[1. ACID 2](#_Toc435710096)

[1-1. Atomicity 2](#_Toc435710097)

[1-2. Consistency 2](#_Toc435710098)

[1-3. Isolation 2](#_Toc435710099)

[1-4. Durability 2](#_Toc435710100)

[2. SQL Server 2008 R2 支持的事务隔离级别 3](#_Toc435710101)

[2-1. 小结 3](#_Toc435710102)

[2-2. 详细说明 3](#_Toc435710103)

[2-3. 为什么会支持有并发问题的隔离级别 3](#_Toc435710104)

# ACID

In computer science, ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee that database transactions are processed reliably. In the context of databases, a single logical operation on the data is called a transaction.

For example, **a transfer of funds from one bank account to another, even involving multiple changes such as debiting one account and crediting another, is a single transaction**.

## Atomicity

Atomicity requires that each transaction be "all or nothing": if one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes. To the outside world, a committed transaction appears (by its effects on the database) to be indivisible ("atomic"), and an aborted transaction does not happen.

## Consistency

The consistency property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof. This does not guarantee correctness of the transaction in all ways the application programmer might have wanted (that is the responsibility of application-level code) but merely that any programming errors cannot result in the violation of any defined rules.

## Isolation

The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e., one after the other. **Providing isolation is the main goal of concurrency control**. Depending on concurrency control method (i.e. if it uses strict - as opposed to relaxed - serializability), **the effects of an incomplete transaction might not even be visible to another transaction**.

## Durability

The durability property ensures that **once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors**. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter). To defend against power loss, transactions (or their effects) must be recorded in a non-volatile memory.

# SQL Server 2008 R2 支持的事务隔离级别

## 小结



可序列化(Serializable)

可重复读(Repeatable reads)

提交读(Read committed)

未提交读(Read uncommitted)

## 详细说明

<http://blog.itpub.net/13651903/viewspace-1082730/>

<http://www.cnblogs.com/chillsrc/archive/2013/04/20/3032787.html>

http://www.cnblogs.com/chenmh/p/3998614.html

## 为什么会支持有并发问题的隔离级别

1. 并非所有系统的数据库都是并发访问的,虽然这种情况比较少（比如自己搭个模拟的环境学学SQL啥的），当然怎么设置都可以，不存在并发问题（企业级的应用都应该是考虑并发的）
2. 有时候是在准确性和运行效率的一种Balance，折衷。

# SQL Server 的每个Query都是一个独立的Session

互相之间的事务隔离级别互不影响，可以用开多个Query窗口的方式模拟并发操作

# 事务隔离级别是针对Session的而不是整个数据库的

数据库有自己的默认的事务隔离级别（SQL Server 是Read Commited），但是每个connection 都可以自己更改的事务隔离级别。

## 如何更改

set transaction isolation level read uncommitted

go

## 如何查看当前session的事务隔离级别

SELECT transaction\_isolation\_level

FROM sys.dm\_exec\_sessions

where session\_id = @@SPID

SELECT CASE transaction\_isolation\_level

WHEN 0 THEN 'Unspecified'

WHEN 1 THEN 'ReadUncommitted'

WHEN 2 THEN 'ReadCommitted'

WHEN 3 THEN 'Repeatable'

WHEN 4 THEN 'Serializable'

WHEN 5 THEN 'Snapshot' END AS TRANSACTION\_ISOLATION\_LEVEL

FROM sys.dm\_exec\_sessions

where session\_id = @@SPID

## 如何查看所有session的事务隔离级别

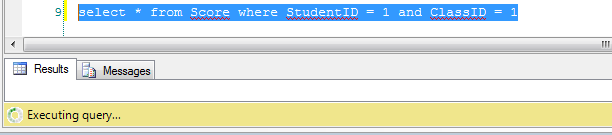
SELECT \*

FROM sys.dm\_exec\_sessions

# 事务的隔离级别内部是如何实现的

## 是通过锁机制

事务内的每一条语句在执行之前，SQL Server 都会试图去取得被影响对象的锁。如果取不到，SQL Server 就会等待。正如同下面



## 隔离级别与锁持续的时间

在基于锁的并发控制中，隔离级别决定了锁的持有时间。

"C"-表示锁会持续到事务提交。 "S" –表示锁持续到当前语句执行完毕。



# 并发带来的几个问题

## 脏读(Dirty reads (Uncommitted Dependency))

脏读不是说读取了旧的数据，恰恰是读了最新的数据，就是读了别人还没有提交的数据。

这个同Cache机制里面的脏读还不太一样，那个应该是读了旧的数据

因为别的事务最终由可能是rollback而不是commit更新

## 不可重复读(non-repeatable read)

在一次事务中，当**一行数据**获取两遍得到不同的结果表示发生了“不可重复读(non-repeatable read)”.

还有就是读完之后，在执行下一次读之前，有别的事务更新了数据并提交

所以不可重复读至少表名事务需要进行两次以上的读，如果只进行一次读，那么不管隔离级别怎样，应该都不会出现不可重复读的险象。

脏读算不算不可重复读的范畴，要看脏读之后是否还有后续的读。如果脏读之后直接更新而没有二次读，有可能的影响是Update Lost（更新丢失），而不会有不可重复读的现象。

## 幻影读(phantom read)

在事务执行过程中，当两个完全相同的查询语句执行得到不同的结果集。这种现象称为“幻影读(phantom read)”

“幻影读(phantom read)”是不可重复读(Non-repeatable reads)的一种特殊场景：当事务1两次执行SELECT ... WHERE检索一定范围内数据的操作中间，事务2在这个表中创建了(如INSERT)了一行新数据，这条新数据正好满足事务1的“WHERE”子句。