TRANSACÇÕES

PARTE I

(Extraído de "SQL Server Books Online")

Transactions Architecture

Microsoft® SQL Server™ 2000 maintains the consistency and integrity of each database despite errors that occur in the system. Every application that updates data in a SQL Server database does so using transactions. A transaction is a logical unit of work made up of a series of statements (selects, inserts, updates, or deletes). If no errors are encountered during a transaction, all of the modifications in the transaction become a permanent part of the database. If errors are encountered, none of the modifications are made to the database.
A transaction goes through several phases:

- Before the transaction starts, the database is in a consistent state.
- The application signals the start of a transaction. This can be done explicitly with the BEGIN TRANSACTION statement. Alternatively, the application can set options to run in implicit transaction mode; the first Transact-SQL statement executed after the completion of a prior transaction starts a new transaction automatically. No record is written to the log when the transaction starts; the first record is written to the log when the application generates the first log record for a data modification.
- The application starts modifying data. These modifications are made one table at a time. As a series of modifications are made, they may leave the database in a temporarily inconsistent intermediate state.
- When the application reaches a point where all the modifications have completed successfully and the database is once again consistent, the application commits the transaction. This makes all the modifications a permanent part of the database.
- If the application encounters some error that prevents it from completing the transaction, it undoes, or rolls back, all the data modifications. This returns the database to the point of consistency it was at before the transaction started.

SQL Server applications can also run in autocommit mode. In autocommit mode each individual Transact-SQL statement is committed automatically if it is successful and rolled back automatically if it generates an error. There is no need for an application running in autocommit mode to issue statements that specifically start or end a transaction.

All Transact-SQL statements run in a transaction: an explicit transaction, an implicit transaction, or an autocommit transaction. All SQL Server transactions that include data modifications either reach a new point of consistency and are committed, or are rolled back to the original point of consistency. Transactions are not left in an intermediate state where the database is not consistent.

Transactions

A transaction is a sequence of operations performed as a single logical unit of work. A logical unit of work must exhibit four properties, called the ACID (Atomicity, Consistency, Isolation, and Durability) properties, to qualify as a transaction:

Atomicity

A transaction must be an atomic unit of work; either all of its data modifications are performed, or none of them is performed.

Consistency

When completed, a transaction must leave all data in a consistent state. In a relational database, all rules must be applied to the transaction's modifications to maintain all data integrity. All internal data structures, such as B-tree indexes or doubly-linked lists, must be correct at the end of the transaction.

Isolation

Modifications made by concurrent transactions must be isolated from the modifications made by any other concurrent transactions. A transaction either sees data in the state it was in before another concurrent transaction modified it, or it sees the data after the second transaction has completed, but it does not see an intermediate state. This is referred to as serializability because it results in the ability to reload the starting data and replay a series of transactions to end up with the data in the same state it was in after the original transactions were performed.

Durability

After a transaction has completed, its effects are permanently in place in the system. The modifications persist even in the event of a system failure.

Specifying and Enforcing Transactions

SQL programmers are responsible for starting and ending transactions at points that enforce the logical consistency of the data. The programmer must define the sequence of data modifications that leave the data in a consistent state relative to the organization's business rules. The programmer then includes these modification statements in a single transaction so that Microsoft® SQL Server™ can enforce the physical integrity of the transaction.

It is the responsibility of an enterprise database system, such as SQL Server, to provide mechanisms ensuring the physical integrity of each transaction. SQL Server provides:

- Locking facilities that preserve transaction isolation.
- Logging facilities that ensure transaction durability. Even if the server hardware, operating system, or SQL Server itself fails, SQL Server uses the transaction logs, upon restart, to automatically roll back any uncompleted transactions to the point of the system failure.
- Transaction management features that enforce transaction atomicity and consistency. After a transaction has started, it must be successfully completed, or SQL Server undoes all of the data modifications made since the transaction started.

Controlling Transactions

Applications control transactions mainly by specifying when a transaction starts and ends. The system must also be able to correctly handle errors that terminate a transaction before it completes.

Transactions are managed at the connection level. When a transaction is started on a connection, all Transact-SQL statements executed on that connection are part of the transaction until the transaction ends.

Starting Transactions

You can start transactions in Microsoft® SQL Server™ as explicit, autocommit, or implicit transactions.

Explicit transactions

Explicitly start a transaction by issuing a BEGIN TRANSACTION statement.

Autocommit transactions

This is the default mode for SQL Server. Each individual Transact-SQL statement is committed when it completes. You do not have to specify any statements to control transactions.

Implicit transactions

Set implicit transaction mode on through either an API function or the Transact-SQL SET IMPLICIT_TRANSACTIONS ON statement. The next statement automatically starts a new transaction. When that transaction is completed, the next Transact-SQL statement starts a new transaction.

Connection modes are managed at the connection level. If one connection changes from one transaction mode to another it has no effect on the transaction modes of any other connection.

Ending Transactions

You can end transactions with either a COMMIT or ROLLBACK statement.

COMMIT

If a transaction is successful, commit it. A COMMIT statement guarantees all of the transaction's modifications are made a permanent part of the database. A COMMIT also frees resources, such as locks, used by the transaction.

ROLLBACK

If an error occurs in a transaction, or if the user decides to cancel the transaction, then roll the transaction back. A ROLLBACK statement backs out all modifications made in the transaction by returning the data to the state it was in at the start of the transaction. A ROLLBACK also frees resources held by the transaction.

Explicit Transactions

An explicit transaction is one in which you explicitly define both the start and end of the transaction. Explicit transactions were also called user-defined or user-specified transactions in earlier versions of Microsoft® SQL Server™.

DB-Library applications and Transact-SQL scripts use the BEGIN TRANSACTION, COMMIT TRANSACTION, COMMIT WORK, ROLLBACK TRANSACTION, or ROLLBACK WORK Transact-SQL statements to define explicit transactions.

BEGIN TRANSACTION

Marks the starting point of an explicit transaction for a connection.

COMMIT TRANSACTION or COMMIT WORK

Used to end a transaction successfully if no errors were encountered. All data modifications made in the transaction become a permanent part of the database. Resources held by the transaction are freed.

ROLLBACK TRANSACTION or ROLLBACK WORK

Used to erase a transaction in which errors are encountered. All data modified by the transaction is returned to the state it was in at the start of the transaction. Resources held by the transaction are freed.

Autocommit Transactions

Autocommit mode is the default transaction management mode of Microsoft® SQL Server™. Every Transact-SQL statement is committed or rolled back when it completes. If a statement completes successfully, it is committed; if it encounters any error, it is rolled back. A SQL Server connection operates in autocommit mode whenever this default mode has not been overridden by either explicit or implicit transactions. Autocommit mode is also the default mode for ADO, OLE DB, ODBC, and DB-Library.

A SQL Server connection operates in autocommit mode until a BEGIN TRANSACTION statement starts an explicit transaction, or implicit transaction mode is set on. When the explicit transaction is committed or rolled back, or when implicit transaction mode is turned off, SQL Server returns to autocommit mode.

Implicit Transactions

When a connection is operating in implicit transaction mode, Microsoft® SQL Server™ automatically starts a new transaction after the current transaction is committed or rolled back. You do nothing to delineate the start of a transaction; you only commit or roll back each transaction. Implicit transaction mode generates a continuous chain of transactions.

After implicit transaction mode has been set on for a connection, SQL Server automatically starts a transaction when it first executes any of these statements:

ALTER TABLE	DROP	INSERT	SELECT
CREATE	FETCH	OPEN	TRUNCATE TABLE
DELETE	GRANT	REVOKE	UPDATE

The transaction remains in effect until you issue a COMMIT or ROLLBACK statement. After the first transaction is committed or rolled back, SQL Server automatically starts a new transaction the next time any of these statements are executed by the connection. SQL Server keeps generating a chain of implicit transactions until implicit transaction mode is turned off.

Implicit transaction mode is set either using the Transact-SQL SET statement, or through database API functions and methods.

Advanced Topics

Mismanagement of transactions often leads to contention and performance problems in systems that have many users. As the number of users in a system increases, it becomes important to have applications that use transactions efficiently. A transaction can hold some locks, such as those protecting updates, until the transaction ends. An application that allows users to control when a transaction ends presents an opportunity for a malicious user to deny access to data that is being locked. For example, it is generally a bad practice for an application to interact with a user while the application has a transaction open unless the application places a limit on how long it will wait for a user response before ending the transaction.

Microsoft® SQL Server™ also supports nesting transactions, transaction savepoints, and bound transactions, which offer programmers additional options for writing efficient transactions.

Isolation Levels

When locking is used as the concurrency control mechanism, it solves concurrency problems. This allows all transactions to run in complete isolation of one another, although there can be more than one transaction running at any time.

Serializability is the database state achieved by running a set of concurrent transactions equivalent to the database state that would be achieved if the set of transactions were executed serially in order.

SQL-92 Isolation Levels

Although serialization is important to transactions to ensure that the data in the database is correct at all times, many transactions do not always require full isolation. For example, several writers are working on different chapters of the same book. New chapters can be submitted to the project at any time. However, after a chapter has been edited, a writer cannot make any changes to the chapter without the editor's approval. This way, the editor can be assured of the accuracy of the book project at any point in time, despite the arrival of new unedited chapters. The editor can see both previously edited chapters and recently submitted chapters.

The level at which a transaction is prepared to accept inconsistent data is termed the isolation level. The isolation level is the degree to which one transaction must be isolated from other transactions. A lower isolation level increases concurrency, but at the expense of data correctness. Conversely, a higher isolation level ensures that data is correct, but can affect concurrency negatively. The isolation level required by an application determines the locking behavior SQL Server uses.

SQL-92 defines the following isolation levels, all of which are supported by SQL Server:

- Read uncommitted (the lowest level where transactions are isolated only enough to ensure that physically corrupt data is not read).
- Read committed (SQL Server default level).
- Repeatable read.
- Serializable (the highest level, where transactions are completely isolated from one another).

If transactions are run at an isolation level of serializable, any concurrent overlapping transactions are guaranteed to be serializable.

These isolation levels allow different types of behavior.

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

Transactions must be run at an isolation level of repeatable read or higher to prevent lost updates that can occur when two transactions each retrieve the same row, and then later update the row based on the originally retrieved values. If the two transactions update rows using a single UPDATE statement and do not base the update on the previously retrieved values, lost updates cannot occur at the default isolation level of read committed.

Adjusting Transaction Isolation Levels

The isolation property is one of the four ACID properties a logical unit of work must display to qualify as a transaction. It is the ability to shield transactions from the effects of updates performed by other concurrent transactions. The level of isolation is actually customizable for each transaction.

Microsoft® SQL Server™ supports the transaction isolation levels defined in SQL-92. Setting transaction isolation levels allows programmers to trade off increased risk of certain integrity problems with support for greater concurrent access to data. Each isolation level offers more isolation than the previous level, but does so by holding more restrictive locks for longer periods.

The transaction isolation levels are:

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

Set transaction isolation level

Controls the default transaction locking behavior for all Microsoft® SQL Server™ SELECT statements issued by a connection.

Syntax

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

Arguments

READ COMMITTED

Specifies that shared locks are held while the data is being read to avoid dirty reads, but the data can be changed before the end of the transaction, resulting in nonrepeatable reads or phantom data. This option is the SQL Server default.

READ UNCOMMITTED

Implements dirty read, or isolation level 0 locking, which means that no shared locks are issued and no exclusive locks are honored. When this option is set, it is possible to read uncommitted or dirty data; values in the data can be changed and rows can appear or disappear in the data set before the end of the transaction. This option has the same effect as setting NOLOCK on all tables in all SELECT statements in a transaction. This is the least restrictive of the four isolation levels.

REPEATABLE READ

Locks are placed on all data that is used in a query, preventing other users from updating the data, but new phantom rows can be inserted into the data set by another user and are included in later reads in the current transaction. Because concurrency is lower than the default isolation level, use this option only when necessary.

SERIALIZABLE

Places a range lock on the data set, preventing other users from updating or inserting rows into the data set until the transaction is complete. This is the most restrictive of the four isolation levels. Because concurrency is lower, use this option only when necessary. This option has the same effect as setting HOLDLOCK on all tables in all SELECT statements in a transaction.

TRANSACÇÕES

PARTE I I

(The Guru's Guide to Transact-SQL)

Inicie duas instâncias do *SQL Query Analyzer* e em cada sessão execute um dos *queries* indicados a seguir.

Transaction Isolation Levels

SQL Server supports four transaction isolation levels. As mentioned earlier, a transaction's isolation level controls how it affects, and is affected by, other transactions. The trade-off is always one of data consistency vs. concurrency. Selecting a more restrictive TIL increases data consistency at the expense of accessibility. Selecting a less restrictive TIL increases concurrency at the expense of data consistency. The trick is to balance these opposing interests so that the needs of your application are met. Use the SET TRANSACTION ISOLATION LEVEL command to set a transaction's isolation level. Valid TILs include READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE.

READ UNCOMMITTED

It permits **dirty reads** (reads of uncommitted changes by other transactions) and **nonrepeatable reads** (data that changes between reads during a transaction). To see how READ UNCOMMITTED permits dirty and nonrepeatable reads, run the following queries simultaneously:

```
Query 1
      select top 5 nome, D.local from departamento as D order by nome
      begin tran
            update departamento set Local = 'UBI'

            select top 5 nome, D.local
            from departamento as D order by nome

            waitfor delay '00:00:05'
      Rollback tran
      select top 5 nome, D.local
      from departamento as D order by nome

Query 2
      Set transaction Isolation level read uncommitted
      Select 'Agora está a vê-los...'

      select top 5 nome, D.local from departamento as D
      where local = 'UBI' order by nome

      if @@RowCount > 0
      Begin
            waitfor delay '00:00:05'

            Select 'Agora não...'

            select top 5 nome, D.local
            from departamento as D
            where local = 'UBI' order by nome
      end
```

Nota: enquanto o primeiro query está a executar (tem-se 5 segundos) se executarmos o segundo query podemos aceder a dados alterados pelo primeiro query mas ainda não confirmados (committed). Após esperar 5 segundos tenta-se ler novamente os mesmos dados. Contudo, como as alterações foram descartadas (rolled back), os dados desapareceram, provocando no segundo query uma leitura que não se pode repetir (nonrepeatable read).

Query 1

```
nome                           local
------------------------------ -----------
Camarote                       Camarate
Comercial                      Lisboa
Informática                    Covilhã
Produção                       Guarda

(4 row(s) affected)

(4 row(s) affected)

nome                           local
------------------------------ ---------
Camarote                       UBI
Comercial                      UBI
Informática                    UBI
Produção                       UBI

(4 row(s) affected)

nome                           local
------------------------------ ----------
Camarote                       Camarate
Comercial                      Lisboa
Informática                    Covilhã
Produção                       Guarda

(4 row(s) affected)
```

Query 2

```
----------------------
Agora está a vê-los...

(1 row(s) affected)

nome                           local
------------------------------ -------
Camarote                       UBI
Comercial                      UBI
Informática                    UBI
Produção                       UBI

(4 row(s) affected)

------------
Agora não...

(1 row(s) affected)

nome                           local
------------------------------ --------

(0 row(s) affected)
```

READ UNCOMMITTED

READ COMMITTED **avoids dirty reads** by initiating share locks on accessed data but permits changes to underlying data during the transaction, possibly resulting in **nonrepeatable reads and/or phantom data**. To see how this works, run the following queries simultaneously:

Query 1

```
Set transaction Isolation level read committed
begin tran
      Select 'Agora está a vê-los...'

      select top 5 nome, D.local
      from departamento as D
      order by nome

      waitfor delay '00:00:05'

      Select 'Agora não...'

      select top 5 nome, D.local
      from departamento as D
      order by nome
rollback tran
```

Query 2

```
Set transaction Isolation level read committed
update departamento
set Nome = 'Camarata'
where nome = 'Camarote
```

Nota: o nome do departamento foi alterado entre as leituras efectuadas durante o primeiro query – a nonrepeatable read!

Query 1

```
----------------------
Agora está a vê-los...
(1 row(s) affected)

nome                           local
------------------------------ --------
Camarote                       Camarate
Comercial                      Lisboa
Informática                    Covilhã
Produção                       Guarda

(4 row(s) affected)

------------
Agora não...

(1 row(s) affected)

nome                           local
------------------------------ --------
Camarata                       Camarate
Comercial                      Lisboa
Informática                    Covilhã
Produção                       Guarda

(4 row(s) affected)
```

Query 2

```
(1 row(s) affected)
```

REPEATABLE READ

REPEATABLE READ initiates locks to prevent other users from changing the data a transaction accesses but doesn't prevent new rows from being inserted, possibly resulting in **phantom rows** appearing between reads during the transaction.

Query 1

```
Set transaction Isolation level repeatable read
begin tran
      Select 'Veja bem...'

      select top 5 nome, D.local
      from departamento as D
      order by nome

      waitfor delay '00:00:05'

      Select 'Veja melhor...'

      select top 5 nome, D.local
      from departamento as D
      order by nome
rollback tran
```

Query 2

```
Set transaction Isolation level repeatable read
Insert departamento
Values( 50, 'YYYYY', 'XXXXX')
```

Query 1

```
-----------
Veja bem...

(1 row(s) affected)

nome                           local
------------------------------ --------
Camarata                       Camarate
Comercial                      Lisboa
Informática                    Covilhã
```

Query 2

```
(1 row(s) affected)
```

```
Produção                       Guarda

(4 row(s) affected)

--------------
Veja melhor...

(1 row(s) affected)

nome                           local
------------------------------ -------
Camarata                       Camarate
Comercial                      Lisboa
Informática                    Covilhã
Produção                       Guarda
YYYYY                          XXXXX

(5 row(s) affected)
```

As you can see, a new row appears between the first and second reads of the `departamento` table, even though REPEATABLE READ has been specified. Though REPEATABLE READ prevents changes to data it has already accessed, it doesn't prevent the addition of new data, thus introducing the possibility of phantom rows.

SERIALIZABLE

SERIALIZABLE **prevents dirty reads and phantom rows** by placing a range lock on the data it accesses. It is the most restrictive of SQL Server's four TILs. It's equivalent to using the HOLDLOCK hint with every table a transaction references.

Query 1

```
Set transaction Isolation level serializable
begin tran
      Select 'Veja bem...'

      select top 5 nome, D.local
      from departamento as D
      order by nome

      waitfor delay '00:00:05'

      Select 'Veja melhor...'

      select top 5 nome, D.local
      from departamento as D
      order by nome
rollback tran
```

Query 2

```
Set transaction Isolation level serializable
Insert departamento
Values(49, 'ZZZZZZ', 'XXXXX')
```

Query 1

```
-----------
Veja bem...

(1 row(s) affected)

nome                           local
------------------------------ Camarata
Camarate
Comercial                      Lisboa
Informática                    Covilhã
Produção                       Guarda
YYYYY                          XXXXX

(5 row(s) affected)
```

Query 2

```
(1 row(s) affected)
```

```
--------------
Veja melhor...

(1 row(s) affected)

nome                           local
------------------------------ --------
Camarata                       Camarate
Comercial                      Lisboa
Informática                    Covilhã
Produção                       Guarda
YYYYY                          XXXXX

(5 row(s) affected)
```

In this example, the locks initiated by the SERIALIZABLE isolation level prevent the second query from running until after the first one finishes. While this provides airtight data consistency, it does so at a cost of greatly reduced concurrency.