- Rezimi izvrsavanja: apliaktivni (umetanje SQL-a u druge jezike), interaktivni (preko terminala).

- Aplikativni SQL podrzava 2 tipa naredbi: staticke i dinamicke.

- ODBC standard (Open DataBase Connectivity) – korisnik komunicira sa ODBC medjusistemom koji dalje komunicira sa SUBP, nezavisno od korisnika.

- CLI – direktni pozivi DB2 funkcija – specificni za DB2 sistem, koriste sve njegove mogucnosti

Zapamcena procedura: poziva se iz DB2 CLI aplikacije i izvrsava se na serveru. Jednom napisanu zapamcenu proceduru moze da pozove bilo koja ODBC ili CLI aplikacija.

Aplikativni (ugradjeni) SQL

Dobija se umetanjem SQL-a u neki visi (maticni) programski jezik.

Princip dualnosti: svaka interaktivna SQL naredba se moze izraziti I u aplikativnom SQL-u, obratno ne vazi.

Razmena podataka se vrsi preko maticnih (host) promenljivih koje se mogu javljati I u naredbama maticnog jezika I u SQL naredbama.

SQLCA (SQL Communication Area) – azurira se nakon izvrsavanja svake SQL naredbe.

EXEC SQL INCLUDE SQLCA;

Najcesce koriscena promenljiva strukture SQLCA je SQLCODE, koja predstavlja indikator uspesnosti izvrsenja SQL naredbi. Ima vrednosti: 0 – uspesno izvrsavanje, pozitivnu vrednost – ako se dogodilo nesto izuzetno, negativnu vrednost – naredba prekinuta zbog greske.

EXEC SQL WHENEVER <uslov> <akcija>

Uslov moze biti: NOT FOUND (SQLCODE = 100), SQLERROR (SQLCODE < 0), SQLWARNING (SQLCODE > 0 AND SQLCODE <> 100)

Akcija moze biti: CONTINUE, GOTO obelezje.

WHENEVER omogucuje programeru da proverava vrednost promenljive SQLCODE nakon svake SQL naredbe.

SQLSTATE, tipa char[5]: sadrzi kod rezultata izvrsenja SQL naredbe.

SQLWARNO0-SQLWARN9, tipa char I sadrze informacije o uzroku upozorenja.

Povezivanje na bazu

CONNECT, CONNECT RESET

Pretprocesiranje se vrsi naredbom PREP (ili PRECOMPILE):

DB2 PREP datoteka.sqc BINDFILE,

kao rezultat dobijamo dve datoteke sa istim imenom I ekstenzijama .c I .bnd.

Ugradnja paketa u bazu vrsi se naredbom BIND:

DB2 BIND datoteka.bnd

Nakon ovoga se prevodi dobijena C datoteka I vrsi se linkovanje sa odgovarajucom bibliotekom DB2 funkcija.

Prica o host promenljivama…. Znamo svi ovo sa vezbi.

EXEC SQL BEGIN DELCARE SECTION

….

EXEC SQL END DECLARE SECTION

koriste se pomocu :hostPromenljiva

SELECT naredba koja vraca jedan red kao rezultat → host promenljive

SELECT koja vraca vise redova → preko kursora

U slucaju da je atribut koji ucitavamo u host promenljive nullable, moramo koristiti I indikatorske promenljive koje sluze za proveru vrednosti host promenljive (da li je null). :host :indikator

KURSORI

Kursori su najcesci vid povezivanja upita iz SQL sa maticnim jezikom.

EXEC SQL DECLARE <naziv> CURSOR FOR <upit>;

EXEC SQL OPEN <naziv>;

EXEC SQL FETCH <naziv> INTO <lista_promenljivih>;

Ukoliko smo procitali sve redove iz relacije, vraca se SQLCODE 100.

EXEC SQL CLOSE <naziv>;

Obicno se koriste u kombinaciji sa beskonacnom petljom da bi iterirali kroz rezultat upita.

FOR READ ONLY (FOR FETCH ONLY)

UPDATE I DELETE se vrse pomocu: WHERE CURRENT OF <naziv>

Ako zelimo da menjamo neke podatke putem kursora, navodimo:

FOR UPDATE OF <lista_atributa>

Kursor je implicitno samo za citanje ako njegov upit sadrzi: dve ili vise tabele u FROM klauzi, GROUP BY ili HAVING, kolonsku f-ju u spoljasnjoj SELECT liniji, skupovnu operaciju (osim UNION ALL), DISTINCT, agregatnu f-ju, ORDER BY stavku.

Dinamicki SQL

Naredbe se preciziraju tek u fazi izvrsavanja programa, zadaju se u obliku niske karaktera koja se dodeljuje host promenljivoj. Imaju prednosti (delovi, pa cak I cele naredbe se mogu napisati na

osnovu informacija koje dostavlja korisnik), medjutim moraju se svaki put iznova analizirati I optimizovati od strane DB2 sistema.

Razlozi za koriscenje dinamickog SQL-a: deo ili kompletna naredba se generisu u vreme izvrsavanja, objekti nad kojima je SQL naredba formulisana ne postoje u fazi preprocesiranja, zelimo da izvrsavanje koristi optimalan plan pristupa podacima → zasnovan na trenutnim statistikama podataka.

Priprema naredbe za izvrsavanje se vrsi statickom naredbom PREPARE, nakon cega se moze izvrsiti naredbom EXECUTE.

Tipovi naredbi:

1) naredba koja nije SELECT: potpuna, parametrizovana

2) SELECT naredba: sa fiksnom listom kolona koje se izdvajaju, sa promenljivom listom kolona -||-

3) nepoznata naredba

Naredba PREPARE

Prevodi tekstualnu formu SQL naredbe u izvrsni oblik, dodeljuje pripremljenoj naredbi naziv I eventualno popunjava SQLDA strukturu.

PREPARE <naziv>

FROM <naredba_string>

EXEC SQL PREPARE S1 FROM :V1;

- Kako se upit moze prepisati da se lakse optimizuje, koji indeks je najbolje koristiti, u kom redosledu treba spojiti tabele da bi se minimizovao pristup disku…

Pripremljenu naredbu je moguce koristiti u narednim naredbama: DESCRIBE, EXECUTE (ne SELECT), DECLARE CURSOR (samo SELECT).

Dinamicka SQL naredba ne sme da sadrzi obracanja host promenljivama. Koristimo oznake parametara. Dve vrste: ?, CAST (? AS <tip>).

Neimenovana oznaka bez opisa tipa → ?: koriste se kada se moze nedvosmisleno odrediti njihov tip

Neimenovana oznaka sa opisom tipa → CAST (? AS <tip>): koristi se kada ? ne moze.

Naredba EXECUTE

Dva oblika:

EXECUTE <naziv> [USING <lista_host_prom>]

EXECUTE <naziv> [USING DESCRIPTOR <ime_deskriptora>]

<ime_dekskriptora> je SQLDA struktura koja sadrzi:

1) polje SQLN – broj alociranih SQLVAR struktura

2) polje SQLDABC – ukupan broj bajtova alociranih za SQLDA strukturu

3) polje SQLD – broj promenljivih koje se upotrebljavaju pri obradi naredbe

4) polje SQLVAR – sadrzi opise pojedinacnih promenljivih

Naredba EXECUTE IMMEDIATE

U jednom koraku priprema I izvrsava dinamicku SQL naredbu.

EXECUTE IMMEDIATE <naziv>

<naziv> → niska sa tekstom SQL naredbe

Svaka naredba koja nije SELECT, ako ne sadrzi oznake parametara.

Obicno se primenjuje kada je SQL naredbu potrebno izvrsiti samo jednom.

Naredba DESCRIBE

Upisuje potrebne informacije o naredbi u SQLDA strukturu.

EXEC SQL INCLUDE SQLDA;

DESCRIBE [OUTPUT] <naziv_naredbe> INTO <ime_deskriptora>

Mora postojati dinamicka SQL naredba sa datim nazivom, kao I alocirana SQLDA struktura.

SQLDA se popunjava na sledeci nacin:

1) polje SQLDABC sadrzi velicinu SQLDA strukture u bajtovima

2) polje SQLD sadrzi broj kolona u rezultatu, ako je naredba SELECT, ili broj oznaka parametara

3) polje SQVAR sadrzi podatke o parametrima

Svaka SQLVAR struktura sadrzi:

1) polje SQLTYPE koje cuva oznaku tipa parametara, I da li moze sadrzati NULL

2) polje SQLLEN koje sadrzi velicinu parametara u bajtovima

3) polje SQLNAME sadrzi naziv kolone ili tekstom zapisan broj koji opisuje originalan polozaj izvedene kolone

Oznake tipova:

SQL_TYP_ + naziv odgovarajuceg SQL tipa (sa izuzetkom REAL, LONG VARCHAR, TIMESTAMP), npr. SQL_TYP_SMALLINT, ako moze biti NULL dodajemo N SQL_TYP_NSMALLINT.

Dinamicki kursori

DECLARE <ime> CURSOR FOR <naredba>

OPEN <ime> [USING <lista_host_prom>]

OPEN <ime> [USING DESCRIPTOR <ime_deskriptora>]

FETCH <ime> INTO <llista_host_prom >

FETCH <ime> USING DESCRIPTOR <ime_deskriptora>

Direktni pozivi funkcija SUBP (DB2 CLI)

DB2 CLI je IBM-ov C I C++ aplikacion programski interfejs za pristup rbp. Koristi pozive f-ja ciji su argumenti SQL naredbe, I dinamicki ih izvrsava.

Za razliku od ugradjenog SQL-a, ne zahteva pretprocesiranje niti vezivanje za bazu. To omogucava pisanje prenosivih aplikacija, koje nisu vezane za neki konkretni proizvod.

Slogovi

Mora se ukljuciti zaglavlje sqlci.h. U programu se moze raditi sa 4 vrste slogova (struktura) u C-u:

1) Okruzenja: slog ovog tipa se pravi od strane aplikacije kao priprema za jednu ili vise konekcija sa bazom podataka

2) Konekcije: slog ovog tipa pravi se da bi se aplikacija povezala sa bazom

3) Naredbe: moze se kreirati jedan ili vise slog naredbi, svaki slog cuva informacije o pojedinacnoj SQL naredbi

4) Opisi/Deskriptori: slogovi ovog tipa cuvaju informacije o kolonama relacije koja je rezultat upita ili o dinamickim parametrima SQL naredbe.

Slogovi se prave putem funkcije:

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

- hType → tip zeljenog sloga, SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT, SQL_HANDLE_DESC.

- hIn → slog elementa viseg nivoa u koji se novoalocirani element smesta. Ako pravimo slog konekcije onda je hIn slog okruzenja… Ako pravimo slog okruzenja onda je SQL_NULL_HANDLE.

- hOut → adresa sloga koji se kreira f-jom

Funkcija vraca SQL_SUCCESS ili SQL_ERROR (celobrojne vrednosti).

Nakon sto slogovi nisu vise potrebni, oslobadjaju se f-jom:

SQLRETURN SQLFreeHandle(SQLSMALLINT hType, SQLHANDLE h), koja redom prima tip I naziv sloga.

Povezivanje na bazu podataka

SQLRETURN SQLConnect(SQLHDBC hdbc,

SQLCHAR db_name, SQLSMALLINT db_name_len,

SQLCHAR user_name, SQLSMALLINT user_name_len,

SQLCHAR password, SQLSMALLINT password_len)

- hbdc → slog konekcije, f-ja SQLAllocHandle(SQL_HANDLE_DBC, …, &hdbc) mora biti pozvana pre ove f-je.

- ostali argumenti su vrlo intuitivni.

SQLRETURN SQLDisconnect(SQLHDBC hdbc), vrsi raskid konekcije sa bazom.

Obrada naredbi

Analogno obradi naredbi u dinamickom SQL-u.

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

- sh → slog naredbe

- st → string koji sadrzi naredbu

- si → duzina niske karaktera st.

Ova funkcija povezuje slog naredbe sh sa samom naredbom st.

SQLRETURN SQLExecute(SQLHSTMT sh), izvrsava se naredba sh.

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

SQLExecDirect(naredba, “SELECT god_izdavanja FROM Knjiga”, SQL_NTS);

^ ima smisla samo kada se naredba izvrsava jednom.

Citanje podataka iz rezultata upita

Kursor se otvara kada se dinamicka select naredba izvrsi pozivom SQLExecute ili SQLExecDirect.

SQLRETURN SQLFetch(SQLHSTMT sh)

naredba na koju referise slog sh mora biti izvrsena, ukoliko nije greska. Ukoliko nema vise rezultata vraca SQL_NO_DATA sto se koristi za izlaz iz petlje.

SQLRETURN SQLBindCol(SQLHSTMT sh,

SQLUSMALLINT colNo, SQLSMALLINT colType,

SQLPOINTER pVar, SQLLEN varSize,

SQLLEN *varInfo)

Koristi se da komponente n-torke povezemo sa promenljivama u maticnom jeziku.

- sh je slog naredbe

- colNo je redni broj komponente cije vrednosti citamo

- colType je kod tipa promenljive u koju se smesta vrednost

- pVar je pokazivac na promenljivu u koju se smesta vrednost

- varSize je velicina u bajtovima promenljive na koju pokazuje pVar

- varInfo je pokazivac na celobr. vrednost koja se koristi da se dodje do dodatnih informacija (npr. Da li je vrednost kolone NULL)

Prosledjivanje parametara upitu

Koraci:

1.) SQLPrepare za pripremu naredbe, ? oznacava mesto parametra

2.) SQLBindParameter za vezivanje parametara

3.) SQLExecute za izvrsavanje upita

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

- sh: slog naredbe

- parNo: redni broj parametra – pocinje od 1

- inputOutputType: tip parametra → SQL_PARAM_INPUT za SQL naredbe koje nisu zapamcene procedure, SQL_PARAM_OUTPUT za zapamcene procedure, SQL_PARAM_INPUT_OUTPUT za ulazno-izlazne zapamcene procedure.

- valueType: je C tip parametra, npr. SQL_C_LONG

- parType: je SQL tip parametra, npr. SQL_INTEGER

- colSize: preciznost parametra, npr. Max duzina niske

- decimalDigits: broj decimala, ako je tip SQL_DECIMAL

- parValPtr: pokazivac na bafer za smestanje parametra

- bufferLen: velicina bafera za smestanje parametra

- indPtr: pokazivac na lokaciju koja sadrzi duzinu parametra koji se cuva u parValPtr

Upotreba ODBC standarda

ODBC drajveri koji su specificni za odredjenu bp se dinamicki ucitavaju. Aplikacija se direktno linkuje sa bibliotekom menadzera drajvera, umesto sa svakom bibliotekom pojedinacno.

Menadzer posreduje izmedju ODBC aplikacije I drajvera.

JDBC

```
import java.sql.*
```

Ucitavamo drajver za sistem za upravljanje bazom podataka koji zelimo da koristimo.

Class.forName(<ime_drajvera>);

Class.forName(“com.ibm.db2.jcc.DB2Driver”); od ovog momenta imamo klasu DriverManager.

Connection getConnection(String URL, String username, String password)

void close() za raskid konekcije.

Izuzecima se rukuje pomocu try-catch blokova, SQLException, SQLWarning

Pravljenje naredbi u JDBC

Moguce je napraviti naredbu pomocu 2 metode, koje se obe primenjuju na objekat klase Connection

1.) createStatement(): vraca objekat klase Statement, moze samo za naredbe bez parametara

2.) prepareStatement(Q): vraca objekat klase PreparedStatement, Q je niska koja sadrzi SQL naredbu. Ukoliko naredba sadrzi parametre mora se koristiti ova metoda.

Metode za izvrsavanje naredbi:

1.) executeQuery(Q): niska Q sadrzi SQL upit, primenjuje se na objekat klase Statement. Vraca objekat klase ResultSet, koji je skup n-torki u rezultatu upita Q.

2.) executeQuery(): koja se primenjuje na objekat klase PreparedStatement. Vraca ResultSet.

3.) executeUpdate(U): niska U je SQL naredba koja nije upit, primenjuje se na objekat klase Statement. Vraca broj redova na koje je naredba uticala.

4.) executeUpdate(): primenjuje se na vec spremnu SQL naredbu koja ne sme biti upit.

Operacije nad kursorima

- next(): kursor se pomera na sledecu n-torku, ako je nema vraca FALSE

- getString(i), getInt(i), getFloat(i): vraca i-tu komponentu n-torke, mora se poklapati tip

- close(): zatvara se objekat klase ResultSet

- wasNull(): provera da li je poslednja procitana vrednost NULL, slicno kao indikatorski promenljive u statickom SQL-u.

- first() I last(): pozicionira kursor na prvi/poslednji red u rezultatu, vraca FALSE ako nema redova

U okviru naredbe prepareStatement()/createStatement() mozemo podesiti smer kretanja kursora:

- TYPE_FORWARD_ONLY: samo unapred po skupu redova u rezultatu

- TYPE_SCROLL_SENSITIVE: unapred I unazad, I izmene napravljene od strane drugih transakcija su odmah vidljive.

- TYPE_SCROLL_INSENSITIVE: isto kao proslo, samo promene nisu odmah vidne

Prosledjivanje parametara

setString(i, v), setInt(i, v)… Isti princip kao I ranije

Zapamcene procedure

Programski blok koji se poziva iz aplikacije na klijentu I izvrsava se na serveru baze podataka.

Prednosti: koriste prednosti mocnih servera, donose poboljsanja performansi statickom SQL-u, smanjuju mrezni saobracaj, poboljsavaju integritet podataka…

Definisanje funkcija I procedura

CREATE PROCEDURE <naziv> (<parametri>)
<lokalne deklaracije>
<telo_procedure>;

CREATE FUNCTION <naziv> (<parametri>) RETURNS <tip>
<lokalne deklaracije>
<telo_funkcije>;

Parametri procedure se ne zadaju samo imenom I tipom, vec im prethodi I naziv moda: IN, OUT, INOUT. Podrazumevano IN. Parametri funkcije mogu biti samo ulazni.

CREATE PROCEDURE Zameni (IN stari_izdavac char(30), IN novi_izdavac char(30))
UPDATE Knjiga
SET izdavac = novi_izdavac
WHERE izdavac = stari_izdavac;

Neke jednostavne forme naredbi

1.) Naredba poziva:

CALL <naziv_procedure> (<lista_argumenata>);

Moze se javiti kao deo programa na maticnom jeziku, npr. Poziv procedure iznad:

EXEC SQL CALL Zameni(‘Prosveta’, ‘Nova Prosveta’);

2.) Naredba vracanja vrednosti:

RETURN <izraz>, samo u funkciji

3.) Deklaracija lokalnih promenljivih:

DECLARE <naziv> <tip>;

4.) Naredba dodele:

SET <promenljiva> = <izraz>;

5.) Grupe naredbi:

Mozemo formirati listu naredbi koje se zavrsavaju sa ‘;’ I ogradjene su kljucnim recima BEGIN I END.

6.) Imenovanje naredbi:

Naredbu mozemo imenovati tako sto joj kao prefiks navedemo ime I stavimo ‘;’.

Naredbe grananja

Podrzano je grananje koriscenjem naredbe IF:

IF <uslov> THEN
    <lista_naredbi>
ELSEIF <uslov> THEN
    <lista_naredbi>
ELSEIF
    . . .
[ELSE
    <lista_naredbi>]
END IF;

Upiti

Upiti se u zapamcenim procedurama mogu koristiti na nekoliko nacina:

1.) U uslovima, odnosno na svakom mestu gde se u SQL-u mogu naci podupiti

2.) Upiti koji vracaju jednu vrednost se mogu naci 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.

Pr:

CREATE PROCEDURE Obrada_knjige(IN sifra_knjige INTEGER)

DECLARE naziv_knjige CHAR(50);

BEGIN

    SELECT naziv INTO naziv_knjige
    FROM Knjiga
    WHERE k_sifra = sifra_knjige;

. . .

/* *dalja obrada* */

. . .

END

Petlje

LOOP
<lista_naredbi>
END LOOP;

LEAVE <naziv_petlje>

Najcesce se u petlji putem kursora citaju n-torke I iz petlje se izlazi kada nema vise n-torki.

DECLARE <naziv> CONDITION FOR SQLSTATE <vrednost>;

DECLARE Not_Found CONDITION FOR SQLSTATE ‘02000’; ← za izlaz iz petlje sa kursorima

For petlja

Samo za iteriranje kroz kursor.

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

    <upit>

DO

    <lista_naredbi>

END FOR;

Naredba ne samo da deklarise kursor vec nas oslobadja otvaranja I zatvaranja kursora, citanja podataka I provere da li vise nema torki.

While petlja

WHILE <uslov> DO

    <lista_naredbi>

END WHILE

Repeat-until petlja

REPEAT

    <lista_naredbi>

UNTIL <uslov>

END REPEAT

Slicno kao do-while u C-u. Uvek se izvrsava barem jednom, zaustavlja se kada se ispuni uslov.

Izuzeci

SQL sistem ukazuje na potencijalne greske putem SQLSTATE. U zapamcenoj proceduri moguce je deklarisati deo koda koji nazivamo hvatac izuzetaka. Poziva se kada SQLSTATE dobije kao neku vrednost jedan od kodova izuzetaka. Sastoji se od:

1.) Liste kodova izuzetaka za koje se poziva hvatac

2.) koda koji se izvrsava kada se uhvati neki od izuzetaka

3.) naznake gde treba ici nakon sto je hvatac zavrsio posao

DECLARE <gde_ici_nakon> HANDLER FOR <lista_uslova>
    <naredba>

- <gde_ici_nakon>: CONTINUE, EXIT, UNDO

- <lista_uslova>: deklarisani uslovi ili izrazi sa SQLSTATE I niskama duzine 5

TRANSAKCIJE

Logicka jedinica posla pri radu sa podacima. Prica sa RBP-a….

COMMIT, ROLLBACK

Upravljac transakcijama → komponenta SUBP koja je zaduzena za kontrolu transakcija

Konkurentnost

Bazu koristi veci broj korisnika istovremeno → vise transakcija istovremeno.

Konfliktne radnje → nad istim objektom, a jedna od radnji je upis

Problemi pri konkurentnom radu

1.) Problem izgubljenih azuriranja

2.) Problem zavisnosti od nepotvrdjenih podataka

3.) Problem nekonzistentne analize

Problem izgubljenih azuriranja

Neka se svaka od transakcije A I B sastoji od citanja I upisa istog sloga datoteke, pri cemu transakcije najpre vrse citanje a zatim I upis podataka.

- Transakcija A cita podatke. Transakcija B cita podatke.

- Transakcija A menja podatke I unosi ih.

- Transakcija B menja podatke I unosi ih.

Jasno je da se gubi efekat transakcije A.

Problem zavisnosti od nepotvrdjenih podataka

- Transakcija B azurira neki slog u bazi

- Transakcija A cita/azurira neki slog u bazi

- Transakcija B ponisti rezultat svog rada

Jasno je da se azuiraranje transakcije A gubi za drugi slucaj. U prvom slucaju transakcija A dobija podatke koji se razlikuju od stvarnih podataka u bazi.

Problem nekonzistentne analize

Jedna transakcija cita nekoliko vrednosti, a druga transakcija azurira neke od tih vrednosti tokom izvrsavanja prve. Primer je da prva transakcija racuna neku agregarnu vrednost skupa, dok druga transakcija menja vrednosti skupa. Ovo vodi do netacnog razultata agregacije.

Sva tri problema se resavaju zakljucavanjem. Zakljucavanje obezbedjuje izvrsavanje transakcije koje je ekvivalentno serijskom. Pojam serijalizabilnosti – skup transakcija proizvodi isti rezultat kao njihovo serijsko izvrsavanje.

Zakljucavanje

Podrazumeva postavljanja katanca na objekat. Kada transakcija zeli da radi sa nekim objektom ona zahteva njegovo zakljucavanje, a kada zavrsi sa radom oslobadja ga.

Svojstva katanca:

1.) Vrsta/Mod/Rezim: odredjuje nacine pristupa dozvoljene vlasniku, kao I konkurentnim korisnicima

2.) Objekat: resurs koji se zakljucava, objekat takodje odredjuje granularnost – opseg katanca (prostor tabela, tabela, red)

3.) Trajanje: vremenski interval u kome se katanac drzi

Pretpostavimo da sistem podrzava 2 vrste katanaca:

- deljivi – S

- ekskluzivni – X

S katanac je potreban za citanje, a X za azuriranje. Ako je postavljen katanac X svaki zahtev za druge katance biva odbiven, ako je postavila S moze se odobriti S ali se X odbija.

- Dvofazni protokol zakljucavanja:

1.) Faza zakljucavanja: u ovoj fazi se zakljucava objekat I nema nijedna operacije otkljucavanja

2.) Faza otkljucavanja: u ovoj fazi se objekat oslobadja I nema nijedne operacije zakljucavanja

- Striktini dvofazni protokol zakljucavanja: pridrzava se dvofaznog protokola I dodatno se X katanci oslobadjaju tek nakon kraja transakcije. Koraci:

1.) Transakcija koja zeli da cita vrstu, mora da postavi S katanac

2.) Transakcija koja zeli da azurira mora da postavi X katanac. Ako vec drzi S katanac mora da trazi unapredjenje u X katanac

3.) Ako je zahtev za postavljanjem katanca odbijen jer je u konfliktu sa nekim vec postavljenim katancem, transakcija ide u stanje cekanja koje traje sve dok se konfliktni katanac ne oslobodi. Sistem mora da garantuje da cekanje nece trajati zauvek.

4.) X katanac se zadrzava do kraja transakcije, S katanac se obicno zadrzava do kraja transakcije

Ako sve transakcije primenjuju dvofazni protokol, tada su svi moguci rasporedi izvrsavanja transakcija serijalizabilni.

Resenje problema izgubljena azuriranja

U trenutku t1 transakcija A postavlja S katanac nad R, a u trenutku t2 transakcija B postavlja S nad R. U trenutku t3 A ne moze da postavi X katanac nad R jer B drzi S. A ide u stanje cekanja. Analogno za B. Ovako dolazimo do mrtve petlje – deadlock.

Resenje problema zavisnosti od nepotvrdjenih podataka

U trenutku t2 transakcija A je sprecena da dobije S/X katanac, jer transakcija B drzi X katanac.

Resenje problema nekonzistentne analize

. . .

Mrtva petlja

Situacija kada dve ili vise transakcija imaju postavljen katanac nad resursom koji je potreban drugim transakcijama, tako da ga nijedna ne dobija. Otkriva se grafom cekanja. Usmereni graf koji ima po jedan cvor za svaku transakciju, a grana (Ti, Tj) oznacava da transakcija j drzi katanac nad resursom koji je potreban transakciji i. U ovakvom grafu mrtva petlja je usmereni ciklus. Nakon sto se pronadje mrtva petlja bira se “zrtva” transakcija koja se ponistava I tako oslobadja sveoje resurse.

Zakljucavanje sa namerom

1.) IS (intent share) katanac: transakcija koja drzi ovu vrstu katanca namerava da postavi S katanac nad pojedinacnim torkama tabele, za svaki red koji se namerava citati potrebno je dobiti S katanac.

2.) IX (intent exclusive) katanac: transakcija koja drzi ovaj katanac moze da cita I menja pojedinacne torke tabele, ali mora prvo da dobije odgovarajucu vrstu katanca nad njima.

3.) S katanac: transakcija tolerise konkurentno citanje ali ne I konkurentno pisanje. Ne namerava da vrsi azuriranje.

4.) SIX (share with intent exclusive) katanac: kombinacija S i IX katanaca – transakcija moze da tolerise konkurentno citanje, ali namerava da vrsi azuriranja redova tabele nad kojima ce postaviti X katanac.

5.) X katanac: transakcija ne tolerise nikakav knkurentan pristup tabeli.

U katanac

Update – U katanac predstavlja hibrid deljenog I ekskluzivnog katanca. On se zahteva kada transakcija zeli da cita ali ce naknadno hteti da menja dati red tabele. Koriscenje U katanca onemogucava da dva poziva transakcije naprave deadlock na redu.

Zakljucavanje u sistemu DB2

DB2 uvek zahteva katanac na tabeli pre nego sto se dozvoli pristup podacima. Moze se zakljucati samo tabela ili I tabela I redovi ili stranice tabele.

ALTER [TABLESPACE|TABLE] naziv LOCKSIZE TABLE, postavlja da se zahteva katanac pri svakom pristupu tabeli.

Neke vrste katanaca:

1.) IN (intent none): vlasnik moze da cita sve podatke u tabeli, ukljucujuci I nepotvrdjene, ali ne moze da ih menja. Druge konkurentne aplikacije mogu da citaju I azuriraju tabelu, ne postavljaju se nikakvi katanci na redovima.

2.) IS (intent share): vlasnik moze citati proizvolajn podatak u tabeli, ako se S katanac moze dobiti nad zeljenim redovima ili stranicama.

3.) IX (intent exclusive): vlasnik moze da cita I menja podatke ako se X katanac moze dobiti nad podacima koje zelimo da menjamo, ili S ili U katanac mzoe dobiti nad podacima koje zelimo da citamo.

4.) SIX (share with intention exclusive): vlasnik moze da cita sve podatke iz tabele I da menja redove ako se moze dobiti X katanac. Za citanje se ne dobijaju katanci nad redovima, konkurentne aplikacije mogu da citaju iz tabele. Dobija se ako aolikacija ima IX/S katanac I trazi S/IX katanac.

5.) S (share): vlasnik moze da cita podatke, nece dobiti katanac nad redovima

6.) U (update): vlasnik moze da cita prooizvoljan podatak, moze da menja podatke ako se moze dobiti X katanac. Ne dobijaju se katanci nad redovima I stranicama.

7.) X (exclusive): vlasnik moze da cita I menja podatke. Ne dobijaju se katanci nad redovima I stranicama.

8.) Z (super exclusive): zahteva se na tabeli u posebnim prilikama – npr. Menjaje strukture tabele. Nijedna druga aplikacija ne moze da cita niti azurira podatke.

Striktni katanac nad tabelom se moze zahtevati:

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

SHARE- S, EXCLUSIVE – X.

Ekskalacija katanaca: objedinjuje proces dobijanja katanca nad tabelom, I oslobadjanja katanca nad redovima. Zeljeni efekat je da se smanje ukupni zahtevi skladistenja za katance.

Parametri konfiguracije BP koji uticu na proces ekskalacije:

1.) locklist: prostor u globalnoj memoriji koji se koristi za skladistenje katanaca

2.) maxlocks: procenat ukupne liste katanaca koje sme da drzi jedna aplikacija

Oba parametra su konfigurabilna. Ekskalacija se desava u dva slucaja:

1.) aplikacija zahteva katanac koji ce prekoraciti parametar maxlocks, u ovoj situaciji menadzer baze podataka pokusava da dobije katanac nad tabelom I oslobodi katance nad redovima te tabele.

2.) aplikacija ne moze da dobije katanac jer je lista katanaca puna. Analogno resenje kao u proslom slucaju.

Ekskalacija moze da ne uspe – SQLCODE -912.

Detekcija prekoracenja vremena cekanja – locktimeout, sprecava se da aplikacija beskonacno ceka na katanac.

Ako se pronadje deadlock, odredjuje se zrtva I na nju se zatim primenjuje ROLLBACK…

Parametar dlchktime definise frekvenciju sa kojom se proverava da li je doslo do deadlocka.

Vrednosti ovih parametara mogu se procitati:

GET DATABASE CONFIGURATION FOR <naziv_baze>

UPDATE DATABASE CONFIGURATION FOR <naziv_baze>
<parametar> <vrednost>
Npr: UPDATE DATABASE CONFIGURATION FOR BPKnjiga LOCKTIMEOUT 30

Svojstva transakcija I nivoi izolovanosti transakcija

Izvrsavanje transakcije moze da se zavrsi: planirano (COMMIT ili eksplicitni ROLLBACK) I neplanirano (ako se dodje do greske – ROLLBACK).

COMMIT – sve se potvrdjuje, upis u log, free katanaca.

Pad transakcije – neplaniran kraj transakcije

ACID svojstva – znamo sa RBP (Atomicnost, Konzistentnost, Izolovanost, Trajnost)

Nivoi izolovanosti

Zbog povecanja konkurentnosti, svojstvo Izolovanosti se realizuje u nekoliko nivoa. Ovako se opisuje koji stepen ometanja transakcija moze da podnese. Ove nivoe nazivamo nivoima izolovanosti transakcije. Ako su transakcije serijalizabilne stepen ometanja ne postoji, tj. nivo izolovanosti je najveci. U realnim sistemima se dopusta nesto nizi nivo izolacije. SQL definise:

1.) SERIALIZABLE – ne mogu se desavati problemi

2.) REPEATABLE READ – moze fantomsko citanje

3.) READ COMMITED – moze neponovljivo I fantomsko citanje

4.) READ UNCOMMITED – mogu sva 3

SERIALIZABLE je najvisi nivo izolovanosti I garantuje serijalizabilnost resenja. Standard definise tri nacina da se serijalizabilnost narusi:

1.) Prljavo citanje: transakcija A azurira slog, B cita slog, A ponistava promene. B cita nepostojeci slog.

2.) Neponovljivo citanje: A cita sloga, B azurira taj slog, A pokusava da procita isti slog. A ponovnim citanjem istog sloga dobija drugaciji rezultat.

3.) Fantomsko citanje: A cita slogove koji zadovoljavaju neki uslov, B unosi novi slog koji zadovoljava uslov. A ponovo cita videce slog koji nije ranije postojao – fantomski slog.

Nivoi izolovanosti u DB2

- REPEATABLE READ – RR

- READ STABILITY – RS

- CURSOR STABILITY – CS
    1. CURRENTLY COMMITED - CC

- UNCOMMITED READ – UR

Odgovaraju nivoima izolovanosti u SQL standardu, ali drugacije ime.

RR – najvisi nivo, zakljucavaju se sve vrste kojima se transakcija obraca, sve do COMMIT/ROLLBACK, transakcija nema uvid u nepotvrdjene promene drugih transakcija

RS – zakljucava samo vrste koje zadovoljavaju zadate uslove upita

CS – zakljucava samo one vrste koje transakcija trenutno cita, podrazumevan u DB2

CC – citaju se samo potvrdjeni podaci

UR – transakcija cita podatke bez zahteva za S katanac, dobar za prisptupanje read-only tabelama.

Utvrdjivanje nivoa izolovanosti za SQL naredbe: staticke I dinamicke… (SET CURRENT ISOLATION za dinamicke)

UPDATE Knjiga SET izdavac = ‘Laguna’ WHERE id_knjige = 20153 WITH RR

BIND <paket> ISOLATION (RR|RS|CS|UR)

SET CURRENT ISOLATION = {RR|RS|CS|UR} – za dinamicke naredbe unutar neke sesije

U DB2 CLI pristupu koristimo SQLSetConnectAttr – potrebno je atribut SQL_ATTR_TXN_ISOLATION postaviti:

- SQL_TXN_SERIALIZABLE

- SQL_TXN_REPEATABLE_READ

- SQL_TXN_READ_COMMITED – podrazumevan

- SQL_TXN_READ_UNCOMMITED

SQLSetConnectAttr(konekcija, SQL_ATTR_TXN_ISOLATION, (void*)SQL_TXN_READ_COMMITED, 0)

u JDBC – setTransactionIsolation(isolation)

Kursori deklarisani sa opcijom WITH HOLD

DECLARE <naziv> CURSOR WITH HOLD FOR <upit>, kursor ostaje otvoren sve dok se eksplicitno ne zatvori ili ROLLBACK.

JDBC – u createStatement() I prepareStatement() mozemo navesti HOLD_CURSORS_OVER_COMMIT.

U DB2 CLI: atribut SQL_ATTR_CURSOR_HOLD postavimo na SQL_CURSOR_HOLD_ON

SQLSetStmtAttr(naredba, SQL_ATTR_CURSOR_HOLD, (void*) SQL_CURSOR_HOLD_OFF, 0)

Oporavak

Aktivnost koja se izvrsava u slucaju da transakcije ne mogu uspesno da se zavrse:

Uzroci:

1.) U samoj transakciji - pad transakcije, prekoracenje neke dozvoljene vrednosti

2.) U sistemu – prestanak elektricnog napajanja – pad sistema

3.) U disku u kome je BP – pad medijuma

Ponistavanje odnosno ponovno izvrsavanje radnji transakcija u cilju uspostavljanja konzistentnog stanja baze predstavlja sistemsku aktivnost oporavka od pada transakcije ili sistema.

U slucaju pada medijuma, oporavak uljucuje prepisivanje podataka na ispravan medijum kao I ponovno izvrsavanje transakcija kompletiranih pre pada medijuma.

Log datoteka – sistemski log: mesto u sistemu u kome se cuvaju informacije o izvrsenim radnjama.

GET DATABASE CONFIGURATION nam daje informacije o logu (mesto, max velicina, …)

Upravljac oporavka – komponenta SUBP zaduzena za oporavak

Niz radnji:

1.) periodicno prepisuje celu bazu na medijum za arhiviranje

2.) pri svakoj promeni baze, upisuje slog promene u log datoteku, sa tipom promene I sa:
- novom I starom vrednoscu pri azuriranju,
- novom vrednoscu pri unosenju u bazu
- starom vrednoscu pri brisanju iz baze

3.) za svaku naredbu DDL-a, upisuje odgovarajuci slog u log

4.) u slucaju pada transakcije/sistema stanje baze moze biti nekonzistentno. Koristi informacije iz log-a da ponisti parcijalno izvrsene transakcije, odnosno ponovo izvrsi kompletirane

5.) u slucaju pada medijuma, najsvezija arhivirana kopija baze se kopira na ispravan medijum, a zatim se koristi log da bi se izvrsile transakcije kompletirane posle poslednjeg arhiviranja I pre pada medijuma.

DO, UNDO, REDO logika za ponavljanje/ponistavanje transakcija.

Pad sistema/medijuma se moze desiti u fazi oporavka od prethodnog pada – UNDO I REDO moraju imati idempotentnost: UNDO(UNDO(x)) = UNDO(x), REDO(REDO(x)) = REDO(x)

Oporavak od pada transakcije

BEGIN TRANSACTION se upisuje u log, ROLLBACK ponistava sve promene na bazi citanjem unazad svih slogova transakcije, do pocetnog sloga. Za svaki slog promena se ponistava pomocu UNDO.

Oporavak od pada sistema

Pomocu log datoteke se ponsitavaju transakcije aktivne u vreme pada sistema.

Procedura oporavka od pada sistema kod ranijih SUBP

Posle odredjenog broja upisanih slogova u log uvode se takozvane tacke preseka stanja. U momentu koji odredjuje tacku preseka stanja fizicki se upisuju podaci I informacije iz log bafera I bafera podataka u log datoteku I u bazu, redom, I u log datoteku se upisuje slog tacke preseka stanja.

Ovaj slog sadrzi info o svim aktivnim transakcija u momentu tacke preseka stanja, adrese poslednjih slogova tih transakcija u log datoteci, kao I niz drugih informacija. Adresa sloga tacke preseka stanja se upisuje u datoteku ponovnog startovanja.

Protokol upisivanja u log unapred (WAL): pri izvrsenju COMMIT-a, najpre se slog fizicki upisuje u log datoteku, pa se zatim podaci iz bafera podataka prepisuju u bazu podataka. Ako dodje do pada

sistema posle upisa COMMIT sloga u log datoteku, a pre nego sto se sadrzaj bafera podataka prepise u bazu, taj sadrzaj se moze restaurirati REDO logikom.

Poboljsanja procedure oporavka od pada sistema

Prethodno opisani postupak ponistava efekte neuspelih transakcija koji mozda nisu ni bili ubelezeni u bazu, odnosno ponovo izvrsava radnje uspelih transakcija od tacke preseka stanja, iako je moguce da su efekti tih radnji upisani u bazu pre pada sistema. Moguce je eliminisati fizicko upisivanje bafera podataka u bazu 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 ponoviti samo one radnje uspelih transakcija ciji efekti nisu upisani u bazu.

Uvodi se jedinstvena oznaka u svakom slogu log datoteke – LSN (Log Sequence Number).

Tbc

SQL podrska za tacke pamcenja

SAVEPOINT <naziv>

Dodatne opcije:

- UNIQUE: isti naziv tacke pamcenja se nece javiti ponovo sve dok je ona aktivna

- ON ROLLBACK RETAIN CURSORS: kad god je moguce operacija ROLLBACK TO SAVEPOINT nece uticati na otvorene kursore

- ON ROLLBACK RETAIN LOCKS: katanci koji su dobijeni nakon tacke pamcenja se ne oslobadjaju ROLLBACK TO SAVEPOINT-om

RELEASE SAVEPOINT <naziv>

povratak na tacku pamcenja: ROLLBACK TO SAVEPOINT [<naziv>];

Objektno-relaciono preslikavanje I alat Hibernate

Alati za objektno-relaciono preslikavanje: vrse automatsko preslikavanje relacionog modela baze podataka na objektni model aplikacije I obrnuto.

Objektno-relaciono preslikavanje

Trajnost/perzistentnost podataka jedan je od osnovnih koncepata koji se podrazumevaju prilikom razvoja aplikacije.Zbog veoma dobro proucenih teorijskih osnova relacionog modela podataka, rbp garantuju I cuvaju integritet podataka.

Nezavisnost podataka: podaci zive duze od svake aplikacije, a relaciona tehnologija omogucava deljenje podataka izmedju aplikacija ili medju delovima jednog sistema.

Koriscenje SQL-a u programskom jeziku Java

JDBC API. Model domena: definise objekte koji predstavljaju probleme iz realnog sveta.

Objektno-relaciono neslaganje

Osnovna razlika je sto model podataka prikazuje podatke, a objektni model ih skriva. Kombinacija objektne I relacione tehnologije moze da dovede do konceptualnih I tehnickih problema – objektno-relacionih neslaganja. Medju najvaznije probleme spadaju:

1.) Neslaganje izmedju nivoa granularnosti u objektno orijentisanom modelu I odgovarajucem relacionom modelu

2.) Neslaganje hijerarhijskog uredjenja podataka: klase se mogu organizovati hijerarhijski, a tebele ne mogu

3.) Neslaganje pojma istovetnosti: npr. Dva reda u tabeli sa istim podacima smatraju se istim, dok se dva objekta sa istim vrednostima mogu razlikovati zbog memorijskih adresa

4.) Neslaganje u nacinima na koje se realizuju veze

5.) Neslaganje u nacinu na koji se podacima pristupa u Javi I relacionim bazama podataka

Objektno relaciono preslikavanje I alat Hibernate

Objektno relaciono preslikavanje (ORP) je automatsko trajno cuvanje objekata iz Java aplikacije u SQL baze podataka, koriscenjem metapodataka koji opisuju preslikavanja izmedju klasa aplikacije I sheme SQL baze podataka. Funkcionise tako sto transformise podatke iz jedne reprezentacije u drugu.

Prednosti alata Hibernate:

1.) Produktivnost: programer se oslobadja vecine zamornog posla niskog nivoa

2.) Pogodnost odrzavanja: automatsko izvrsavanje ORP-a smanjuje broj linija koda, cineci na taj nacin sistem razumljivijim I laksim za refaktorisanje

3.) Performanse: Hibernate pruza razne vidove optimizacije, programeri mogu da se posvete optimizovanju nekoliko uskih grla koja preteknu

4.) Nezavisnost od proizvodjaca: Nezavisnost od SUBP-a, pomaze kada se tokom razvoja koristi lokalna baza a testiranje I produkcija se rade na drugacijoj bazi

Java Persistence API – JPA. JPA specifikacijom se definise:

- Mogucnost zadavanja metapodataka preslikavanja: kako se trajne klase I njihova svojstva odnose na shemu baze podataka.

- API za izvodjenje osnovnih CRUD (Create, Read, Update, Delete) operacija na instancama trajnih klasa

- Jezik I API za zadavanje upita koji se odnose na klase I svojstva klasa

- Kako mehanizam trajnosti interaguje sa mehanizmom transakcija

Hibernate

Svaki red tabele – jedna instanca klase. Primer znamo sa vezbi… (Unos nove instance sa Session objektima I session.save()).

Koriscenje Hibernate okruzenja

Standardni koraci prilikom pravljenja Hibernate aplikacije:

1.) Konfigurisanje konekcije na bazu podataka

2.) Kreiranje definicija preslikavanja

3.) Trajno cuvanje klasa

Konfigurisanje konekcije na bazu podataka

Da bi se povezali na bazu, Hibernate mora da zna detalje o bazi podataka, tabelama, klasama…

Ove informacije se mogu proslediti posredstvom XML datoteke ili nekog drugog tekstualnog fajla.

Sesija I transakcija

Hibernate na osnovu prosledjenih informacija o konfiguraciji pravi kreatora sesija – SessionFactory.

Klasa koja omogucava kreiranje sesija, moze je na bezbedan nacin koristiti vise niti u isto vreme. Za svaku bazu podataka novi kreator sesija. Zadatak sesije je da vodi racuna o svim operacijama nad bazom podataka. Objekat tipa Session nije bezbedno koristiti od strane vise niti u isto vreme.

Fabrika.openSession();, zatvaramo sa session.close();

Transakcije – Transaction klasa, beginTransaction();

Dva pristupa u radu sa transakcijama:

1.) Kontrolisano okruzenje – CMT transakcije kod kojih kontejner moze da kreira I upravlja radom aplikacije u svojim transakcijama

2.) Nekontrolisano okruzenje – JDBC transakcije kod kojih mozemo da upravljamo mehanizmom transakcija

U drugom pristupu moramo mi zvati commit I rollback.

Preslikavanje klasa na tabele

1.) Koriscenjem XML datoteka preslikavanja

2.) Koriscenjem anotacija

XML datoteke preslikavanja

…

Anotacije

Anotacije predstavljaju metapodatke o preslikavanju objekata na tabele I one se dodaju klasi na nivou izvornog koda. @Entity, @Table(name=””, schema=””), @Id, @Column(name=””, nullable=bool, unique=bool)

@Transient – ukoliko neko polje klase koje nije konstanta niti staticka clanica nije potrebno trajno pamtiti u tabeli

@GeneratedValue – razliciti nacini generisanja identifikatora

@GeneratedValue(strategy=)

1.) GenerationType.AUTO: strategija generisanja zavisi od konkretnog dijalekta baze podataka, najcesce GenerationType.SEQUENCE

2.) GenerationType.IDENTITY: strategija se oslanja na automatsko inkrementiranje kolone baze podataka, ostavlja se bazi da generise novu vrednost za svaku operaciju umetanja

3.) GenerationType.SEQUENCE: naredna vrednost identifikatora se izdvaja na osnovu datog niza vrednosti definisanog u okviru baze podataka

4.) GenerationType.TABLE: slicno kao SEQUENCE, simulira niz cuvanjem I azuriranjem njegove tekuce vrednosti u tabeli baze podataka

Ovako generisanu klasu ukljucujemo sa addAnnotatedClass();

Slozeni identifikatori

Slozeni primarni kljuc – kombinacija kolona je pk

Pravimo klasu koju prosledjujemo kao primarni kljuc. Klasa mora biti oznacena kao @Embeddable, implementirati interfejs Serializable I da ima definisan podrazumevani konstruktor. Takodje je pozeljno preopteretiti metode equals() I hashCode().

Trajno cuvanje kolekcija

List, Set, Array, Map…

Interfejsi: java.util.Collection, java.util.List, java.util.Set, java.util.Array, java.util.Map…

Trajno cuvanje pomocu XML datoteka

. . .

Trajno cuvanje kolekcija koriscenjem anotacija

Pre nego sto kodu koji sadrzi kolekcije dodelimo anotacije, treba ga pripremiti na jedan od dva nacina:

1.) Koriscenjem stranog kljuca

2.) Koriscenjem spojne tabele: postoji spojna tabela preslikavanja koja sadrzi primarne kljuceve obe tabele.

. . .

Asocijacije (veze)

Pod asocijacijom(vezom) podrazumevacemo veze izmedju tabela relacione baze podataka. One se najcesce realizuju koriscenjem mehanizama primarnih I stranih kljuceva. Dva svojstva:

1.) Visestrukost: koliki broj objekata je povezan sa koliko ciljnih objekata – jedan prema jedan, jedan prema vise, vise prema vise

2.) Usmerenost: mogu biti jednosmerne I dvosmerne

Preslikavanje putem XML datoteka

Modelovanje veze tipa “jedan prema jedan”: Dva nacina:

1.) koriscenjem primarnog kljuca

2.) koriscenjem stranog kljuca

<one-to-one>

HQL

Upitni jezik specijalno dizajniran za zadavanje upita nad Java objektima.

Za razliku od SQL-a podrzava I koncepte asocijacija, nasledjivanja I polimorfizma, I rezultat upita vraca u vidu objekata.

Query API za rad sa objektno-relacionim upitima. Klasa Query – centralni deo API-ja.

Session session = fabrika.openSession();

Query upit = session.createQuery(“FROM Knjiga”)

SQL vs HQL:

- U HQL-u kljucna rec SELECT se ne pise ako izdvajamo celu tabelu.

- U SQL-u se javljaju relacione tabele, HQL koristi nazive klasa.

Query<Knjiga> upit = session.createQuery(“FROM Knjiga”, Knjiga.class);

List<Knjiga> knjige = upit.list(); ← dobijamo rezultate upita

setMaxResults(), setFirstResult() . . .

uniqueResult() - ukoliko znamo da postoji samo jedan red u rezultatu upita.

Imenovani parametri

FROM Knjiga WHERE naziv = :nazivId;

upit.setParameter(“nazivId”, knjiga);

setParameter() - 1. argument naziv imenovanog parametra, 2. njegovu vrednost, opcioni 3. je tip parametra.

Za parametre mozemo vezati I Java kolekcije, ne samo promenljive.

List izdavaci = new ArrayList();

izdavaci.add(“Prosveta”); izdavaci.add(“Sluzbeni glasnik”);

Query<Knjiga> upit = session.createQuery(“FROM Knjiga WHERE izdavac IN (:lista_izdavaca)”, Knjiga.class);

upit.setParameterList(“lista_izdavaca”, izdavaci);

List<Knjiga> knjige = upit.list();

Kljucna rec Select

Ukoliko ne zelimo da da izdvojimo kompletne redove, vec samo pojedine kolone.

Query<Knjiga> upit = session.createQuery(“SELECT k.naziv FROM Knjiga AS k”, Knjiga.class)

List<String> nazivi = upit.list();

Kada se vraca vise kolona:

String upit = “SELECT k.naziv, k.izdavac FROM Knjiga AS k”

Query<Object[]> upit = session.createQuery(upit, Object[].class)

. . .

Ovakav rad nije najelegantniji, stoga nam HQL omogucava da kreiramo objekat ciljne klase direktno iz upita:

String upit = “SELECT new Knjiga(k.naziv, k.izdavac) FROM Knjiga AS k”

List<Knjiga> knjige = session.createQuery(upit).list(); . . .

Spajanje tabela

Dozvoljeno je uvodjenje novih imena – aliasa.

FROM Knjiga AS k WHERE k.naziv = :naziv; AS je opciono.

HQL podrzava:

- INNER JOIN

- LEFT OUTER JOIN

- RIGHT OUTER JOIN

Pored ovakvog eksplicitnog spajanja, postoji I implicitno koje se realizuje operatorom tacka “.”

Agregatne funkcije

AVG, MAX, MIN, COUNT(*), … kao I u SQL

List lista = session.createQuery(“SELECT max(cena) FROM Knjiga”).list();

Polimorfni upiti

FROM Knjiga as k

ne vraca samo instance klase Knjiga vec I svih njenih potklasa.

FROM java.lang.Object o; → izdvojio bi sve trajne objekte

Azuriranje, brisanje I umetanje

Podatke je moguce azurirati I obrisati metodom executeUpdate(); Vraca broj redova na koje je upit imao uticaja.

Jedino je moguce umetnuti entitete dobijene naredbom SELECT

String insertUpit = “

INSERT INTO Knjiga_backup(id_knjige, naziv, izdavac, god_izdavanja)

SELECT id_knjige, naziv, izdavac, god_izdavanja

FROM Knjiga WHERE god_izdavanja=1990”

Query upit = sesija.createQuery(insertUpit);

int uspeh = upit.executeUpdate();

Imenovani upiti

@NamedQuery(name = “Izdvoji_knjige”, query = “FROM Knjiga”)

metod getNamedQuery();

Query upit = session.getNamedQuery(“Izdvoji_knjige”);

List knjige = upit.list();

@NamedQueries(value = {@NamedQuery(name = “Izdvoji_knjige”, query = “FROM Knjiga”),

@NamedQuery(name = ”Izdvoji_knjige_za_naziv”, value = “FROM Knjiga WHERE naziv = :naziv”)})

Moze se koristiti I cist SQL, createNativeQuery() metod klase NativeQuery.

NativeQuery upit = session.createNativeQuery(“SELECT * FROM Knjige”);

List Knjige = upit.list();

@NamedNativeQuery, @NamedNativeQueries

Administracija baze podataka

Baze podataka se kreiraju unutar instance menadzera baze podataka. To je aplikacija koja je deo SUBP-a I koja obezbedjuje osnovne funkcionalnosti rada sa bazama podataka – kreiranje, preimenovanje, brisanje, …

Instanca je logicko okruzenje baze podataka koje omogucava upravljanje bazama podataka. U okviru instance moguce je bazu uneti u katalog I postaviti joj konfiguracione parametre.

Naredba db2ilist – listaju se sve instance na sistemu. Naredbom get instance prikazuju se detalji trenutne instance.

set DB2INSTANCE <naziv> - promena trenutne instance, db2start/db2stop

Prostori tabela su skladisne strukture koje sadrze tabele, indekse I velike objekte.

Svaka baza podataka pravi se sa 3 podrazumevana sistemska prostora tabela:

1.) SYSCATSPACE: tabele sistemskog kataloga DB2

2.) TEMPSPACE1: privremene sistemske tabele

3.) USERSPACE1: inicijalni prostor za definisanje korisnickih tabela I indeksa

Prostori tabela sastoje se iz jednog ili vise kontejnera, koji mogu biti naziv direktorijuma, naziv uredjaja ili naziv datoteke.

dftdbpath – podrazumevana vrednost putanje baze podataka

GET DATABASE MANAGER CONFIGURATION ili skraceno GET DBM CFG

UPDATE DATABASE MANAGER CONFIGURATION ili UPDATE DBM CFG

Grupa skladista – imenovani skup putanja za skladistenje tabela I drugih objekata u bazi

IBMSTOGROUP – podrazumevana grupa skladista

SELECT * FROM SYSCAT.STOGROUPS – ispis svih grupa skladista

CREATE STOGROUP stg1 ON ‘/data1’

CREATE TABLESPACE ts1 USING STOGROUP stg1

SELECT * FROM SYSCAT.TABLESPACES – ispis svih prostora tabela

DB2 podrzava naredna 3 nacina upravljanja prostorima tabela:

1.) SMS (System Managed Space): menadzer fajl sistema os-a alocira I upravlja prostorom u kome je sacuvana tabela

2.) DMS (Database Managed Space): menadzer baza podataka upravlja prostorom za skladistenje

3.) Automatsko skladistenje

CREATE TABLESPACE <prostor_tabela>, U bazama u kojima nije moguce automatsko skladistenje potrebno je navesti MANAGED BY DATABASE ili MANAGED BY SYSTEM pri kreiranju. Inace se podrazumeva MANAGED BY AUTOMATIC STORAGE.

Sheme SYSCAT I SYSSTAT sadrze prvi I drugi skup pogleda, redom. Pogledi iz SYSCAT se mogu samo citati, dok pogledi iz SYSSTAT sadrzi skup statistickih informacija koje koristi SQL optimizator.

CATALOG DATABASE, UNCATALOG DATABASE

Naredba CREATE DATABASE

Inicijalizuje novu bazu podataka, pravi tri inicijalna prostora tabela, kreira sistemske tabele I alocira log datoteku oporavka.

CREATE DATABASE knjige3
AUTOMATIC STORAGE YES
ON /bpauto31, /bpauto32, /bpauto33
DBPATH ON /bpknjige3

Putanja baze bi bila /bpknjige, skaldistenje automatsko I to sa 3 putanje /bpauto1-3

Moguce je zadati I zemlju, kodiranje I uparivanje koje ce se koristiti.

CREATE DATABASE knjige4 AUTOMATIC STORAGE yes
USING CODESET utf-8 TERRITORY rs

ALIAS <alias_bp>

Sheme: SYSCAT, SYSFUN, SYSIBM, SYSSTAT

Premestanje podataka

Naredbe IMPORT I EXPORT

IMPORT – umetanje podataka iz ulazne datoteke u bazu, EXPORT – kopiranje iz baze u izlaznu datoteku.

EXPORT TO <ime_datoteke> OF <tip_datoteke>
[MODIFIED BY <modifikator>]
[MESSAGES <datoteka_za_poruke>]
<select_naredba>

EXPORT TO temp OF del
MODIFIED BY coldel,
SELECT * FROM Knjiga;

```
IMPORT FROM <ime_datoteke> OF <tip_datoteke>
[MODIFIED BY <modifikator>]
[ALLOW NO ACCESS|ALLOW WRITE ACCESS]
[MESSAGES <datoteka_za_poruke>]
[COMMITCOUNT (n|AUTOMATIC)] [RESTARTCOUNT n]
(INSERT|INSERT_UPDATE|REPLACE|REPLACE_CREATE)
INTO <naziv_tabele>
```

Podrazumevano se prilikom uvoza postavlja X katanac na tabeli da bi se zastitili podaci.

- COMMITCOUNT n izvodi operaciju COMMIT nakon svakih n uvezenih redova.

Primer:

```
IMPORT FROM knjiga_dat.ifx OF ixf
MESSAGES knjiga_poruke.txt
INSERT INTO knjiga
```

Naredba LOAD

Naredbom IMPORT se u stvari izvodi operacija INSERT. Za razliku od nje naredbom LOAD se formatirane stranice direktno upisuju u bazu.

Sastoji se od cetiri faze:

1.) Faze punjenja: tabele se pune podacima I prikupljaju se kljucevi indeksa I statistike tabela

2.) Faze izgradnje: Formiraju se indeksi na osnovu kljuceva indeksa iz prve faze

3.) Faze brisanja: iz tabele se brisu redovi koji krse ogranicenja jedinstvenog ili primarnog kljuca

4.) Faze kopiranja indeksa: indeksirani podaci se kopiraju iz sistemskog privremenog prostora tabela u originalni prostor tabela.

```
LOAD FROM <datoteka> OF [ASC|DEL|IXF] [MODIFIED BY <mod>]
[SAVECOUNT n] [ROWCOUNT n] [WARNINGCOUNT n]
[MESSAGES <datoteka_za_poruke>]
(INSERT|REPLACE|RESTART|TERMINATE) INTO <naziv_tabele>
[FOR EXCEPTION <tabela_izuzetaka>]
[ALLOW NO ACCESS| ALLOW READ ACCESS [USE naziv_prostora_tabela]]
```

- Navodjenjem SAVECOUNT n, tacke pamcenja se postavljaju na svakih n redova

- ROWCOUNT n – ucitava se samo prvih n redova

- WARNINGCOUNT n – zaustavlja se nakon n upozorenja

- INSERT | REPLACE | RESTART | TERMINATE: podaci se dodaju u bazu bez izmene vec postojecih | brisu se postojeci I ucitavaju novi | ponovo se zapocinje prethodno prekinut LOAD | terminise se prethodno prekinut LOAD I vraca se na tacku u kojoj je ona zapocela.

Pravljenje rezervnih kopija I oporavak

Naredba BACKUP DATABASE

```
BACKUP DATABASE <alijas_bp> [USER <username> [USING <password>]]
TO <dir/dev>
```

Pravi se kopija podataka koja se cuva na nekom drugom medijumu.

<alijas_bp>: alijas baze ciju kopiju pravimo

```
alijas_bp.tip.inst.DBPARTnnn.timestamp.redni_br
```

Inkrementalno pravljenje rezervnih kopija: uzastopne kopije sadrze sadrze samo onaj deo koji je menjan od prethodne.

```
BACKUP DATABASE bp_knjiga INCREMENTAL DELTA
```

Naredba RESTORE

Moguce je povratiti bazu podataka cija je rezervna kopija napravljena naredbom BACKUP.

RESTORE DATABASE bp_knjiga

```
RESTORE DATABASE bp_knjiga FROM /dev/backup – putanjom
```

```
RESTORE DATABASE bp_knjiga
TAKEN AT 20200202010101
```

Ukoliko postoji vise od jedne rezervne kopije, moramo navesti kada je kreirana ona koju zelimo.

```
RESTORE DATABASE bp_knjiga
INCREMENTAL AUTOMATIC
TAKEN AT 20200202010101
```

Vracamo bazu na osnovu inkrementalnih rezervnih kopija.

Naredba RECOVER DATABASE

Vrsi se povrataka baze na neku rezervnu kopiju, a zatim se ponovo izvrsavaju transakcije koje su uspesno zavrsene do nekog vremenskog trenutka ili do kraja loga.

`RECOVER DATABASE bp_knjiga`

`RECOVER DATABASE bp_knjiga TO 2020-02-02-01.01.01,` povratak na dati vremenski trenutak.

Aktivnosti na odrzavanju baze podataka

Naredba RUNSTATS

Naredba RUNSTATS se moze iskoristiti za sakupljanje novih statistika tabela I indeksa I azuriranje statistika u sistemskom katalogu. Moguce je dobiti info o:

1.) Broju stranica koje sadrze redove

2.) Broju stranica koje su u upotrebi

3.) Broju redova u tabeli

4.) Vrednostima statistika koje se ticu raspodele podataka

5.) Detaljne karakteristike za indekse koji nam govore koliko se efikasno pristupa podacima preko indeksa

`RUNSTATS ON TABLE knjiga,` osnovne karakteristike za tabelu knjiga

```
RUNSTATS ON TABLE knjiga
AND SAMPLED DETAILED INDEXES ALL
```

Naredbe REORG I REORGCHK

Naredba REORG moze se upotrebiti za reorganizaciju tabela I indeksa da bi se popravila efikasnost skladistenja I smanjili troskovi pristupa.

`REORG TABLE knjiga USE TEMPSPACE1`

Naredba REORGCHK se koristi da dobijanje preporuka koje tabele I indeksi bi imali koristi od reorganizacije.

```
REORGCHK [UPDATE STATISTICS | CURRENT STATISTICS]
ON [SCHEMA <naziv_sheme> |
TABLE [USER | SYSTEM | ALL | <naziv_tabele>]]
```

- UPDATE STATISTICS poziva RUNSTATS da bi se azurirale statistike o tabelama I indeksima

- CURRENT STATISTICS: koriste se trenutno raspolozive vrednosti statistika