Creating a Memory-Optimized Table and a Natively Compiled Stored Procedure

- 03/16/2017
- 5 minutes to read

Applies to: SQL Server (all supported versions)

This topic contains a sample that introduces you to the syntax for In-Memory OLTP.

To enable an application to use In-Memory OLTP, you need to complete the following tasks:

- Create a memory-optimized data filegroup and add a container to the filegroup.
- Create memory-optimized tables and indexes. For more information, see CREATE TABLE (Transact-SQL).
- Load data into the memory-optimized table and update statistics after loading the data and before creating the compiled stored procedures. For more information, see Statistics for Memory-Optimized Tables.
- Create natively compiled stored procedures to access data in memory-optimized tables. For more information, see CREATE PROCEDURE (Transact-SQL). You can also use a traditional, interpreted Transact-SQL to access data in memory-optimized tables.
- As needed, migrate data from existing tables to memory-optimized tables.

Background on In-Memory objects

For information on how to use SQL Server Management Studio to create memory-optimized tables, see SQL Server Management Studio Support for In-Memory OLTP.

Natively compiled stored procedures

Natively compiled stored procedures are Transact-SQL stored procedures compiled to native code, and that access memory-optimized tables. Natively compiled stored procedures allow for efficient execution of the queries and business logic in the stored procedure. For more details about the native compilation process, see Native Compilation of Tables and Stored Procedures. For more information about migrating disk-based stored procedures to natively compiled stored procedures, see Migration Issues for Natively Compiled Stored Procedures.

Note

One difference between interpreted (disk-based) stored procedures and natively compiled stored procedures is that an interpreted stored procedure is compiled at first execution, whereas a natively compiled stored procedure is compiled when it is created. With natively compiled stored procedures, many

error conditions can be detected at create time and will cause creation of the natively compiled stored procedure to fail (such as arithmetic overflow, type conversion, and some divide-by-zero conditions). With interpreted stored procedures, these error conditions typically do not cause a failure when the stored procedure is created, but all executions will fail.

Code example in T-SQL

The following code sample requires a directory called c:\Data\ .

SQLCopy

```
CREATE DATABASE imoltp
GO

-------------------------------------
-- create database with a memory-optimized
-- filegroup and a container.

ALTER DATABASE imoltp ADD FILEGROUP imoltp_mod
    CONTAINS MEMORY_OPTIMIZED_DATA;

ALTER DATABASE imoltp ADD FILE (
    name='imoltp_mod1', filename='c:\data\imoltp_mod1')
    TO FILEGROUP imoltp_mod;

ALTER DATABASE imoltp
    SET MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT = ON;
GO

USE imoltp
GO

-- Create a durable (data will be persisted) memory-optimized table
-- two of the columns are indexed.
```

```
CREATE TABLE dbo.ShoppingCart (
    ShoppingCartId INT IDENTITY(1,1) PRIMARY KEY NONCLUSTERED,
    UserId INT NOT NULL INDEX ix_UserId NONCLUSTERED
        HASH WITH (BUCKET_COUNT=1000000),
    CreatedDate DATETIME2 NOT NULL,
    TotalPrice MONEY
    ) WITH (MEMORY_OPTIMIZED=ON)
GO

-- Create a non-durable table. Data will not be persisted,
-- data loss if the server turns off unexpectedly.

CREATE TABLE dbo.UserSession (
   SessionId INT IDENTITY(1,1) PRIMARY KEY NONCLUSTERED
        HASH WITH (BUCKET_COUNT=400000),
   UserId int NOT NULL,
   CreatedDate DATETIME2 NOT NULL,
   ShoppingCartId INT,
   INDEX ix_UserId NONCLUSTERED
        HASH (UserId) WITH (BUCKET_COUNT=400000)
    )
    WITH (MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY)
GO

-- insert data into the tables
INSERT dbo.UserSession VALUES (342, SYSDATETIME(), 4);
INSERT dbo.UserSession VALUES (65, SYSDATETIME(), NULL)
INSERT dbo.UserSession VALUES (8798, SYSDATETIME(), 1)
INSERT dbo.UserSession VALUES (80, SYSDATETIME(), NULL)
INSERT dbo.UserSession VALUES (4321, SYSDATETIME(), NULL)
INSERT dbo.UserSession VALUES (8578, SYSDATETIME(), NULL)
```

```
INSERT dbo.ShoppingCart VALUES (8798, SYSDATETIME(), NULL)
INSERT dbo.ShoppingCart VALUES (23, SYSDATETIME(), 45.4)
INSERT dbo.ShoppingCart VALUES (80, SYSDATETIME(), NULL)
INSERT dbo.ShoppingCart VALUES (342, SYSDATETIME(), 65.4)
GO

-- Verify table contents.

SELECT * FROM dbo.UserSession;
SELECT * FROM dbo.ShoppingCart;
GO

-- Update statistics on memory-optimized tables;

UPDATE STATISTICS dbo.UserSession  WITH FULLSCAN, NORECOMPUTE;
UPDATE STATISTICS dbo.ShoppingCart WITH FULLSCAN, NORECOMPUTE;
GO

-- in an explicit transaction, assign a cart to a session
-- and update the total price.
-- SELECT/UPDATE/DELETE statements in explicit transactions.

BEGIN TRAN;
   UPDATE dbo.UserSession SET ShoppingCartId=3 WHERE SessionId=4;
   UPDATE dbo.ShoppingCart SET TotalPrice=65.84 WHERE ShoppingCartId=3;
COMMIT;
GO

-- Verify table contents.

SELECT *
    FROM dbo.UserSession u
```

```
      JOIN dbo.ShoppingCart s on u.ShoppingCartId=s.ShoppingCartId
   WHERE u.SessionId=4;
GO

-- Natively compiled stored procedure for assigning
-- a shopping cart to a session.

CREATE PROCEDURE dbo.usp_AssignCart @SessionId int
   WITH NATIVE_COMPILATION, SCHEMABINDING
AS
BEGIN ATOMIC
   WITH (TRANSACTION ISOLATION LEVEL = SNAPSHOT,
      LANGUAGE = N'us_english');

 DECLARE @UserId INT,
        @ShoppingCartId INT;

 SELECT @UserId=UserId, @ShoppingCartId=ShoppingCartId
 FROM dbo.UserSession WHERE SessionId=@SessionId;

 IF @UserId IS NULL
   THROW 51000, N'The session or shopping cart does not exist.', 1;

 UPDATE dbo.UserSession
   SET ShoppingCartId=@ShoppingCartId WHERE SessionId=@SessionId;
END
GO

EXEC usp_AssignCart 1;
GO

-- natively compiled stored procedure for inserting
```

```
-- a large number of rows this demonstrates the
-- performance of native procs
CREATE PROCEDURE dbo.usp_InsertSampleCarts @InsertCount int
    WITH NATIVE_COMPILATION, SCHEMABINDING
AS
BEGIN ATOMIC
    WITH (TRANSACTION ISOLATION LEVEL = SNAPSHOT,
        LANGUAGE = N'us_english');

  DECLARE @i int = 0;

  WHILE @i < @InsertCount
  BEGIN
    INSERT INTO dbo.ShoppingCart VALUES (1, SYSDATETIME() , NULL)
    SET @i += 1
  END

END
GO

-- insert 1,000,000 rows
 EXEC usp_InsertSampleCarts 1000000
 GO

---- verify the rows have been inserted
 SELECT COUNT(*) FROM dbo.ShoppingCart
 GO

-- Sample memory-optimized tables for
-- sales orders and sales order details.
CREATE TABLE dbo.SalesOrders
(
```

```
  so_id INT NOT NULL PRIMARY KEY NONCLUSTERED,
  cust_id INT NOT NULL,
  so_date DATE NOT NULL INDEX ix_date NONCLUSTERED,
  so_total MONEY NOT NULL,
  INDEX ix_date_total NONCLUSTERED
      (so_date DESC, so_total DESC)
) WITH (MEMORY_OPTIMIZED=ON);
GO

CREATE TABLE dbo.SalesOrderDetails
(
  so_id INT NOT NULL,
  lineitem_id INT NOT NULL,
  product_id INT NOT NULL,
  unitprice MONEY NOT NULL,

  CONSTRAINT PK_SOD PRIMARY KEY NONCLUSTERED
      (so_id,lineitem_id)
) WITH (MEMORY_OPTIMIZED=ON)
GO

-- Memory-optimized table type for collecting
-- sales order details.

CREATE TYPE dbo.SalesOrderDetailsType AS TABLE
(
  so_id INT NOT NULL,
  lineitem_id INT NOT NULL,
  product_id INT NOT NULL,
  unitprice MONEY NOT NULL,

  PRIMARY KEY NONCLUSTERED (so_id,lineitem_id)
```

```
) WITH (MEMORY_OPTIMIZED=ON)
GO

-- stored procedure that inserts a sales order,
-- along with its details.

CREATE PROCEDURE dbo.InsertSalesOrder
    @so_id INT, @cust_id INT,
    @items dbo.SalesOrderDetailsType READONLY
WITH NATIVE_COMPILATION, SCHEMABINDING
AS BEGIN ATOMIC WITH
(
  TRANSACTION ISOLATION LEVEL = SNAPSHOT,
  LANGUAGE = N'us_english'
)
  DECLARE @total MONEY
  SELECT @total = SUM(unitprice) FROM @items

  INSERT dbo.SalesOrders
      VALUES (@so_id, @cust_id, getdate(), @total)

  INSERT dbo.SalesOrderDetails
      SELECT so_id, lineitem_id, product_id, unitprice
      FROM @items
END
GO

-- Insert a sample sales order.
DECLARE @so_id INT = 18,
      @cust_id INT = 8,
      @items dbo.SalesOrderDetailsType;
```

```
INSERT @items  VALUES
	(@so_id, 1, 4, 43),
	(@so_id, 2, 3, 3),
	(@so_id, 3, 8, 453),
	(@so_id, 4, 5, 76),
	(@so_id, 5, 4, 43);

EXEC dbo.InsertSalesOrder @so_id, @cust_id, @items;
GO

-- verify the content of the tables
SELECT
	so.so_id,
	so.so_date,
	sod.lineitem_id,
	sod.product_id,
	sod.unitprice
FROM dbo.SalesOrders so
   JOIN dbo.SalesOrderDetails sod on so.so_id=sod.so_id
ORDER BY so.so_id, sod.lineitem_id
```