доц. д-р Цветанка Георгиева-Трифонова

ТРАНСАКЦИИ

СЪДЪРЖАНИЕ

- Свойства на трансакциите
- Проверка за грешки в трансакциите
- Нива на изолация на трансакциите
- Вложени блокове трансакции
- Точки на записване в трансакциите

Свойства на трансакциите

- *Трансакция* (*transaction*)
  - последователност от SQL конструкции, които биват потвърдени или отхвърлени като едно цяло;
- Свойства на трансакциите (ACID)
  - Атомарност (*atomicity*)
  - Съгласуваност (*consistency*)
  - Изолиране (*isolation*)
  - Дълготрайност (*durability*)

ТРАНСАКЦИИ

- `BEGIN TRANSACTION`
  - начало на всяка трансакция, състояща се от множество конструкции;
- `COMMIT TRANSACTION`
  - потвърждава промените в трансакцията;
- `ROLLBACK TRANSACTION`
  - отменя промените в трансакцията.

```
DECLARE @sum money, @st varchar(20)
BEGIN TRANSACTION
   UPDATE SaleDetails
      SET Quantity = 200
   WHERE SaleID = 1 AND ProductID = 5

   SELECT @sum = SUM(price *quantity*(1-discount))
   FROM SaleDetails WHERE SaleID = 1

   IF @sum > 100
      BEGIN
         ROLLBACK TRANSACTION
         SET @st = LTRIM(STR(@sum - 100, 20, 2))
         RAISERROR('Общата сума на продажбата
                    надхвърля 100 с %s.', 11, 1, @st)
      END
   ELSE
      BEGIN
        COMMIT TRANSACTION
        SET @st = LTRIM(STR(@sum, 20, 2))
        RAISERROR('Общата сума на продажбата е %s.',
                  11, 1, @st)
      END
```

ПРОВЕРКА ЗА ГРЕШКИ В ТРАНСАКЦИИТЕ

```
BEGIN TRY
  BEGIN TRANSACTION

    INSERT INTO Sales (CustomerID, EmployeeID, SaleDate)
    VALUES (18, 1, GETDATE())

    INSERT INTO SaleDetails
                (SaleID, ProductID, Price, Quantity)
    VALUES (@@IDENTITY, 1, 6, 10)

    UPDATE Products
       SET Stock = Stock - 10
    WHERE ProductID = 18

    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION
END CATCH
```

ПОСЛЕДОВАТЕЛНОСТ НА ПРОВЕРКИТЕ ЗА ЦЯЛОСТ

1. Присвояват се подразбиращите се стойности.
2. Проверяват се нарушения на NOT NULL.
3. Изчисляват се ограниченията CHECK.
4. Извършват се проверки на FOREIGN KEY в таблиците, които имат референция.
5. Извършват се проверки на FOREIGN KEY в таблиците, към които има референция.
6. UNIQUE (PRIMARY KEY) се проверява за коректност.
7. Активират се тригерите.

КОНТРОЛ НА ЕДНОВРЕМЕННАТА РАБОТА (*CONCURRENCY CONTROL*)

- ***Заключвания*** (*locks*)
  - механизмите, които СУБД използва, за да контролира едновременното извършване на модификации на данните от различни потребители.
- Видове
  - ***песимистични заключвания*** – предотвратяват модифициране на данните от страна на потребителите, което може да се отрази върху модификациите, извършени от други потребители.
  - ***оптимистични заключвания*** – осигуряват едновременна работа на потребителите, при която не се предотвратява никоя операция, но се анулират тези, които са създали конфликти.

НИВА НА ИЗОЛАЦИЯ НА ТРАНСАКЦИИТЕ

- *Ниво на изолация*
  - определя колко дълго трансакцията трябва да държи заключванията на данните за защита от евентуални промени, извършвани от други едновременни трансакции.
  - `READ UNCOMMITTED` – прочитане на непотвърдени данни;
  - `READ COMMITTED` – прочитане на потвърдени данни;
  - `REPEATABLE READ` – повторяемо четене;
  - `SERIALIZABLE` – последователно.

НИВА НА ИЗОЛАЦИЯ НА ТРАНСАКЦИИТЕ (2)

- `SET TRANSACTION ISOLATION LEVEL` **`READ UNCOMMITTED`**
  - допуска се, ако една заявка бъде изпълнена няколко пъти, тя всеки път да връща различни резултати, независимо от това дали едновременните трансакции са били потвърдени;
  - възможно е прочитане на данни, които логически никога не са съществували (т. нар. мръсно четене (*dirty read*));
  - съгласуваността на съвместното изпълнение е висока, но не се поддържа съгласуваност на данните.

НИВА НА ИЗОЛАЦИЯ НА ТРАНСАКЦИИТЕ – ПРИМЕР

Трансакция 1	Стойност на Stock	Трансакция 2
SELECT Stock FROM Products WHERE ProductID = 87	100	
UPDATE Products SET Stock = Stock - 10 WHERE ProductID = 87	90	SELECT Stock FROM Products WHERE ProductID = 87
ROLLBACK TRANSACTION	100	

НИВА НА ИЗОЛАЦИЯ НА ТРАНСАКЦИИТЕ (3)

- `SET TRANSACTION ISOLATION LEVEL` **`READ COMMITTED`**
  - допуска се, ако една заявка бъде изпълнена няколко пъти, тя всеки път да връща различни резултати, но само когато едновременните трансакции са били потвърдени;
  - конекцията, четяща данните, трябва да изчака, докато трансакцията по обновяване на данните приключи;
  - постига се по-висока съгласуваност на данните, тъй като непотвърдените данни не се виждат от други потребители, но съгласуваността на съвместното изпълнение е понижена.

НИВА НА ИЗОЛАЦИЯ НА ТРАНСАКЦИИТЕ – ПРИМЕР (2)

Трансакция 1	Стойност на Stock	Трансакция 2
	100	SELECT Stock FROM Products WHERE ProductID = 87
UPDATE Products SET Stock = Stock - 10 WHERE ProductID = 87 COMMIT TRANSACTION	90	SELECT Stock FROM Products WHERE ProductID = 87
	90	

НИВА НА ИЗОЛАЦИЯ НА ТРАНСАКЦИИТЕ (4)

- `SET TRANSACTION ISOLATION LEVEL` **`REPEATABLE READ`**
  - допуска се появяване на нови редове, които удовлетворяват условието на заявката, т.е. така наречените *фантоми* (*phantom insert*);
  - във всички останали случи се гарантира, че ако дадена заявка бъде изпълнена повторно, данните няма да са се променили.

НИВА НА ИЗОЛАЦИЯ НА ТРАНСАКЦИИТЕ – ПРИМЕР (3)

Трансакция 1	Трансакция 2
	SELECT SUM(quantity) FROM SaleDetails WHERE ProductID = 87
INSERT INTO SaleDetails (SaleID, ProductID, Quantity, Price) VALUES (23, 87, 15.500, 5.90) COMMIT TRANSACTION	
	SELECT SUM(quantity) FROM SaleDetails WHERE ProductID = 87

НИВА НА ИЗОЛАЦИЯ НА ТРАНСАКЦИИТЕ (5)

- SET TRANSACTION ISOLATION LEVEL **SERIALIZABLE**
  - гарантира се, че ако дадена заявка бъде изпълнена отново, междувременно няма да са добавени редове, т.е. няма да се появят фантоми;
  - изпълняването на трансакции по едно и също време е еквивалентно на изпълняването им последователно без значение от реда им;
  - съгласуваността на съвместното изпълнение е значително намалена, тъй като втората конекция трябва да изчака, докато първата приключи трансакцията напълно, но съгласуваността на данните е висока.

ВЛОЖЕНИ БЛОКОВЕ ТРАНСАКЦИИ

- Блокове, започващи с `BEGIN TRANSACTION` и завършващи с `COMMIT TRANSACTION` или `ROLLBACK TRANSACTION`, могат да бъдат влагани един в друг.
  - `ROLLBACK TRANSACTION` превърта назад всички нива на трансакцията, а не само нейния най-вътрешен блок;
  - `COMMIT TRANSACTION`
    - потвърждава всички нива на трансакцията, ако конструкцията съответства на най-външния блок;
    - в противен случай не прави нищо за потвърждаване на трансакцията.

ВЛОЖЕНИ БЛОКОВЕ ТРАНСАКЦИИ (2)

- `@@TRANCOUNT`
  - системна функция, връщаща дълбочината на вложените `BEGIN TRANSACTION` блокове:
    - има стойност 0, ако няма нито една активна трансакция;
    - `BEGIN TRANSACTION` увеличава с единица;
    - `COMMIT TRANSACTION` намалява с единица;
    - `ROLLBACK TRANSACTION` превърта назад цялата трансакция и установява `@@TRANCOUNT` на 0.

ВЛОЖЕНИ БЛОКОВЕ ТРАНСАКЦИИ – ПРИМЕР

```
SELECT @@TRANCOUNT -- Резултатът е 0.
BEGIN TRANSACTION A
    SELECT @@TRANCOUNT -- Резултатът е 1.
    -- изпълняват се конструкции за модифициране на данни
    BEGIN TRANSACTION B
        SELECT @@TRANCOUNT -- Резултатът е 2.
        -- изпълняват се конструкции за модифициране на данни
    ROLLBACK TRANSACTION B
    /* Неуспешна конструкция с грешка 6401: Cannot roll back B.
   No transaction or savepoint of that name was found. */
    SELECT @@TRANCOUNT -- Резултатът е 2.
    -- изпълняват се конструкции за модифициране на данни
ROLLBACK TRANSACTION A -- тази конструкция е успешна
SELECT @@TRANCOUNT -- Резултатът е 0.
```

ВЛОЖЕНИ БЛОКОВЕ ТРАНСАКЦИИ – ПРИМЕР (2)

```
SELECT @@TRANCOUNT -- Резултатът е 0.
BEGIN TRANSACTION A
    SELECT @@TRANCOUNT -- Резултатът е 1.
    -- изпълняват се конструкции за модифициране на данни
    BEGIN TRANSACTION B
        SELECT @@TRANCOUNT -- Резултатът е 2.
        -- изпълняват се конструкции за модифициране на данни
    ROLLBACK TRANSACTION -- прекратява трансакцията
    SELECT @@TRANCOUNT -- Резултатът е 0.
    -- изпълняват се конструкции за модифициране на данни
ROLLBACK TRANSACTION
/* Неуспешна конструкция с грешка 3903: The ROLLBACK
   TRANSACTION request has no corresponding BEGIN TRANSACTION.
   */
SELECT @@TRANCOUNT -- Резултатът е 0.
```

ВЛОЖЕНИ БЛОКОВЕ ТРАНСАКЦИИ – ПРИМЕР (3)

```
SELECT @@TRANCOUNT -- Резултатът е 0.
BEGIN TRANSACTION A
    SELECT @@TRANCOUNT -- Резултатът е 1.
    -- изпълняват се конструкции за модифициране на данни
    BEGIN TRANSACTION B
        SELECT @@TRANCOUNT -- Резултатът е 2.
        -- изпълняват се конструкции за модифициране на данни
    COMMIT TRANSACTION B
/* Не потвърждава трансакцията, но намалява стойността на
   @@TRANCOUNT. */
    SELECT @@TRANCOUNT -- Резултатът е 1.
    -- изпълняват се конструкции за модифициране на данни
COMMIT TRANSACTION A
-- Потвърждава промените и затваря трансакциите.
SELECT @@TRANCOUNT -- Резултатът е 0.
```

ТОЧКИ НА ЗАПИСВАНЕ В ТРАНСАКЦИИТЕ

- *Точка на записване* (*savepoint*)
  - точка, до която модификациите в трансакцията могат да бъдат отменени;
  - не е възможно да бъде използвана за потвърждаване на някакви промени в базата от данни;
  - `SAVE TRANSACTION` *`savepoint_name`* – конструкция за добавяне на точки на записване в SQL Server.

ТОЧКИ НА ЗАПИСВАНЕ В ТРАНСАКЦИИТЕ – ПРИМЕР

```
SELECT @@TRANCOUNT -- Резултатът е 0.
BEGIN TRANSACTION A
    SELECT @@TRANCOUNT -- Резултатът е 1.
    -- изпълняват се конструкции за модифициране на данни
    SAVE TRANSACTION B
        SELECT @@TRANCOUNT /* Резултатът е 1, тъй като
    функцията не се влияе от точката на записване. */
        -- изпълняват се конструкции за модифициране на данни
    ROLLBACK TRANSACTION B
    SELECT @@TRANCOUNT -- Резултатът е 1.
    /* Последният ROLLBACK засяга само точката на записване, а
    не трансакцията. */
    -- изпълняват се конструкции за модифициране на данни
ROLLBACK TRANSACTION A
/* успешно превъртане на трансакцията, отхвърляне на промените
    и затваряне на трансакциите */
SELECT @@TRANCOUNT -- Резултатът е 0.
```

ЗАДАЧИ

- *1.* Да се напише следния код за проверка за грешки в трансакциите, но с използване на конструкцията TRY...CATCH вместо оператора GOTO.

```
BEGIN TRANSACTION
  INSERT INTO Sales (CustomerID, EmployeeID, SaleDate)
  VALUES (1, 1, GETDATE())
  IF @@ERROR <> 0 GOTO error
  INSERT INTO SaleDetails (SaleID, ProductID, Price, Quantity)
  VALUES (@@IDENTITY, 18, 6, 10)
  IF @@ERROR <> 0 GOTO error
  UPDATE Products
     SET Stock = Stock - 10
  WHERE ProductID = 18
  IF @@ERROR <> 0 GOTO error
  COMMIT TRANSACTION
  GOTO Finish
  error:
  ROLLBACK TRANSACTION
  Finish:
  GO
  SELECT * FROM Products WHERE ProductID = 18
  GO
```

2. Да се определи резултата от изпълнението на следния код, при условие че съществува продукт с идентификатор 1, но не съществува продукт с идентификатор 2000:

```
DECLARE @Ident int
BEGIN TRANSACTION
   INSERT INTO Sales (CustomerID, EmployeeID, SaleDate)
   VALUES (1, 1, DATEADD(day, -1, GETDATE()))
   SET @Ident = @@IDENTITY
   SAVE TRANSACTION ProductA
   INSERT INTO SaleDetails (SaleID, ProductID, Price, Quantity)
   VALUES (@Ident, 2000, 50, 25)
   IF @@ERROR <> 0 ROLLBACK TRANSACTION ProductA
   SAVE TRANSACTION ProductB
   INSERT INTO SaleDetails (SaleID, ProductID, Price, Quantity)
   VALUES (@Ident, 1, 15.50, 2)
   IF @@ERROR <> 0 ROLLBACK TRANSACTION ProductB
COMMIT TRANSACTION
SELECT SUM(Price*Quantity) AS Total
FROM SaleDetails
WHERE SaleID = @Ident
GO
```