SQL SERVER OPTIMIZATION TECHNIQUE

PART 1 - UNDERSTANDING DATABASE AND INDEX BASICS

Technical Presentation By Santhosh

Introduction

Almost every developer, has taken part in the performance issues/tuning sometime in his/her development life and I am not an exception by any means.

"It's been months since our team have developed and deployed a site successfully in the internet. The client is pretty satisfied so far as the site was able to attract thousands of users to register and use the site within a small amount of time. Our client, management, team and me - everybody is happy.

As the number of users in the site started growing at a rapid rate on a daily basis, problems started occurring. Life is not a bed of roses. E-mails started to arrive from the client complaining that the site is performing too slowly (some of them were angry mails). The client claimed that they started losing users.

The team start's investigating the application. Soon they discover that the production database was performing extremely slowly when the application was trying to access/update data. Looking into the database, you find that the database tables have grown large in size and some of them were containing hundreds of thousands of rows. The testing team performed a test on the production site, and they found that the order submission process was taking 5 long minutes to complete, whereas it used to take only 2/3 seconds to complete in the test site before production launch."

This is the same old story for thousands of application projects developed worldwide.

We have a database to optimize, let's start!

When a database based application performs slowly, there is a 90% probability that the data access routines of that application are not optimized, or not written in the best possible way. So, you need to review and optimize your data access/manipulation routines for improving the overall application performance.

Let us start our optimization mission in a step-by-step process:

Step 1: Apply proper indexing in the table columns in the database

- Well, some could argue whether implementing proper indexing should be the first step in the performance optimization process for a database. But I would prefer applying indexing properly in the database in the first place, because of the following two reasons:
- This will allow you to achieve the best possible performance in the quickest amount of time in a production system.

Applying/creating indexes in the database will not require you to do any application modification, and thus will not require any build and deployment.

Of course, this quick performance improvement can be achieved if you find that indexing is not properly done in the current database. However, if indexing is already done, I would still recommend you to go through this step.

What is indexing?

Let us learn indexing correlating with a real life short story. As we know, there is music gallery in Brigade Road. Just imagine, It had thousands of music albums, but the albums were not arranged in any order in the rack. So, each time a customer asked for an album to the owner/sales person, they had no way but to check every album to find the required album that the customer needed. Finding the desired album used to take hours, and most of the time, the customer who asked for the album had to wait for a long time.

[Sounds like a table that has no primary key. When data is searched for in a table, the database engine has to scan through the entire table to find the corresponding row, which performs very slow.]

Life was getting miserable for the shop owner as the number of albums and customer’s asking for albums increased day by day. Finally, a wise guy(may be db consultant) came to the shop, and seeing the owner's miserable life, he advised the owner to number each album and arrange the album racks according to their numbers. "What benefit would I get?", asked the owner. The wise guy answered, "Well, now if somebody gives you a album number and asks for that album, you will be able to find the racks quickly that contains the album's number, and within that rack, you can find that album very quickly as albums are arranged according to their number".

[Numbering the albums sounds like creating a primary key in a database table. When you create a primary key in a table, a clustered index tree is created, and all data pages containing the table rows are physically sorted in the file system according to their primary key values. Each data page contains rows, which are also sorted within the data page according to their primary key values. So, each time you ask for any row from the table, the database server finds the corresponding data page first using the clustered index tree (like finding the album rack first) and then finds the desired row within the data page that contains the primary key value (like finding the album within the rack).]

The excited owner instantly started numbering the albums and arranging them across different album shelves. He spent a whole day to do this arrangement, but at the end of the day, he tested and found that a album now could be found using the number within no time at all! The owner was extremely happy.

[That's exactly what happens when you create a primary key in a table. Internally, a clustered index tree is created, and the data pages are physically sorted within the data file according to the primary key values. As you can easily understand, only one clustered index can be created for a table as the data can be physically arranged only using one column value as the criteria (primary key). It's like the albums can only be arranged using one criterion (album number here).]

Wait! The problem was not completely solved yet. The very next day, a person asked a album by the album's name (he didn't have the album's number, all he had was the album's name). The poor owner had no way but to scan all the numbered albums from 1 to N to find the one the person asked for. He found the album in the 87th shelf. It took 30 minutes for the owner to find the album. Earlier, he used to take 2-3 hours to find a album when they were not arranged in the shelves, so that was an improvement. But, comparing to the time to search a album using its number (30 seconds), this 30 minute seemed to be a very high amount of time to the owner. So, he asked the wise man how to improve on this.

[This happens when you have a Product table where you have a primary key ProductID, but you have no other index in the table. So, when a product is to be searched using the Product Name, the database engine has no way but to scan all physically sorted data pages in the file to find the desired item.]

The wise man told the owner:
"Well, as you have already arranged your albums using their serial numbers, you cannot re-arrange them. Better create a catalog or index where you will have all the album's names and their corresponding serial numbers. In this catalog, arrange the album names in their alphabetic number and group the album names using each alphabet so that if any one wants to find a album named “Dhoom 3", you just follow these steps to find the album:

- Jump into the section "D" of your album name "catalog" and find the album name there.
- Read the corresponding serial number of the album and find the album using the serial number (the owner already know how to do this).

Spending some hours, he immediately created the "album name" catalog, and with a quick test, he found that he only required a minute (30 seconds to find the album's serial number in the "album name" catalog and 30 seconds to find the album using the serial number) to find a album using its name.

The owner thought that people might ask for albums using several other criteria like album name and/or author's name etc., so he created a similar catalog for author names, and after creating these catalogs, the owner could find any album using a common album finding criteria (serial number, album name, author's name) within a minute. The miseries of the owner ended soon and the solution is been provided, and lots of people started gathering at the shop for albums as they could get albums really fast, and the shop became very popular.

By this time, I am sure you have understood what indexes really are, why they are important, and how they work internally.

For example, if we have a "Products" table, along with creating a clustered index (that is automatically created when creating the primary key in the table), we should create a non-clustered index on the *ProductName* column.

If we do this, the database engine creates an index tree for the non-clustered index (like the "album name" catalog in the story) where the product names will be sorted within the index pages. Each index page will contain a range of product names along with their corresponding primary key values. So, when a product is searched using the product name in the search criteria, the database engine will first seek the non-clustered index tree for product name to find the primary key value of the album. Once found, the database engine then searches the clustered index tree with the primary key to find the row for the actual item that is being searched.

This is called a B+ Tree (Balanced tree). The intermediate nodes contain a range of values and directs the SQL engine where to go while searching for a specific index value in the tree, starting from the root node.
The leaf nodes are the nodes which contain the actual index values. If this is a clustered index tree, the leaf nodes are the physical data pages. If this is a non-clustered index tree, the leaf nodes contain index values along with the clustered index keys (which the database engine uses to find the corresponding row in the clustered index tree).

Usually, finding a desired value in the index tree and jumping to the actual row from there takes an extremely small amount of time for the database engine. So, indexing generally improves the data retrieval operations.

Conclusion

Make sure that every table in your database has a primary key.

This will ensure that every table has a clustered index created (and hence, the corresponding pages of the table are physically sorted in the disk according to the primary key field). So, any data retrieval operation from the table using the primary key, or any sorting operation on the primary key field or any range of primary key values specified in the where clause will retrieve data from the table very fast.

Create non-clustered indexes on columns which are:

- Frequently used in the search criteria
- Used to join other tables
- Used as foreign key fields
- Of having high selectivity (column which returns a low percentage (0-5%) of rows from a total number of rows on a particular value)
- Used in the ORDER BY clause

Thank You !!!