Implementing In-Memory SQL Database Objects

Lab 4 – Creating Native Stored Procedures

Overview

You are planning to optimize some database workloads by using the in-memory database capabilities of SQL Server 2016. You will create memory-optimized tables and natively compiled stored procedures to optimize OLTP workloads.

Before starting this lab, you should view **Module 4 – Implementing Native Stored Procedures** in the course *Implementing In-Memory 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:

- A SQL Server instance with the AdventureWorksDW sample database. Review the Getting Started document for information about how to provision this.
- The lab files for this course

Challenge 1: Create a Memory Optimized Table

The Adventure Works website, through which customers can order goods, uses the InternetSales database. The database already includes tables for sales transactions, customers, and payment types. You need to add a table to support shopping cart functionality. The shopping cart table will experience a high volume of concurrent transactions, so, to maximize performance, you want to implement it as a memory-optimized table.

Add a Filegroup for Memory-Optimized Data

1. Add a filegroup for memory-optimized data to the InternetSales database.
2. Add a file for memory-optimized data to the InternetSales database. You should store the file in the filegroup that you created in the previous step.

Create a Memory-Optimized Table

1. Create a memory-optimized table named ShoppingCart in the InternetSales database, with the durability option set to SCHEMA_AND_DATA.
2. The table should include the following columns:

- SessionID: integer
- TimeAdded: datetime
- CustomerKey: integer
- ProductKey: integer
- Quantity: integer

3. The table should include a composite primary key nonclustered index on the SessionID and ProductKey columns.
4. To test the table, insert the following rows, and then write and execute a SELECT statement to return all of the rows.

SessionID	TimeAdded	CustomerKey	ProductKey	Quantity
1	<Time>	2	3	1
1	<Time>	2	4	1

For <Time>, use whatever the current time is.

Challenge 2: Using Natively Compiled Stored Procedures

The Adventure Works website now includes a memory-optimized table. You now want to create a natively compiled stored procedure to take full advantage of the performance benefits of in-memory tables.

Create Natively Compiled Stored Procedures

1. Create a natively compiled stored procedure named AddItemToCart. The stored procedure should include a parameter for each column in the ShoppingCart table, and should insert a row into the ShoppingCart table by using a SNAPSHOT isolation transaction.
2. Create a natively compiled stored procedure named DeleteItemFromCart. The stored procedure should include SessionID and ProductKey parameters, and should delete matching rows from the ShoppingCart table by using a SNAPSHOT isolation transaction.

3. Create a natively compiled stored procedure named EmptyCart. The stored procedure should include SessionID parameters, and should delete matching rows from the ShoppingCart table by using a SNAPSHOT isolation transaction.
4. To test the AddItemToCart procedure, write and execute a Transact-SQL statement that calls AddItemToCart to add the following items, and then write and execute a SELECT statement to return all of the rows in the ShoppingCart table.

SessionID	TimeAdded	CustomerKey	ProductKey	Quantity
1	<Time>	2	3	1
1	<Time>	2	4	1
3	<Time>	2	3	1
3	<Time>	2	4	1

For <Time>, use whatever the current time is.

5. To test the DeleteItemFromCart procedure, write and execute a Transact-SQL statement that calls DeleteItemFromCart to delete any items where SessionID is equal to 3 and the product key is equal to 4, and then write and execute a SELECT statement to return all of the rows in the ShoppingCart table.
6. To test the EmptyCart procedure, write and execute a Transact-SQL statement that calls EmptyCart to delete any items where SessionID is equal to 3, and then write and execute a SELECT statement to return all of the rows in the ShoppingCart table.
7. Close SQL Server Management Studio without saving any changes.