

Complementos de Bases de Dados

2022/2023

Licenciatura em Engenharia Informática



Laboratório 4 – Transações e Concorrência

Objetivos:

- Demonstração dos níveis de isolamento de transações no SQL Server

Resumo Teórica

De uma forma geral uma transação (Ti) é constituída por um conjunto de operações de leitura e escrita de dados. O SGBD monitoriza o estado de uma transação o que permite elevado nível de eficiência quando são geradas múltiplas transações concorrentes.

SET TRANSACTION ISOLATION LEVEL

O SSMS suporta os níveis de isolamento do standard SQL através da instrução:

```
-- Syntax for SQL Server Database
```

```
SET TRANSACTION ISOLATION LEVEL nível
```

```
/*
```

Nível pode assumir os seguintes valores:

READ COMMITTED (default): *Ti* só lê dados efetivados, mas outras transações podem escrever em dados lidos por *Ti*

READ UNCOMMITTED: *Ti* pode ler dados que ainda não foram efetivados

REPEATABLE READ: *Ti* só lê dados efetivados, mas outras transações podem criar dados no intervalo lido por *Ti*

SNAPSHOT: *Ti* só lê dados efetivados, todas as transações vêem os mesmos dados

SERIALIZABLE: *Ti* executa com completo isolamento

```
*/
```

Efeitos secundários admitidos por nível de isolamento:

Isolation level	Dirty Read	Nonrepeatable Read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Snapshot	No	No	No
Serializable	No	No	No

Enunciado:

Crie a tabela *Customer*:

```
SELECT * INTO Customer FROM SalesLT.Customer WHERE CustomerID < 1000
```

ETAPA 1: READ UNCOMMITTED e COMMITTED

- 1.1. É possível obter os valores que ainda estão bloqueados ou não confirmados por alterações de outra transação. No SSMS, abra duas janelas e execute as seguintes instruções:

Janela #1: Execute a seguinte transação, sem a finalizar:

```
BEGIN TRANSACTION
UPDATE dbo.Customer
SET     EmailAddress = 'new@estsetubal.ips.pt.pt'
WHERE   CustomerId = 5
```

Janela #2: Tente obter a leitura do valor:

```
SELECT EmailAddress
FROM    dbo.Customer
WHERE   CustomerId = 5
```

Analise os resultados obtidos, irá constatar que NÃO consegue obter nenhuma leitura.

Apresente uma resposta devidamente fundamentada?

- 1.2. Na Janela #2 do ponto anterior, tente novamente obter o valor, mas executando a seguinte instrução:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
SET NOCOUNT ON
GO
```

```
SELECT EmailAddress
FROM    dbo.Customer
WHERE   CustomerId = 5
```

OU

```
SELECT EmailAddress
FROM    dbo.Customer (NOLOCK)
WHERE   CustomerId = 5
```

Comente este procedimento. Apresente uma resposta devidamente fundamentada?

Termine a transação na Janela #1 executando a instrução

```
ROLLBACK
```

- 1.3. As seguintes instruções pretendem demonstrar a importância das transações em execuções concorrenciais. Abra duas janelas e execute as seguintes **funções de seguida**:

Janela #1: execute as seguintes *queries*, separadas por 10 segundos, à mesma tabela

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
SET NOCOUNT ON
GO
```

```
BEGIN TRAN
```

```
SELECT  EmailAddress
FROM    dbo.Customer
WHERE   CustomerId = 5
```

```
WAITFOR DELAY '00:00:10'
```

```
SELECT  EmailAddress
FROM    dbo.Customer
WHERE   CustomerId = 5
```

```
COMMIT TRAN
```

```
-- obs.: se necessário, poderá aumentar o delay para 15 ou 20 segundos
```

Janela #2: alterar o EmailAddress enquanto o procedimento anterior está em execução

```
BEGIN TRAN
UPDATE  dbo.Customer
SET     EmailAddress = 'new@estsetubal.ips.pt.pt'
WHERE   CustomerId = 5
COMMIT
```

Qual o resultado das duas *queries* na Janela #1? Comente.

ETAPA 2: REPEATABLE READ

Este nível de isolamento ultrapassa as limitações dos isolamentos anteriores. Abra duas janelas e execute as seguintes **funções de seguida**:

Janela #1: execute as seguintes *queries*, separadas por 10 segundos, à mesma tabela

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
SET NOCOUNT ON
GO
```

```
BEGIN TRAN
```

```
SELECT  EmailAddress
FROM    dbo.Customer
WHERE   CustomerId = 5
```

```
WAITFOR DELAY '00:00:10'
```

```
SELECT  EmailAddress
FROM    dbo.Customer
WHERE   CustomerId = 5
```

```
COMMIT TRAN
```

Janela #2: alterar o valor do EmailAddress enquanto o procedimento anterior está em execução

```
BEGIN TRAN
UPDATE  dbo.Customer
SET      EmailAddress = 'update@estsetubal.ips.pt.pt'
WHERE   CustomerId = 5
COMMIT
```

- 2.1) Qual o resultado das duas *queries* na Janela #1? Comente.
- 2.2) Em que situações se deve utilizar o nível de isolamento REPEATABLE READ, apresente uma resposta devidamente fundamentada?
- 2.3) Altere o comando do ponto 2 para inserir uma nova linha na mesma tabela.

```
INSERT INTO dbo.Customer
VALUES (0, 'Mr.', 'FirstName', null, 'LastName', null, 'CompanyName',
'SalesPerson', 'EmailAddress', 'Phone', '', '', NEWID(), GETDATE());
```

Execute novamente as instruções acima (i.e., Janela #1 & Janela #2) e comente o resultado.

ETAPA 3: SERIALIZABLE

Para ultrapassar o problema das leituras fantasma, é necessário um nível de isolamento **SERIALIZABLE**. Este nível, para além de assegurar o isolamento do **READ COMMITTED** e **REPEATABLE READ**, permite também que transações concorrentes executem como se fosse em série. Contudo, o impacto é a redução da concorrência do SGBD e portanto, menor performance.

Tendo por base as instruções executadas na Etapa 2, faça as seguintes alterações:

- Substituir: `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ`
- Por: `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE`

Comente o resultado obtido.

ETAPA 4: CONTROLO DE SESSÕES

Views de sistema sobre concorrência

4.1) Identifique o número de sessão associado às janelas #1 e #2 da Etapa 1.

Deverá executar de novo ambas e numa terceira janela (i.e., janela #3), executar a seguinte *query*:

```
SELECT -- use * to explore
  request_session_id      AS spid,
  resource_type            AS restype,
  resource_database_id    AS dbid,
  resource_description    AS res,
  resource_associated_entity_id AS resid,
  request_mode            AS mode,
  request_status          AS status
FROM sys.dm_tran_locks;
```

Comente os resultados obtidos

4.2) Observe e explore o resultado da seguinte *query*, comente os resultados obtidos.

```
SELECT -- use * to explore
  session_id AS spid, -- session ID (SPID)
  blocking_session_id,
  command,
  sql_handle,
  database_id,
  wait_type,
  wait_time,
  wait_resource
FROM sys.dm_exec_requests
WHERE blocking_session_id > 0;
```

- 4.3) Execute a seguinte *query*, substituindo X e Y pelos números de sessão identificados no ponto 4.1; comente os resultados obtidos.

```
SELECT session_id, text
FROM sys.dm_exec_connections
CROSS APPLY sys.dm_exec_sql_text(most_recent_sql_handle) AS ST
WHERE session_id IN (X, Y);
```

- 4.4) Execute as duas *queries* da Etapa 1 (i.e., janelas #1 e #2) e identifique os processos bloqueados utilizando o procedimento de sistema:

```
EXEC sp_who2
```

Parar a sessão que está a criar o bloqueio aplique o método KILL. Comente os resultados obtidos.

```
-- Example: checking the status of a query using the sp_who2
```

Results		Messages											
	SPID	Status	Login	HostName	BlkBy	DBName	Command	CPUTime	DiskIO	LastBatch	ProgramName	SPID	REQUESTID
1	84	RUNNABLE	kashish\Test	KASHISH	.	SQLShackDemo	INSERT	17093	3826	08/16 09:48:52	Microsoft SQL Server Management Studio - Query	84	0

```
-- Use the KILL command to KILL SPID.
-- Example: execute this command in a new query window to kill SPID 84
```

```
KILL 84
```

```
-- After executing the KILL SPID command, SQL Server starts the ROLLBACK process
for the selected query.
-- Check the "Status" using the sp_who2
```

	SPID	Status	Login	HostName	BlkBy	DBName	Command	CPUTime	DiskIO	LastBatch	ProgramName	SPID	REQUESTID
1	84	ROLLBACK	kashish\Test	KASHISH	.	SQLShackDemo	AWAITING COMMAND	53280	14775	08/16 09:49:59	Microsoft SQL Server Management Studio - Query	84	0

(fim de enunciado)