Pennsylvania

Department of Human Services, Insurance & Aging

Bureau of Information Systems

Transaction Management in .NET

Version 1.7

September 30, 2015

**Table of Contents**

[Introduction 3](#_Toc370289614)

[Purpose 3](#_Toc370289615)

[What is a Transaction 4](#_Toc370289616)

[Distributed Transactions in Visual Basic .NET 5](#_Toc370289617)

[Introduction to Distributed Transactions 5](#_Toc370289618)

[Transaction Defined 5](#_Toc370289619)

[Transactions in ADO.NET 6](#_Toc370289620)

[Transactions in T/SQL 6](#_Toc370289621)

[Distributed Transactions Defined 6](#_Toc370289622)

[Transactional Components in .NET 7](#_Toc370289623)

[Transactional Web Services 7](#_Toc370289624)

[Summary 7](#_Toc370289625)

[Online Transaction Processing (OLTP) 8](#_Toc370289626)

[Online Transaction Processing Design Considerations 8](#_Toc370289627)

[Guidelines 9](#_Toc370289628)

[Considerations 10](#_Toc370289629)

[References 11](#_Toc370289630)

[Document Change Log 11](#_Toc370289631)

Transaction Management

Introduction

The technology responsible for making balanced, predictable exchanges is called transaction processing. Transactions ensure that data-oriented resources are not permanently updated unless all operations within the transactional unit complete successfully. By combining a set of related operations into a unit that either completely succeeds or completely fails, you can ensure data consistency, ensure orphan records are not created, simplify error recovery and make your application more reliable.

A transaction can be confined to a single data resource, such as a database or message queue. It is common for these data resources to provide local transaction capabilities.

It can also span multiple data resources referred to as Distributed transactions. It provides the capability to incorporate several distinct operations occurring on different systems into a single pass or fail action.

Purpose

The purpose of this document is to review the choices available for designing transaction in .NET and Enterprise services enabled applications.

What is a Transaction

Enterprise applications often require concurrent access to distributed data shared amongst multiple components, to perform operations on data. Such applications should maintain integrity of data (as defined by the business rules of the application) under the following circumstances:

* distributed access to a single resource of data, and
* access to distributed resources from a single application component.

In such cases, it may be required that a group of operations on (distributed) resources be treated as one unit of work. In a unit of work, all the participating operations should either succeed or fail and recover together. This problem is more complicated when

* a unit of work is implemented across a group of distributed components operating on data from multiple resources, and/or
* the participating operations are executed sequentially or in parallel threads requiring coordination and/or synchronization.

In either case, it is required that success or failure of a unit of work be maintained by the application. In case of a failure, all the resources should bring back the state of the data to the previous state (*i.e.,* the state prior to the commencement of the unit of work).

The concept of a transaction, and a transaction manager (or a transaction processing service) simplifies construction of such enterprise level distributed applications while maintaining integrity of data in a unit of work.

A transaction is a unit of work that has the following properties:

* **ATOMICITY:** A transaction should be done or undone completely and unambiguously. In the event of a failure of any operation, effects of all operations that make up the transaction should be undone, and data should be rolled back to its previous state.
* **CONSISTENCY:** A transaction should preserve all the invariant properties (such as integrity constraints) defined on the data. On completion of a successful transaction, the data should be in a consistent state. In other words, a transaction should transform the system from one consistent state to another consistent state. For example, in the case of relational databases, a consistent transaction should preserve all the integrity constraints defined on the data.
* **ISOLATION:** Each transaction should appear to execute independently of other transactions that may be executing concurrently in the same environment. The effect of executing a set of transactions serially should be the same as that of running them concurrently. This requires two things:
  + During the course of a transaction, intermediate (possibly inconsistent) state of the data should not be exposed to all other transactions.
  + Two concurrent transactions should not be able to operate on the same data. Database management systems usually implement this feature using locking.
* **DURABILITY:** The effects of a completed transaction should always be persistent.

These properties, called as **ACID** properties, guarantee that a transaction is never incomplete, the data is never inconsistent, concurrent transactions are independent, and the effects of a transaction are persistent.

Distributed Transactions in Visual Basic .NET

Introduction to Distributed Transactions

Online Transaction Processing (OLTP) systems are being installed into business organizations at an increasing rate. These systems update database records in real-time at the moment of a business transaction, like a bank ATM machine that must immediately record a cash withdrawal or a Web site for a retailer that must immediately reduce the amount of available inventory to reflect an order.

In contrast, there are Message Queue (MQ) systems that use less urgency to complete a transaction. For example, once an order is received over a Web site or by telephone, the fulfillment department for a retailer can ship the order as soon as possible. Although the amount of available inventory needs to be reduced in an accessible database, the actual order does not have to ship before the customer hangs up the telephone or leaves the Web site. The fulfillment of the order and the database records that log the shipment may be handled through a Message Queue.

<http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope.aspx>

Transaction Defined

A transaction is any modification to any record in a persistent data store like a database table. When you insert, update, or delete a record, it is called a transaction. On the other hand, when you select a record to read, it is not a transaction, because nothing changes in the database.

Some business rules require that a group of modifications be treated as a single transaction. For example, in accounting software, a debit and a credit must succeed or fail together because the accounts must "balance." If a debit succeeds and the credit fails, the accounts will not balance.

This kind of balanced modification collectively represents a single transaction. If an error occurs in the transaction, the entire transaction must be rolled back. Microsoft® SQL Server™, for instance, maintains a transaction log of modified records so that they can be returned to their original state if an error occurs during a transaction.

A transaction is initiated from a single program. This type of program executes a command to begin and commit (complete) a transaction. Between these two commands, the program can modify any quantity of records within the database. This kind of multiple-step modification collectively represents a single transaction.

Transactions in ADO.NET

The Connection object in ADO.NET provides the commands to begin and commit a transaction in a .NET program. This Connection object transaction includes all records that are modified between the begin and commit commands. However, the Connection object can be connected to only a single database.

For more information on ADO.NET transactions, refer to Transactions in ADO.NET in the Visual Studio .NET documentation.

Transactions in ADO.NET:

<http://msdn.microsoft.com/en-us/library/2k2hy99x.aspx>

Transactions in T/SQL

T/SQL provides commands that begin and commit a transaction in a SQL Server stored procedure. This kind of transaction includes all records that are modified between the two commands, begin and commit. Like the Connection object in ADO.NET, T/SQL transactions are applied to a single database.

For more information on T/SQL transactions refer to SQL Server Books Online.

Distributed Transactions Defined

A distributed transaction involves more than one database. A single distributed transaction may involve a company's inventory, customer relations management, and accounts receivable databases. For example, when an order is received on a Web site, the program may need to modify product inventory levels in one database, customer profile information in another database, and an account balance in yet another database.

In .NET, the System.EnterpriseServices assembly provides the commands to begin and commit a distributed transaction. This type of transaction includes all the records that are modified between these commands. This assembly transaction works with connections to SQL Server or other relational database servers that support distributed transactions.

Before your component can participate in a distributed transaction, it will need a strong name. This strong name is created by the Sn.exe tool outside of the IDE.

For more information on the Strong Name tool, see <http://msdn.microsoft.com/en-us/library/k5b5tt23.aspx>

For more information on assemblies in the Microsoft .NET Framework, see Understanding Assemblies and the .NET Framework SDK.

Understanding Assemblies:

http://msdn.microsoft.com/en-us/library/ms973231.aspx

Transactional Components in .NET

In this section, you will learn how to build a transactional component that is called from a form. In this example, Visual Basic .NET is used to create both the component and the form. The information you supply in the form will then be applied to the database as part of the transaction.

VB.NET Example:   
<http://msdn.microsoft.com/en-us/library/ms973833.aspx>

Transactional Web Services

One of the new project types in .NET is Web services. A Web service is essentially a function that gets invoked remotely over the Internet. Typically, a transactional component runs on an application server; client computers invoke the transaction by remotely invoking a transactional component.  
  
Participating in Transactions in XML Web Services Created Using ASP.NET

<http://msdn.microsoft.com/en-us/library/vstudio/85f292h1(v=vs.100).aspx>

Summary

Online Transaction Processing (OLTP) systems require transactions that complete at the same moment as a business transaction. Transactions can be written in ADO.NET or T/SQL when a single database is modified.

A distributed transaction is a transaction that involves multiple databases. Using .NET, you can write a distributed transaction as a transactional component or a transactional Web service. A transactional component is called within a .NET program such as a Windows Form.

Your program can commit a transaction automatically or manually. Using the ContextUtil object in conjunction with the SetComplete() and SetAbort() methods, your program can manually control the transaction. When used with exception handling, the manual approach provides a more elegant solution than automation.

.NET provides a rich set of tools for building enterprise applications. The support for distributed transactions is one of many features included to make this a great platform for development.

Online Transaction Processing (OLTP)

Online Transaction processing database applications are optimal for managing changing data, and usually have a large number of users who will be simultaneously performing transactions that change real-time data. Although individual requests by users for data tend to reference few records, many of these requests are being made at the same time. Common examples of these types of databases are airline ticketing systems and banking transaction systems. The primary concerns in this type of application are concurrency and atomicity.

Concurrency controls in a database system ensure that two users cannot change the same data, or that one user cannot change a piece of data before another user is done with it. For example, if you are talking to an airline ticket agent to reserve the last available seat on a flight and the agent begins the process of reserving the seat in your name, another agent should not be able to tell another passenger that the seat is available.

Atomicity ensures that all of the steps involved in a transaction complete successfully as a group. If any step fails, no other steps should be completed. For example, a banking transaction may involve two steps: taking funds out of your checking account and placing them into your savings account. If the step that removes the funds from your checking account succeeds, you want to make sure that the funds are placed into your savings account or put back into your checking account.

Online Transaction Processing Design Considerations

Transaction processing system databases should be designed to promote:

* Controlled data placement.
* I/O bottlenecks are a big concern for OLTP systems due to the number of users modifying data all over the database. Determine the likely access patterns of the data and place frequently accessed data together. Use filegroups and RAID (redundant array of independent disks) systems to assist in this.
* Short transactions to minimize long-term locks and improve concurrency.
* Avoid user interaction during transactions. Whenever possible, execute a single stored procedure to process the entire transaction. The order in which you reference tables within your transactions can affect concurrency. Place references to frequently accessed tables at the end of the transaction to minimize the duration that locks are held.
* Online backup.
* OLTP systems are often characterized by continuous operations (24 hours a day, 7 days a week) for which downtime is kept to an absolute minimum. Although Microsoft® SQL Server™ 2008 can back up a database while it is being used, schedule the backup process to occur during times of low activity to minimize effects on users.
* High normalization of the database.
* Reduce redundant information as much as possible to increase the speed of updates and hence improve concurrency. Reducing data also improves the speed of backups because less data needs to be backed up.
* Little or no historical or aggregated data.
* Data that is rarely referenced can be archived into separate databases, or moved out of the heavily updated tables into tables containing only historical data. This keeps tables as small as possible, improving backup times and query performance.
* Careful use of indexes.
* Indexes must be updated each time a row is added or modified. To avoid over-indexing heavily updated tables, keep indexes narrow. Use the Index Tuning Wizard to design your indexes.
* Optimum hardware configuration to handle the large numbers of concurrent users and quick response times required by an OLTP system.

Guidelines

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Category** | **Technology** | **Advantages** | **Disadvantages** | **When to Use** |
| Manual | ADO.Net | Fast  Easy to transition to ADO.Net version 2’s technologies  Database agnostic.  Moderately easy to migrate to COM+/DTC transactions  Easy control over beginning and end of transaction. | Cannot support multiple resource managers.  Holds connection open for duration of transaction. | Business logic resides outside of database.  Transaction will never update more than one resource manager, nor will it participate in a calling transaction that updates more than one resource manager. |
| DB | Fast | Cannot support multiple resource managers.  Holds connection open for duration of transaction | Business logic resides in the database.  Transaction will never update more than one resource manager, nor will it participate in a calling transaction that updates more than one resource manager. |
| Automatic | COM+/DTC | Supports multiple resource updates | Much slower  Can leave persistent distributed transaction locks in DB that will require manual intervention. | Transactions that participate in multiple resource updates.  Transactions update a non database resource.  Connection object cannot be passed around. |

Considerations

Each transaction technique offers trade-offs with respect to application performance and code maintainability. Running a database transaction implemented in a stored procedure offers the best performance as it needs only a single round trip to the database and gives the flexibility of explicitly controlling the transaction boundary.

***Manual transactions*** using ADO.NET transaction objects are easy to code and give you the flexibility of controlling the transaction boundaries with explicit instructions to begin and end the transaction. The trade off for this ease and flexibility is that it incurs a performance cost for extra round trips to the database to complete the transaction.

***Automatic transactions*** will be the only choice if your transaction spans multiple transaction-aware resource managers which could include SQL Server databases, MSMQ Message Queues, etc. It greatly simplifies the application design and reduces coding requirements. However, since COM+ service does all the coordination work, it may have some extra overhead.

References

Introduction to Distributed Transactions

<http://technet.microsoft.com/en-us/library/cc753874(v=WS.10).aspx>

Distributed Transactions in Visual Basic .NET

<http://msdn.microsoft.com/en-us/library/ms254973.aspx>

Using COM+ Services in .NET

<http://msdn.microsoft.com/en-us/library/ms973809.aspx>

Understanding Enterprise Services (COM+) in .NET

<http://msdn.microsoft.com/en-us/library/ms973847.aspx>

Document Change Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Change Date** | **Version** | **CR #** | **Change Description** | **Author and Organization** |
| 06/15/03 | 1.0 |  | Initial creation. | Deloitte Consulting |
| 07/17/03 | 1.1 |  | Modifications suggested in the Componentware meeting (7/16/03) | Deloitte Consulting |
| 07/27/03 | 1.2 |  | Research and verification | Deloitte Consulting |
| 8/26/03 | 1.3 |  | Scope changed to include all transaction processing | Deloitte Consulting |
| 11/10/03 | 1.4 |  | Added table for guidelines | Deloitte Consulting |
| 01/20/04 | 1.5 |  | Updated and Finalized for Submission | Deloitte Consulting |
| 10/21/13 | 1.6 |  | Updated outdated links | Steve Isleib |
| 09/30/15 | 1.7 |  | Name change from DPW to DHS | Virjean Dauksha |