Lab Manual

T2762-2, Developing SQL Databases, Part 2

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***
- 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 1: Advanced table designs

Ex 1. Use Data Compression

Evaluate the Production.TransactionHistory table. Feel free to install and use sp_tableinfo and sp_indexinfo that you can find at https://karaszi.com/articlesandutilities.

- What indexes does it have?
- How much space is each part currently using.

Estimate the storage size for the data and each index for the Production.TransactionHistory table for both row and page compression. Use either the GUI in Object Explorer or sp_estimate_data_compression_savings directly. (Note that this is a tiny table, but the same principals would apply for a larger table.)

- What would you compress?
- With what compression level?

Compress the actual data using page compression

Ex 2. Create a temporal table

You will create a “copy” of the Production.TransactionHistory table. Names the table dbo.TransactionHistoryTprl.

Generate a create script of the current table from Object Explorer. We’ll keep it simple so remove all constraint, identity and such stuff except for the primary key which is required for a temporal table.

Add the code to the above script to make it temporal. One tip is to right-click the table folder and select New, Temporal Table to get some skeleton code. Or use an example online, for instance from the documentation: https://learn.microsoft.com/en-us/sql/relational-databases/tables/creating-a-system-versioned-temporal-table?view=sql-server-ver16#creating-a-temporal-table-with-a-user-defined-history-table

Perform a few data modifications (with some seconds between them):

```
INSERT INTO TransactionHistoryTprl
(TransactionID, ProductID, ReferenceOrderID, ReferenceOrderLineID,
TransactionDate, TransactionType, Quantity, ActualCost, ModifiedDate)
VALUES (1, 784, 11111, 0, '2013-07-31', N'W', 1, 0.0000, '2013-07-31')

WAITFOR DELAY '00:00:01.100'
GO

INSERT INTO TransactionHistoryTprl
(TransactionID, ProductID, ReferenceOrderID, ReferenceOrderLineID,
TransactionDate, TransactionType, Quantity, ActualCost, ModifiedDate)
VALUES (2, 785, 22222, 77, '2013-08-01', N'W', 2, 0.0000, '2013-08-01')

WAITFOR DELAY '00:00:01.100'
GO

UPDATE TransactionHistoryTprl
SET Quantity = 55
WHERE TransactionID = 1

WAITFOR DELAY '00:00:01.100'
GO

DELETE FROM TransactionHistoryTprl
WHERE TransactionID = 1
```

Now check out the history table and also a SELECT using the FOR SYSTEM_TIME AS OF clause.

Lab 1 answer suggestions

Ex 1. Use Data Compression

```
--Space usage
EXEC sp_spaceused 'Production.TransactionHistory'

--Or, using https://karaszi.com/sptableinfo-list-tables-and-space-usage
EXEC sp_tableinfo 'TransactionHistory'
EXEC sp_indexinfo 'TransactionHistory'

--Index information
EXEC sp_helpindex 'Production.TransactionHistory'

--Evaluate compression ratio
EXEC sp_estimate_data_compression_savings 'Production', 'TransactionHistory', NULL,
NULL, ROW
EXEC sp_estimate_data_compression_savings 'Production', 'TransactionHistory', NULL,
NULL, PAGE

--Page-compress the actual data (clustered index)
ALTER TABLE Production.TransactionHistory REBUILD WITH(DATA_COMPRESSION = PAGE)
--ALTER INDEX can also be used, considering the table has a clustered index:
-- ALTER INDEX PK_TransactionHistory_TransactionID ON Production.TransactionHistory
REBUILD WITH(DATA_COMPRESSION = PAGE)


--Verify compression, using method of your choice
EXEC sp_indexinfo 'TransactionHistory'

--Remove page compression
ALTER TABLE Production.TransactionHistory REBUILD WITH(DATA_COMPRESSION = NONE)
```

Ex 2. Create a temporal table

```
ALTER TABLE TransactionHistoryTprl SET (SYSTEM_VERSIONING = OFF);
GO

DROP TABLE IF EXISTS TransactionHistoryTprlHistory
DROP TABLE IF EXISTS TransactionHistoryTprl
GO

CREATE TABLE dbo.TransactionHistoryTprl(
       TransactionID int PRIMARY KEY NOT NULL,
       ProductID int NOT NULL,
       ReferenceOrderID int NOT NULL,
       ReferenceOrderLineID int NOT NULL,
       TransactionDate datetime NOT NULL,
       TransactionType nchar(1) NOT NULL,
       Quantity int NOT NULL,
       ActualCost money NOT NULL,
       ModifiedDate datetime NOT NULL
  , ValidFrom DATETIME2 GENERATED ALWAYS AS ROW START NOT NULL
  , ValidTo DATETIME2 GENERATED ALWAYS AS ROW END NOT NULL
  , PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.TransactionHistoryTprlHistory))
GO

INSERT INTO TransactionHistoryTprl
(TransactionID, ProductID, ReferenceOrderID, ReferenceOrderLineID,
TransactionDate, TransactionType, Quantity, ActualCost, ModifiedDate)
VALUES (1, 784, 11111, 0, '2013-07-31', N'W', 1, 0.0000, '2013-07-31')

WAITFOR DELAY '00:00:01.100'
GO

INSERT INTO TransactionHistoryTprl
(TransactionID, ProductID, ReferenceOrderID, ReferenceOrderLineID,
TransactionDate, TransactionType, Quantity, ActualCost, ModifiedDate)
VALUES (2, 785, 22222, 77, '2013-08-01', N'W', 2, 0.0000, '2013-08-01')

WAITFOR DELAY '00:00:01.100'
GO

UPDATE TransactionHistoryTprl
SET Quantity = 55
WHERE TransactionID = 1

WAITFOR DELAY '00:00:01.100'
GO

DELETE FROM TransactionHistoryTprl
WHERE TransactionID = 1

SELECT * FROM TransactionHistoryTprl
SELECT * FROM TransactionHistoryTprlHistory

--Adjust the datetime value below
SELECT *
FROM TransactionHistoryTprl
FOR SYSTEM_TIME AS OF '2022-11-15 14:17:00.0034'
```

Lab 2: Columnstore indexes

Note: You will use the *AdventureworksDW* database for this lab.

Ex 1. Test and note performance for a query

The main table and the table of interest for this lab is the FactResellerSalesXL table. Investigate what indexes you have for that table. The actual data should be about 2.4 GB.

You have the below query

1. Execute the query twice using SET STATISTICS TIME ON and note the CPU and elapse time for the second execution.
2. Change database compatibility level to 2017 (140).
3. Execute the query twice and note the CPU and elapse time for the second execution. This is the performance you will see if you don't have Enterprise Edition.

Why is there a difference between the two compatibility levels?

Ex 2. Create a Columnstore index

Create a columnstore index on the FactResellerSalesXL table to support our query. Note the size of this index.

Re-execute your query, the same way you did in exercise 1 and again note the CPU and execution time (for both compatibility level 140 and 150).

Remove the index when you are done.

Lab 2 answer suggestions

Ex 1. Test and note performance for a query

On my machine with 8 CPU cores, I got below numbers:

- Compatibility level 140: CPU time = 22952 ms, elapsed time = 3755 ms.
- Compatibility level 140: CPU time = 2611 ms, elapsed time = 396 ms.

```
--Structure, size and indexes for the table
EXEC sp_indexinfo 'FactResellerSalesXL'

--Set to compatibility level 2017
ALTER DATABASE CURRENT SET COMPATIBILITY_LEVEL = 140

SET STATISTICS TIME ON
GO

--Execute query twice, note CPU and execution time on second execution.
SELECT
 SUM(s.OrderQuantity * s.UnitPrice)
,COUNT(DISTINCT s.CurrencyKey)
,COUNT(s.Freight)
,COUNT(p.EnglishProductName)
,COUNT(t.SalesTerritoryCountry)
,t.SalesTerritoryCountry
,d.CalendarYear
FROM FactResellerSalesXL AS s
 INNER JOIN DimDate AS d ON s.OrderDateKey = d.DateKey
 INNER JOIN DimProduct AS p ON s.ProductKey = p.ProductKey
 INNER JOIN DimSalesTerritory AS t ON s.SalesTerritoryKey = t.SalesTerritoryKey
GROUP BY d.CalendarYear, t.SalesTerritoryCountry
ORDER BY CalendarYear DESC

--Set to compatibility level 2019
ALTER DATABASE CURRENT SET COMPATIBILITY_LEVEL = 150


--Execute query twice, note CPU and execution time on second execution.
```

Ex 2. Create a Columnstore index

On my machine with 8 CPU cores, I got below numbers:

- Compatibility level 140: CPU time = 1095 ms, elapsed time = 220 ms.
- Compatibility level 140: CPU time = 1140 ms, elapsed time = 219 ms.

```
CREATE COLUMNSTORE INDEX cs_FactResellerSalesXL
ON FactResellerSalesXL(OrderQuantity, UnitPrice, CurrencyKey, Freight, OrderDateKey,
ProductKey, SalesTerritoryKey)

--Structure, size and indexes for the table
EXEC sp_indexinfo 'FactResellerSalesXL'

--Re-execute our query with both compatibility levels

--Remove the index when we are done
```

Lab 4: XML and JSON in SQL Server

Ex 1. Use FOR XML and FOR JSON

Note: You will use the *AdventureworksDW* database for this exercise.

You have below query:

```
SELECT p.EnglishPromotionName, p.DiscountPct, f.SalesAmount, f.DiscountAmount,
f.Freight
FROM DimPromotion AS p
INNER JOIN FactResellerSales AS f
ON f.PromotionKey = p.PromotionKey
WHERE f.SalesAmount > 10000
ORDER BY EnglishPromotionName
```

Use above query and generate a hierarchical XML document with:

- A root named DiscountsInfo
- The columns from the DimPromotion table should be a sub-element from above named Promo
- The columns form the FactResellerSales should be a sub-element from above named SalesInfo

Tip: you can use the AUTO mode. Feel free to also use the PATH mode if you have the time.

Now use the same query, but return the data as JSON instead. Tip: run the JSON version in Azure Data Studio, since SSMS doesn't format JSON nicely.

Ex 2. Using the XML data type in a column

Note: You will use the *Adventureworks* database for this exercise.

Use below query

```
SELECT *
FROM Production.ProductModel
WHERE CatalogDescription IS NOT NULL
```

Note that there are two XML columns. We are interested in the CatalogDescription column. Investigate that XML data for some of the rows.

Challenge, if time permits: filter so that we only get the rows where the Material element equals to Aluminum. Tip: add below at the beginning inside the value method (inside the string):

declare namespace p1="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/ProductModelDescription";

Lab 4 answer suggestions

Ex 1. Use FOR XML and FOR JSON

```
SELECT Promo.EnglishPromotionName, Promo.DiscountPct, SalesInfo.SalesAmount,
SalesInfo.DiscountAmount, SalesInfo.Freight
FROM DimPromotion AS Promo
INNER JOIN FactResellerSales AS SalesInfo
ON SalesInfo.PromotionKey = Promo.PromotionKey
WHERE SalesInfo.SalesAmount > 10000
ORDER BY EnglishPromotionName
FOR XML AUTO, ROOT('DiscountsInfo')
```

```
SELECT Promo.EnglishPromotionName, Promo.DiscountPct, SalesInfo.SalesAmount,
SalesInfo.DiscountAmount, SalesInfo.Freight
FROM DimPromotion AS Promo
INNER JOIN FactResellerSales AS SalesInfo
ON SalesInfo.PromotionKey = Promo.PromotionKey
WHERE SalesInfo.SalesAmount > 10000
ORDER BY EnglishPromotionName
FOR JSON AUTO, ROOT('DiscountsInfo')
```

Ex 2. Using the XML data type in a column

```
SELECT *
FROM Production.ProductModel
WHERE CatalogDescription IS NOT NULL
AND CatalogDescription.value('
declare namespace p1="http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/ProductModelDescription";
p1:ProductDescription[1]/p1:Specifications[1]/Material[1]', 'varchar(30)') =
'Aluminum'
```

Lab 6: BLOBs and full-text indexes

Ex 1. Create a full-text index

You will create a full-text catalog and index on the Production.ProductDescription table, including the Description column.

Create a full-text catalog. Name it for instance ft_catalog.

Take a quick look at the data in the Production.ProductDescription table. (Note that there are rows with descriptions for other languages than English. We will not limit our searches to a certain language.)

```
SELECT *
FROM Production.ProductDescription
```

Find out and note the name of the index for the primary key on this table.

Based on above information, create a full-text index on the table and include the Description column. Use the English language for the index.

Ex 2. Performing full-text search

Write a query for above table that finds all rows having something to do with “high level of performance”, including synonyms to the word.

Now you want to sort the result so that the best hits are returned at the top in the result. I.e., you will use the FREETEXTTABLE() table function instead of the FREETEXT() predicate.

Feel free to also test the CONTAINS predicate and CONTAINSTABLE table function, if time permits.

Lab 6 answer suggestions

Ex 1. Create a full-text index

```
--Cleanup. if the stuff already exists
DROP FULLTEXT INDEX ON Production.ProductDescription
GO
DROP FULLTEXT CATALOG ft_catalog
GO

--Create full-text catalog
CREATE FULLTEXT CATALOG ft_catalog
WITH ACCENT_SENSITIVITY = ON
AS DEFAULT


--Note the name of the primary key (index):
EXEC sp_helpindex 'Production.ProductDescription'
-- PK_ProductDescription_ProductDescriptionID

--Create the full-text index on the table
CREATE FULLTEXT INDEX ON Production.ProductDescription
 (Description) --The columns to include in the index
KEY INDEX PK_ProductDescription_ProductDescriptionID
ON ft_catalog –Optional, if we have a default catalog
WITH
 CHANGE_TRACKING = AUTO

GO
```

Ex 2. Performing full-text search

```
--All rows having "high level of performance" or synonyms in the description column
SELECT *
FROM Production.ProductDescription AS pd
WHERE FREETEXT(Description, N'high level of performance')

--Use FRETTEXTTABLE instead, first only return the KEY and RANK columns
SELECT *
FROM FREETEXTTABLE(Production.ProductDescription, Description, 'high level of
performance')

--Use above to join to the table and sort on the RANK column
SELECT pd.ProductDescriptionID, pd.Description, ft.[RANK]
FROM FREETEXTTABLE(Production.ProductDescription, Description, 'high level of
performance') AS ft
INNER JOIN Production.ProductDescription AS pd ON pd.ProductDescriptionID = ft.[KEY]
ORDER BY [RANK] DESC
```