Lab 3 Helper Transactions

Ioana Ciuciu,
ioana.ciuciu@ubbcluj.ro

Info about the lab ☺

- Lab requirements available here:
  - www.cs.ubbcluj.ro/~sabina
- 2 weeks delay = 1 point penalty
- Max 2 lab assignments / lab
- Final lab grade: ((GradeLab1-PenaltyLab1) + (GradeLab2-PenaltyLab2) + (GradeLab3-PenaltyLab3))/3
- No lab delivery during weeks 13, 14 and during the exams (sesiune)
- During retake session (restante): max 2 labs, with a penalty of 35%, only if the practical exam is retaken (except when the student has 10 p. for the practical exam)
- Attendance: 6 labs out of 7 (https://www.cs.ubbcluj.ro/wp-content/uploads/Hotarare-CDI-29.04.2020.pdf)
- Practical exam: weeks 13, 14 (in order to promote, a grade >= 5 is needed)

Prerequisites

- Visual Studio – installed

- For Linux users
  - Virtual machine

  - ! For the practical exam, an app using Windows Forms will be required!

Transactions

Transactions

- A **transaction** is a single unit of work

- If a transaction is **successful**
  - All of the data modifications made during the transaction are **committed** and become a **permanent part of the database**

- If a transaction encounters **errors**
  - The transaction must be **canceled** or **rolled back** and all of the data modifications are erased

Transaction modes in SQL Server

Transaction mode	Remarks
Autocommit transactions	- each individual statement is a transaction
Explicit transactions	- each transaction is explicitly started with the BEGIN TRANSACTION statement - each transaction is ended with a COMMIT or a ROLLBACK statement
Implicit transactions	- a new transaction is implicitly started when the prior transaction completes BUT - each transaction is explicitly completed with a COMMIT or a ROLLBACK statement
Batch-scoped transactions	- only applicable to multiple active result sets (MARS)

Transactions

- BEGIN TRANSACTION
  - Marks the starting point of an explicit, local transaction
  - Explicit transactions start with the BEGIN TRANSACTION statement and end with the COMMIT or ROLLBACK statement
- Syntax

```
BEGIN { TRAN | TRANSACTION }
    [ { transaction_name | @tran_name_variable }
      [ WITH MARK [ 'description' ] ]
    ]
[ ; ]
```

Transactions

- BEGIN TRANSACTION
- Examples
  - Using an explicit transaction:

```
BEGIN TRANSACTION;
DELETE FROM HumanResources.JobCandidate
        WHERE JobCandidateID = 13;
COMMIT;
```

  - Rolling back a transaction:

```
CREATE TABLE ValueTable (id INT);
BEGIN TRANSACTION;
INSERT INTO ValueTable VALUES(1);
 INSERT INTO ValueTable VALUES(2);
ROLLBACK;
```

Transactions

- BEGIN TRANSACTION
- Examples
  - Naming a transaction

```
DECLARE @TranName VARCHAR(20);
SELECT @TranName = 'MyTransaction';

BEGIN TRANSACTION @TranName;
USE AdventureWorks2012;
DELETE FROM AdventureWorks2012.HumanResources.JobCandidate
    WHERE JobCandidateID = 13;

COMMIT TRANSACTION @TranName;
GO
```

Transactions

- COMMIT TRANSACTION
  - Marks the end of a successful implicit or explicit transaction
  - If @@TRANCOUNT is 1
    - COMMIT TRANSACTION makes all data modifications since the start of the transaction a permanent part of the database, frees the transaction's resources, and decrements @@TRANCOUNT to 0
  - If @@TRANCOUNT is greater than 1,
    - COMMIT TRANSACTION decrements @@TRANCOUNT only by 1 and the transaction stays active
- Syntax

```
COMMIT [ { TRAN | TRANSACTION } [ transaction_name |
@tran_name_variable ] ] [ WITH ( DELAYED_DURABILITY = { OFF | ON } ) ]
[ ; ]
```

Transactions

- COMMIT TRANSACTION
- Example

```
BEGIN TRANSACTION;
 DELETE FROM HumanResources.JobCandidate
       WHERE JobCandidateID = 13;
COMMIT TRANSACTION;
```

Transactions

- ROLLBACK TRANSACTION
  - Rolls back an explicit or implicit transaction to the beginning of the transaction, or to a savepoint inside the transaction
  - Also frees resources held by the transaction
  - Does not include changes made to local variables or table variables (not erased by this statement)

- Syntax

```
ROLLBACK { TRAN | TRANSACTION }
[ transaction_name | @tran_name_variable
| savepoint_name | @savepoint_variable ]
[ ; ]
```

Transactions

- ROLLBACK TRANSACTION

- Example

```
USE tempdb;
GO
CREATE TABLE ValueTable ([value] INT);
GO

DECLARE @TransactionName VARCHAR(20) = 'Transaction1';

BEGIN TRAN @TransactionName
        INSERT INTO ValueTable VALUES(1), (2);
ROLLBACK TRAN @TransactionName;

INSERT INTO ValueTable VALUES(3),(4);

SELECT [value] FROM ValueTable;

DROP TABLE ValueTable;
```

Here is the result set:

```
value
-----
3
4
```

Transactions

- SAVE TRANSACTION
  - Sets a savepoint within a transaction.
- Syntax

```
SAVE { TRAN | TRANSACTION } { savepoint_name | @savepoint_variable }
[ ; ]
```

Transactions – SAVE TRANSACTION Example

```
USE [Stagiu]
GO

CREATE PROCEDURE
[dbo].[insertIntoManyToMany_recover]
        --list of parameters
AS
BEGIN
SET NOCOUNT ON;
 DECLARE @errors VARCHAR(50) = '',
        @rollBackPointStudents int,
        @rollBackPointCourses int,
        @rollBackPointStudentsCourses int
```

```
BEGIN TRAN
BEGIN TRY
    --do the necessary parameter validations

    --INSERT INTO Students ...
    SAVE TRANSACTION InsertIntoStudents


    --INSERT INTO Courses ...
    SAVE TRANSACTION InsertIntoCourses

    --INSERT INTO StudentsCourses..

    …
```

@@TRANCOUNT

- Returns the number of BEGIN TRANSACTION statements that have occurred on the current connection

- BEGIN TRANSACTION increments @@TRANCOUNT by 1

- ROLLBACK TRANSACTION decrements @@TRANCOUNT to 0
  - except for ROLLBACK TRANSACTION *savepoint_name*, which does not affect @@TRANCOUNT

- COMMIT TRANSACTION decrement @@TRANCOUNT by 1

@@TRANCOUNT

- Examples

```
PRINT @@TRANCOUNT
-- The BEGIN TRAN statement will increment the
-- transaction count by 1.
BEGIN TRAN
    PRINT @@TRANCOUNT
    BEGIN TRAN
        PRINT @@TRANCOUNT
-- The COMMIT statement will decrement the transaction
count by 1.
    COMMIT
    PRINT @@TRANCOUNT
COMMIT
PRINT @@TRANCOUNT

--Results
--0
--1
--2
--1
--0
```

```
PRINT @@TRANCOUNT
-- The BEGIN TRAN statement will increment the
-- transaction count by 1.
BEGIN TRAN
    PRINT @@TRANCOUNT
    BEGIN TRAN
        PRINT @@TRANCOUNT
-- The ROLLBACK statement will clear the
@@TRANCOUNT variable
-- to 0 because all active transactions will be rolled back
ROLLBACK
PRINT @@TRANCOUNT

--Results
--0
--1
--2
--0
```

Transaction isolation levels

- Controls the locking and row versioning behavior of Transact-SQL statements issued by a connection to SQL Server

- Syntax

```
SET TRANSACTION ISOLATION
LEVEL
 { READ UNCOMMITTED
 | READ COMMITTED
 | REPEATABLE READ
 | SNAPSHOT
 | SERIALIZABLE
 }
```

- Remark
  - takes effect at execute or run time, and not at parse time

Transaction isolation level

- READ UNCOMMITED
  - The least restrictive of the isolation levels
  - A transaction can read rows that have been modified by other transactions but not yet committed
  - Allows *dirty reads*

Transaction isolation level

- READ COMMITED
  - The default isolation level
  - A transaction cannot read data that has been modified but not committed by other ongoing transactions
  - Allows *unrepeatable reads*

Transaction isolation levels

- REPEATABLE READ
  - Transactions cannot read data that has been modified but not yet committed by other transactions
  - and
  - No other transactions can modify data that has been read by the current transaction until the current transaction completes
  - Holds S locks and X locks until the end of the transaction
    - Concurrency is lower than the default READ COMMITTED isolation level
    - Use this option only when necessary
  - Doesn't allow *dirty reads*, *unrepeatable reads*
  - *Phantom reads* can occur

Transaction isolation levels

- SNAPSHOT
  - Data read by any statement in a transaction will be the transactionally consistent version of the data that existed at the start of the transaction (working on a snapshot of the data)

Transaction isolation levels

- SERIALIZABLE
  - Highest (most restrictive) isolation level
  - Statements cannot read data that has been modified but not yet committed by other transactions
  - No other transactions can modify data that has been read by the current transaction until the current transaction completes
  - Other transactions cannot insert new rows with key values that would fall in the range of keys read by any statements in the current transaction until the current transaction completes

  - Holds locks (including key range locks) during the entire transaction
  - Because concurrency is lower, use this option only when necessary
  - Doesn't allow *dirty reads*, *unrepeatable reads*, *phantom reads*

Transaction isolation levels

- Reminder (see Seminar 3)

concurrency probl. / isolation level	Chaos	Read Uncommitted	Read Committed	Repeatable Read	Serializable
Lost Updates?	Yes	No	No	No	No
Dirty Reads?	Yes	Yes	No	No	No
Unrepeatable Reads?	Yes	Yes	Yes	No	No
Phantoms?	Yes	Yes	Yes	Yes	No

- Remark
  - *Lost updates* occur due to the update of the same record by two different transactions at the same time

References

- lecture / seminar notes
- https://docs.microsoft.com/en-us/sql/t-sql/statements/set-transaction-isolation-level-transact-sql
- https://docs.microsoft.com/en-us/sql/t-sql/language-elements/transactions-transact-sql