Faculty of Computer & Information Sciences
Ain Shams University
Subject: DBA 271 File Organization
Year: (2nd year)undergraduate
Academic year: 2nd term 2019-2020

# Research Topic (6)

# Title: Database Indexes

## Introduction (Ref. 1)

**INDEXING** is a data structure technique that permits you to rapidly retrieve records from a database file and optimize the performance of the database by limiting the number of disk accesses required when a query is processed. An Index is a little table having only two columns. The first column includes a copy of the primary or candidate key of a table. The second column of the table contains a set of pointers for holding the location of the disk block where that particular key value stored.

## 1. Different Types of Database Indexes: (Ref. 2)

- Primary Index
  - Dense Index
  - Sparse Index
- Secondary Index
  - Non-clustered Index
- Clustering Index
  - Non-clustered Index
- Other types
  - Unique
  - Full-text
  - Spatial

## 2. Comparison Between Index Types: (Ref. 2) (Ref. 3)

- **Primary Index:**

  Primary Index is defined on an ordered and organized data file that contains the key fields of the table and a pointer to the non-key fields of the table. The primary index is created automatically when the table is created in the database and also considered as the key of the relation.

<u>The primary indexing is divided into another two types:</u>

- **Dense Index:**

  In dense index, there is an index record for each search key value in the database. This helps you to search quicker but requires more space to store index records. Index records contain search key value and a pointer to the real record on the disk.

- **Sparse Index:**

  In sparse index, index records are not made for each search key. An index record here contains a search key and a real pointer to the data on the disk. To look through a record. Sparse index helps you to resolve the issues of dense indexing, we initially continue by index record and reach at the actual location of the data. It needs less space. If the data we are searching for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.

- **Secondary Index:**

  Secondary Index is generated by a field which is a candidate key and has a unique value in each record, or a non-key with duplicate values.

  It's also known as a non-clustering index.

  - **Non-clustered:**

    A non clustered index is made using clustered index. Each record line in the non clustered index has non clustered key value and a row locator. It's known that all locator positions to the data row in the clustered index that has a key value.

  - **Filtered:**

    A non clustered index. Totally improved and optimized for query data from a well characterized subset of data. A filter is used to predicate a portion of rows in the table to be indexed.

- **Clustering Index:**

  The importance of clustered index is to sort and store the rows data of a table based on the order of clustered index key. Clustered index key used to be implemented in B-tree index structure.

- **Other types:**
  - **Unique:** Unique index guarantees the availability and accessibility of only non-duplicate values and therefore, every row is unique.
  - **Full-text:** Its support is efficient in searching and looking through words in string data. This sort of indexes is utilized in certain database managers.
  - **Spatial:** It encourages the ability for performing operations in efficient way on spatial objects. To do this, the column ought to be of geometry type.

## 3. Clustered vs. Non-Clustered Indexes: (Ref. 4) (Ref. 5)

| Clustered Index | Non-Clustered Index |
|---|---|
| • Can be used only one per table | • Can be used many times per table |
| • Usually made on the primary key | • Usually made on any key |
| • A Clustered index always has index id of 0 | • A Non-Clustered index always has index > 0 |
| • Faster to read than non clustered as data is physically stored in index order | • Faster to insert and update operations than a clustered index |
| • Do not need extra space to store logical structure | • Use extra space to store logical structure |
| • The leaf node contains data pages of the table on which it is made. | • The leaf nodes consists of Index pages which contain Clustering Key or RID to find Data Row. |

### Clustered indexes

physically order the data on the disk. This implies no additional data is needed for the index, however, there can be only one clustered index (clearly). Accessing data utilizing a clustered index is quickest.

**Non-clustered indexes**

have a copy of the data from the indexed columns kept ordered along with pointers to the actual data rows (pointers to the clustered index if there is one). This implies getting to data through a non-clustered index has to go through an extra layer of indirection. However, if you select only the data that's accessible in the indexed columns you can get and recover the data back directly from the copied index data (that's why it's a smart thought to select just the columns that you need and not use ).

The two kinds of indexes will improve performance and execution when select data with fields that utilize the index but will hinder update and insert operations. As a result of the slower insert and update clustered indexes ought to be set on a field that is ordinarily incremental i.e. Id or Timestamp.

---

## 4. Description of the Implementation of the code: (Ref. 6)

I'll discuss the steps that I did to reach the final output:

1. I made a XML file called "people" and put all the data in it. The data are (ID, Name, City, Country) and wrote all the code that used to make the file and its data in a function called "fill_all_data" in class "Data". I wrote 40 records and assumed that city is unique for each record to optimize searching using city's name.

2. I sorted the data that in the XML file according to the "ID" in ascending order.

3. I divided the sorted data into 4 clusters (XML files), each file contains 10 records, and the files are "cluster_1", "cluster_2", "cluster_3", "cluster_4". I implemented that in code by making a for loop that loops over all records that in the xml file "people" and increase a counter in each iteration. When I reached 10 records , I put them in xml file and by the way in the rest files.

4. I made a XML file called "Entry_Indices_Table" that carries out the entry indices and the files' names for the fourth files to make it easy for me when I search using the ID or the name of the city.

5. The output of the code will be two sentences to enable the user to choose what he wants. If he wants to search by ID, he should enter "1".And if he wants to search by City's name, he should enter "2". Then the output will be the number of iterations.

6. The logic of how the number of iteration is calculated:
   - **Firstly,** I'm using a counter to search on the small file that have the entry indices to the fourth files.
   - **Secondly,** When I get the entry id of the particular file that I'll search on. I'll make another counter to count the number of iterations until I get the wanted record.
   - **Thirdly,** I'll sum up the two counters that I got in the first and second steps then display them.

**NOTICE**

**There's another detailed description of the implementation in the comments of the code.**

## 5. Test Cases:

| Test Case | No of records in data | No of comparisons made to search for a record | |
|---|---|---|---|
| | | Search using ID | Search using City |
| 1 | 8 | ID:13<br>Num. of Iterations: **5** | City: Beni Ebeid<br>Num. of Iterations: **13** |
| 2 | 10 | ID: 15<br>Num. of Iterations: **7** | City: Cairo<br>Num. of Iterations: **15** |
| 3 | 39 | ID: 40<br>Num. of Iterations: **13** | City: 10th of Ramadan<br>Num. of Iterations: **40** |
| 4 | 15 | ID: 12<br>Num. of Iterations: **4** | City: Beni Mazar<br>Num. of Iterations: **12** |
| 5 | 26 | ID: 26<br>Num. of Iterations: **9** | City: El Husseiniya<br>Num. of Iterations: **26** |

## 6. A comment on the results that in the previous table:

The number of iterations when I search using city's name is bigger than the number of iterations when I search using the ID.

So from here , we'll notice the importance of the Clustered Indexing. So I should divide the big file into small files (clusters) to decrease the number of iterations and increase the speed of it.

## Conclusion

Indexing has an incredible role in databases — it structures the data so it very well may be questioned quicker. In the end, you'll have more data and information on writing or deleting over the speed of reading or keeping the write/delete speed even with the reading speed by not utilizing indexes.

## References:

**[1] Geeks for Geeks** (Website): Link

**[2] Tutorials Point** (Website): Link

**[3] Guru99** (Website): Link

**[4] Stack overflow** (Website): Link

**[5] C# Corner , blog by Jaipal Reddy** (Blog): Link

**[6] Microsoft Docs** (Website): Link