Telerik Academy

Transactions in ADO.NET and Entity Framework

Databases

Telerik Software Academy

http://academy.telerik.com

Telerik Academy

Table of Contents

1. Transactions in ADO.NET
   - Starting, Committing and Aborting Transactions
   - Distributed Transactions and `TransactionScope`
   - Implicit transactions
2. Transactions in Entity Framework (EF)
   - Optimistic Concurrency in EF

Telerik Academy

Using Transactions in ADO.NET

Using Transactions

- Working with transactions in ADO.NET
  - Beginning a transaction:

```
SqlTransaction trans =
  dbConnection.BeginTransaction();
```

  - Including a command in a given transaction:

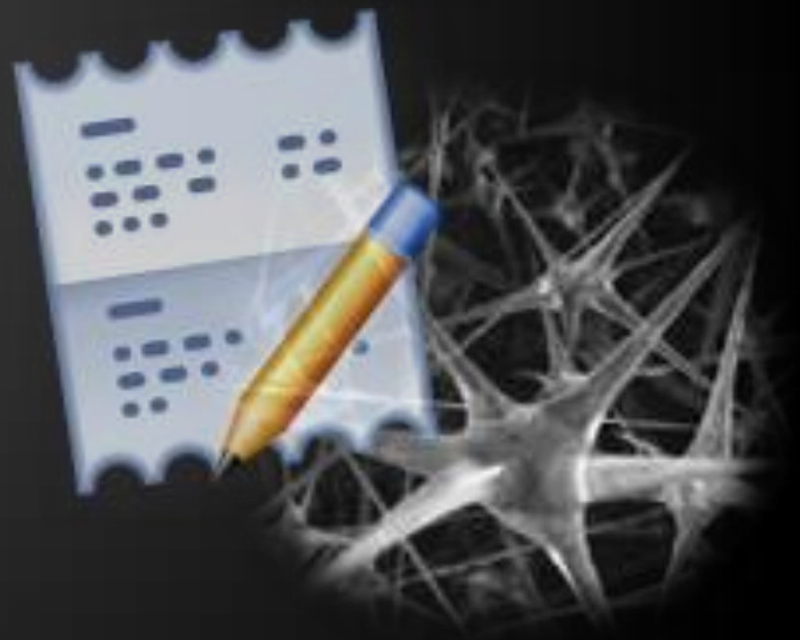
```
sqlCommand.Transaction = trans;
```

  - Committing / aborting a transaction:

```
trans.Commit();
trans.Rollback();
```

Using Transactions (2)

- The level of isolation is specified by the enumeration `IsolationLevel`:
  - `ReadUncommited`
  - `ReadCommited`
  - `RepeatableRead`
  - `Serializable`
  - `Snapshot`
  - `Chaos`
- Example:



```
SqlTransaction trans = dbConnection.
    BeginTransaction(IsolationLevel.Serializable);
```

ADO.NET Transactions – Example

```
SqlTransaction trans = dbCon.BeginTransaction(
  IsolationLevel.ReadCommitted);

SqlCommand cmd = dbCon.CreateCommand();
cmd.Transaction = trans;
try
{
  // Perform some SQL commands here …
  trans.Commit();
}
catch (SqlException e)
{
  Console.WriteLine("Exception: {0}", e.Message);
  trans.Rollback();
  Console.WriteLine("Transaction cancelled.");
}
```

Telerik Academy

ADO.NET Transactions

Live Demo

Telerik Academy

TransactionScope Class

- In ADO.NET you can implement implicit transactions using `TransactionScope`
  - Each operation in transaction scope joins the existing ambient transaction
  - MSDTC service may be required to be running
- Advantages of `TransactionScope`
  - Easier to implement
  - More efficient
  - Supports distributed transactions (MSDTC)

Using TransactionScope

- When instantiating TransactionScope
  - You either join the existing (ambient) transaction
  - Or create a new ambient transaction
  - Or you don't apply transactional logic at all
- The transaction coordinator (MSDTC) determines to which transaction to participate based on
  - If there is an open existing transaction
  - The value of the TransactionScopeOption parameter

Telerik Academy

The TransactionScope Class

- `TransactionScopeOption` specifies which transaction to be used (new / existing / none)

TransactionScopeOption	Transaction Already Exists?	Action
`Required` (default)	No	New transaction is created
	Yes	Existing ambient transaction is used
`RequiresNew`	No	New transaction is explicitly created
	Yes	
`Suppress`	No	No transaction is used at all
	Yes	

TransactionScope – Example

Create a new transaction

```
void RootMethod()
{
  using (TransactionScope tran = new TransactionScope())
  {
     /* Perform transactional work here */
     SomeMethod();
     tran.Complete();
  }
}

void SomeMethod()
{
  using (TransactionScope transaction =
    new TransactionScope(TransactionScopeOption.Required))
  {
     /* Perform transactional work here */
     transactions.Complete();
  }
}
```

Joins the existing transaction

Completing a TransactionScope

- The method `Complete()` informs the transaction coordinator that it is acceptable to commit the transaction
  - If not called, the transaction will be rolled back
  - Calling it doesn't guarantee that the transaction will be committed
- At the end of the `using` block of the root transaction, it is committed
  - Only if the root TS and all joined TS have called `Complete()`

Telerik Academy

TransactionScope

Live Demo

Telerik Academy

Using Transactions in Entity Framework

Telerik Academy

Transactions in Entity Framework (EF)

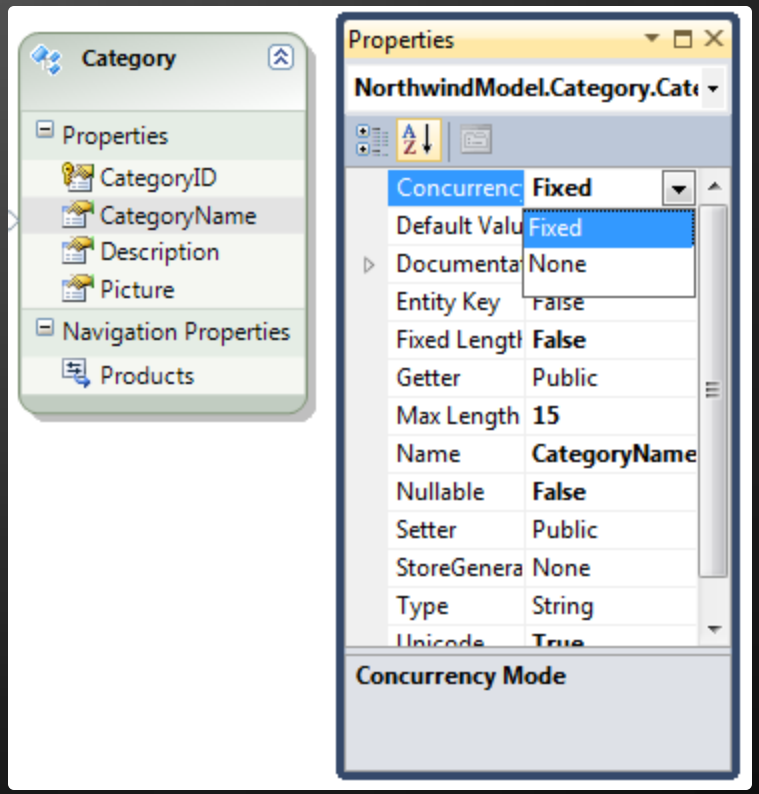
- In EF `ObjectContext.SaveChanges()` always operates in a transaction
  - Either all changes are persisted, or none of them
- Enable optimistic concurrency control by `ConcurrencyMode=Fixed` for certain property
  - `OptimisticConcurrencyException` is thrown when the changes cannot be persisted
  - Conflicts can be resolved by `ObjectContext.Refresh(StoreWins / ClientWins)`
- You still can use `TransactionScope` in EF

Transactions in EF – Example

```
NorthwindEntities context = new NorthwindEntities();

// Add a valid new category
Category newCategory = new Category() {
  CategoryName = "New Category",
  Description = "New category, just for testing"
};
context.Categories.AddObject(newCategory);

// Add an invalid new category
Category newCategoryLongName = new Category() {
  CategoryName = "New Category Looooooooong Name",
};
context.Categories.AddObject(newCategoryLongName);

// The entire transaction will fail due to
// insertion failure for the second category
context.SaveChanges();
```

Optimistic Concurrency in EF

- Enabling optimistic concurrency for e certain property of an entity in EF:



Telerik Academy

OptimisticConcurrencyException – Example

```
NorthwindEntities context = new NorthwindEntities();
Category newCategory = new Category() {
  CategoryName = "New Category" };
context.Categories.AddObject(newCategory);
context.SaveChanges();

// This context works in different transaction
NorthwindEntities anotherContext = new NorthwindEntities();

Category lastCategory =
  (from cat in anotherContext.Categories
   where cat.CategoryID == newCategory.CategoryID
   select cat).First();
lastCategory.CategoryName = lastCategory.CategoryName + " 2";
anotherContext.SaveChanges();

// This will cause OptimisticConcurrencyException if
// Categories.CategoryName has ConcurrencyMode=Fixed
newCategory.CategoryName = newCategory.CategoryName + " 3";
context.SaveChanges();
```

Telerik Academy

Transactions in ADO.NET and Entity Framework

Exercises

1. Suppose you are creating a simple engine for an ATM machine. Create a new database "ATM" in SQL Server to hold the accounts of the card holders and the balance (money) for each account. Add a new table `CardAccounts` with the following fields:

```
Id (int)
CardNumber (char(10))
CardPIN (char(4))
CardCash (money)
```

   Add a few sample records in the table.

Exercises (2)

2. Using transactions write a method which retrieves some money (for example $200) from certain account. The retrieval is successful when the following sequence of sub-operations is completed successfully:
   - A query checks if the given `CardPIN` and `CardNumber` are valid.
   - The amount on the account (`CardCash`) is evaluated to see whether it is bigger than the requested sum (more than `$200`).
   - The ATM machine pays the required sum (e.g. `$200`) and stores in the table `CardAccounts` the new amount (`CardCash = CardCash - 200`).

Exercises (3)

Why does the isolation level need to be set to "repeatable read"?

3. Extend the project from the previous exercise and add a new table `TransactionsHistory` with fields (`Id`, `CardNumber`, `TransactionDate`, `Ammount`) containing information about all money retrievals on all accounts.

   Modify the program logic so that it saves historical information (logs) in the new table after each successful money withdrawal.

   What should the isolation level be for the transaction?

Exercises (4)

4. * Write unite tests for all your transactional logic. Ensure you test all border cases.