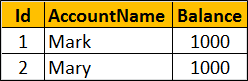
**In this video we will discuss**  
1. What a transaction is  
2. The problems that might arise when tarnsactions are run concurrently  
3. The different transaction isolation levels provided by SQL Server to address concurrency side effects   
  
   
  
**What is a transaction**  
A transaction is a group of commands that change the data stored in a database. A transaction, is treated as a single unit of work. A transaction ensures that, either all of the commands succeed, or none of them. If one of the commands in the transaction fails, all of the commands fail, and any data that was modified in the database is rolled back. In this way, transactions maintain the integrity of data in a database.   
  
   
  
**Example :** The following transaction ensures that both the UPDATE statements succeed or both of them fail if there is a problem with one UPDATE statement. 

-- Transfer $100 from Mark to Mary Account

BEGIN TRY

    BEGIN TRANSACTION

         UPDATE Accounts SET Balance = Balance - 100 WHERE Id = 1

         UPDATE Accounts SET Balance = Balance + 100 WHERE Id = 2

    COMMIT TRANSACTION

    PRINT 'Transaction Committed'

END TRY

BEGIN CATCH

    ROLLBACK TRANSACTION

    PRINT 'Transaction Rolled back'

END CATCH

Databases are powerful systems and are potentially used by many users or applications at the same time. Allowing concurrent transactions is essential for performance but may introduce concurrency issues when two or more transactions are working with the same data at the same time.  
  
**Some of the common concurrency problems**

* Dirty Reads
* Lost Updates
* Nonrepeatable Reads
* Phantom Reads

We will discuss what these problems are in detail with examples in our upcomning videos  
  
One way to solve all these concurrency problems is by allowing only one user to execute, only one transaction at any point in time. Imagine what could happen if you have a large database with several users who want to execute several transactions. All the transactions get queued and they may have to wait a long time before they could get a chance to execute their transactions. So you are getting poor performance and the whole purpose of having a powerful database system is defeated if you serialize access this way.   
  
At this point you might be thinking, for best performance let us allow all transactions to execute concurrently. The problem with this approach is that it may cause all sorts of concurrency problems (i.e Dirty Reads, Lost Updates, Nonrepeatable Reads, Phantom Reads) if two or more transactions work with the same data at the same time.  
  
SQL Server provides different **transaction isolation levels**, to balance concurrency problems and performance depending on our application needs.

* Read Uncommitted
* Read Committed
* Repeatable Read
* Snapshot
* Serializable

**The isolation level that you choose for your transaction**, defines the degree to which one transaction must be isolated from resource or data modifications made by other transactions. Depending on the isolation level you have chosen you get varying degrees of performance and concurrency problems. The table here has the list of isoltaion levels along with concurrency side effects.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Isolation Level** | **Dirty Reads** | **Lost Update** | **Nonrepeatable Reads** | **Phantom Reads** |
| **Read Uncommitted** | Yes | Yes | Yes | Yes |
| **Read Committed** | No | Yes | Yes | Yes |
| **Repeatable Read** | No | No | No | Yes |
| **Snapshot** | No | No | No | No |
| **Serializable** | No | No | No | No |

If you choose the lowest isolation level (i.e Read Uncommitted), it increases the number of concurrent transactions that can be executed at the same time, but the down side is you have all sorts of concurrency issues. On the other hand if you choose the highest isolation level (i.e Serializable), you will have no concurrency side effects, but the downside is that, this will reduce the number of concurrent transactions that can be executed at the same time if those transactions work with same data.  
  
In our upcoming videos we will discuss the concurrency problems in detail with examples