SQL

Última atualização a *8 de junho de 2020*

Transações

Em SQL, um comando individual é considerado uma transação.

No entanto, existem comandos que auxiliam a geri-las.

> `BEGIN TRAN[SACTION] [<trans_name>|<@trans_name_variable>]`
>
> Inicia e configura características de uma transação
>
> Pode ter como nome uma variável do tipo VARCHAR
>
> `COMMIT [TRAN[SACTION]] [<trans_name>|<@trans_name_variable>]`
>
> Encerra a transação
>
> `ROLLBACK [TRAN[SACTION]] [<trans_name>|<@trans_variable>]`
>
> Desfaz ações da tra nsação

Se por alguma razão a transação não terminar da forma esperada (com um comando COMMIT ou ROLLBACK) é realizado um **ROLLBACK implícito**, ou seja, a transação será revertida. + info

```
-- Exemplo
BEGIN TRANSACTION
    UPDATE authors SET au_lname = upper(au_lname) WHERE au_lname = 'White'
IF @@ROWCOUNT = 2
    COMMIT TRAN
ELSE
    BEGIN
        PRINT 'A transaction needs to be rolled back'
        ROLLBACK TRAN
    END
```

Existem ainda pontos internos aos quais é possível retornar, denominados **savepoints**. +

> `SAVE TRAN[SACTION] [<savepoint name>|<@savepoint variable>]`
>
> `ROLLBACK [TRAN[SACTION]] [<savepoint_name>|<@savepoint_name_variable>]`

```
BEGIN TRANSACTION

   INSERT INTO TestTable( ID, Value )
   VALUES  ( 1, N'10')
   -- this will create a savepoint after the first INSERT
   SAVE TRANSACTION FirstInsert

   INSERT INTO TestTable( ID, Value )
   VALUES  ( 2, N'20')
```

```
   -- this will rollback to the savepoint right after the first INSERT was
done
   ROLLBACK TRANSACTION FirstInsert

-- this will commit the transaction leaving just the first INSERT
COMMIT
```

Recuperação de falhas

Existem comandos SQL que nos permitem gerir as falhas e a sua recuperação.

Isolamento

Para gerir o isolamento de uma transação devemos declará-lo aquando da definição da mesma, definindo o seu nível, que pode variar entre:

> SERIALIZABLE isolamento completo
>
> REPEATABLE READ só lê dados *commited* e outras transações não podem modificar dados lidos por si
>
> READ COMMITTED só lê dados *commited* mas outras transações podem alterar dados lidos por si
>
> > É o nível por defeito em SQL Server
>
> READ UNCOMMITTED pode ler dados que ainda não foram *commited*
>
> SNAPSHOT vê imagem dos dados que existiam antes da transação ter sido iniciada (não vê alterações *comitted* entretanto)

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
GO
BEGIN TRANSACTION;
...
COMMIT TRANSACTION;
```

Transações encadeadas

Podemos ter transações dentro de transações, sendo para isso fundamental a variável `@@TRANCOUNT` que conta o número de transações ativas.

`VIP` Um *rollback* dentro de uma transação interna reverte todas as operações até ao primeiro BEGIN TRAN, sendo por isso preferível criar *savepoints*!

Stored Procedures

Tal como nas transações, um *rollback* num SP reverte as suas operações, mas também as exteriores ao SP. Devemos por isso também utilizar *savepoints*.

```
CREATE PROC Name
AS
BEGIN
    BEGIN TRAN
    SAVE TRAN Savepoint1
    -- ...
    ROLLBACK TRAN Savepoint1
    COMMIT TRAN
END
```

Gestão de erros em transações

Numa transação, quando ocorre um erro numa operação, é feito o *rollback* da transação. No entanto, se definirmos a variável `XACT_ABORT` com o valor `OFF`, apesar de ser levantado o erro e a transação interrompida, as operações realizadas antes da ocorrência do erro não são desfeitas.

```
BEGIN TRAN
...
SET XACT_ABORT OFF
-- valor default é ON
COMMIT TRAN
```

Uma alternativa a esta abordagem é o bloco **try...catch**, que na ocorrência de um erro dentro do bloco *try* desfaz as operações realizadas nesse bloco, mas salta para o bloco *catch*, realizando as operações dentro deste e depois do mesmo.

```
BEGIN TRY
    INSERT into dbo.customers_deleted select * from deleted;
    DELETE from customers where custID =@id;
END TRY
BEGIN CATCH
    raiserror ('Delete Error', 16, 1);
END CATCH
Print 'Cheguei aqui...';
COMMIT TRAN
```