Top 10 steps to optimize data access in SQL Server: Part I (use indexing)

Al-Farooque Shubho, 28 Apr 2009

4.83 (196 votes) Rate this:

As part of a series of articles on several data access optimization steps in SQL Server, this article focuses on using indexing to optimize data access.

Introduction

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

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

You start investigating the application. Soon you 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. Almost every developer, including me, has taken part in the story sometime in his/her development life. I know why such situations take place, and I can tell you what to do to overcome this.

Let's face it. If you are part of this story, you must have not written the data access routines in your application in the best possible way, and it's time to optimize those now. I want to help you do this by sharing my data access optimization experiences and findings with you in this series of articles. I just hope this might enable you to optimize your data access routines in existing systems, or to develop data access routines in an optimized way in your future projects.

Scope

Please note that the primary focus of this series of articles is "data access performance optimization in transactional (OLTP) SQL Server databases". But, most of the optimization techniques are roughly the same for other database platforms.

Also, the optimization techniques I am going to discuss are applicable for software application developers only. That is, as a developer, I'll focus on the issues that you need to follow to make sure that you have done everything that you could do to optimize the data access code you have written or you are going to write in future. Database Administrators (DBAs) also have a great role to play in optimizing and tuning database performance. But, optimization scopes that fall into a DBA's area are out of scope for these articles.

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:

1. This will allow you to achieve the best possible performance in the quickest amount of time in a production system.
2. 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?

I believe you know what indexing is. But, I've seen many people being unclear on this. So, let us try to understand indexing once again. Let us read a small story.

Long ago, there was a big library in an ancient city. It had thousands of books, but the books ware not arranged in any order in the book shelves. So, each time a person asked for a book to the librarian, the librarian had no way but to check every book to find the required book that the person wanted. Finding the desired book used to take hours for the librarian, and most of the time, the persons who asked for the book had to wait for a long time.

[Hmm... 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 librarian as the number of books and persons asking for books increased day by day. Then one day, a wise guy came to the library, and seeing the librarian's measurable life, he advised him to number each book and arrange the book shelves according to their numbers. "What benefit would I get?", asked the librarian. The wise guy answered, "Well, now if somebody gives you a book number and asks for that book, you will be able to find the shelves quickly that contains the book's number, and within that shelf, you can find that book very quickly as books are arranged according to their number".

[Numbering the books 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 book shelf first) and then finds the desired row within the data page that contains the primary key value (like finding the book within the shelf).]

"This is exactly what I need!" The excited librarian instantly started numbering the books and arranging them across different book shelves. He spent a whole day to do this arrangement, but at the end of the day, he tested and found that a book now could be found using the number within no time at all! The librarian 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 books can only be arranged using one criterion (book number here).]

Wait! The problem was not completely solved yet. The very next day, a person asked a book by the book's name (he didn't have the book's number, all he had was the book's name). The poor librarian had no way but to scan all the numbered books from 1 to N to find the one the person asked for. He found the book in the 67^{th} shelf. It took 20 minutes for the librarian to find the book. Earlier, he used to take 2-3 hours to find a book when they were not arranged in the shelves, so that was an improvement. But, comparing to the time to search a book using its number (30 seconds), this 20 minute seemed to be a very high amount of time to the librarian. 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 librarian: "Well, as you have already arranged your books using their serial numbers, you cannot re-arrange them. Better create a catalog or index where you will have all the book's names and their corresponding serial numbers. In this catalog, arrange the book names in their alphabetic number and group the book names using each alphabet so that if any one wants to find a book named "Database Management System", you just follow these steps to find the book:

1. *Jump into the section "D" of your book name "catalog" and find the book name there.*
2. *Read the corresponding serial number of the book and find the book using the serial number (you already know how to do this).*

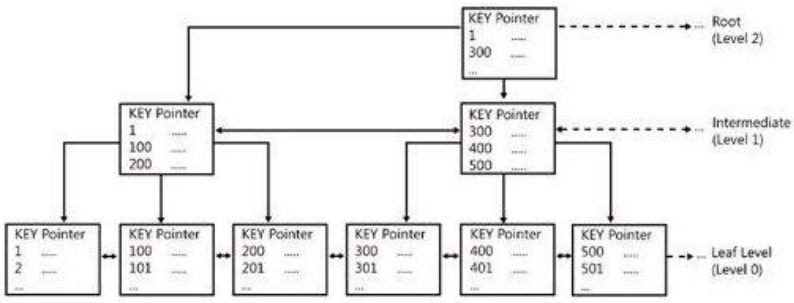
"You are a genius!", exclaimed the librarian. Spending some hours, he immediately created the "Book name" catalog, and with a quick test, he found that he only required a minute (30 seconds to find the book's serial number in the "Book name" catalog and 30 seconds to find the book using the serial number) to find a book using its name.

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

The librarian started passing his life happily ever after. The story ends.

By this time, I am sure you have understood what indexes really are, why they are important, and what their inner workings are. 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 "book 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 book. 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.

Following is how an index tree looks like:



Index tree structure

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.

Time to apply indexing in your database to retrieve results fast!

Follow these steps to ensure proper indexing in your database:

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:

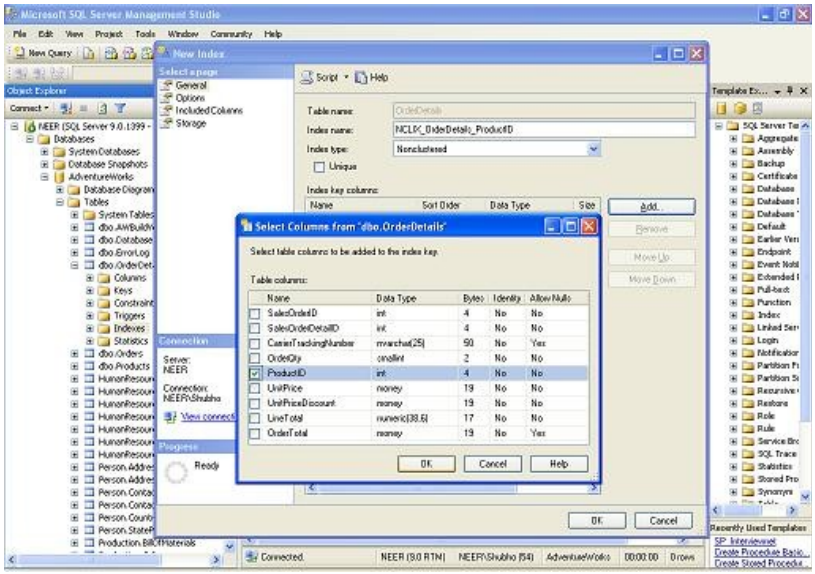
- Frequently used in the search criteria
- Used to join other tables
- Used as foreign key fields
- Of having high selectivity (column which returns a low percentage (0-5%) of rows from a total number of rows on a particular value)
- Used in the `ORDER BY` clause
- Of type XML (primary and secondary indexes need to be created; more on this in the coming articles)

Following is an example of an index creation command on a table:

Hide Copy Code

```
CREATE INDEX
NCLIX_OrderDetails_ProductID ON
dbo.OrderDetails(ProductID)
```

Alternatively, you can use SQL Server Management Studio to create an index on the desired table



Creating an index using SQL Server Management Studio

Step 2: Create the appropriate covering indexes

So you have created all the appropriate indexes in your database, right? Suppose, in this process, you have created an index on a foreign key column (*ProductID*) in the *Sales(SelesID,SalesDate,SalesPersonID,ProductID,Qty)* table. Now, assuming that the *ProductID* column is a "highly selective" column (selects less than 5% of the total number of rows rows using any *ProductID* value in the search criteria), any `SELECT` query that reads data from this table using the indexed column (*ProductID*) in the `where` clause should run fast, right?

Yes, it does, compared to the situation where no index is created on the foreign key column (*ProductID*), in which case, a full table scan is done (scanning all related pages in the table to retrieve desired data). But still, there is further scope to improve this query.

Let's assume that the Sales table contains 10,000 rows, and the following SQL selects 400 rows (4% of the total rows):

Hide Copy Code

```
SELECT SalesDate, SalesPersonID FROM Sales WHERE ProductID = 112
```

Let's try to understand how this SQL gets executed in the SQL execution engine:

1. The *Sales* table has a non-clustered index on the *ProductID* column. So it "seeks" the non-clustered index tree for finding the entry that contains *ProductID*=112.
2. The index page that contains the entry *ProductID* = 112 also contains all the clustered index keys (all primary key values, that is *SalesIDs*, that have *ProductID* = 112 assuming that the primary key is already created in the Sales table).
3. For each primary key (400 here), the SQL Server engine "seeks" into the clustered index tree to find the actual row locations in the corresponding page.
4. For each primary key, when found, the SQL Server engine selects the *SalesDate* and *SalesPersonID* column values from the corresponding rows.

Please note that in the above steps, for each of the primary key entries (400 here) for *ProductID* = 112, the SQL Server engine has to search the clustered index tree (400 times here) to retrieve the additional columns (*SalesDate*, *SalesPersonID*) in the query.

It seems that, along with containing clustered index keys (primary key values), if the non-clustered index page could also contain two other column values specified in the query (*SalesDate*, *SalesPersonID*), the SQL Server engine would not have to perform steps 3 and 4 above and, thus, would be able to select the desired results even faster just by "seeking" into the non-clustered index tree for the *ProductID* column and reading all the three mentioned column values directly from that index page.

Fortunately, there is a way to implement this feature. This is what is called "covered index". You create "covered indexes" in table columns to specify what additional column values the index page should store along with the clustered index key values (primary keys). Following is an example of creating a covered index on the *ProductID* column in the Sales table:

Hide Copy Code

```
CREATE INDEX NCLIX_Sales_ProductID--Index name
ON dbo.Sales(ProductID)--Column on which index is to be created
INCLUDE(SalesDate, SalesPersonID)--Additional column values to include
```

Please note that covered index should be created including a few columns that are frequently used in the `Select` queries. Including too many columns in the covered indexes would not give you too much benefit. Rather, doing this would require too much memory to store all the covered index column values, resulting in over consumption of memory and slow performance.

Use the Database Tuning Advisor's help while creating covered index

We all know, when a SQL is issued, the optimizer in the SQL Server engine dynamically generates different query plans based on:

- Volume of data
- Statistics
- Index variation
- Parameter value in TSQL
- Load on server

That means, for a particular SQL, the execution plan generated in the production server may not be the same execution plan that is generated in the test server, even though the table and index structure are the same. This also indicates that an index created in the test server might boost some of your TSQL performance in the test application, but creating the same index in the production database might not give you any performance benefit in the production application! Why? Well, because the SQL execution plans in the test environment utilizes the newly created indexes and thus gives you better performance. But the execution plans that are being generated in the production server might not use the newly created index at all for some reasons (for example, a non-clustered index column is not "highly" selective in the production server database, which is not the case in the test server database).

So, while creating indexes, we need to make sure that the index would be utilized by the execution engine to produce faster results. But, how can we do this?

The answer is, we have to simulate the production server's load in the test server, and then need to create the appropriate indexes and test those. Only then, if the newly created indexes improve performance in the test environment, these will most likely improve performance in the production environment.

Doing this should be hard, but fortunately, we have some friendly tools to do this. Follow these instructions:

1. Use SQL Profiler to capture traces in the production server. Use the Tuning template (I know, it is advised not to use SQL Profiler in a production database, but sometimes you have to use it while diagnosing performance problems in production). If you are not familiar with this tool, or if you need to learn more about profiling and tracing using SQL Profiler, read http://msdn.microsoft.com/en-us/library/ms181091.aspx.
2. Use the trace file generated in the previous step to create a similar load in the test database server using the Database Tuning Advisor. Ask the Tuning Advisor to give some advice (index creation advice in most cases). You are most likely to get good realistic (index creation) advice from the Tuning Advisor (because the Tuning Advisor loads the test database with the trace generated from the production database and then tried to generate the best possible indexing suggestion). Using the Tuning Advisor tool, you can also create the indexes that it suggests. If you are not familiar with the Tuning Advisor tool, or if you need to learn more about using the Tuning Advisor, read http://msdn.microsoft.com/en-us/library/ms166575.aspx.

Step 3: Defragment indexes if fragmentation occurs

OK, you created all the appropriate indexes in your tables. Or, may be, indexes are already there in your database tables. But you might not still get the desired good performance according to your expectations.

There is a strong chance that index fragmentation has occurred.

What is index fragmentation?

Index fragmentation is a situation where index pages split due to heavy insert, update, and delete operations on the tables in the database. If indexes have high fragmentation, either scanning/seeking the indexes takes much time, or the indexes are not used at all (resulting in table scan) while executing queries. Thus, data retrieval operations perform slow.

Two types of fragmentation can occur:

- **Internal Fragmentation**: Occurs due to data deletion/update operations in the index pages which end up in the distribution of data as a sparse matrix in the index/data pages (creates lots of empty rows in the pages). Also results in an increase of index/data pages that increases query execution time.
- **External Fragmentation**: Occurs due to data insert/update operations in the index/data pages which end up in page splitting and allocation of new index/data pages that are not contiguous in the file system. That reduces performance in determining the query result where ranges are specified in the "`where`" clauses. Also, the database server cannot take advantage of the read-ahead operations as the next related data pages are not guaranteed to be contiguous, rather these next pages could be anywhere in the data file.

How to know whether index fragmentation has occurred or not?

Execute the following SQL in your database (the following SQL will work in SQL Server 2005 or later databases). Replace the database name '*AdventureWorks*' with the target database name in the following query:

Hide Copy Code

```
SELECT object_name(dt.object_id) Tablename,si.name
IndexName,dt.avg_fragmentation_in_percent AS
ExternalFragmentation,dt.avg_page_space_used_in_percent AS
InternalFragmentation
FROM
(
```

```
    SELECT object_id,index_id,avg_fragmentation_in_percent,avg_page_space_used_in_percent
    FROM sys.dm_db_index_physical_stats (db_id('AdventureWorks'),null,null,null,'DETAILED'
)
WHERE index_id <> 0) AS dt INNER JOIN sys.indexes si ON si.object_id=dt.object_id
AND si.index_id=dt.index_id AND dt.avg_fragmentation_in_percent>10
AND dt.avg_page_space_used_in_percent<75 ORDER BY avg_fragmentation_in_percent DESC
```

The above query shows index fragmentation information for the '*AdventureWorks*' database as follows:

	Tablename	IndexName	ExternalFragmentation	InternalFragmentation
1	OrderDetails	CLIDX_OrderDetails	99.6875	57.1753150481838
2	SalesOrderDetail	AK_SalesOrderDetail_rowguid	75	36.03595255745
3	OrderDetails	CLIDX_OrderDetails	66.6666666666667	55.9838645910551
4	ProductReview	IX_ProductReview_ProductID_Name	66.6666666666667	41.1086483815172
5	WorkOrderRouting	PK_WorkOrderRouting_WorkOrderID_ProductID_Operat...	66.6666666666667	54.2006424511984
6	CountryRegion	PK_CountryRegion_CountryRegionCode	50	69.2611811218186
7	Vendor	PK_Vendor_VendorID	50	58.3271559179639
8	ProductProductPhoto	PK_ProductProductPhoto_ProductID_ProductPhotoID	50	59.1302199159871
9	StoreContact	IX_StoreContact_ContactID	50	51.1428218433407
10	StoreContact	IX_StoreContact_ContactTypeID	50	69.749196336002
11	ProductReview	PK_ProductReview_ProductReviewID	50	63.4667655053126
12	Employee	AK_Employee_NationalIDNumber	50	53.1134173461824
13	EmployeeAddress	PK_EmployeeAddress_EmployeeID_AddressID	50	73.4247590808806
14	EmployeeDepartme...	PK_EmployeeDepartmentHistory_EmployeeID_StartDate...	50	73.1158883123301
15	EmployeePayHistory	PK_EmployeePayHistory_EmployeeID_RateChangeDate	50	74.1536940943909
16	SalesPersonQuota...	PK_SalesPersonQuotaHistory_SalesPersonID_QuotaDate	50	53.3419817148505

Index fragmentation information

Analyzing the result, you can determine where the index fragmentation has occurred, using the following rules:

- *ExternalFragmentation* value > 10 indicates external fragmentation occurred for the corresponding index
- *InternalFragmentation* value < 75 indicates internal fragmentation occurred for the corresponding index

How to defragment indexes?

You can do this in two ways:

- Reorganize the fragmented indexes: execute the following command to do this:

Hide Copy Code

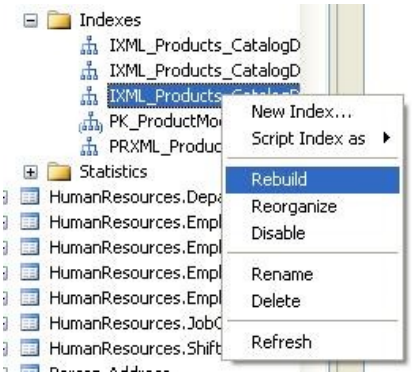
```
ALTER INDEX ALL ON TableName REORGANIZE
```

- Rebuild indexes: execute the following command to do this:

Hide Copy Code

```
ALTER INDEX ALL ON TableName REBUILD WITH (FILLFACTOR=90,ONLINE=ON)
```

You can also rebuild or reorganize individual indexes in the tables by using the index name instead of the 'ALL' keyword in the above queries. Alternatively, you can also use SQL Server Management Studio to do index defragmentation.



Rebuilding index using SQL Server Management Studio

When to reorganize and when to rebuild indexes?

You should "reorganize" indexes when the External Fragmentation value for the corresponding index is between 10-15 and the Internal Fragmentation value is between 60-75. Otherwise, you should rebuild indexes.

One important thing with index rebuilding is, while rebuilding indexes for a particular table, the entire table will be locked (which does not occur in the case of index reorganization). So, for a large table in the production database, this locking may not be desired, because rebuilding indexes for that table might take hours to complete. Fortunately, in SQL Server 2005, there is a solution. You can use the *ONLINE* option as *ON* while rebuilding indexes for a table (see the index rebuild command given above). This will rebuild the indexes for

the table, along with making the table available for transactions.

Last words

It's really tempting to create an index on all eligible columns in your database tables. But, if you are working with a transactional database (an OLTP system where update operations take place most of the time), creating indexes on all eligible columns might not be desirable every time. In fact, creating heavy indexing on OLTP systems might reduce the overall database performance (as most operations are update operations, updating data means updating indexes as well).

A rule of thumb can be suggested as follows: if you work on a transactional database, you should not create more than 5 indexes on the tables on an average. On the other hand, if you work on a data warehouse application, you should be able to create up to 10 indexes on the tables on an average.

What's next?

Applying indexing properly in your database would enable you to increase performance a lot in a small amount of time. But there are lots of other things you should do to optimize your database, including some advanced indexing features in SQL Server databases. These will be covered in the other optimization steps provided in the next articles.

Take a look at the next optimization steps in the article "**Top 10 steps to optimize data access in SQL Server: Part II (re-factor TSQLs and apply best practices)**". Have fun.

License

Share

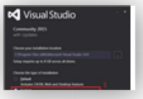
About the Author

Al-Farooque Shubho
Founder SmartAspects
Bangladesh

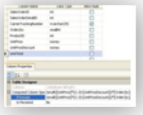
I write codes to make life easier, and that pretty much describes me.

You may also be interested in...

Top 10 steps to optimize data access in SQL Server: Part II (Re-factor TSQL and apply best practices)



Create Your Own SQL Server Management Studio 17 (SSMS) Extension



Top 10 steps to optimize data access in SQL Server: Part III (Apply advanced indexing and denormalization)

Getting Started with Django

Comments and Discussions

You must Sign In to use this message board.

Search Comments

Spacing Relaxed Layout Open All Per page 25 Update

First Prev Next

Best Way I understand the Index

Member 13249830 **15-Jun-17 1:20**

Hi Al-Farooque, Thanks for such a neat article regarding the Index. I have read lot article on Index but the best way I understand if buy your article. Its clean and clear. Thou I am very late to your page. But this is awesome way to understand the Index first and then can go from here to higher.
Question : I just wanted to know from the DMV query what that value indicate eg External if its 98 means 98% pages are not in order and Internal 80% indicate 80% of pages are empty.

Sign In · View Thread

Very useful

Mohsin Azam **22-Feb-17 21:37**

Very useful article for developers and other IT nerds.
Thanks for article series.

Sign In · View Thread

First Scenario of above explaination

Member 12947955 **12-Jan-17 18:31**

Hi,

First of all I would say that the article is too helpful. Properly written in a laymen language.

I am kind of new to SQL, my question is when the librarian arranged the books according to book number, how the customer is supposed to know the number.

Also, how the time is improved while searching for book name with only Primary key (Book number) created on it. In short how book number helped a little in finding a book using book name.

Let me know if my query is incomplete or not clear.

Thanks
Ankit

Sign In · View Thread

My vote of 5

Humayun Kabir Mamun **17-Nov-15 0:51**

Nice...

Sign In·View Thread

My Vote 5

Ravikiran72p 23-Aug-15 8:15

a simple thanks is not enough for this article. Thank you very very very much. Very nice article.

Sign In·View Thread

My Vote 5

Dharmesh .S. Patil 22-Jul-15 20:59

Nice

Sign In·View Thread

What if table have primary key on more than one columns ..

chirag solanki 90851 16-Oct-14 21:46

Is any problem if table have primary key on more than one column ..

```
CREATE TABLE [dbo].[SALE_DATA](
[CO_CODE] [varchar](2) NOT NULL,
[LC_CODE] [varchar](2) NOT NULL,
[CO_YEAR] [varchar](9) NOT NULL,
[DB_CODE] [varchar](4) NOT NULL,
[TRN_TYPE] [varchar](1) NOT NULL,
[BIL_NO] [varchar](15) NOT NULL,
[BIL_DT] [datetime] NULL,
[BILL_TIME] [datetime] NULL,
[TYPE] [varchar](1) NULL,
[AC_CODE] [varchar](10) NULL,
[CUST_ACCD] [varchar](8) NULL,
[AC_NAME] [varchar](60) NULL,
[PHONE] [varchar](15) NULL,
[ADDRESS] [varchar](255) NULL,
[RET_AMT] [float] NULL,
[NET_AMT] [float] NULL,
[DISC_PRC] [float] NULL,
[DISC_AMT] [float] NULL,
[TAXABLE_AMT] [float] NULL,
[TAX_CODE] [varchar](2) NULL,
[TAX_PRC] [float] NULL,
[TAX_AMT] [float] NULL,
[SCH_PRC] [float] NULL,
[SCH_AMT] [float] NULL,
[ROF_AMT] [float] NULL,
[TOT_AMT] [float] NULL,
[CRN_AMT] [float] NULL,
[CRNADJ_AMT] [float] NULL,
[CRN_USEAMT] [float] NULL,
[CSH_AMT] [float] NULL,
...
CONSTRAINT [PK_SALE_DATA] PRIMARY KEY CLUSTERED
(
[CO_CODE] ASC,
[LC_CODE] ASC,
[CO_YEAR] ASC,
[DB_CODE] ASC,
[TRN_TYPE] ASC,
[BIL_NO] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

This is my sales data strucutre which have 6 primary key .. And table is really slow on data selection..

Any suggetion ?

Sign In · View Thread

My vote of 5

Sibeesh KV 29-Sep-14 21:18

My vote of 5

Sign In · View Thread

Very Good!

Omar Gameel Salem 16-Jul-14 14:46

I like your way of explaining things, you really engage the reader
5+

Sign In · View Thread

Thanks for the article. Here's another good read

Member 10244503 21-May-14 1:46

Thanks for your article!

Here's another one which touches on more techniques, and not so much indexing:

http://www.codemag.com/Article/1208111[^]

Sign In · View Thread

how can an index bring benefit, if the search is by another column?

Paul Pacurar 16-Feb-14 21:45

As in example, you said it was an improvement to have an index, even if the book was searched by name (not a key). But index only contains a number, there is no relation between that number and the title of the book (read any other column on a table). How can that index bring a benefit (20 minutes instead of 2 hours, in the example) ?

Sign In · View Thread

Re: how can an index bring benefit, if the search is by another column?

LuckyLouis 14-May-15 15:24

I have the same question. Anyone can explain why this will be faster?

Louis

Sign In · View Thread

Re: how can an index bring benefit, if the search is by another column?

Member 12947955 12-Jan-17 18:37

Same Question.

Sign In · View Thread

Thanks a lot

King_Fisher 10-Feb-14 20:49

its very nice to understand,thanks man

Sign In · View Thread

Nice Article

Skylem 25-Nov-13 19:46

As an aspiring DBA in the future, this article helps me a lot. Thanks comrade.

Sign In · View Thread

Thanks

Shafeeque Ahammed 28-Oct-13 0:22

Thanks Farooq, It was very usefull.

Sign In · View Thread

Really Helpful

sund7wells 4-Sep-13 23:36

Your article very helpful to understand the Cluster and Non-Cluster index types in very simple and effective way.
Thanks for sharing.

Sign In · View Thread

Question

Member 9901458 29-Apr-13 20:14

Some interviewer asked me

write an index for data retrieval from a table

can you help me in this?

Sign In · View Thread

Helpful

Dhaval Joshi 8-Apr-13 18:21

This is very helpful article for someone who is new to SQL perf optimization. Thanks!

Sign In · View Thread

My vote of 5

amish kumar 11-Mar-13 6:57

very very useful

Sign In · View Thread

useful article

Member 9579330 5-Feb-13 22:03

thks, this article is very useful to me. i' m look forward to reading other ones related.

Sign In · View Thread

My vote of 5

Jair Avilés 4-Dec-12 6:51

Thank you so much for your help.

This article is utterly useful for me.

Regards...

Sign In · View Thread

My vote of 5

eupendra 21-Oct-12 6:36

Excellent article

Sign In · View Thread

My vote of 5 — **Akram El Assas** — **8-Oct-12 20:20**

First class article. Thanks!

Sign In · View Thread

Ebrahim.... — **Ebrahims** — **13-May-12 6:31**

Very Good....

Sign In · View Thread

Last Visit: 31-Dec-99 18:00 Last Update: 10-May-18 10:28 Refresh 1 2 3 Next »

General News Suggestion Question Bug Answer Joke Praise Rant Admin