EXPORT Naredbom IMPORT moguce je izvrsiti umetanje podataka iz ulazne datoteke u tabelu, pri cemu ulazna datoteka sadrzi podatke iz druge baze podataka ili programa. Naredbu EXPORT moguce je koristiti za kopiranje podataka iz tabele u izlaznu datoteku za potrebe koriscenja podataka od strane druge baze podataka ili programa za tabelarnu obradu podataka. Ova datoteka se moze kasnije koristiti za popunjavanje tabela, cime se dobija zgodan metod za migriranje podataka sa jedne baze podataka na drugu. Moguci su razliciti formati ovih datoteka:

• DEL – ASCII format u kome su podaci medusobno razdvojeni specijalnim karakterima za razdvajanje redova i kolona (eng. Delimited ASCII);

• ASC – ASCII format bez razdvajanja kolona (eng. Non-delimited ASCII); podrzane su dve varijante ovog formata: sa fiksnom duzinom podataka i sa fleksibilnom duzinom podataka – u tom slucaju podaci su medusobno razdvojeni karakterom za razdvajanje redova. Ovaj format se ne moze koristiti sa naredbom EXPORT;

• IXF – verzija integrisanog formata za razmenu podatka (eng. Integrated Exchange Format). To je binarni format, koji se moze koristiti za prebacivanje podataka izmedu operativnih sistema. Ovaj format se najcesce koristi za izvoz podataka iz tabele, pri cemu je kasnije moguce te iste podatke uneti u istu ili neku drugu tabelu. Kod ovog formata podataka, tabela ne mora da postoji pre pocetka naredbe IMPORT, dok je kod DEL i ASC formata potrebno da tabela bude definisana (sa listom kolona i njihovim tipovima) pre nego sto se izvrsi naredba IMPORT.

Naredba LOAD:

Pored naredbe IMPORT, podatke je moguce dodati u tabelu i koriscenjem naredbe LOAD. Ove dve naredbe imaju slicnu namenu, ali se u mnogo cemu razlikuju. Naredbom IMPORT se u stvari izvodi operacija INSERT, te su stoga njene mogucnosti analogne pozivanju naredbe INSERT. Za razliku od nje, naredbom LOAD se formatirane stranice direktno upisuju

u bazu podataka, te je na ovaj nacin moguce efikasno prebaciti veliku kolicinu podataka u novokreirane tabele ili u tabele koje vec sadrze podatke. Naredba LOAD se sastoji iz cetiri faze:

1. faze punjenja

2. faze izgradnje

3. faze brisanja

4. faze kopiranja indeksa

Tokom faze punjenja, tabele se pune podacima i prikupljaju se kljucevi indeksa i statistike tabela. Tacke pamcenja se uspostavljaju nakon intervala koji se zadaje kroz opciju SAVECOUNT naredbe LOAD. Generisu se poruke kojima se ukazuje na to koliko puta su ulazni redovi uspesno dodati u trenutku tacke pamcenja.

U drugoj fazi, fazi izgradnje, formiraju se indeksi na osnovu kljuceva indeksa sakupljenih tokom faze punjenja. Kljucevi indeksa se sortiraju tokom faze punjenja.

Tokom faze brisanja, iz tabele se brisu redovi koji narusavaju ogranicenja jedinstvenog kljuca ili primarnog kljuca. Ovi obrisani redovi se cuvaju u tabeli izuzetaka ako je zadata (ona mora biti kreirana pre poziva naredbe LOAD), a poruke o odbacenim redovima se pamte u datoteci poruka. Nakon zavrsetka naredbe LOAD pozeljno je pregledati ovu datoteku, razresiti svaki problem i uneti izmenjene redove u tabelu.

Tokom faze kopiranja indeksa, indeksirani podaci se kopiraju iz sistemskog privremenog prostora tabela u originalni prostor tabela. Ovo ce se desiti samo ako je prilikom poziva naredbe LOAD zadat sistemski privremeni prostor tabela za pravljenje indeksa opcijom ALLOW READ ACCESS.

Postoje cetiti moda pod kojima se naredba LOAD moze izvrsavati: INSERT, REPLACE, RESTART, TERMINATE.

PRAVLJENJE REZERVNIH KOPIJA I OPORAVAK:

Naredba BACKUP DATABASE:

Naredbom BACKUP DATABASE pravi se kopija podataka iz baze podataka i cuva je na nekom drugom medijumu. Ovi podaci se mogu koristiti u slucaju pada medijuma i stete na originalnim podacima. Mogu ce je napraviti rezervnu kopiju cele baze podataka, particije ili samo odredenih prostora tabela. Prilikom pravljenja rezervne kopije baze podataka, nije neophodno biti povezan na bazu podataka ;

Naredba RESTORE Naredbom RESTORE moguce je povratiti bazu podataka cija je rezervna kopija bila napravljena koriscenjem naredbe BACKUP. Najjednostavniji oblik naredbe

RESTORE zahteva samo navodenje alijasa baze podataka koju zelimo da povratimo. Na primer, RESTORE DATABASE bp_knjiga

Naredba RECOVER DATABASE

Naredbom RECOVER DATABASE vrsi se povratak baze podataka na neku rezervnu kopiju a zatim ponovo izvrsavaju transakcije koje su uspesno zavrsene do odredenog vremenskog trenutka ili do kraja loga. Ona koristi informacije iz datoteke istorije da odredi sliku rezervne kopije koju treba iskoristiti u odredenom trenutku, te sam korisnik ne mora da zada odredjenu sliku rezervne kopije. Ako je zahtevana slika rezervne kopije inkrementalna rezervna kopija, naredba RECOVER ce pozvati inkrementalnu automatsku logiku da izvede povratak podataka. Ako se zahteva oporavak u nekom vremenskom trenutku, ali najranija slika rezervne kopije iz datoteke istorije je kasnija od tog vremenskog trenutka, naredba RECOVER DATABASE vratice gresku.RECOVER DATABASE bp_knjiga

AKTIVNOSTI NA ODRZAVANJU BAZE PODTAKA:

Naredba RUNSTATS:

Naredba RUNSTATS se moze iskoristiti za sakupljanje novih statistika tabela i indeksa i azuriranje statistika u sistemskom katalogu. Ako sadrzaj tabele raste ili se dodaju novi indeksi, vazno je azurirati vrednosti ovih statistika (kao sto su broj redova, broj stranica i prosecnu duzinu reda) tako da oslikavaju ove promene. DB2 baza podataka se moze konfigurisati da automatski odredi tabele i indekse za koje je potrebno izracunati nove statistike. Na ovaj nacin moguce je dobiti informacije o:

• broju stranica koje sadrze redove,

• broju stranica koje su u upotrebi,

• broju redova u tabeli,

• vrednostima statistika koje se ticu raspodele podataka,

• detaljne statistike za indekse kojima se odreduje koliko je efikasno pristupiti podacima preko indeksa, . . . Na primer, osnovne statistike o tabeli knjiga mozemo dobiti pozivom naredbe: RUNSTATS ON TABLE knjiga

Naredbe REORG I REORGCHK:

Naredba REORG se moze iskoristiti za reogranizaziju tabela i indeksa da bi se popravila efikasnost skladistenja i smanjili troskovi pristupa. Reorganizacija tabele vrsi defragmentaciju podataka i uklanja prazne prostore; time se koristi manje stranica, cime se stedi prostor na disku i sadrzaj tabele se moze procitati sa manjim brojem ulazno/izlaznih operacija. Moze se takode vrsiti i preuredenje redova u tabeli, npr. podaci se mogu preurediti u skladu sa nekim indeksom, da bi se mogli izdvojiti sa sto manjim brojem

operacija citanja. Veliki broj izmena na podacima u tabeli moze da degradira i performanse indeksa

REORG TABLE knjiga USE TEMPSPACE1

REORG INDEX knjigaInd ON TABLE knjiga

REORG INDEXES ALL FOR TABLE knjiga

Naredba REORGCHK se moze koristiti za dobijanje preporuka koje tabele i koji indeksi bi imali koristi od reorganizacije. Nakon toga se moze iskoristiti naredba REORG za implementiranje kompresije postojecih tabela i indeksa. Na primer, naredbom:

REORGCHK ON SCHEMA AUTO