6 ways of doing locking in .NET (Pessimistic and optimistic)

<http://www.c-sharpcorner.com/UploadFile/shivprasadk/6-ways-of-doing-locking-in-net-pessimistic-and-optimistic/>

By **[Shivprasad koirala](http://www.codeproject.com/script/Membership/View.aspx?mid=1335831)**, 2 Oct 2010

|  |  |
| --- | --- |
|  |  |

Top of Form

6 ways of doing locking in .NET (Pessimistic and optimistic)

|  |  |
| --- | --- |
|  |  |

Introduction

## Introduction

This article talks about 6 ways of doing locking in .NET. It starts with concurrency problems and then discusses about 3 ways of doing optimistic locking. As optimistic locking does not solve the concurrency issues from roots, it introduces pessimistic locking. It then moves ahead to explain how isolation levels can help us implement pessimistic locking. Each isolation level is explained with sample demonstration to make concepts clearer.

This is a small Ebook for all my .NET friends which covers topics like WCF,WPF,WWF,Ajax,Core .NET,SQL, Entity framework, Design patterns , Agile etc you can download the same from [here](http://tinyurl.com/4nvp9t)  or else you can catch me on my daily free training @ from [here](http://tinyurl.com/y4mbsn6)

### Why do we need locking?

In multi-user environment it's possible that multiple users can update the same record at the same time causing confusion between users. This issue is termed as concurrency.   
1a.JPG

### How can we solve concurrency problems?

Concurrency problems can be solved by implementing proper "Locking strategy". Locks prevent action on a resource to be performed when some other resource is already performing some action on it.

### What kind of confusion is caused because of concurrency?

There are 4 kinds of major problems caused because of concurrency, below table shows the details of the same.

|  |  |  |
| --- | --- | --- |
| **Problems** | **Short description** | **Explanation** |
| Dirty reads | "Dirty Read" occurs when one transaction is reading a record, which is part of a half, finished work of other transaction. | * User A and user B are seeing value as "5". * User B changes the value "5" to "2". * User A is still seeing the value as "5"....Dirty read has happened. |
| Unrepeatable read | In every data read if you get different values then it's an "Unrepeatable Read" problem. | * User A is seeing value as "5". * User B changes the value"5" to "2". * User A refreshes see values "5", he is surprised....unrepeatable read has happened. |
| Phantom rows | If "UPDATE" and "DELETE" SQL statements does not affect the data then it can be "Phantom Rows" problem. | * User A updates all value "5' to "2". * User B inserts a new record with value "2". * User A selects all record with value "2' if all the values have changed, he is surprised to still find value "2" records.....Phantom rows have been inserted. |
| Lost updates | "Lost Updates" are scenario where one updates which is successfully written to database is overwritten with other updates of other transaction. | * User A updates all value form "5" to "2". * User B comes and updates all "2" values to "5". * User A has lost all his updates. |

### So how can we solve the above problems?

By using optimistic or pessimistic locking, this article discusses the same.

2a.JPG

### What is Optimistic locking?

2.JPG

As the name suggests "optimistic" assumes that multiple transactions will work without affecting each other. In other words no locks are enforced while doing optimistic locking. The transaction just verifies that no other transaction has modified the data. In case of modification the transaction is rolled back.

### How does optimistic lock work?

You can implement optimistic locking by numerous ways but the fundamentals of implementing optimistic locking remains the same. It's a 5 step process as shown below:-

* Record the current timestamp.
* Start changing the values.
* Before updating check whether anyone else has changed the values by checking the old time stamp and new time stamp.
* If it's not equal rollbacks or else commit.

3.JPG

### What are the different solutions by which we can implement optimistic locking?

There are 3 primary ways by which we can implement optimistic locking in .NET:-

* **Datasets:**- Dataset by default implement optimistic locking. They do a check of old values and new values before updating.
* **Timestamp Data type:** - Create a timestamp data type in your table and while updating check if old timestamp is equal to new timestamp.
* **Check old and new value:** - Fetch the values, do the changes and while doing the final updates check if the old value and current values in database are equal. If they are not equal then rollback or else commits the values.

### Solution number 1:- Datasets

As said in the previous section dataset handles optimistic concurrency by itself. Below is a simple snapshot where we held the debug point on Adapter's update function and then changed the value from the SQL Server. When we ran the "update" function by removing the break point it threw "Concurrency" exception error as shown below.   
    
4.JPG  
  
If you run the profiler at the back end you can see it fires the update statement checking of the current values and the old values are same.

exec sp\_executesql N'UPDATE [tbl\_items] SET [AuthorName] = @p1 WHERE (([Id] =  
 @p2) AND ((@p3 = 1 AND [ItemName] IS NULL) OR ([ItemName] = @p4)) AND ((@p5 =   
1 AND [Type] IS NULL)   
OR ([Type] = @p6)) AND ((@p7 = 1 AND [AuthorName] IS NULL) OR ([AuthorName] =   
@p8)) AND ((@p9 = 1 AND [Vendor] IS NULL) OR ([Vendor] = @p10)))',N'@p1   
nvarchar(11),@p2 int,@p3   
int,@p4 nvarchar(4),@p5 int,@p6 int,@p7 int,@p8 nvarchar(18),@p9 int,@p10   
nvarchar(2)',@p1=N'this is new',@p2=2,@p3=0,@p4=N'1001',@p5=0,@p6=3,@p7=0,@p8=N'This is Old   
Author',@p9=0,@p10=N'kk'

In this scenario we were trying to change the field value "AuthorName" to "This is new" but while updating it makes a check with the old value "This is old author". Below is the downsized code snippet of the above SQL which shows the comparison with old value.

,@p8=N'This is Old Author'

### Solution number 2:- Use timestamp data type

The other way of doing optimistic locking is by using 'TimeStamp' data type of SQL Server. Time stamp automatically generates a unique binary number every time you update the SQL Server data. Time stamp data types are for versioning your record updates.   
    
5.JPG  
  
To implement optimistic locking we first fetch the old 'TimeStamp' value and when we are trying to update we check if the old time stamp is equal to the current time stamp as shown in the below code snippet.

update tbl\_items set itemname=@itemname where CurrentTimestamp=@OldTimeStamp

We then check if any updates has happened, in case updates has not happened then we raise a serious error '16' using SQL Server 'raiserror' as shown in the below code snippet.

if(@@rowcount=0)  
begin  
raiserror('Hello some else changed the value',16,10)  
end

If any concurrency violation takes place you should see the error propagated when you call 'ExecuteNonQuery' to the client side as shown in the below figure.   
    
6.JPG

### Solution number 3:- Check old values and new values

Many times we would like to check concurrency on only certain fields and omit fields like identity etc. For those kind of scenarios we can check the old value and the new value of the updated fields as shown in the below code snippet.

update tbl\_items set itemname=@itemname where itemname=@OldItemNameValue

### But it looks like by using optimistic locking concurrency problems are not really solved?

Yes, you said right. By using optimistic locking you only detect the concurrency problem. To solve concurrency issues from the roots itself we need to use pessimistic locking. Optimistic is like prevention while pessimistic locking is actually the cure.

### What is pessimistic locking?

7.JPG  
  
Pessimistic locking assumes that concurrency / collision issues will happen so a lock is placed on the records and then data is updated.

### How can we do pessimistic locking?

We can do pessimistic locking by specifying "IsolationLevel" in SQL Server stored procedures, ADO.NET level or by using transaction scope object.

### What kind of locks can be acquired by using pessimistic locking?

There are 4 kinds of locks you can acquire Shared, Exclusive, Update and intent. The first two are actual locks while the other two   
are hybrid locks and marker.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **When to use?** | **Reads Allowed** | **Writes Allowed** |
| Shared lock | When you want only to read and you do not want any other transactions to do update. | Yes | No |
| Exclusive | When you want to modify data and you do not want anyone to read the transaction, neither you want anyone to update. | No | No |
| Update lock | This is a hybrid lock. This lock is used when you want to do update operation which passes through multiple phases before the actual update happens. It first starts with shared lock in the read phase and then on the actual update it acquires an exclusive lock. |  |  |
|  | Read phase | Yes | No |
|  | Manipulating phase | Yes | No |
|  | Update phas | No | No |
| Intent Lock ( Demand locks) | Intent lock is for lock hierarchy. This lock is used when you want to lock resources down in the hierarchy. For example a shared intent lock on a table means shared locks are placed on pages and rows with the table. | NA | NA |
| Schema locks | When you are changing table structure. | No | No |
| Bulk update locks | Used when you are doing bulk updates | Table level No | Table level No |

### The update lock is confusing can you explain in detail?

The other locks are pretty straight forward; the update lock is confusing because of its hybrid nature. Many times before we update we read the record. So during read the lock is shared and while actually updating we would like to have an exclusive lock. Update locks are more of transient locks.

9.JPG

### So what are the different types of isolation levels and when should be used when?

There are 4 kinds of transaction isolation level, below is a simple table which shows when to use them and what locks they put.

|  |  |  |  |
| --- | --- | --- | --- |
| **Isolation Level** | **Read** | **Update** | **Insert** |
| Read Uncommitted | Reads data which is yet not committed. | Allowed | Allowed |
| Read Committed ( Default) | Reads data which is committed. | Allowed | Allowed |
| Repeatable Read | Reads data which is committed. | Not Allowed | Allowed |
| Serializable | Reads data which is committed. | Not Allowed | Not Allowed |

### How can we specify Isolation?

Isolation levels are features of RDBMS software, in other words they fundamentally really belong to SQL Server and not to Ado.NET, EF or LINQ. Said and done that you can always set the transaction isolation level from any of these components.   
    
11.JPG  
  
**Middle tier**

In middle tier you can specify isolation level using transaction scope object.

TransactionOptions TransOpt = New TransactionOptions();  
TransOpt.IsolationLevel = System.Transactions.IsolationLevel.ReadCommitted;   
using(TransactionScope scope = new   
TransactionScope(TransactionScopeOption.Required, TransOptions))  
{  
  
}

**ADO.NET**

You can also specify transaction isolation level using "SqlTransaction" object in ADO.NET.

SqlTransaction objtransaction =   
objConnection.BeginTransaction(System.Data.IsolationLevel.Serializable);

**SQL Server**

You can also specify isolation level in TSQL using 'SET TRANSACATION ISOLATION LEVEL' as shown in the below code snippet.

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

### Which transaction isolation level solves which problems from Concurrency?

Below is a chart which shows which transaction isolation level solves which problems of concurrency.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Read committed(S)** | **Repeatable read(I)** | **Serializable** | **Read Uncommitted** |
| Dirty reads | Solves | Solves | Solves | X |
| Lost updates | X | Solves | Solves | X |
| Non repeatable reads | X | Solves | Solves | X |
| Phantom rows | X | X | Solves | X |

### Solution 4:- Can we see how dirty reads are solved using Read Committed?

Some important Key points about read committed:-

* It's the default transaction isolation level for SQL Server.
* Its reads only committed data. In other words any uncommitted data is not read and blocked until the commit happens. Below figure explains the same in more detail. You can see the update

13.JPG  
  
If you want to see above things practically do the following:- 

* Open 2 Query windows fire an update transaction but do not commit.
* In the second window try firing select query it will show a blocked query as shown in the figure below.

14.JPG

### So is Read uncommitted opposite of Read Committed?

Yes, read uncommitted is opposite to read committed. When you set the transaction isolation level to read uncommitted, uncommitted data is also read.   
Some important key points for read committed:-

* Uncommitted is see so dirty read possible.
* No locks held.
* Useful when locking is not important and more important is concurrency and throughput.

If you want to test the same, fire the below SQL statement which is doing an update and roll back. The roll back happens after 20 seconds halt. Within that time if you fire a select query you will get the uncommitted data and after 20 seconds you will see the old data this committed data is rolled back.

set transaction isolation level read uncommitted  
Begin Tran  
  
Update customer set CustomerName='Changed' where CustomerCode='1001'  
WAITFOR DELAY '000:00:20'  
rollback tran

set transaction isolation level read uncommitted  
select \* from Customer where CustomerCode='1001'

### Solution 5:- Can we see how lost update and non-repeatable read are solved using repeatable read?

By setting isolation level to repeatable read no one can read and update the data. Some key points about repeatable read isolation level are as follows:- 

* Only committed data is read when repeatable transaction isolation level is set for select queries.
* When you select a record using repeatable read no one other transaction can update the record, but selects are possible.
* If repeatable transaction is set in update query, until the transaction finishes no one can read or update the same.
* When select and update query is set to repeatable read other transaction can insert new records. In other words phantom rows are possible.

If you want to test this isolation level, fire the syntax below and try firing select and update queries they will be blocked and after 50 seconds you should see the data.

set transaction isolation level repeatable read  
Begin Tran  
Update customer set CustomerName='Changed' where CustomerCode='1001'  
WAITFOR DELAY '000:00:50'  
rollback tran

If you fire the select query below in repeatable read mode you will not be able to update for 50 seconds until the transaction finishes.

set transaction isolation level repeatable read  
begin tran  
select \* from Customer where CustomerCode='1001'  
WAITFOR DELAY '000:00:50'  
commit tran

One important note you can add new records of customer code 1001, in other words phantom rows are possible.

### Solution 6:- How are Phantom row problems addressed using Serializable Isolation level?

This is the highest level of isolation level; in this other transactions cannot update, select and insert records. Some key points for serializable transaction are:-

* No other transaction can insert, update, delete or select when isolation level is serializable.
* Lot of blockings but all concurrency issues are solved.

set transaction isolation level serializable  
begin tran  
select \* from Customer where CustomerCode='1001'  
WAITFOR DELAY '000:00:50'  
commit tran

### In what scenarios should we use optimistic and pessimistic locking?

Working...

### What is a dead lock and how does shared lock avoid the same?

Still working...

### What are Lock hints?

Still working...

Bottom of Form