

Multiversionarea

Seminar 4

Monitorizarea blocărilor

SQL Server Extended Events

- Este un sistem de monitorizare a performanței care utilizează resurse minime
- Oferă două interfețe grafice pentru utilizatori care pot fi folosite pentru a crea, afișa, modifica și analiza datele sesiunii: *New Session* și *New Session Wizard*

Procedura stocată sistem *sp_lock* (această funcționalitate se află în maintenance mode)

- Oferă informații despre blocări (locks)

Dynamic management view-ul sistem *sys.dm_tran_locks*

- Returnează informații despre resursele *lock manager* active în prezent
- Fiecare înregistrare reprezintă o cerere activă în prezent către *lock manager* pentru o blocare ce a fost acordată sau așteaptă să fie acordată

Monitorizarea blocărilor

Dynamic management view-ul sistem *sys.dm_tran_active_transactions*

- Returnează informații despre tranzacții pentru instanța SQL Server

Tipuri de resurse:

- RID – identificator de înregistrare
- Key – interval de chei într-un index (blocări key range)
- Pagină – pagină de 8 KB din tabele/indecși
- HoBT – Heap or balanced tree
- Tabel, fișier, bază de date
- Metadata
- Aplicație

Query Governor

SET QUERY_GOVERNOR_COST_LIMIT

- Suprascrie valoarea actuală configurată pentru **query governor cost limit** (pentru conexiunea curentă)
- *Query cost* se referă la timpul estimat (în secunde) necesar execuției unei interogări pe o anumită configurație hardware
- *Query optimizer*-ul estimează numărul de secunde necesare pentru execuția unei interogări și în cazul în care valoarea estimată este mai mare decât limita stabilită, interogarea nu va fi executată
- Valoarea implicită este 0 (această valoare permite execuția tuturor interogărilor)

DBCC LOG

- Returnează informații despre logul de tranzații
- Sintaxa:

DBCC LOG (<dbname>,<output id>)

<output id> este o valoare cuprinsă între 0 și 4 (specifică nivelul de detaliere)

<dbname> este numele bazei de date

- Exemplu:

DBCC LOG (SGBDIR,2)

Niveluri de izolare în SQL Server

- **READ UNCOMMITTED**: fără blocări la citire
- **READ COMMITTED**: este nivelul de izolare implicit și menține blocările pe durata execuției instrucțiunii (elimină dirty reads)
- **REPEATABLE READ**: menține blocările pe durata tranzacției (elimină unrepeatable reads)
- **SERIALIZABLE**: menține blocările și blocările key range pe durata întregii tranzacții (elimină phantom reads)
- **SNAPSHOT**: lucrează pe un snapshot al datelor
- Sintaxa SQL:

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

Multiversionarea

- Într-un sistem de gestiune a bazelor de date cu multiversionare, fiecare scriere a unui item x produce o nouă versiune (copie) a lui x
- La fiecare citire a lui x , sistemul de gestiune a bazelor de date selectează una din versiunile lui x pentru citire
- Deoarece nu există suprascrieri între operațiile de scriere, iar operațiile de citire pot citi orice versiune, sistemul de gestiune a bazelor de date are mai multă flexibilitate în controlul ordinii scrierilor și citirilor

Versionarea la nivel de înregistrare (RLV)

- A fost introdusă în SQL Server 2005
- Este utilă când este nevoie de date comise, dar nu neapărat de cea mai nouă versiune
- **Read Committed Snapshot** Isolation și **Full Snapshot** Isolation
 - Cititorul nu blochează niciodată
 - Cititorul primește valoarea comisă anterior
- Toate versiunile mai vechi sunt stocate în baza de date *tempdb*
- Pe baza versiunilor mai vechi se poate construi un **snapshot** al datelor

Read Committed Snapshot Isolation

Toate operațiunile văd înregistrările comise la începerea execuției lor =>

- Snapshot al datelor la nivel de comandă
- Citire consistentă la nivel de comandă
- Disponibil la utilizarea nivelului de izolare READ COMMITTED (default) cu opțiunea **READ_COMMITTED_SNAPSHOT** setată pe ON
- Setarea opțiunii READ_COMMITTED_SNAPSHOT pe ON:

```
ALTER DATABASE database_name
```

```
SET READ_COMMITTED_SNAPSHOT ON;
```

Full Snapshot Isolation

Toate operațiile văd înregistrările comise la începerea execuției tranzacției =>

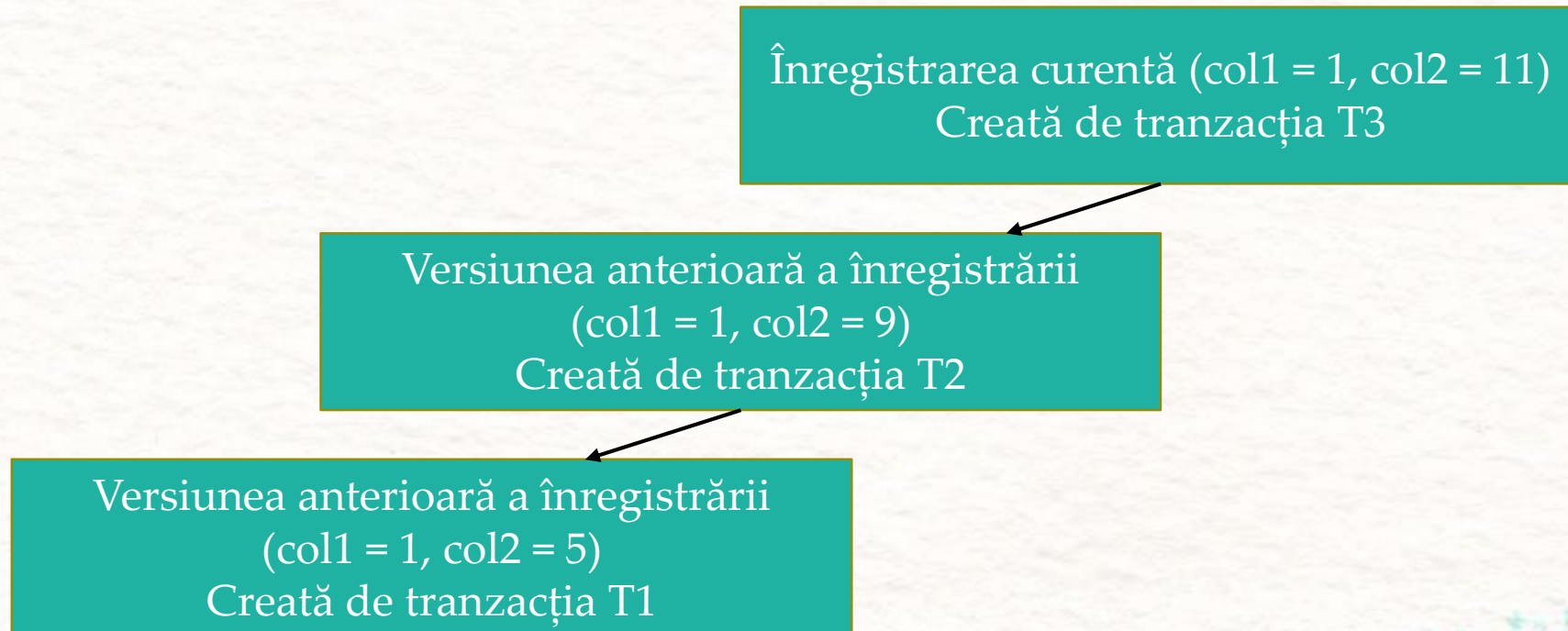
- Snapshot al datelor la nivel de tranzacție
- Citire consistentă la nivel de tranzacție
- Nivelul de izolare SNAPSHOT poate fi folosit dacă opțiunea **ALLOW_SNAPSHOT_ISOLATION** este setată pe ON
- Setarea opțiunii ALLOW_SNAPSHOT_ISOLATION pe ON:

```
ALTER DATABASE database_name
```

```
SET ALLOW_SNAPSHOT_ISOLATION ON;
```


Versionarea la nivel de înregistrare

- O înregistrare conține TSN (transaction sequence number)
- Toate versiunile sunt stocate într-o listă înlănțuită



Versionarea la nivel de înregistrare

Avantaje

- Nivelul de concurență este mai ridicat
- Mărește performanța triggerelor și a creării indecșilor

Dezavantaje

- Cerințe de gestiune suplimentare pentru monitorizarea utilizării *tempdb*
- Performanța mai scăzută a operațiilor de modificare
- Viteza cititorilor este afectată de costul traversării listelor înlănțuite
- Se rezolvă conflictul dintre cititori și scriitori, dar tot nu sunt permisi scriitorii simultani

Triggere și versionarea la nivel de înregistrare

Triggerele au acces la două pseudo-tabele:

- Tabelul *deleted* care conține înregistrări șterse sau versiuni vechi ale înregistrărilor modificate
- Tabelul *inserted* care conține înregistrări inserate sau versiuni noi ale înregistrărilor modificate

Înainte de SQL Server 2005:

- Tabelul *deleted* era creat pe baza logului de tranzacții (afectează performanța)

Prin utilizarea versionării la nivel de înregistrare:

- Pentru tabele cu triggere relevante sunt versionate schimbările

Crearea indecșilor și versionarea la nivel de înregistrare

În versiunile anterioare de SQL Server, crearea sau reconstruirea indecșilor însemna:

- Tabel blocat exclusiv și date complet inaccesibile (index clustered)
- Tabel disponibil doar pentru citire și index indisponibil (index nonclustered)

Cu versionare la nivel de înregistrare:

- Indecșii sunt creați și reconstruiți online
- Toate cererile sunt procesate pe date versionate

Niveluri de izolare și anomalii de concurență

Nivel de izolare	Dirty Reads	Unrepeatable Reads	Phantom Reads	Update conflict	Model de concurență
Read Uncommitted	Da	Da	Da	Nu	Pesimist
Read Committed Locking	Nu	Da	Da	Nu	Pesimist
Read Committed Snapshot	Nu	Da	Da	Nu	Optimist
Repeatable Read	Nu	Nu	Da	Nu	Pesimist
Versionare la nivel de înregistrare	Nu	Nu	Nu	Da	Optimist
Serializable	Nu	Nu	Nu	Nu	Pesimist

Instrucțiunea MERGE

- Instrucțiunea MERGE oferă posibilitatea de a compara înregistrări dintr-un tabel sursă cu înregistrări dintr-un tabel destinație
- Comenzile INSERT, UPDATE și DELETE pot fi executate pe baza rezultatului acestei comparații
- Tabelul *Filme*:

Cod_film	Titlu	An	Durata
1	IT	2017	NULL
2	IT	NULL	NULL
3	IT	NULL	135

Instrucțiunea MERGE – Sintaxa generală

MERGE Definiție_Tabel AS Destinație

USING (Tabel_sursă) AS Sursă

ON (Termeni de căutare)

WHEN MATCHED THEN UPDATE SET sau DELETE

WHEN NOT MATCHED [BY TARGET] THEN INSERT

WHEN NOT MATCHED BY SOURCE THEN UPDATE SET sau DELETE

Instrucțiunea MERGE – Exemplu

MERGE Filme

```
USING (SELECT MAX(Cod_film) Cod_film, Titlu, MAX(An) An,  
MAX(Durata) Durata FROM Filme GROUP BY Titlu) MergeFilme  
ON Filme.Cod_film=MergeFilme.Cod_film  
WHEN MATCHED THEN UPDATE SET Filme.Titlu=MergeFilme.Titlu,  
Filme.An=MergeFilme.An, Filme.Durata=MergeFilme.Durata  
WHEN NOT MATCHED BY SOURCE THEN DELETE;
```


Instrucțiunea MERGE – Exemplu

- Rezultat:

Cod_film	Titlu	An	Durata
3	IT	2017	135

PIVOT/UNPIVOT

- Schimbă o expresie *table-valued* într-un alt tabel
- PIVOT rotește o expresie *table-valued* transformând valorile unice dintr-o coloană din expresie în mai multe coloane în output și calculează valori agregate acolo unde este necesar, pe valorile oricăror coloane rămase care sunt dorite în rezultatul final
- UNPIVOT realizează operația opusă, rotind coloanele dintr-o expresie *table-valued* în valori de coloană

Sintaxa PIVOT

```
SELECT <non-pivoted column>, [first pivoted column] AS <column  
name>, [second pivoted column] AS <column name>, ... [last pivoted  
column] AS <column name>
```

```
FROM (<SELECT query that produces the data>) AS <source query>
```

```
PIVOT
```

```
(<aggregation function>(<column being aggregated>) FOR
```

```
[<column that contains values that become column headers>]
```

```
IN ( [first pivoted column], [second pivoted column], ... [last  
pivoted column]))
```

```
AS <alias for the pivot table> <optional ORDER BY clause>;
```

PIVOT - Exemplan

- Tabelul *Students*:

student_id	name	city
1	Jack	New York
2	Jane	Los Angeles
3	Rose	New York
4	Jill	New York
5	Anne	Los Angeles
6	John	London

PIVOT - Exemplu

- Pentru a afișa numărul de studenți pentru fiecare oraș, vom executa următoarea interogare:

```
SELECT city, COUNT(student_id) AS [number of students]  
FROM Students GROUP BY city;
```

- Obținem următorul rezultat:

city	number of students
London	1
Los Angeles	2
New York	3

PIVOT - Exemplu

- Vom transforma valorile unice din coloana *city* în coloane ale tabelului output:

```
SELECT 'number of students' AS 'city', [New York],  
[Los Angeles], [London] FROM (SELECT city, student_id FROM  
Students) AS SourceTable PIVOT (COUNT(student_id) FOR city  
IN ([Los Angeles], [London], [New York])) AS PivotTable;
```

- Rezultat:

city	New York	Los Angeles	London
number of students	3	2	1

Probleme de concurență – Exemplu (RLV)

- În SQL Server, vom crea o nouă bază de date numită 'SGBDPC'
- După ce baza de date a fost creată, vom seta opțiunile `READ_COMMITTED_SNAPSHOT` și `ALLOW_SNAPSHOT_ISOLATION` pe ON (pentru a activa row versioning):

```
ALTER DATABASE SGBDPC
```

```
SET READ_COMMITTED_SNAPSHOT ON;
```

```
ALTER DATABASE SGBDPC
```

```
SET ALLOW_SNAPSHOT_ISOLATION ON;
```

- După aceea, vom crea un nou tabel numit *Movies*:

Probleme de concurență – Exemplu (RLV)

```
CREATE TABLE Movies  
(  
    movie_id INT PRIMARY KEY IDENTITY,  
    title VARCHAR(100),  
    year INT  
);
```

- Vom insera două înregistrări:

```
INSERT INTO Movies (title, year) VALUES  
('IT', 2017), ('Red Sparrow', 2018);
```

- Vom crea un tabel numit *Actors*:

Probleme de concurență – Exemplu (RLV)

```
CREATE TABLE Actors  
(  
    actor_id INT PRIMARY KEY IDENTITY,  
    firstname VARCHAR(100),  
    lastname VARCHAR(100)  
);
```

- Vom insera o înregistrare:

```
INSERT INTO Actors (firstname, lastname)  
VALUES ('Bill', 'Skarsgard');
```

Probleme de concurență – Exemplu (RLV)

- Dacă executăm următoarele interogări, putem vedea următoarele result set-uri:

```
SELECT * FROM Movies;
```

	movie_id	title	year
1	1	IT	2017
2	2	Red Sparrow	2018

```
SELECT * FROM Actors;
```

	actor_id	firstname	lastname
1	1	Bill	Skarsgard

- Exemplu unrepeatable reads:
- Deschidem două ferestre noi de interogare (două conexiuni)
- În prima fereastră de interogare punem următoarea tranzacție:

Probleme de concurență – Exemplu (RLV)

```
BEGIN TRAN;  
  
SELECT * FROM Movies;  
  
WAITFOR DELAY '00:00:06';  
  
SELECT * FROM Movies;  
  
COMMIT TRAN;
```

- În a doua fereastră de interogare punem următoarea tranzacție:

```
BEGIN TRAN;  
  
WAITFOR DELAY '00:00:03';  
  
UPDATE Movies SET year=2019 WHERE title='IT';  
  
COMMIT TRAN;
```

- Începem execuția primei și a celei de a doua tranzacții (în această ordine)

Probleme de concurență – Exemplu (RLV)

- În prima fereastră de interogare putem vedea următorul rezultat:

	movie_id	title	year
1	1	IT	2017
2	2	Red Sparrow	2018

	movie_id	title	year
1	1	IT	2019
2	2	Red Sparrow	2018

- După cum putem observa, aceeași interogare executată de două ori în aceeași tranzacție a returnat două valori diferite pentru aceeași înregistrare (unrepeatable reads)

Probleme de concurență – Exemplu (RLV)

- Exemplu phantom reads:
- Deschidem două ferestre noi de interogare (două conexiuni)
- În prima fereastră de interogare punem următoarea tranzacție:

```
BEGIN TRAN;
```

```
SELECT * FROM Movies WHERE year BETWEEN 2017 AND 2020;
```

```
WAITFOR DELAY '00:00:06';
```

```
SELECT * FROM Movies WHERE year BETWEEN 2017 AND 2020;
```

```
COMMIT TRAN;
```

Probleme de concurență – Exemplu (RLV)

- În a doua fereastră de interogare punem următoarea tranzacție:

```
BEGIN TRAN;
```

```
WAITFOR DELAY '00:00:03';
```

```
INSERT INTO Movies (title, year) VALUES  
('Black Panther', 2018);
```

```
COMMIT TRAN;
```

- Începem execuția primei și a celei de a doua tranzacții (în această ordine)

Probleme de concurență – Exemplu (RLV)

- În prima fereastră de interogare putem vedea următorul rezultat:

	movie_id	title	year
1	1	IT	2019
2	2	Red Sparrow	2018
	movie_id	title	year
1	1	IT	2019
2	2	Red Sparrow	2018
3	3	Black Panther	2018

- În aceeași tranzacție, o interogare care specifică un interval de valori în clauza WHERE a fost executată de două ori și numărul de înregistrări incluse în cel de-al doilea result set este mai mare decât numărul de înregistrări incluse în primul result set (phantom reads)

Probleme de concurență – Exemplu (RLV)

- Exemplu deadlock:
- Deschidem două ferestre noi de interogare (două conexiuni)
- În prima fereastră de interogare punem următoarea tranzacție:

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
```

```
BEGIN TRAN;
```

```
UPDATE Movies SET year=2017 WHERE title='IT';
```

```
WAITFOR DELAY '00:00:05';
```

```
UPDATE Actors SET firstname='Alexander' WHERE  
lastname='Skarsgard';
```

```
COMMIT TRAN;
```


Probleme de concurență – Exemplu (RLV)

- În a doua fereastră de interogare punem următoarea tranzacție:

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
```

```
BEGIN TRAN;
```

```
UPDATE Actors SET firstname='Alex' WHERE  
lastname='Skarsgard';
```

```
WAITFOR DELAY '00:00:05';
```

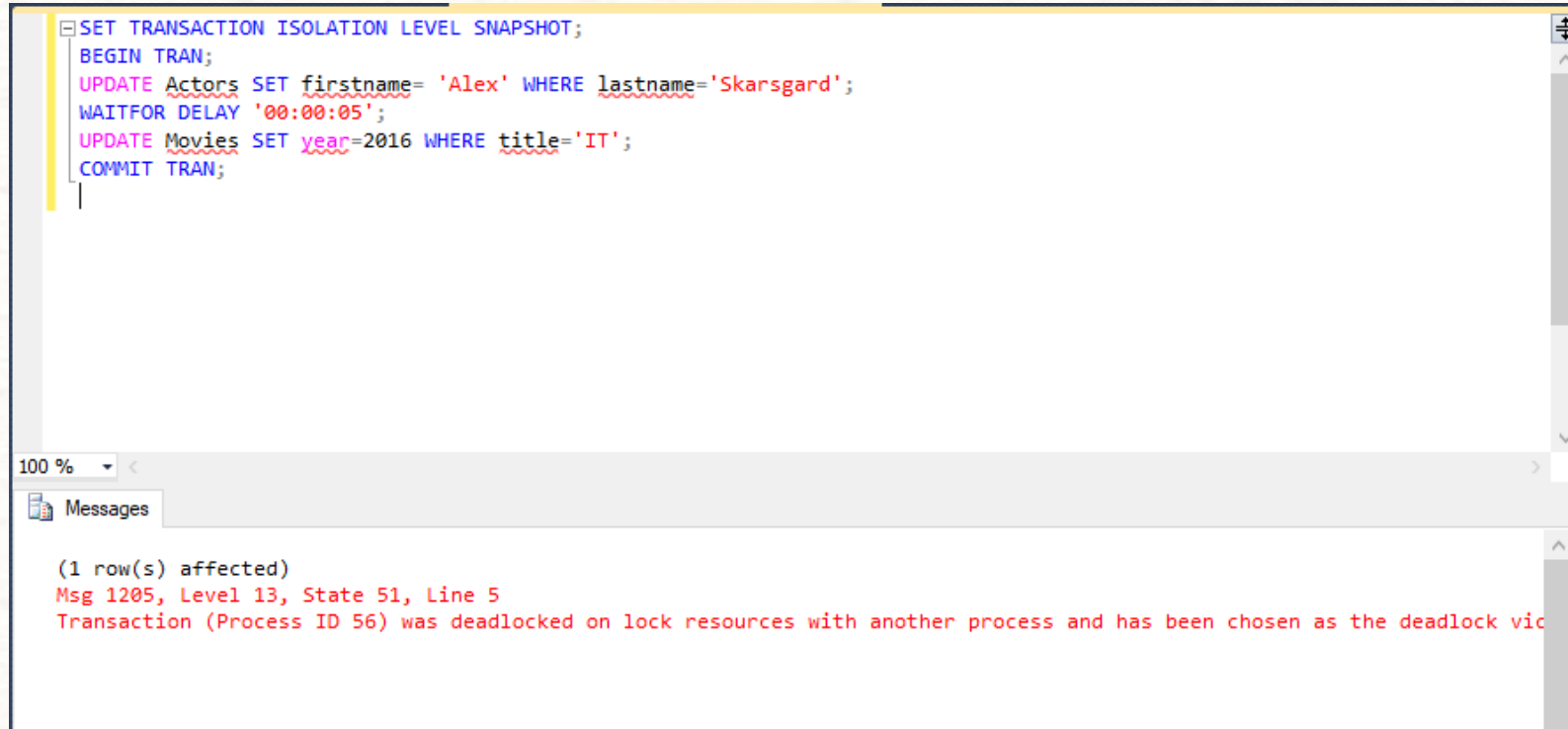
```
UPDATE Movies SET year=2016 WHERE title='IT';
```

```
COMMIT TRAN;
```

- Începem execuția primei și a celei de a doua tranzacții (în această ordine)

Probleme de concurență – Exemplu (RLV)

- A avut loc un deadlock, iar a doua tranzacție a fost aleasă drept victimă a deadlock-ului și face un rollback:



```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;  
BEGIN TRAN;  
UPDATE Actors SET firstname= 'Alex' WHERE lastname='Skarsgard';  
WAITFOR DELAY '00:00:05';  
UPDATE Movies SET year=2016 WHERE title='IT';  
COMMIT TRAN;
```

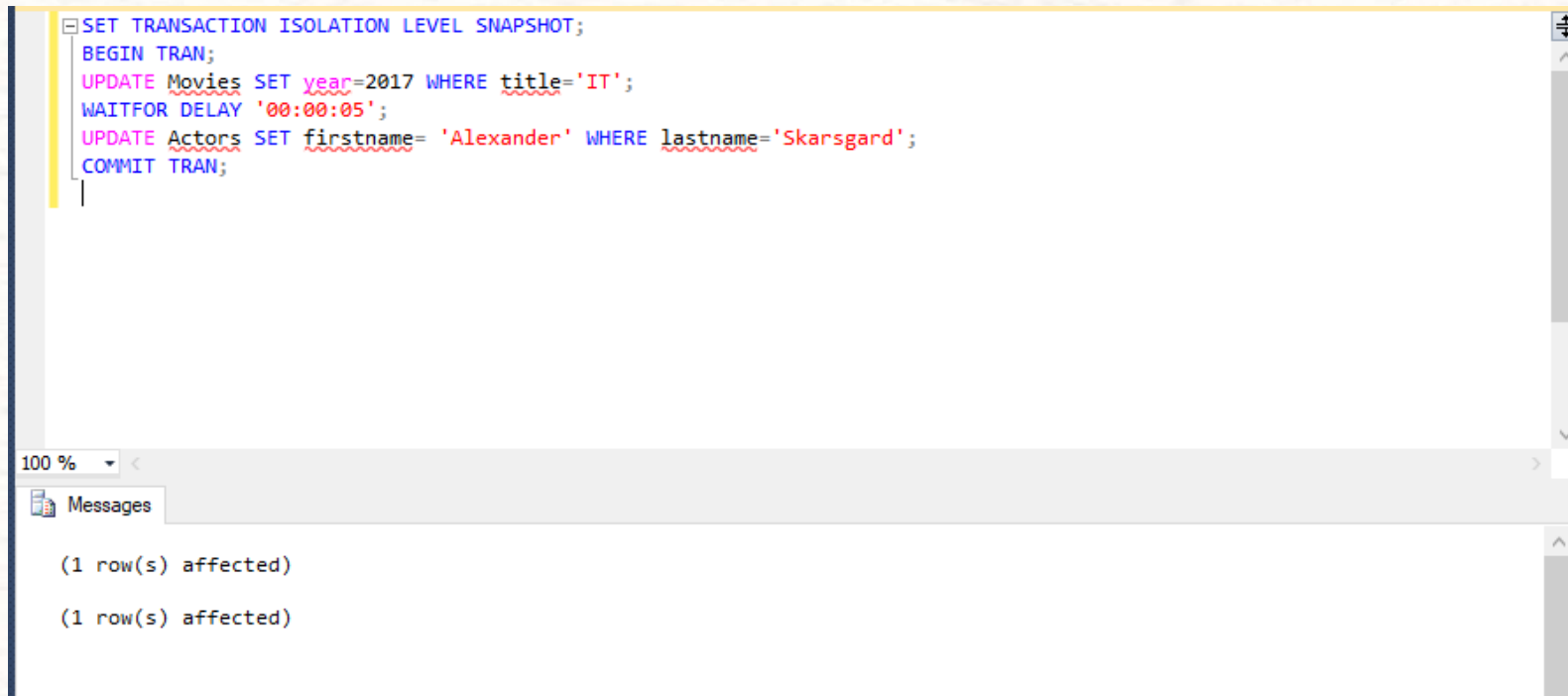
100 %

Messages

(1 row(s) affected)
Msg 1205, Level 13, State 51, Line 5
Transaction (Process ID 56) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. ROLLBACK TRANSACTION

Probleme de concurență – Exemplu (RLV)

- Deoarece a doua tranzacție a fost aleasă drept victimă a deadlock-ului, prima tranzacție a fost executată cu succes:



```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;  
BEGIN TRAN;  
UPDATE Movies SET year=2017 WHERE title='IT';  
WAITFOR DELAY '00:00:05';  
UPDATE Actors SET firstname= 'Alexander' WHERE lastname='Skarsgard';  
COMMIT TRAN;
```

100 %

Messages

(1 row(s) affected)

(1 row(s) affected)

Probleme de concurență – Exemplu (RLV)

- Dacă executăm următoarele interogări, putem vedea rezultatul final:

```
SELECT * FROM Movies;
```

```
SELECT * FROM Actors;
```

	movie_id	title	year
1	1	IT	2017
2	2	Red Sparrow	2018
3	3	Black Panther	2018

	actor_id	firstname	lastname
1	1	Alexander	Skarsgard

Probleme de concurență – Exemplu (RLV)

- Exemplu update conflict:
- Deschidem două ferestre noi de interogare (două conexiuni)
- În prima fereastră de interogare punem următoarea tranzacție:

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
```

```
BEGIN TRAN;
```

```
WAITFOR DELAY '00:00:03';
```

```
UPDATE Movies SET title='The Snowman' WHERE year=2017;
```

```
COMMIT TRAN;
```

Probleme de concurență – Exemplu (RLV)

- În a doua fereastră de interogare punem următoarea tranzacție:

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
```

```
BEGIN TRAN;
```

```
UPDATE Movies SET title='Thor Ragnarok' WHERE  
year=2017;
```

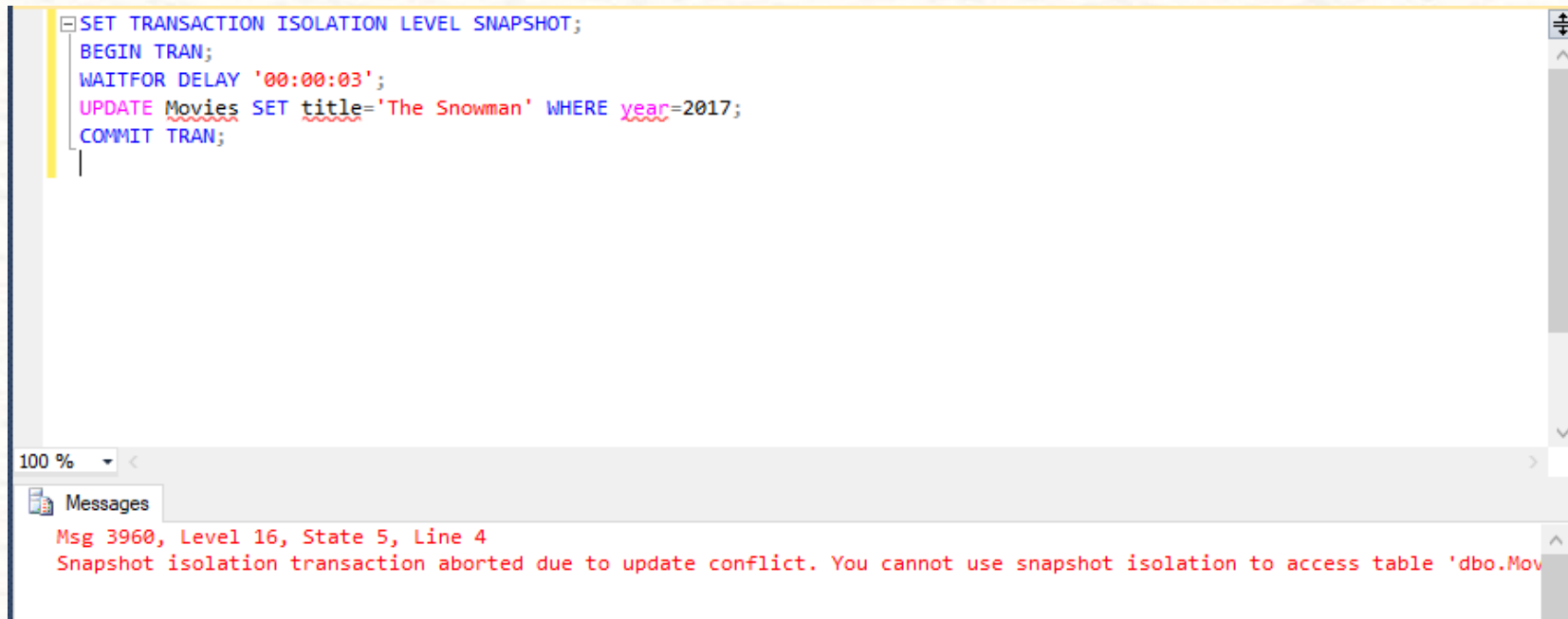
```
WAITFOR DELAY '00:00:03';
```

```
COMMIT TRAN;
```

- Începem execuția primei și a celei de a doua tranzacții (în această ordine)

Probleme de concurență – Exemplu (RLV)

- A avut loc un update conflict și prima tranzacție a eșuat:



```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;  
BEGIN TRAN;  
WAITFOR DELAY '00:00:03';  
UPDATE Movies SET title='The Snowman' WHERE year=2017;  
COMMIT TRAN;
```

100 %

Messages

Msg 3960, Level 16, State 5, Line 4
Snapshot isolation transaction aborted due to update conflict. You cannot use snapshot isolation to access table 'dbo.Mov

Probleme de concurență – Exemplu (RLV)

- Deoarece prima tranzacție a eșuat, a doua tranzacție a fost efectuată cu succes:



The screenshot shows a SQL Server Enterprise Manager query window. The query text is as follows:

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;  
BEGIN TRAN;  
UPDATE Movies SET title='Thor Ragnarok' WHERE year=2017;  
WAITFOR DELAY '00:00:03';  
COMMIT TRAN;
```

Below the query window, the Messages pane shows the result of the execution:

(1 row(s) affected)

Probleme de concurență – Exemplu (RLV)

- Dacă executăm următoarea interogare, putem vedea rezultatul final:

```
SELECT * FROM Movies;
```

	movie_id	title	year
1	1	Thor Ragnarok	2017
2	2	Red Sparrow	2018
3	3	Black Panther	2018