

Base de Données II,

Chap 4: Transactions and Triggers

Outline

- 1 Transactions
- 2 Triggers overview
- 3 Examples of triggers

Create transactions

Group of operations often must be treated as atomic unit.

```
1 begin <tran | transaction> <nameOfTransaction>
```

```
2 .
```

```
3 .
```

```
4 .
```

Create transactions

Group of operations often must be treated as atomic unit.

```
1 begin <tran | transaction> <nameOfTransaction>
```

```
2 .
```

```
3 .
```

```
4 .
```

```
1 begin tran tr
```

```
2 update Account set balance = balance - 100 where AccId=100
```

```
3 update Account set balance = balance + 100 where AccId=200
```

Transactions

- **commit** : used to save changes invoked by a transaction to the database

```
1 begin tran tr
2 update Account set balance = balance - 100 where AccId=100
3 update Account set balance = balance + 100 where AccId=200
4 commit <tran tr>
```

- **rollback** : used to undo transactions that have not already been saved to the database.

```
1 begin tran tr
2 update Account set balance = balance - 100 where AccId=100
3 update Account set balance = balance + 100 where AccId=200
4 rollback <tran tr>
```

Transactions

- **save transaction** : sets a save point within a transaction

```
1 begin transaction
2 print 'First Transaction'
3 insert into Article values (111, ...)
4
5 save transaction Savepoint1
6 print 'Second Transaction'
7
8 insert into Article values (222, ...)
9
10 rollback transaction Savepoint1
11 print 'Rollback'
12
13 commit transaction
14 print 'Complete'
15
16 — Output
17 First Transaction
18 Second Transaction
19 Rollback
20 Complete
```

Transaction: autocommit

- TSQL operates by default in the **autocommit** mode;
- If a statement completes **successfully**, it is **committed**;
- If it encounters any **error**, it is **rolled back**;
- **Remark 1:** the effects/changes made to the database by the statement in question cannot be rolled back.
- **Remark 2:** To disable the autocommit mode, use:
SET IMPLICIT_TRANSACTIONS OFF

ACID properties of Transactions

- **Atomicity**: either all of transactions are executed or all are rolled back
Account transfer debit and credit both succeed or fail
- **Consistency**: data is in a consistent state when a transaction starts and when it ends
When transferring funds from one account to another, the total value of funds in both the accounts is the same at the start and end of each transaction.
- **Isolation**: results not seen by other transactions until the transaction is complete
Account transfer debit and credit either both seen or neither is seen
- **Durability**: data is persistent even if hardware or software crashes: what is written on the disk is correct
Account balance is maintained

Transactions: @@TRANCOUNT

@@TRANCOUNT : returns the **number of begin transaction** statements that have occurred on the current connection.

```
1 print @@TRANCOUNT — 0
2 begin tran
3     print @@TRANCOUNT — 1
4     begin tran
5         print @@TRANCOUNT — 2
6     commit
7     print @@TRANCOUNT — 1
8 commit
9 print @@TRANCOUNT — 0
```

- begin transaction statement increments @@TRANCOUNT by 1;
- rollback transaction decrements @@TRANCOUNT to 0;
- rollback transaction savepoint_name does not affect @@TRANCOUNT;
- commit transaction decrements @@TRANCOUNT by 1.

Transactions: @@TRANCOUNT

```
1 begin transaction
2 print 'First Transaction: ' + convert(varchar, @@TRANCOUNT)
3 insert into Article values (111, ...)
4
5 save transaction Savepoint1
6 print 'Second Transaction: ' + convert(varchar, @@TRANCOUNT)
7
8 insert into Article values (222, ...)
9
10 rollback transaction Savepoint1
11 print 'Rollback: ' + convert(varchar, @@TRANCOUNT)
12
13 commit transaction
14 print 'Complete: ' + convert(varchar, @@TRANCOUNT)
```

1 Output

```
3 First Transaction: 1
4 Second Transaction: 1
5 Rollback: 1
6 Complete: 0
```

Transactions

```
1 begin tran
2 insert Order values (212, '2/2/2000)
3 if @@ERROR = 0
4     begin
5         insert OrderDetails values(212, '1231', 10)
6         if @@ERROR = 0
7             begin
8                 insert OrdersDetails values (212, '1233', 5)
9                 if @@ERROR = 0
10                    commit tran
11             else
12                rollback tran
13         end
14     else
15        rollback tran
16 end
17 else
18    rollback tran
```

Nested transactions

Be careful: **rollback inside nested transaction !!**

```
1 print @@TRANCOUNT — 0
2 begin transaction
3     print @@TRANCOUNT — 1
4
5     begin transaction
6         print @@TRANCOUNT — 2
7         rollback
8     print @@TRANCOUNT — 0
9
10 commit —error
11 print @@TRANCOUNT — 0
```

The ROLLBACK statement inside the inner transaction statement will clear the @@TRANCOUNT variable to 0 because all active transactions will be rolled back.

Outline

- 1 Transactions
- 2 Triggers overview
- 3 Examples of triggers

Triggers

- **Trigger**: procedure that starts **automatically** if **specified changes** occur to the DB;
- Three parts of a trigger:
 - **Event**: activates the trigger (e.g. insert, delete or update of the database)
 - **Condition**: tests whether the trigger should run (a Boolean statement or a query)
 - **Action**: what happens if the trigger runs (deletion, update, insertion)

When the event happens, the system will check the constraint, and if satisfied, will perform the action

Need for triggers

- Suppose that instead of allowing **negative account balances**, the bank deals with overdrafts by
 - setting the account balance to zero
 - creating a loan in the amount of the overdraft
 - giving this loan a loan number identical to the account number of the overdrawn account

- The condition for executing the trigger is an update to the account relation that results in a negative balance value.

Elements of triggers

- Timing of action execution: before, after or instead of triggering event
- The action can refer to both the **old and new state** of the database
- Update events may specify a particular column or set of columns
- The action can be performed either for
 - once for every tuple, or
 - once for all the tuples that are changed by the database operation.

Triggers: syntax

```
1 create trigger name_trigger
2 on <tableName | viewName>
3 <after | instead of> <List Of Events>
4 as
5 begin
6   <TSQL statement>
7 end
```

Possible events: **insert**, **update** or **delete**.

Triggers: types

- 1 **AFTER**: triggers are executed **after** the instruction takes place;
- 2 **INSTEAD OF**: triggers do not execute the triggering instruction, but **executes custom code** in place of it

Triggers: syntax

Inside the `<TSQL statements>`, special tables allow accessing the **newly created** and the **deleted** rows:

inserted and **deleted**

- In the case of **delete**:
deleted contains the removed rows
- In the case of **insert**:
inserted contains the inserted rows
- In the case of **update**:
deleted contains the deleted rows
inserted contains the inserted rows

Example

```
1 create trigger tr_1
2 on table table1
3 after insert, update, delete
4 as
5 begin
6     declare @i as int, @d as int
7     select @i = count(*) from inserted;
8     select @d= count(*) from deleted;

10    if @i=1 and @d=1
11        print 'UPDATE of at least one row identified'
12    else if @i=1 and @d=0
13        print 'INSERT of at least one row identified'
14    else if @i=0 and @d=1
15        print 'DELETE of at least one row identified'
16    else
17        print 'No rows affected'
18 end
```

Two possible actions

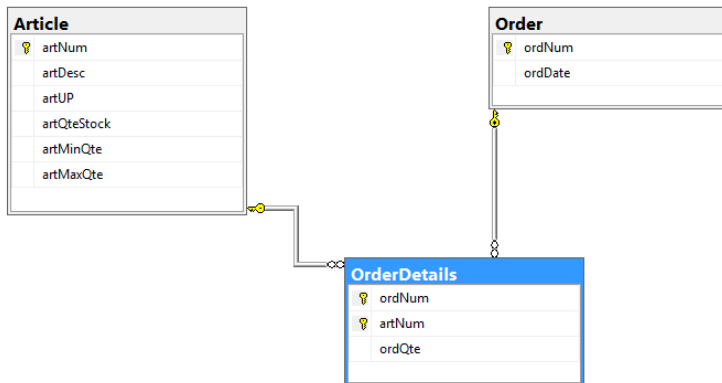
When a constraint violation is detected, two types of actions are possible:

- 1 **Abort**: the transaction is canceled with a rollback statement and an error is raised
- 2 **Repair**: an update statement modifies the database to make it consistent with the integrity constraints.

Outline

- 1 Transactions
- 2 Triggers overview
- 3 Examples of triggers

Database to use



Triggers: example 1

Example

The ordered quantity of an item cannot be greater than the maximum quantity

how can this happen?

Constraint violating events:

- When adding an new order
- When modifying the quantity of an existing order

Triggers: example 1

Example 1

The ordered quantity of an item cannot be greater than the maximum quantity

Triggers: example 1

Example 1

The ordered quantity of an item cannot be greater than the maximum quantity

Solution with after trigger and rollback

```
1 create trigger trQteOrder
2 On OrderDetails
3 after insert , update
4 as
5 begin
6 if exists(select * from inserted newE, Article a
7           where newE.artNum= a.artNum and
8             a.artMaxQte < newE.qte)
9     begin
10         print 'the ordered quantity should be less than the maximum quantity'
11         ROLLBACK
12     end
13 end
```

Triggers: problem with solution 1

Example 1: problem

Problem when **using ROLLBACK statement within the trigger!**

The following error occurs:

The transaction ended in the trigger. The batch has been aborted.

Triggers: problem with solution 1

Example 1: problem

Problem when **using ROLLBACK statement within the trigger!**

The following error occurs:

The transaction ended in the trigger. The batch has been aborted.

Example 1: why

????

Triggers: problem with solution 1

Example 1: problem

Problem when **using ROLLBACK statement within the trigger!**

The following error occurs:

The transaction ended in the trigger. The batch has been aborted.

Example 1: why

- When a trigger executes, an **implicit transaction** is started
- Using ROLLBACK TRANSACTION inside of triggers, will **reset @@TRANCOUNT to 0**
- Then a trigger completes execution with an **@@TRANCOUNT = 0**, an error occurs and the batch is terminated.

Triggers: example 1

Example 1

The ordered quantity of an item cannot be greater than the maximum quantity

Triggers: example 1

Example 1

The ordered quantity of an item cannot be greater than the maximum quantity

Solution with after trigger without rollback

```
1 create trigger trQteOrder
2 On OrderDetails
3 after insert , update
4 as
5 begin
6 if exists (select * from inserted newE, Article a
7           where newE.artNum= a.artNum and
8             a.artMaxQte < newE.qte)
9     begin
10         print 'the ordered quantity should be less than the maximum quantity '
11         delete from OrderDetails where artNum in (select artNum from inserted)
12         and OrdNum in (select ordNum from inserted)
13     end
14 end
```

Triggers: example 1

Example 1

The ordered quantity of an item cannot be greater than the maximum quantity

Triggers: example 1

Example 1

The ordered quantity of an item cannot be greater than the maximum quantity

Solution with instead of trigger

```
1 create trigger trQteOrder
2 On OrderDetails
3 instead of insert , update
4 as
5 begin
6 if exists(select * from inserted newE, Article a
7           where newE.artNum= a.artNum and
8           a.artMaxQte > newE.qte)
9     begin
10       insert into OrdersDetails select * from inserted
11     end
12 end
```

Triggers: example 2

Example 2

Modification of orders is not allowed.

```
1 Create trigger Tr_Prohibit_Modif_Com
2 On OrdersDetails
3 after update
4 as
5 rollback
```

Update statement

Update statement

Update statement is used to react to the change only if **certain columns were modified**.

Example

Reject an attempt to modify a primary key value.

Triggers: example 3

Example 3

Prohibiting changing the order number and checking whether the entered date for the order date is greater than or equal to today's date

```
1 create trigger tr_Prohibit_modif
2 on Order
3 after update
4 as
5     if update(ordNum)
6     begin
7         print 'ordNum cannot be modified'; rollback
8     end
9     if update(ordDate)
10    begin
11        if((select count(ordDate) from inserted where datediff(day,ordDate,
12            getdate()))>0) <> 0)
13        begin
14            print 'ordDate cannot be greater than the current date'; rollback
15        end
```

Triggers: example 4

Example 4

Preventing deletion of an order without removing its details

```
1 create trigger Tr_Prevent_Remove
2 On Order
3 instead of delete
4 as
5 begin
6     declare @a int
7     select @a = count(artNum) from OrdrDetails, deleted
8         where OrderDetails.ordNum =deleted.ordNum)
9     if @a=0
10         delete from Order where ordNum=(select ordNum from deleted)
11     else
12         print 'Cannot not remove the order since it has related order details'
13 end
```

Triggers: example 5

Example 5

Tracking modifications to a table.

```
1 create table audit_trail
2 (
3 source_table      char(32),
4 user_id           char(8),
5 action_code       char(1),
6 action_time       datetime,
7 old_value         varchar(20),
8 new_value         varchar(20),
9 field            varchar(10)
10 )
11
12 create table tableA
13 (
14 id int,
15 name char(10)
16 )
```

Triggers: example 5

```
1 create trigger trigger_audit
2 on tableA
3 after update
4 as
5 begin
6     declare @old char(10), @new char(10), @f char(20)
7     if update(id)
8     begin
9         set @f='id';
10        select @new = id from inserted;
11        select @old = id from deleted;
12    end
13    else if update(name)
14    begin
15        set @f='name';
16        select @new = name from inserted;
17        select @old = name from deleted;
18    end
19    else
20    begin
21        set @f='';set @new = '';set @old = '';
22    end
23    insert into audit_trail values('tableA',USER_NAME(),'u',GETDATE(),@old,@new, @f)
24 end
```

Disable/Enable triggers

- `drop trigger trigger_name`
- `disable trigger triggerName on tableName`
- `enable trigger triggerName on tableName`
- `disable trigger all on <tableName | databaseName>`
- `enable trigger all on <tableName | databaseName>`