

Transakcia, Kurzory

1) TRANSAKCIA

1a) Databázová transakcia, Vlastnosti ACID

1b) Explicitná transakcia

1c) LOG protokol

1d) Príklad

2) Kurzory

2a) Deklarácia kurzora

2b) Vnorené kurzory

1) TRANSAKCIA

1a) Databázová transakcia, Vlastnosti ACID

1b) Explicitná transakcia

1c) LOG protokol

1d) Príklad

1a) Databázová transakcia je samostatný celok príkazov na prevod dát. ____

Ak transakcia je

- úspešná, potom všetky operácie v rámci transakcie s dátami sú vykonané a uložené do DB.
- neúspešná, potom operácie v rámci transakcie sú anulované a zmeny v dátach sú odstránené

Transakcia spĺňa **vlastnosti ACID**:

- **A** – **Atomicity** - transakcia je atomická operácia: vykoná sa buď ako **celok** alebo vôbec nie
- **C** – **Consistency – dôslednosť** - transakciou sa nenaruší žiadne **integritné** obmedzenie v rámci DB
- **I** – **Isolation** – vrátenie transakcie **nevyvolá** ďalšiu transakciu
- **D** – **Durability – trvalosť** - úspešné transakcie sú **uložené** do databázy

1b) Explicitná transakcia:

Formy transakcie

- Autocommit transactions – každý individuálny príkaz je transakcia;
- Implicit transactions - nová transakcia je implicitne zahájená po dokončení predchádzajúcej transakcie pomocou COMMIT alebo ROLLBACK;
- Batch-scoped transactions – platí len pre multiple active result sets (MARS);
- **Explicit transactions** – každá transakcia je explicitne počatá s BEGIN TRANSACTION a explicitne ukončená s COMMIT alebo ROLLBACK príkazom

- **BEGIN TRANSACTION**
- **COMMIT or ROLLBACK**

Commit označuje **koniec úspešnej** implicitnej alebo explicitnej transakcie.

Ak **@@TRANCOUNT** je 1, COMMIT TRANSACTION zabezpečí, aby všetky zmeny dát vykonané od začiatku transakcie stali natrvalo súčasťou databázy, uvoľní prostriedky potrebné pre transakcie, a @@TRANCOUNT nastaví na 0. Ak @@TRANCOUNT je väčšie ako 1, zníži sa @@TRANCOUNT iba o 1 a transakcia zostane aktívna.

ROLLBACK TRANSACTION sa využíva na vymazanie všetkých zmien dát vykonané od začiatku transakcie alebo uloženého bodu v prípade **neúspešnej** transakcie. Príkaz tiež uvoľní prostriedky držané transakciou.

1c) LOG protokol

Do transaction LOG protokolu sa zaznamenávajú všetky **transakcie** a **zmeny/úpravy** databázy vykonané každou transakciou. Ak dôjde k zlyhaniu systému a dôjde k strate kontaktu s klientom a *transakcia* sa nerealizuje úplne, databáza sa vráti do pôvodného konzistentného stavu pomocou

log súboru. Ak server zlyhá, databáza môže zostať v stave, že niektoré *úpravy* neboli nikdy zapísané z vyrovnávacej pamäte do dátových súborov.

Koncepcne log súbor je **reťazcom** (protokolových) log záznamov. **Fyzicky**, postupnosť log záznamov je uložená efektívne do fyzických **súborov**, ktoré implementujú transaction log.

Niektoré **typy obnovení** (recovery)

- Individuálne obnovenie transakcie.
- Obnova všetkých nedokončených transakcií.
- Rolovanie obnovej databázy, súboru, skupiny súborov alebo stránky späť do bodu zlyhania.

Obnova DB servera pozostáva z troch fáz

1. fáza analýzy analyzuje protokol transakcií pomocou kontrolných bodov, a vytvorí tabuľku špinavých stránok a tabuľku aktívnych transakcií (ATT).
2. fáza Redo sa stará o **neúplné úpravy** zaznamenané v protokole, ktoré nie sú úplne zapísané do dátových súborov. Špinavé stránky sa tu zapisujú do dátových súborov.
3. fáza Undo roluje späť **neúplné transakcie** nájdené v ATT, aby sa zaistilo zachovanie integrity databázy. Po rollback sa databáza prepne do režimu online.

Pages do pamati - WHERE, TRANS, LOCK, Zmeny v pamati, Zmeny do LOG, Commit

1d) Príklad

V rámci transakcie s využitím uložených procedúr budeme vykonať dve operácie

- vkladanie údajov (insert)
- vymazanie údajov (delete)

Pri vytvorení databázy **TransakciaDB** ukážeme ako je možné nastaviť niektoré parametre, ako napr. jej počiatočnú veľkosť a uvedieme dve úspešné transakcie a tri neúspešné.

A) Najprv vytvoríme DB **TransakciaDB**, v nej dve tabuľky **Osoba** a **OsobaUdaje** a vložíme do nich dva, dvariadky

```
(1, 'Bobo Bibi', 'IBM ABC'),  
(2, 'Tata Tutu', 'Micro XYZ')
```

```
(1, 'Roznava'),  
(2, 'Kosice')
```

Cesta k súborom databáz MS SQL Server

```
SELECT top 1 physical_name[ ] FROM sys.master_files WHERE file_id = 1;
```

Skontroluj

- obsah adresára - Search DATA z C:\Program Files\Microsoft SQL Server
C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA
- Adresar ma byt existujuci: C:\Csaba_____Vyuka

```
USE Master
```

```
GO
```

```
IF DB_ID ('TransakciaDB') IS NOT NULL DROP DATABASE TransakciaDB
```

```
GO
```

```

CREATE DATABASE TransakciaDB
---- Ak pristup pre subory .mdf a .ldf nie je zakazany, inak zakomentuj:
ON PRIMARY(
NAME = N'TransakciaData',
FILENAME = N'C:\Csaba\_____Vyuka\TransakciaDB.mdf',
-- SIZE = 3MB, -- = 5MB
MAXSIZE = 10MB,
FILEGROWTH = 20%
)

LOG ON(
NAME = N'TransakciaLog',
FILENAME = N'C:\Csaba\_____Vyuka\TransakciaDB.ldf',
SIZE = 1MB,
MAXSIZE = 5MB,
FILEGROWTH = 1MB
)
GO

USE TransakciaDB
GO
create table Osoba(
    idOsoba int primary key not null,
    meno nvarchar(10) not null,
    firma nvarchar(15)
)
create table OsobaUdaje(
    idOsoba int foreign key references dbo.Osoba(idOsoba),
    adresa nvarchar(30)
)
Insert into Osoba values
(1, 'Bobo Bibi', 'IBM ABC'),
(2, 'Tata Tutu', 'Micro XYZ')
Insert into OsobaUdaje values
(1, 'Roznava'),
(2, 'Kosice')
GO

```

B) Vytvoríme uloženú procedúru `sp_InsertDelete`, ktorá pomocou **begin transaction** ... vloží a deletuje Osobu s kontrolou chýb a **commit** či **rollback**.

```

IF OBJECT_ID ( 'sp_InsertDelete', 'P' ) IS NOT NULL
    DROP PROCEDURE sp_InsertDelete;
GO
create procedure sp_InsertDelete
    @idOs int,
    @meno nvarchar(10),
    @firma nvarchar(15),
    @idOsOld int

```

```

as
    declare @erIns int
    declare @erDel int
    declare @er int

-- 1)
begin transaction
-- a) Pridaj osobu
insert into Osoba (idOsoba, meno, firma) values(@idOs, @meno, @firma)

-- Mozna chyba po Insert
set @erIns = @@error
if @erIns > 0 set @er = @erIns

-- b) Maz osobu
delete from Osoba where idOsoba = @idOsOld

-- Mozna chyba po Delete
set @erDel = @@error
if @erDel > 0 set @er = @erDel

-- 2)
-- If error, roll back
if @er <> 0
begin
rollback
RAISERROR('Csaba - Transaction rolled back', 11,1)
end
else
begin
commit
print 'Csaba - Transaction committed'
end
print 'Csaba - INSERT error number:' + cast(@erIns as nvarchar(8))
print 'Csaba - DELETE error number:' + cast(@erDel as nvarchar(8))
return @er
GO

```

C) Teraz vložíme 2 osoby s novými id a vymažeme osobu s id 33. Operácie s danými hodnotami nenarúšajú integritu dát, preto systém ich vykoná a nehlási chybu.

--In 2008 DELETE returns error number 0 even though it has not deleted any rows

```

select * from osoba
select * from OsobaUdaje

```

-- Pozri Messages:

```

exec sp_InsertDelete 3, 'Fero', null, 33
exec sp_InsertDelete 4, 'Jano', null, 33

```

```
select * from osoba
select * from OsobaUdaje
```

Ani jeden z nasledujúcich troch príkazov sa nevykoná a systém hlási rôzne chyby.

- Tu chceme vkladať osobu s id, ktorá už existuje a vymazať osobu s existujúcim id, ale neexistujúcim cudzím kľúčom.

```
exec sp_InsertDelete 4, 'Stevo', null, 3 -- Err Primary Key
```

- Tu by sme chceli vložiť osobu s novým id a mazať inú osobu s existujúcim primárnym a cudzím kľúčom.

```
exec sp_InsertDelete 5, 'Stevo', null, 2 -- Err Delete Reference error
```

- Tu ide o snahu vložiť osobu s id, ktoré už existuje a mazať inú osobu s existujúcim primárnym a cudzím kľúčom.

```
exec sp_InsertDelete 4, 'Stevo', null, 2 -- Err PK + Err Delete
```

2) Kurzory

2a) Deklarácia kurzora

2b) Vnorené kurzory

Kurzor je DB prostriedok na získanie prístupu k jednotlivým **riadkom** tabuľky DB. Efektívne spracovanie tabuliek po riadkoch pomocou kurzora môže byť problematické, lebo DB-vé optimalizačné systémy samostatne ťažko nájdu optimálnu verziu užívateľského kódu s kurzorom. Napriek tomu, DB systémy podporujú kurzory (MS SQL, Oracle viac) na základe štandardu SQL-92.

SQL ako vysokoúrovňový deklaratívny jazyk vs. nízkoúrovňový cyklus.

Syntax.

1) Deklarácia

DECLARE kur1 CURSOR FOR SELECT ...

2) Otvorenie + použitie: získanie

3) Zatvorenie

Podrobnejšie:

ISO Syntax

```
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR
    FOR select_statement
    [ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]
[;]
```

Transact-SQL Extended Syntax

```
DECLARE cursor_name CURSOR [ LOCAL | GLOBAL ]
    [ FORWARD_ONLY | SCROLL ]
    [ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
    [ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
    [ TYPE_WARNING ]
    FOR select_statement
    [ FOR UPDATE [ OF column_name [ ,...n ] ] ]
[;]
```

Štandardne: **FORWARD_ONLY => FETCH NEXT FROM kur**

SCROLL => FETCH FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE

FETCH

```
[ [ NEXT | PRIOR | FIRST | LAST
    | ABSOLUTE { n | @nvar }
    | RELATIVE { n | @nvar }
    ]
    FROM
    ]
{ { [ GLOBAL ] cursor_name } | @cursor_variable_name }
[ INTO @variable_name [ ,...n ] ]
```

Skalárne funkcie pre kurzory

@@CURSOR_ROWS – konštanta vráti počet riadkov v kurzore

- 0 – a) kurzor nebol otvorený alebo bol zatvorený resp. dealokovaný
b) ani jeden riadok. Vyskúšajte!
- 1 – kurzor je dynamický
- n – počet asynchrone vrátených riadkov
- n – kurzor je úplne naplnený

@@FETCH_STATUS – používa sa v cykle na zistenie, či kurzor ešte obsahuje riadok

- 0 – FETCH príkaz bol úspešný
- 1 – FETCH príkaz bol úspešný alebo riadok bol za množinou výsledkov
- 2 – riadok chýba

Príklad 1: Vráťte prvú DB zo zoznamu DB pomocou sysdatabases a **kurzora**.

```
-- 1/3) Deklaracia
DECLARE db_cursor CURSOR FOR
SELECT name FROM master.dbo.sysdatabases
      WHERE name NOT IN ('master', 'model', 'msdb')
-- 2/3) Otvorenie a použitie, získanie
OPEN db_cursor
-- Výsledok sa buď automaticky vypíše alebo uloží do premennej
-- DECLARE @jaj VARCHAR(50)
FETCH NEXT FROM db_cursor -- INTO @jaj; print @jaj
-- 3/3) Zatvorenie a uvoľnenie
CLOSE db_cursor
DEALLOCATE db_cursor
```

Príklad 2: Pokračovanie – vráťme zoznam všetkých (okrem niektorých systémových) DB pomocou kurzora - **cyklus**.

```
-- 1/3)
DECLARE db_cursor CURSOR
      FORWARD_ONLY
FOR
SELECT name FROM master.dbo.sysdatabases
      WHERE name NOT IN ('master', 'model', 'msdb')
-- 2/3)
OPEN db_cursor
DECLARE @dbName VARCHAR(50)
FETCH NEXT FROM db_cursor INTO @dbName
print cast(@@CURSOR_ROWS as char(10))
WHILE @@FETCH_STATUS = 0
BEGIN
    print @dbName
    FETCH NEXT FROM db_cursor INTO @dbName
END
```

```
-- 3/3)
CLOSE db_cursor
DEALLOCATE db_cursor
```

Príklad 3: Vráťte názvy všetkých (okrem niektorých) databáz s príslušnými tabuľkami pomocou **vnorených cyklov**.

```
-- Pomocou dvoch kurzorov, pritom jeden je retazcovy prikaz.
-- Vysledok do MESSAGES!!!
DECLARE @db VARCHAR(255)
DECLARE @tab VARCHAR(255)
DECLARE @cmd NVARCHAR(500)
DECLARE kurDB CURSOR FOR
SELECT name FROM master.dbo.sysdatabases
    WHERE name NOT IN ('master', 'msdb', 'model', 'ReportServer',
        'ReportServerTempDB', 'tempdb') and name !=
        'AdventureWorks2014'
    ORDER BY name
OPEN kurDB
FETCH NEXT FROM kurDB INTO @db
WHILE @@FETCH_STATUS = 0
BEGIN
    print ''
    print @db + ':'
    SET @cmd = 'DECLARE kurTab CURSOR FOR SELECT table_name FROM ['
        + @db +
        '].INFORMATION_SCHEMA.TABLES'
    EXEC (@cmd) -- vytvor kurTab!
    OPEN kurTab
    FETCH NEXT FROM kurTab INTO @tab
    WHILE @@FETCH_STATUS = 0
    BEGIN
        print ' ' + @tab
        FETCH NEXT FROM kurTab INTO @tab
    END
    CLOSE kurTab DEALLOCATE kurTab
    FETCH NEXT FROM kurDB INTO @db
END
CLOSE kurDB DEALLOCATE kurDB
```

Riešenie bez kurzora

Každej DB zodpovedajúce tabuľky (nedokumentovaná SP).

```
sp_msforeachdb 'select "?" AS db, * from [?].sys.tables'
```

Príklady na cvičení:

- Vkladať riadky do tabuľky pomocou kurzoru:
- Vytvoriť tabuľku t1 s tromi riadkami a tabuľku t2 do ktorej prenesiete riadky pomocou kurzora
- Stoličky