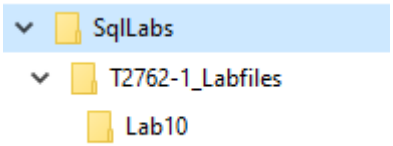
Lab Manual

T2762-1, Developing SQL Databases, Part 1

Lab environment overview

- **Tip**: open this lab manual inside the lab virtual machine. This makes it easier to copy and paste from this document.
    - For Virsoft VM environment, pasting from outside the VM requires you to right-click the notepad at the top left and select "paste…"
- Your machine name is ***NORTH***.
- If not already done for you, log in to Windows using
    - ***Student***
    - ***myS3cret***
- Copy the lab files to your virtual machine. You find them here: https://karaszi.com/training
    - Use a web browser in the above virtual machine
- Extract the files, so you in the root of your C: drive end up with a folder structure looking like below



- Use ***SQL Server Management Studio (SSMS) or Azure Data Studio (ADS)*** to do the labs, whichever you prefer.
- Login to the SQL Server named “North” using Windows authentication.
    - (There is a default SQL Server instance on the machine and there is a Windows login in your SQL Server for your Windows account, which is a sysadmin.)
- You will be using the database named **Adventureworks**, unless other is specified.

- You can always revert/reset your databases to "default".
    - There are three bat files in the C:\SqlLabs folder that does this (RESTORE), one for each database.
        - You might need to download these files first, from https://files.karaszi.com/rot/courses/RestoreLabDatabases.zip
    - Note: **Run as Administrator**
- The lab answers are *not* designed to be used independently. Use the lab instructions, and check the answers when you get stuck, etc.

Lab 2: Designing and implementing tables

Ex 1. Create a schema

1. Create a schema named mySchema.

Ex 2. Create a table

You will decide on data types and nullability for the ***Persons*** table and create that table. Below you find the information (columns) that we will have in this table. Create the table in the ***mySchema*** schema. Use whatever data types you find suitable (there is no one right answer).

Don't worry about Primary Keys and such since we will discuss that in the next module.

1. CustomerID. This will store integers (1, 2, 3 etc).
2. FirstName
3. LastName
4. MiddleName (should allow NULL)
5. CompanyName
6. Phone
7. CreationDate

Ex 3. Add a column

You realized that you also want to have option to store who last modified a row.

1. Add a ModifiedBy column.

Lab 1 answer suggestions

Ex 1. Create a schema

```
DROP SCHEMA IF EXISTS mySchema
GO

CREATE SCHEMA mySchema
GO
```

Ex 2. Create a table

```
DROP TABLE IF EXISTS mySchema.Persons
GO

CREATE TABLE mySchema.Persons
(CustomerID int NOT NULL
,FirstName nvarchar(30) NOT NULL
,LastName nvarchar(40) NOT NULL
,MiddleName nvarchar(10) NULL
,CompanyName nvarchar(100) NOT NULL
,Phone varchar(20) NOT NULL
,CreationDate date NOT NULL)
GO
```

Ex 3. Add a column

```
ALTER TABLE mySchema.Persons
ADD ModifiedBy varchar(30) NULL
```

Lab 3: Ensuring data integrity using constraints

Ex 1. Add identity property

You have below table structure. Change it so that the CustomerID column has the identity property, start value 1 and increment 1. After (re)-creating the table, perform an insert into the table.

```
DROP TABLE IF EXISTS mySchema.Persons
GO

CREATE TABLE mySchema.Products
(CustomerID int NOT NULL
,FirstName nvarchar(30) NOT NULL
,LastName nvarchar(40) NOT NULL
,MiddleName nvarchar(10) NULL
,CompanyName nvarchar(100) NOT NULL
,Phone varchar(20) NOT NULL
,CreationDate date NOT NULL
,ModifiedBy varchar(30) NULL
)
GO
```

Ex 2. Add default constraints

Using your modified CREATE script from above step, add below defaults. Name the default constraints so you more easily can drop them later.

- The CreationDate column should have default for when the row is inserted.
- The ModifiedBy column should have default for the user name doing the insert (you can use the SUSER_SNAME() function for this).

Test above by doing an INSERT and then SELECT.

Ex 3. Add primary key and foreign key constraints

Using your CREATE script from above step you will now add primary key and foreign key constraints. I.e., you will drop the table (if exists) and re-create it.

First, execute below. You will later add a foreign key that references to below table:

```
DROP TABLE IF EXISTS mySchema.Regions
GO

CREATE TABLE mySchema.Regions
(RegionID int IDENTITY(1, 1) NOT NULL PRIMARY KEY
,RegionName varchar(40) NOT NULL)
GO

INSERT INTO mySchema.Regions(RegionName)
VALUES('Way up north')
GO
```

Now, use the CREATE script from above step and add below:

- Make the CustomerID column a primary key.
- Add a RegionID column that has a foreign key that references above table.
- Perform an INSERT to your table.
- Feel free to play around with DELETE and UPDATE if time permits.

Lab 3 answer suggestions

Ex 1. Add identity property

```
DROP TABLE IF EXISTS mySchema.Persons
GO

CREATE TABLE mySchema.Persons
(CustomerID int IDENTITY(1, 1) NOT NULL
,FirstName nvarchar(30) NOT NULL
,LastName nvarchar(40) NOT NULL
,MiddleName nvarchar(10) NULL
,CompanyName nvarchar(100) NOT NULL
,Phone varchar(20) NOT NULL
,CreationDate date NOT NULL
,ModifiedBy varchar(30) NULL
)
GO
```

Ex 2. Add default constraints

```
DROP TABLE IF EXISTS mySchema.Persons
GO

CREATE TABLE mySchema.Persons
(CustomerID int IDENTITY(1,1) NOT NULL
,FirstName nvarchar(30) NOT NULL
,LastName nvarchar(40) NOT NULL
,MiddleName nvarchar(10) NULL
,CompanyName nvarchar(100) NOT NULL
,Phone varchar(20) NOT NULL
,CreationDate date NOT NULL CONSTRAINT d_CreationDate DEFAULT (GETDATE())
,ModifiedBy varchar(30) NULL CONSTRAINT d_ModifiedBy DEFAULT (SUSER_SNAME())
)
GO

INSERT INTO mySchema.Persons
(FirstName, LastName, MiddleName, CompanyName, Phone)
VALUES('Hunter', 'Thomson', 'S', 'Gonzo', '2222-222')

SELECT * FROM mySchema.Persons
```

Ex 3. Add primary key and foreign key constraints

```
DROP TABLE IF EXISTS mySchema.Persons
DROP TABLE IF EXISTS mySchema.Regions
GO

CREATE TABLE mySchema.Regions
(RegionID int IDENTITY(1, 1) NOT NULL PRIMARY KEY
,RegionName varchar(40) NOT NULL)
GO

INSERT INTO mySchema.Regions(RegionName)
VALUES('Way up north')
GO

DROP TABLE IF EXISTS mySchema.Persons
GO

CREATE TABLE mySchema.Persons
(CustomerID int IDENTITY(1,1) CONSTRAINT PK_Persons PRIMARY KEY NOT NULL
,FirstName nvarchar(30) NOT NULL
,LastName nvarchar(40) NOT NULL
,MiddleName nvarchar(10) NULL
,CompanyName nvarchar(100) NOT NULL
,Phone varchar(20) NOT NULL
,CreationDate date NOT NULL CONSTRAINT d_CreationDate DEFAULT (GETDATE())
,ModifiedBy varchar(30) NULL CONSTRAINT d_ModifiedBy DEFAULT (SUSER_SNAME())
,RegionID int NOT NULL CONSTRAINT FK_Persons__Regions FOREIGN KEY REFERENCES
mySchema.Regions(RegionID)
)
GO

INSERT INTO mySchema.Persons
(FirstName, LastName, MiddleName, CompanyName, Phone, RegionID)
VALUES('Hunter', 'Thomson', 'S', 'Gonzo', '2222-222', 1)
```

Lab 4: Introduction to indexes

Note: This lab should be done in the **AdventureworksDW** database.

Ex 1. Identify heap tables

Find the heap tables in the database. A tip is to select from sys.indexes. Disregard various types of supporting tables (tip: the only potentially relevant tables left has “Fact” somewhere in the table name).

How many rows do we have in these tables?

Ex 2. Convert a heap table to a clustered table

There should be one table with more than 10,000 rows. You will convert this into a clustered table.

- Sample some data (SELECT)
- Investigate the data types for the columns
- Investigate the current indexes
- Create a clustered index. We currently have no information about access pattern on the table, se we will just pick the FinanceKey as key for the index.

Ex 3. Create a single-column index

The below query is frequently executed in the database. Create a supporting index. Use SET STATISTICS IO ON to determine number pf page reads for the query before and after creating that index.

Feel free to look at the execution plans if you have that experience, but don’t spend a significate amount of time on the execution plan.

```
SELECT SUM(f.SalesAmount) AS sumSales
FROM FactResellerSalesXL AS f
WHERE ShipDateKey = 20050413
```

Remove the index when you are done with the exercise.

Ex 4. Create a multi-column index

The below query is frequently executed in the database. Create a supporting index. Use SET STATISTICS IO ON to determine number pf page reads for the query before and after creating that index.

We want the number of pages to be read to be well under 1,000 pages. Neither of the search arguments alone are very selective.

Feel free to look at the execution plans if you have that experience, but don’t spend a significate amount of time on the execution plan.

```
SELECT SUM(f.SalesAmount) AS sumSales, COUNT(*) AS no_rows
FROM FactResellerSalesXL AS f
WHERE ProductKey = 564 AND Resellerkey = 494
```

Lab 4 answer suggestions

Ex 1. Identify heap tables

```
USE AdventureworksDW

--Do we have any heap tables?
SELECT OBJECT_SCHEMA_NAME(i.object_id) AS schema_, OBJECT_NAME(i.object_id) AS
table_name_
FROM sys.indexes AS i
WHERE type_desc = 'HEAP'

--How many rows in the potentially relevant heap tables
SELECT COUNT(*) FROM FactFinance
SELECT COUNT(*) FROM NewFactCurrencyRate
```

Ex 2. Convert a heap table to a clustered table

```
--Sample some data
SELECT TOP(10) * FROM FactFinance

--Investigate data types
EXEC sp_help 'FactFinance'

--What indexes exists?
EXEC sp_indexinfo 'FactFinance' --Found at www.karaszi.com
EXEC sp_helpindex 'FactFinance'

--Convert the table to a clustered table
CREATE INDEX FinanceKey ON FactFinance(FinanceKey)

--Verify our index
EXEC sp_helpindex 'FactFinance'
```

Ex 3. Create a single-column index

```
SET STATISTICS IO ON

--Approx 330897 pages read
SELECT SUM(f.SalesAmount) AS sumSales
FROM FactResellerSalesXL AS f
WHERE ShipDateKey = 20050413


CREATE INDEX ShipDateKey ON FactResellerSalesXL(ShipDateKey)
GO

--Approx 174 pages read
SELECT SUM(f.SalesAmount) AS sumSales
FROM FactResellerSalesXL AS f
WHERE ShipDateKey = 20050413


--Cleanup
DROP INDEX IF EXISTS ShipDateKey ON FactResellerSalesXL
```

Ex 4. Create a multi-column index

```
SET STATISTICS IO ON

--Approx 330897 pages read
SELECT SUM(f.SalesAmount) AS sumSales, COUNT(*) AS no_rows
FROM FactResellerSalesXL AS f
WHERE ProductKey = 564 AND Resellerkey = 494


--Create single column index
CREATE INDEX ProductKey ON FactResellerSalesXL(ProductKey)
GO
--Run the SELECT query, check number of pages read (118603)

--Drop the index
DROP INDEX IF EXISTS ProductKey ON FactResellerSalesXL
GO


--Create single column index
CREATE INDEX Resellerkey ON FactResellerSalesXL(Resellerkey)
GO
--Run the SELECT query, check number of pages read (39338)

--Drop the index
DROP INDEX IF EXISTS Resellerkey ON FactResellerSalesXL
GO

--Create composite index
CREATE INDEX ProductKey__Resellerkey ON FactResellerSalesXL(ProductKey, Resellerkey)
GO
--Run the SELECT query, check number of pages read (109)

--Cleanup
DROP INDEX IF EXISTS ProductKey__Resellerkey ON FactResellerSalesXL
```

Lab 5: Index strategies

Note: This lab should be done in the **AdventureworksDW** database.

Ex 1. Turn on Query Store

Turn on Query Store. Use the defaults except for the statistics collection internal, which you will set to 1 minute. This is not suitable for production environment, but suits our lab environment.

Turn it on for all below databases:

- Adventureworks
- AdventureworksDW
- AdventureworksLT

Note: There will be no labs to look in the query store, since it is best used after collecting data for a while and when you have a real-world query load. Feel free to check out the query store reports at any time, for instance after doing the last exercise in this lab.

Ex 2. Create a covering index

You have below query:

```
SELECT SUM(f.SalesAmount) AS sumSales, COUNT(*) AS no_rows
FROM FactResellerSalesXL AS f
WHERE ProductKey = 330 AND Resellerkey = 669
```

It will read 330897 with no supporting index. With a composite index created in above lab, it will read 488 pages. You need to get it down to under 10 page reads. Create a covering index to achieve that.

Ex 3. Verify above index in the execution plan

Check out the actual execution plans for above query. Feel free to do the same for the queries in the prior lab if time permits.

How can you determine that the index covers the query?

Drop the index created in the previous exercise then you are done with the lab.

Lab 5 answer suggestions

Ex 1. Turn on Query Store

```
USE master
GO
ALTER DATABASE Adventureworks SET QUERY_STORE = ON
GO
ALTER DATABASE Adventureworks SET QUERY_STORE (OPERATION_MODE = READ_WRITE,
INTERVAL_LENGTH_MINUTES = 1)
GO
ALTER DATABASE AdventureworksDW SET QUERY_STORE = ON
GO
ALTER DATABASE AdventureworksDW SET QUERY_STORE (OPERATION_MODE = READ_WRITE,
INTERVAL_LENGTH_MINUTES = 1)
GO
ALTER DATABASE AdventureworksLT SET QUERY_STORE = ON
GO
ALTER DATABASE AdventureworksLT SET QUERY_STORE (OPERATION_MODE = READ_WRITE,
INTERVAL_LENGTH_MINUTES = 1)
GO
```

Ex 2. Create a covering index

```
SET STATISTICS IO ON

--Approx 330897 pages read
SELECT SUM(f.SalesAmount) AS sumSales, COUNT(*) AS no_rows
FROM FactResellerSalesXL AS f
WHERE ProductKey = 330 AND Resellerkey = 669


--Create covering index
CREATE INDEX ProductKey__Resellerkey___SalesAmount ON FactResellerSalesXL(ProductKey,
Resellerkey)
INCLUDE (SalesAmount)
GO


--Run the SELECT query, check number of pages read (5)
```

Ex 3. Verify above index in the execution plan

You can determine that the index covers the query by seeing an index seek (or scan, if that were the case) with ***no lookups***.

```
--Cleanup
DROP INDEX IF EXISTS ProductKey__Resellerkey___SalesAmount ON FactResellerSalesXL
```

Lab 6: Views

Ex 1. Create a view

You have below query. It is used very frequently, and for convenience purposes create a view named ProdSales for the query.

```
SELECT p.EnglishProductName, SUM(f.SalesAmount) AS sumSales
FROM FactResellerSalesXL AS f
INNER JOIN DimProduct AS p ON f.ProductKey = p.ProductKey
GROUP BY p.EnglishProductName
```

Ex 2. Create an indexed view, if time permits

The performance when using the view isn't satisfactory.

Check the number of pages read from below query. Feel free to check out the execution plan as well (with no deeper analysis).

```
SET STATISTICS IO ON

SELECT * FROM ProdSales
```

You have about 300,000 page reads and you want to improve this.

Try to create an index on the view. It will fail. Use the error messages and see if you can re-do the view so you finally can create an index on it. Feel free to check the lab answers if you get tired of the trial-and-error. 😊

Check the performance improvements using SET STATISTICS IO (and also look at the execution plan of you feel like it).

Delete the view when you are done with the lab.

Lab 6 answer suggestions

Ex 1. Create a view

```
CREATE OR ALTER VIEW ProdSales
AS
SELECT p.EnglishProductName, SUM(f.SalesAmount) AS sumSales
FROM FactResellerSalesXL AS f
INNER JOIN DimProduct AS p ON f.ProductKey = p.ProductKey
GROUP BY p.EnglishProductName
GO

--Test the view
SELECT * FROM ProdSales
```

Ex 2. Create an indexed view, if time permits

```
--Re-create the view so we are allowed to create an index on it
CREATE OR ALTER VIEW ProdSales
WITH SCHEMABINDING
AS
SELECT p.EnglishProductName, SUM(ISNULL(f.SalesAmount, 0)) AS sumSales, COUNT_BIG(*)
AS numSales
FROM dbo.FactResellerSalesXL AS f
INNER JOIN dbo.DimProduct AS p ON f.ProductKey = p.ProductKey
GROUP BY p.EnglishProductName
GO

--Number of I/O operations without the index is 330,000
SELECT * FROM ProdSales
GO

--Create the index
CREATE UNIQUE CLUSTERED INDEX EnglishProductName ON ProdSales(EnglishProductName)

--Number of I/O operations with the index is 5
SELECT * FROM ProdSales
GO

--Cleanup
DROP VIEW ProdSales
```

Lab 7: Stored procedures

Ex 1. Create a simple stored procedure

Create a stored procedure that from the Production.Product table returns below column for products that has a ListPrice > 1000:

- `ProductID`
- Name
- ListPrice

Test the procedure.

Ex 2. Create a procedure with a parameter

Modify above procedure so you pass the value for the ListPrice as a parameter to the procedure (@minListPrice). Make sure that the parameter has the same data type as the ListPrice column. Make the parameter optional with a default value of 1000.

Also, adjust your code so it will execute without errors regardless of whether the procedure already exists or not.

Test the procedure.

Ex 3. Handle parameter sniffing issue, if time permits

Do this exercise in the **AdventureworksDW** database.

Execute below to create a parameterized stored procedure.

```
USE AdventureworksDW
GO

DROP INDEX IF EXISTS DiscountAmount ON  FactResellerSalesXL
GO

CREATE INDEX DiscountAmount ON  FactResellerSalesXL(DiscountAmount)
GO

CREATE OR ALTER PROC ListProductsByDiscount
@DiscountAmount float
AS
SELECT AVG(UnitPrice)
FROM FactResellerSalesXL AS f
WHERE DiscountAmount = @DiscountAmount
GO
```

Execute below to execute that procedure once. Note the number of pages read from the execution of the procedure (the output from STATISTICS IO). Feel free to also check the execution plan.

```
SET STATISTICS IO ON
GO

EXEC ListProductsByDiscount @DiscountAmount = 120
GO
```

Now execute it as per below. It will take a while (some 30 seconds to a minute). Note number of pages (and plan if you want to).

```
EXEC ListProductsByDiscount @DiscountAmount = 0
GO
```

You see some 35 million page reads.

- Why is that?
- How can you improve the situation?
- Implement one (or more) improvements and test it.

Lab 7 answer suggestions

Ex 1. Create a simple stored procedure

```
CREATE PROC ListProductsbyPrice
AS
SET NOCOUNT ON
SELECT p.ProductID, p.Name, p.ListPrice
FROM Production.Product AS p
WHERE p.ListPrice > 1000
GO

EXEC ListProductsbyPrice
```

Ex 2. Create a procedure with a parameter

```
CREATE OR ALTER PROC ListProductsbyPrice
 @minListPrice money = 1000
AS
SET NOCOUNT ON
SELECT p.ProductID, p.Name, p.ListPrice
FROM Production.Product AS p
WHERE p.ListPrice > @minListPrice
GO

EXEC ListProductsbyPrice
EXEC ListProductsbyPrice 3000
EXEC ListProductsbyPrice @minListPrice = 3500
```

Ex 3. Handle parameter sniffing issue, if time permits

The problem was that for the first execution a plan was created with an index seek, with a lookup for each row. That was fine with a selectivity of just a few rows as for the first execution. But for the second execution the plan was re-used and with a selectivity of almost 9 million rows we did just as many lookups.

Here are some ways that can improve the situation (in no particular order)

- Specify RECOMPLIE hint for the query so you get an “optimal” plan for each execution but you pay for plan recompile.
- Or, use OPTIMIZE FOR for the query so you don’t have to pay for recompile for each execution.
- Move to SQL Server 2022 and see if parameter sensitive parameter sniffing helps
- Do some clever re-write and make it two different statements based on the selecticity we pass in.
- Work with some index strategy so we aren’t that sensitive in the first place.

Here is an example for the first option above:

```
CREATE OR ALTER PROC ListProductsByDiscount
@DiscountAmount float
AS
SELECT AVG(UnitPrice) AS avgPrice, COUNT(*) AS numberOfProducts
FROM FactResellerSalesXL AS f
WHERE DiscountAmount = @DiscountAmount
OPTION(RECOMPILE)
GO

SET STATISTICS IO ON
GO

EXEC ListProductsByDiscount @DiscountAmount = 120
GO

EXEC ListProductsByDiscount @DiscountAmount = 0
GO
```

Lab 8: User-defined functions

Ex 1. Create and use a scalar function

Create a scalar function named dbo.SalesForProduct that sums the sales for a certain product number (ProductID), based on below query:

```
SELECT SUM(d.UnitPrice)
FROM Sales.SalesOrderDetail AS d
WHERE d.ProductID = @productID
```

Add the code around this to make it a working function.

Test your function with below code:

```
SELECT dbo.SalesForProduct(707)
GO

SELECT p.Name, dbo.SalesForProduct(p.ProductID)
FROM Production.Product AS p
GO
```

If time permits: Re-write above so a join is used instead, and no function is required. Check out the performance difference using STATISTICS IO and STATISTICS TIME.

Ex 2. Create and use a table function

Crete an in-line table function that returns the two cheapest products for a certain product sub category. You will pass in the name of the product sub category as a parameter to the function. Return the following columns (p means the product table and c means the ProductSubCategory able):

- p.Name, p.ProductNumber, p.ListPrice, p.Size
- c.Name AS SubCategoryName

Here is a starting point:

```
SELECT *
FROM Production.Product AS p
 INNER JOIN Production.ProductSubcategory AS c
 ON p.ProductSubcategoryID = c.ProductSubcategoryID
```

Test your function using below queries:

```
SELECT * FROM CheapProducts('Handlebars')


SELECT cp.ListPrice, cp.Name, cp.Size, cp.SubCategoryName
FROM Production.ProductSubcategory AS ps
CROSS APPLY CheapProducts(ps.Name) AS cp
ORDER BY SubCategoryName, ListPrice
```

Lab 8 answer suggestions

Ex 1. Create and use a scalar function

```
--Create the function
CREATE OR ALTER FUNCTION SalesForProduct(@productID int)
RETURNS money
AS
BEGIN
DECLARE @SoldFor money

SET @SoldFor =
       (SELECT ISNULL(SUM(d.UnitPrice), 0)
       FROM Sales.SalesOrderDetail AS d
       WHERE d.ProductID = @productID)
RETURN @SoldFor
END
GO

--Test it
SELECT dbo.SalesForProduct(707)
GO

SELECT p.Name, dbo.SalesForProduct(p.ProductID)
FROM Production.Product AS p
GO


SET STATISTICS IO ON
SET STATISTICS TIME ON
GO

--Compare performance between the two
SELECT p.Name, dbo.SalesForProduct(p.ProductID)
FROM Production.Product AS p
GO

SELECT p.Name, ISNULL(SUM(d.UnitPrice), 0)
FROM Production.Product AS p
INNER JOIN Sales.SalesOrderDetail AS d ON d.ProductID = p.ProductID
GROUP BY p.Name
```

Ex 2. Create and use a table function

```
CREATE OR ALTER FUNCTION CheapProducts(@SubCategoryName Name)
RETURNS TABLE
AS
RETURN(
SELECT TOP(2) p.Name, p.ProductNumber, p.ListPrice, p.Size, c.Name AS SubCategoryName
FROM Production.Product AS p
 INNER JOIN Production.ProductSubcategory AS c
 ON p.ProductSubcategoryID = c.ProductSubcategoryID
WHERE c.Name = @SubCategoryName
ORDER BY ListPrice ASC
)
GO
```

Lab 9: Triggers

Ex 1: Create a trigger

Create trigger on the Sales.SalesPerson table, that will fire on UPDATE. Whenever any rows are modified, if the SalesQuote column is modified, then you should add history data to the Sales.SalesPersonQuotaHistory table.

Make sure that the trigger takes multi-row updates in consideration. Of, if you feel that is out of reach for you, check number of rows modified by the firing UPDATE and if more than 1 row, exit the trigger with a rollback and an error message.

- The value for the QuotaDate column should be set to GETDATE()
- The value for the rowguid column should be set to NEWID()
- The value for the ModifiedDate column should be set to GETDATE()

Ex 2: Test your trigger

You will now test your trigger. Below are three UPDATE statements that will test modifying 0 rows, 1 row and many rows. After each there's a SELECT statement that will see what history rows that has been generated the last hour.

If you happen to trash the data (this happens to all of us 😊), you can always restore the whole database to the initial state. See the "Lab environment overview" at the beginning of this lab document.

```
--Test the trigger, modify 0 rows
UPDATE Sales.SalesPerson
SET SalesQuota = 5000
WHERE BusinessEntityID = -999

--Check history done last hour
SELECT * FROM Sales.SalesPersonQuotaHistory WHERE ModifiedDate > DATEADD(DAY, -1,
GETDATE())
GO

--Test the trigger, modify 1 row
UPDATE Sales.SalesPerson
SET SalesQuota = 3000
WHERE BusinessEntityID = 279

--Check history done last hour
SELECT * FROM Sales.SalesPersonQuotaHistory WHERE ModifiedDate > DATEADD(DAY, -1,
GETDATE())
GO

--Test the trigger, modify many rows
UPDATE Sales.SalesPerson
SET SalesQuota = 5000
WHERE Bonus > 4000

--Check history done last hour
SELECT * FROM Sales.SalesPersonQuotaHistory WHERE ModifiedDate > DATEADD(DAY, -1,
GETDATE())
GO
```

Lab 9 answer suggestions

Ex 1: Create a trigger

```
CREATE OR ALTER TRIGGER Sales.trSalesPersonQuotaChange
ON Sales.SalesPerson
AFTER UPDATE
AS
IF @@ROWCOUNT = 0
BEGIN
       RETURN
END
SET NOCOUNT ON
IF UPDATE(SalesQuota)
BEGIN
  INSERT INTO Sales.SalesPersonQuotaHistory(BusinessEntityID, QuotaDate, SalesQuota,
rowguid, ModifiedDate)
  SELECT d.BusinessEntityID, GETDATE(), d.SalesQuota, NEWID(), GETDATE()
  FROM deleted AS d
END
RETURN
GO
```

Ex 2: Test your trigger

See the lab instructions.

Lab 10: Concurrency and transactions

Ex 1: Improve concurrency

You have a situation where users are complaining about bad performance. You suspect that you have concurrency issues, i.e., blocking. Below is the workload for this lab, i.e., this represents the workload on your SQL Server:

Close all query windows in SSMS. Open below two files in SSMS, so you now have (only) two query windows:

- C:\SqlLabs\T2762-1_LabFiles\Lab05\DoUpdates.sql
- C:\SqlLabs\ T2762-1_LabFiles\Lab05\DoSelects.sql

Optional: Put them side-by-side in SSMS (alt-W, V if you prefer keyboard shortcut).

- Window menu
- New Vertical Tab Group

Execute both query windows simultaneous. Start the DoUpdates.sql scripts first, and start the other pretty directly after you started the first one. They will run for about 1 minute. Switch to the "messages" tab in each query window to get some text feedback while the script is running.

Take a note of how many SELECTs you managed to do in one minute.

You suspect that you have blocking issues. *How can you verify that?* (Feel free to check the answer suggestions if it isn't obvious…)

Set the read_committed_snapshot database setting to 1. You cannot have any open query windows while you change this setting, the ALTER command will be blocked. If you use the GUI in SSMS, it will close the connections for you.

Re-run the tests. Did you get an increased number of SELECTs?

When you're done, reset the versioning setting for the database back to default!

Lab 10 answer suggestions

Follow the lab instructions.

How can you determine if you have lots of blocking?
Check out the wait stats. You can use Glenn Berry's query for this (search in his file for "dm_os_wait", at the time of writing this lab it is query number 39).

To change the database setting to get versioning instead of blocking:

```
USE master
GO
ALTER DATABASE Adventureworks SET SINGLE_USER WITH ROLLBACK IMMEDIATE
WAITFOR DELAY '00:00:02'
ALTER DATABASE Adventureworks SET READ_COMMITTED_SNAPSHOT ON
WAITFOR DELAY '00:00:02'
ALTER DATABASE Adventureworks SET MULTI_USER WITH ROLLBACK IMMEDIATE
GO
```

When you're done, reset the versioning setting for the database back to default:

```
USE master
GO
ALTER DATABASE Adventureworks SET SINGLE_USER WITH ROLLBACK IMMEDIATE
WAITFOR DELAY '00:00:02'
ALTER DATABASE Adventureworks SET READ_COMMITTED_SNAPSHOT OFF
WAITFOR DELAY '00:00:02'
ALTER DATABASE Adventureworks SET MULTI_USER WITH ROLLBACK IMMEDIATE
GO
```