Performance Evaluation in SQL Server

Author: Simonely Hidalgo Lorenzo
Advisor: Dr. Nelliud Torres Batista
Electrical & Computer Engineering and Computer Science Department

Abstract

Abstract - A database table with millions of rows could take a long time to retrieve, insert, update, and delete data. The evaluation in this paper consists of create indexes, apply normalization process, and create surrogate key to improve the performance of retrieving data. Explain the differences between multiple types of indexes and which scenarios we can use for each of them. To evaluate the improvements, one table was created in SQL server with 45 million rows. The analysis describes the resources and I/O statistics used by Microsoft SQL Server Management Studio. For non-indexed tables is categorized sequentially searched and indexed table that are compared as B-tree index.

Introduction

The tables indexes are compared with the index card in a traditional library where we can see a lot shelve with books. But database indexes will affect another transaction as Insert, Update and Delete. Those different scenarios will be discussed and evaluated in this paper. In this paper will be discussed a different type of index, how to create the index in SQL server, the benefits of each index and demonstration of the performance improvement using index. In the normalization section it discusses the rules, and finally the composite key can affect the database performance, creating a surrogate key can bring a benefit but also a drawback.

Background

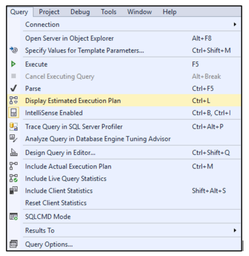
Indexes are the means to providing an efficient access path between the user and the data, by providing this path, the user can ask for data and the database will know where to go to retrieve [1]. Creating and maintaining an index file is a major issue in database, by using an appropriate indexing mechanism, the query processing algorithms may not have to search the entire database [2]. An index contains keys built from one or more columns in the table or view [3]. These keys are stored in a structure (B-tree) that enables SQL Server to find the row or rows associated with the key values quickly and efficiently [3]. Clustered Index is created with a column or combination of columns and are stored in orders to obtain a fast retrieval of the rows. The common type of Index are Cluster/Non-cluster index, column store index , XML indexes and spatial index. The variations are primary key, unique index, filtered indexes and, partitioned indexes. The database normalization has multiples benefits. When the normalization rules are applied the benefits are, avoid anomalies in the data, reduce large table into smaller tables avoiding data redundancy, maintain the data integrity reducing multiples entries and updates, the Insert and Update operations will be more quickly. With less data then maximize the storage capacity.

Problem

During a database operations (Select, Insert, Delete and, Update) could be take long respond. In this project will be evaluated each operation with creation of tables index and tables normalizations to compare which of them affect the time response and resources consumption for each operation.

Methodology

To evaluate the queries results performance, two tools provided by Microsoft SQL Server Management Studio were used. The first tool is “Estimate Execution Plan” tool found in the Query menu tab where we are focus in the IO Cost and CPU Cost sections.



The second is “IO statistics”, for obtain those statistics values in the output it needs to be turned on (Set Statistics IO on) left figure below before to execute of the SQL Statement. The two values from I/O statistics results that we are using to measure the performance are scan counts and logical reads see right figure below. Scan count is number of seeks or scans started after reaching the leaf level in any direction to retrieve all the values to construct the final dataset for the output. The logical reads are number of pages read from the data cache.

```
set statistics io on
set statistics time on
```

```
Table 'PRODUCT'. Scan count 13, logical reads 1923853,

SQL Server Execution Times:
  CPU time = 9689 ms,  elapsed time = 9827 ms.
```

To demonstrate the index improvements one table was created named PRODUCT with 45 million rows in SQL server, one of the columns created is Id and has integer values. The same query statement for SELECT, INSERT, DELETE and UPDATE was performed with non-index and index column.
To evaluate the resources consumed in normalized and denormalized scenarios I used “Display Estimate Execution Plan” tool, IO Statistics and the execution of the system stored procedures “sp_spaceused” to obtain the space used during the normalization and denormalized tables.

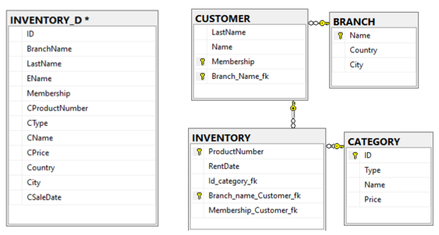
```
EXEC sp_spaceused INVENTORY
```

79 %

Results | Messages

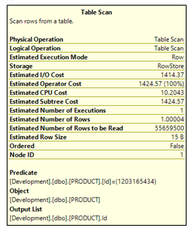
	name	rows	reserved	data	index_size	unused
1	INVENTORY	4535813	3372112 KB	3371632 KB	16 KB	464 KB

To perform the evaluation of the normalization rules one table denormalized was created, after applying the normalization rules four additional tables were created to decomposing into smaller relational schema with desirable properties. Both scenarios were evaluated for the same SQL statements.

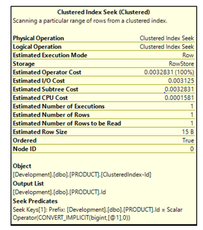


Results and Discussion

Index Results
The results of the SELECT statement where the index has not been created is showed below in the left side. In this scenario the search will do a sequential table scan until reach the id desired, also the I/O cost for this execution is 1414.37 and CPU cost is 10.2043. Comparing this result with a clustered index in the right figure. We can prove the improvement using the same SQL query. The results show the I/O cost of 0.003125 and the CPU Cost 0.0001581.



No index Results



Index Results

In the below table we show the execution time for each SQL statement with non index and indexed column.

SQL Time Statistics

SQL Statement	Number of Rows	Execution Time (ms)	
		No Index	Index
SELECT	1	90277	72
INSERT	9763855	39864	62894
DELETE	9763855	47616	83215
UPDATE	9763855	45486	62184

Normalization
we can see the space reduction, for denormalized table named “INVENTORY_D” the are found in table below. The consume was 5.3 GB and the sum of the four tables created during the normalization was 3.5 GB so the reduction in space was 2.8 GB for a 33%.

Space Used During Normalization

Table Name	Total Rows	Data Space (kb)	Total Space Used (kb)	Sum of tables used (Gb)
Category	3000791	206376	206384	3.5
Inventory	44535813	3321736	3321744	
Branch	15	8	16	
Customer	391795	20152	20168	
Inventory_D	4453583	5326576	5326592	5.3

The I/O statistic results that measure the query performance for SQL are shown in tables 6 and 7 where we can see the scan count and the logical read that SQL used during the query execution and shown why denormalized tables was faster than normalized. But for the Insert, Update and Delete in the normalized tables are faster than denormalized.

Normalization Time Statistics

SQL Statement	Number of Rows	Execution Time (ms)	
		Denormalized	Normalized
SELECT	8,451,155	104,610	115,720
INSERT	8,451,276	33,520	27,315
DELETE	8,451,397	19,350	14,250

Normalization SQL I/O Statistics (Logical Reads)

SQL Statement	Number of Rows	Logical Reads	
		Denormalized	Normalized
SELECT	8,451,155	665,822	1,028,578
INSERT	8,451,276	9,268,718	8,940,630
DELETE	8,451,397	8,940,749	9,232,663

Conclusions

We saw the Indexing, normalization and their behavior in the database performance. Each of one have their benefits and drawbacks. we need to be clear of how the data will be consumed if it for transactional or analytical purpose. For example, the improvement was noticed significantly when the cluster index was created in term of execution time for one simple query executed was 9 seconds faster but inserting was slower. So probably some of cons doesn’t apply to your application. In the data warehouse the normalization makes the data retrieve slow but in a transactional system where need to maintain the integrity of the data, make insert, delete, and update then the normalization is beneficial. Using the SQL tools provided by Microsoft SQL Server Management Studio was useful to evaluate which SQL Statements have better performance according with the applications or business needs.

Future Work

Perform the analysis in each types of index to evaluate the performance and resources consumed in each of them, collect the result in a transactional database and in a Datawarehouse database to evaluate the behaviors in a dedicate business rules.

Acknowledgements

I would like to express my sincere gratitude to Dr. Nelliud Torres Batista for their assistance at every stage of the research project and Isabel Batteria for the format review of the papers.

References

[1] J. Strate and T. Krueger, *Expert Performance Indexing for SQL Server 2012*. New York: Apress, 2012. [Online]. Available: https://www.apress.com/gp/book/9781430237419

[2] B. Thuraisingham, *Database and Applications Security*, Boca Raton, FL: Taylor & Francis Group, 2005.

[3] Microsoft, “Clustered and nonclustered indexes described,” December 14, 2020 [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver15