05-06 – Adatbázis-programozás (1)

|  |  |
| --- | --- |
| PL/SQL | T-SQL |
| Alapműveletek, vezérlőszerkezetek | |
| declare  konstansok; változók; kurzorok; kivételek;  begin  if (*feltétel*) then ...  elsif (*feltétel*) then ...  else ... end if;  while *feltétel* loop ... end loop;  for *i* in [reverse] 1..5 loop ... end loop;  exception  when no\_data\_found then  raise\_application\_error(-20000, ’Hiba!’);  end; | declare  @valtozo1 int  begin  if *feltétel* begin ... end  else if *feltétel* begin ... end  while *feltétel* begin ... end  raiserror(’Hiba: %d’, 16, 1, 1234)  begin try ... end try  begin catch ... end catch  end |
| Tábla- és nézetműveletek | |
| insert into *táblanév*  values (1,to\_date(’2009-05-31’,’yyyy-mm-dd’));  delete from *tablanev* where *feltétel*; | create table táblanév (  id int identity primary key,  nev nvarchar(50) not null constraint nev\_fk  references *másiktábla*(*nev*),  cim nvarchar(100)  )  insert into {*táblanév*|*nézetnév*} (*nev*, *cim*)  values (’Gipsz Jakab’, ’Budapest’)  create view *nézetnév* as select *név* from *tábla*; |
| Tárolt eljárás/függvény létrehozása | |
| create or replace {procedure|function} *procnev*  (param1 in nvarchar2, param2 out int)  [return int]  as  begin  [return 42;]  end *procnev*; | create {procedure|function} *procnev*  @param1 nvarchar(50), @param2 float  [returns int]  as  [return 42] |
| Függvényhívás, változónak értékadás | |
| declare  nev nvarchar(200);  szam int;  begin  nev := ’Gipsz Jakab’;  procnev(nev => param1);  end; | declare  @nev nvarchar(200)  begin  declare @szam int;  set @szam = 42;  exec *procnev*(@param1 = ’Gipsz Jakab’)  exec *procnev* ’Gipsz Jakab’  end |

05-06 – Adatbázis-programozás (2)

|  |  |
| --- | --- |
| Kurzorok | |
| declare  cursor *kurzornev*(param1 int) is  select *nev*, *cim* from *tabla1*  where id = param1;  cnev tabla1.nev%type;  begin  for *kurzorrekord* in *kurzornev*(42) loop  ...  end loop;  end; | begin  declare *kurzornev* cursor for  select *nev, cim* from inserted;  declare *@nev* nvarchar(50), *@cim* nvarchar(50)  open *kurzornev*  fetch *kurzornev* into @nev, @cim  while @@fetch\_status = 0 begin  ...  fetch *kurzornev* into @nev, @cim  end  close *kurzornev*  deallocate *kurzornev*  end |
| Triggerek | |
| create or replace trigger *trignev*  {before|after} {insert or update or delete}  on *tablanev*  [for each row]  [when feltétel]  begin  update *tábla* set *nev* = :new.*nev*  where *sorszam* = 1;  end | create trigger  on {*tablanev*|*nezetnev*}  {for|after|instead of} {insert,update,delete}  as  if columns\_updated() begin ... end |
| Egyebek | |
| create sequence *szekvencianév*  start with 10 increment by 1;  dbms\_output.put\_line(*valtozo* || ’ darab’); | print ’Dátum: ’ + convert(nvarchar, @*datum*) |

07 – Lekérdezés-optimalizálás (1)

Blokkolvasás költsége:

* keresési idő: amíg a fej megtalálja a cilindert
* rotációs késés: keresett blokk a fej alá fordul
* adatátviteli idő

Csökkentés:

* korai beolvasás (🡪bufferelés)
* cilinder alapú szervezés
* RAID
* lift algoritmus

Logikai terv – elemző fa:

* relációk (levelekben)
* műveletek (csomópontokban)
* adatáramlás levéltől gyökér felé

Lekérdezés-feldolgozó szerkezete:

* elemző (SQL fordítása, logikai terv készítése)
* optimalizáló (fizikai terv készítése, táblabejárások, join-ok)
* sorfordító (fizikaiból I/O terv készítése)
* végrehajtó

Allekérdezések:

* nem korrelatív: allekérdezés külön végezhető (pl. „in” operátor)
* korrelatív: allekérdezést a fő lekérdezés minden sorára le kell futtatni (pl. „exists”)

Nézetet tartalmazó lekérdezés: nem optimalizálódik, tárolt tervet használ.

Elemző fa átalakítása

* szelekció lelfelé
* projekció felfelé
* join-használata Descartes-szorzat helyett
* join egyik attribútuma mindig tábla

I/O-t befolyásoló tényezők

* B(R) – R reláció által foglalt blokkok száma
* T(R) – R reláció sorainak száma
* V(R,a) – R reláció „a” attribútumainak variáltsága

Táblaelérések

* Full table scan
* Index scan (szelekció és rendezés esetén)
* Nested loops join
  + nagy tábláknál: a táblák egy-egy blokkját tartja memóriában
  + kis tábláknál: egész a mem.ban
* Hash join
  + 1. lépés: kisebb R beolvasása; vödrös hash, kulcs a join attribútuma
  + 2. lépés: nagyobbik S beolvasása, kapcsolódó rekordok keresése a vödrös hash-ben
* Sort merge join – kicsi táblákhoz, index kell
  + R, S beolvasása
  + join kulcs szerinti rendezés
  + rendezett listák összefésülése

07 – Lekérdezés-optimalizálás (2)

Oracle Server

Full Table Scan

* nincs index a táblán
* sok rekord lesz az eredményben
* kicsi a tábla

RowID scan

* sorazonosító alapú beolvasás
* indexből jön a sorazonosító

Unique index scan – preferált

* PK használata
* ha az index összes oszlopára szűrünk

Index range scan

* index alapú szűrés
* =, <, >, like
* index szerint rendezett eredmény

Full index scan

* index szerinti teljes végigolvasás
* kulcsok szerint rendezett eredmény

Fast full index scan

* ha csak az indexbeli attribútumok kellenek
* táblát el sem éri ilyenkor

Index join

* ha több index is van a táblán, amik együtt tartalmazzák az összes szükséges attribútumot
* tábla indexeinek hash-alapú join-ja

Bitmap index scan (lásd 07/33. dia)

* enum oszlop értékkészletéhez indexoszlopok, bennük igaz/hamis értékek

Nested loops join – preferált

* ha kevés külső sor kell

Hash join

* equi-join esetén
* egyik reláció befér a memóriába

Sort merge join

* rendezett forrástáblák
* ha egy későbbi műveletnek amúgy is rendezett eredmény kell

Indexek

* B\*-fa: egyszerű | összetett | függvény alapú
* bitmap index
* index organized table (blokksorrend index szerint)

SQL hint

* preferált index megadása
* index tiltása
* join típus megadás
* join sorrend megadás
* táblaelérési mód megadás
* párhuzamos végrehajtás
* lekérdezés átírása: OR helyett több lekérdezés + UINON ALL

MS SQL Server

Table scan

* ha nincs semmilyen index
* táblára vonatkozó szűrési feltételt is kiértékeli

Clustered index scan – preferált table scan helyett

* nyalábolt adatolvasás
* adatblokkok index szerint rendezve
* PK mentén létrejön

Nonclustered index scan

* mint a clustered
* = operátor kiértékeléséhez

Clustered/Nonclustered index seek

* hasonló az index scan-hez
* B\*-fa bejárása egy kezdőelemtől
* <, >, between operátorokhoz

Join-ok

* nested loops
* hash match
* merge join

Indexek

* B\*-fa: egyszerű | összetett |clustered
  + clustered: adatblokkok sorrendje index szerinti
  + egy lehet belőle egy táblán
  + definiált PK mentén automat. létrejön

Jó tanácsok

* join mindig jobb, mint:
  + in, not in
  + exists, not exists (ezek helyett: in)
* nézeteket kerüljük, ne join-oljuk
* OR helyett: több lekérdezés + UNION ALL
* UNION helyett UNION ALL

08 – ADO.NET (1)

|  |  |
| --- | --- |
| Kapcsolat alapú   * egyszerűbb konkurrenciakezelés * adatok mindenhol a legfrissebbek * folyamatos hálózati kapcsolat * nehezen skálázható   Pesszimista ütközéskezelés: adatbázis szintű zárak használata (pl. tranzakcióval) | Kapcsolat nélküli   * ütközések lehetnek * adatok nem mindenhol a legfrissebbek * kapcsolat csak az adatmanipuláció idejére * skálázható   Optimista ütközéskezelés: ütközések feloldása. Ütközésvizsgálat:   * tartalom alapján: TimeStamp-et nézva vagy teljes tartalomvizsgálattal * DataAdapter-en állítjuk be |
| DataReader   * egyirányú, csak olvasható * ha az adatot azonnal feldolgozzuk   Használat:   1. kapcsolat megnyitása 2. parancs futtatása 3. eredmény feldolgozása 4. reader lezárása 5. kapcsolat bontása   Kommunikációs stack:   * SqlDataReader * SqlCommand * SqlConnection * SQL Server | DataSet   * lokális adatcache * ha az adatok közt navigálunk, módosítjuk őket   Használat:   1. kapcsolat megnyitása 2. DataSet feltöltése 3. kapcsolat lezárása 4. DataSet feldolgozása 5. kapcsolat megnyitása 6. változások visszaírása 7. kapcsolat lezárása   Kommunikációs stack:   * DataSet * SqlDataAdapter * SqlConnection * SQL Server |

Connection pooling: minél később megnyitni, minél előbb lezárni a kapcsolatot.

IDbConnection: Open, Close, BeginTransaction(isolation\_level)

IDbCommand: CommandType (tárolt eljárás, tábla teljes tartalma, SQL query), CommandText, Connection, Transaction, Parameters; ExecuteReader, ExecuteScalar, ExecuteNonQuery, ExecuteXmlReader

SqlCommand command = new SqlCommand();

command.Connection = connection;

command.CommandText = "SalesByCategory";

command.CommandType = CommandType.StoredProcedure;

SqlParameter parameter = new SqlParameter();

parameter.ParameterName = "@CategoryName";

parameter.SqlDbType = SqlDbType.NVarChar;

parameter.Direction = ParameterDirection.Input;

parameter.Value = categoryName;

command.Parameters.Add(parameter);

using (SqlConnection conn = new SqlConnection(connString))

{

connection.Open();

SqlDataReader reader = command.ExecuteReader();

...

08 – ADO.NET (2)

Kapcsolat nélküli modell nehézségei

* Állapotkövetés – DataRowState:
  + unchanged: változatlan
  + added: újonnan hozzáadott
  + deleted: újonnan törölt
  + modified: módosított
  + detached: nem része a DataRowCollection-nek
* Verziókövetés – DataRowVersion:
  + original: eredeti
  + current: aktuális
  + default: DataRowState-től függ
  + proposed: szerkesztás alatt álló
* Változáskövetés – DataTable, DataSet, DataAdapter szintjén
  + GetChangeSet()
  + Merge()
  + Load()
* FEGURM probléma: egyszerű adatfrissítés 🡪 6 lépés: Fill / Edit / Get changes / Update / Refresh / Merge  
  Megoldás: DataSet.Load() – FillOption:
  + OverwriteChanges – original és current módosítása
  + PreserveChanges – original módosítása
  + Upser – current módosítása
* PK – szerveroldalon generált, mégis kell kliensoldalon új sor hozzáadásakor  
  Megoldás: ideiglenes azonosítókkal, de FK-k miatt ez nem mindig elég

09 – ORM

Hierarchia leképezése egy közös táblába

Egyszerű hierarchiák leképezéséhez.

Megvalósítás

* az objektum minden attribútumát (örököltekkel) felsoroljuk
* objektumtípusok (osztályok) tárolása:
  + minden típushoz egyedi azonosító VAGY
  + IsClass1, IsClass2, … oszlopok
* egyedi példányazonosító tárolása
* bővítés kezelése: új attribútumok felvitele

Jellemzők

* ☺ egyszerű
* ☺ könnyű új osztályt vinni a hierarchiába
* ☺ könnyen követhető, melyik objektum milyen osztályok példánya
* ☹ helypazarló
* ☹ egy osztály megváltozása az egész tárolást változtatja
* ☹ nehezen áttekinthető

Minden valós osztály leképezése saját táblába

Ritkán változó struktúrák leképezéséhez.

Megvalósítás

* minden osztálynak saját tábla
* az osztály minden attribútumának eltárolása
* egyedi példányazonosító tárolása
* ha változik egy attribútum, csak a hierarchiaszint mentén kell végigvinni

Jellemzők

* ☺ átlátható
* ☺ gyors adatelérés
* ☺ jól illeszkedik az objektummodellhez
* ☹ osztály módosítása 🡪 hierarchiaszintek módosítása
* ☹ több szerepet is betöltő példányok kezelése nehézkes

Minden osztály leképezése saját táblába

Komplex öröklési hierarchia és gyakran változó struktúrák leképezéséhez.

Megvalósítás

* a táblák követik az osztályhierarchiát
* szülő-gyerek viszony leképezése külső kulcsokkal
* egyedi példányazonosító tárolása

Jellemzők

* ☺ osztályok struktúrája könnyen módosítható
* ☹ összetett DB-séma
* ☹ egy példány adatai több táblában vannak (öröklés miatt) 🡪 összetett lekérdezés

Osztályok és hierarchiaszintek általános leképezése

Komplex alkalmazásokban, kis mennyiségű adat mellett, ahol akár futási időben is minden változhat.

Megvalósítás

* metaadat-központú
* általános séma: tetszőleges hierarchia leírható – hierarchia 🡪 metaadat, példányok 🡪 attribútumok
* class, attribute, attributeType, value, inheritance, stb… táblák

Jellemzők

* ☺ perzisztencia-kezelő rendszerekhez illeszkedik
* ☺ flexibilis, bármi leírható benne
* ☹ nehéz összeszedni egy példány adatait
* ☹ sok adat esetén nem hatékony

Kapcsolatok leképzése

1-1: külső kulcs valamelyikbe

1-\*: külső kulcs az 1-re

\*-\*: kapcsolótáblával

Rekurzió (más néven reflexió; önmagát tartalmazó osztályok): leképezés hasonlóan \*-\* kapcsolathoz

Statikus tagok leképezése

* mindnek külön tábla: gyors, de sok kicsi tábla
* mind egy táblába, különböző oszlopokba: gyors, de sok ütközés
* osztályonként egy tábla, az értékek külön oszlopokban: gyors, de sok kicsi tábla
* általános static-tábla: egyszerű
  + minden tulajdonság új rekord:   
    class, name, value hármasokkal
  + konverzió megoldandó