

Managing SQL Database Transactions and Concurrency

Lab 1 – SQL Server Transactions

# Overview

In this lab, you need to evaluate how transactions from multiple users affect your system.

Before starting this lab, you should view **Module 1 – Creating Stored Procedures** in the course *Creating Programmatic SQL Database Objects*. Then, if you have not already done so, follow the instructions in the **Getting Started** document for this course to set up the lab environment.

If you find some of the challenges difficult, don’t worry – you can find suggested solutions for all of the challenges in the **Lab Solution** folder for this module.

# What You’ll Need

To complete the labs, you will need the following:

* An Azure SQL Database instance with the AdventureWorksLT sample database. Review the Getting Started document for information about how to provision this.
* The lab files for this course

# Challenge 1: Read Uncommitted Isolation Level

In this exercise, you will investigate the Read Uncommitted isolation level.

## Create Two Sessions

1. Open SQL Server Management Studio, connect to your AdventureworksLT database, and open two query windows.
2. Arrange your query windows side-by-side so that you can see both of them. You can right click a query tab and click **New Verical Tab Group** to achieve this.
3. Save the query on the left as **Query A** and save the query on the right as **Query B**.

## Test the Read Uncommitted Isolation Level

1. In **Query A**, type and execute the following query:

BEGIN TRANSACTION

UPDATE SalesLT.Product

SET ListPrice= 11111

WHERE ProductID= 680;

GO

1. In **Query B**, type and execute the following query:

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

GO

SELECT ProductID, ListPrice

FROM SalesLT.Product

WHERE ProductID= 680;

GO

1. Note that the updated ListPrice is displayed.
2. In **Query A**, type and execute the following query:

ROLLBACK;

GO

1. In **Query B**, type and execute the following query:

SELECT ProductID, ListPrice

FROM SalesLT.Product

WHERE ProductID= 680;

GO

1. Note that the updated ListPrice never actually existed.

## Test the Read Committed Isolation Level

1. In **Query A**, type and execute the following query:

BEGIN TRANSACTION

UPDATE SalesLT.Product

SET ListPrice= 11111

WHERE ProductID= 680;

GO

1. In **Query B**, type and execute the following query:

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

GO

SELECT ProductID, ListPrice

FROM SalesLT.Product

WHERE ProductID= 680;

GO

1. Note that the original ListPrice is displayed.
2. In **Query A**, type and execute the following query:

ROLLBACK;

GO

1. Note that this is the default behavior in Azure. The default behavior in an on-premises system is to block the second transaction.

## Test the Serializable Isolation Level

1. In **Query A**, type and execute the following query:

BEGIN TRANSACTION

UPDATE SalesLT.Product

SET ListPrice= 11111

WHERE ProductID= 680;

GO

1. In **Query B**, type and execute the following query:

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

GO

SELECT ProductID, ListPrice

FROM SalesLT.Product

WHERE ProductID= 680;

GO

1. Note that the query is blocked.
2. In **Query A**, type and execute the following query:

ROLLBACK;

GO

1. Note that **Query B** can now complete.