SQL Server Indexes

## Why Use Indexes?

## Uniqueness

Indexes enforce uniqueness across columns (primary key constraints).

## Performance

Indexes improve performance of queries

Bad indexes, design, and queries effect performance.

Indexes must be created based on workloads.

Indexes must be created based on queries (not necessarily on database designs).

# Index Approaches

Two ways to approach indexes: New System (just being created), under performing older system.

## New System

Index created based on unique requirements (primary keys).

Unique constraints are defined based on what columns need to be unique and can be enforced via primary keys or unique indexes.

Indexes can also be created on foreign key columns (may be changed).   
  
Can create indexes to support expected common queries.

Need to get expectation on common use to determine how to index.   
  
Indexes may change between expected and actual query usage.

## In Use System

First need to identify: Most Resource Intensive Queries and important queries (frequently used, used by key players, etc.).

Index based on these. These expectations are frequently re-evaluated.

# Index Architecture Basics

## Index Architecture

Index architecture trees?

Data is stored in 8kb chunks called pages (this is fixed) within the index file.   
  
Most system queries and procedures return item sizes in system pages.   
  
Index key: Columns index is built on (index is organized by these columns).

I.e. an ocean liner database organized by name and class of ship (I.e. Name: Titanic, Class: Olympic)

## Index Tree Structure

### Leaf level

The individual pages of an index are known as leaves. These represent the smallest level of organization and each leaf is equivalent to one 8kb page within the index (data file) itself.

### Intermediate Level

Above the page level is the intermediate level. The intermediate level consists of files consisting of the first row of each leaf. Each Intermediate file will consist of pages until there are either no more pages, or the page is full (i.e. if the pages start with Titanic: Olympic, Lusitania: Lusitania, Bismarck: Vaterland, and Empress of Ireland: Empress, and each Intermediate contains two rows the intermediate pages will look like: Titanic: Olympic, Lusitania: Lusitania and Bismarck: Vaterland, Empress of Ireland: Empress).   
  
An Intermediate level will consist of 0 or more pages. If all leaves would fit on one page, there is no Intermediate level.

### Root Level

The level above Intermediate is called root. Like the Intermediate level, the Root page consists of the first row for each page of the Intermediate level. Unlike the Intermediate level, there will ever only be 0 or 1 root pages. If there is only a single leaf page, then there will be no root pages. Otherwise there will be 1 AND ONLY EVER 1 root page.

## Index Operations

There are various operations that can be performed on an index during an execution plan.

### Forward Seek

Navigation down an index from the root to towards the leave (through the various intermediate levels as appropriate). This action is moving down the tree, or forward towards the lower levels.   
  
Seeks begin at the root level. The index will look at the root level until it finds a key bigger than the key it is searching for. It will then take the key from the root that is the closest match to the given key, and go to that page in the Intermediate level.

Starting at the page beginning with the root’s closest match, it repeats the process in the intermediate level until it finds the closest match.   
  
Finally, it takes the closest match on the intermediate level, and looks on the leaf that begins with that page until it finds the row(s) matching the index key. It will keep scanning until it finds a row greater than its key. Finding a greater row, signifies all matching rows have been found, since indexes group keys by ascending order.

I.e. if looking for the key, Titanic, it will start in the root until it finds a key closest to Titanic (say Lusitania, Olympic, Vaterland are the root entries. It will then move to the intermediate page beginning with Olympic, and scan until it finds a match bigger than Titanic and begin on the leaf that begins with the key before the bigger match. It will then scan that page until it finds “Titanic”.   
  
The key for a seek is called a “seek predicate”. Rows that do not match a seek predicate are never even read.

### Index Scan

This is a traversal across the leaves without moving throughout the hierarchy. This is a sideways or lateral movement like a scan made by a sensor.   
  
A scan is often used on a command (such as summing), or if there is a where cause to the query that requires all leaf pages be read (i.e. navigating through the index is not helpful). For example, if the index keys are Liner Name and year of maiden voyage, and the user wants all ships sailing after 1912, this will be performed as a scan. Rows of 1912 will be scattered across various leaves, and so each leaf must be read, and returned if matching the condition.   
  
A key difference between a seek and a scan, is that all rows are read in a scan, and then discarded if they don’t match the predicate, or returned if they do. A seek will only read rows matching the predicate.

### Key Lookup

### A single-row seek done on a clustered index.

## Indexes in Execution Plans

Indexes can be created on specific columns as defined via query. The execution plan when querying an index will show any index operations used during that query (seek, scan, etc). By hovering over the index operation, various statistics can be seen about the index. Particular points are how many rows are red vs returned (ideally these rows will match. Otherwise, there is wasted work performed), The costs of the index int erms of CPU, and operators, etc and predicates applied to the operation. Right Click -> Properties for more detail on this information.   
  
This can be done for scans as well. If statistics are turned on for messages, certain details about the action can also be seen such as many logical (page) reads are performed on an index. For example a seek with 2 logical reads will have read the index, and the page containing the record in question.   
  
Some seeks may have multiple predicates, if additional predicates need to be applied.

## Clustered Indexes (for Table Designs)

### Introduction

What is a Cluster Index?

Clustered Index’s leaf rows are comprised of the actual datapages of the table itself. The clustered index defines the physical storage of the table.   
  
Clustered Indexes are updated whenever a table is.

There can only be one clustered index per table. A table without a clustered index is called a heap.   
  
Two schools of thoughts regarding Clustered Indexes.   
  
Create the clustered index to organize the table, or create it on the most frequent access path of the table.

### Clustered Indexed Guidelines for Organizing the Table

### Narrow

Refers to size and bytes of the index key columns.

Lowest practical size for a clustered index key is 4 bytes, and the maximum allowed size is 900 bytes.   
  
The bigger, and wider a clustered index key is, the more levels are needed in the key (this causes more overhead).   
  
Clustered index is added to every other index on the table. This means that any overhead in the clustered index is added to the non-clustered indexes and can easily add up quite quickly.

### Unique

Must be unique as it’s used by the rows address. This means that the index must created with either the unique constraint, or primary key constraint.   
  
IF the key is not created as unique, then an extra hidden column is added to make it unique. Column is 4-byte signed integer. This is set to null at the first occurrence of a value or set of values and then given a value of 1 for each subsequent occurrence of that value or set of values. It’s possible to run out of unique identifiers but will require 2.1 billion occurrences of the same value or set of values within the table (i.e. have Titanic in the table 2.1 billion times).   
  
Clustered index keys should be applied to mostly unique tables (i.e. Liner name and class, rather than just name. It’s likely to have several ships of the same name, but not likely to have ships with the same name within the same class or launched in the same year.

### Unchanging

The clustered index key defines where a row is found in the index (i.e. if basing off the name of ships, Titanic is found pages with the letter t (or any subset it fits into).

If the key for a row changes, (i.e. an update statement), then that update is split into two operations. The first operation removes the row from any indexes (clustered and non-clustered), and then added back to the appropriate leaf and intermediate levels reflective of the new index key. These updates can cause page-splits and partially empty pages on the leaf level as the keys are moved around which can cause slowdowns on the indexes.

### Ever-increasing

If an index has to go into a page that’s already full, a page-split occurs. The more page-splits within an index, the slower the index.   
  
By having index key columns continually growing (i.e. an autoincrement ID field), page-splits will not occur as new records are only inserted into the last row of the index.   
  
If an index needs to insert into a full page, the page is cut in half, the second half moved to a new page, the new row inserted onto the original page, and any intermediate rows updated as necessary.

## Clustered Index Demo, Guidelines and Table Impacts

sys,dn\_db\_partition\_Stats shows the table (or index) in page counts in used page count, reserved page count, and row\_count

### Clustered Index Trade Offs

### Pros of indexes designed to organize a table

Table will be small as possible.

Minimal page splits.

Less update overhead.

Additional non-clustered indexes needed

### Pros of indexes designed to Support Queries

Table can take up more space than necessary

May lead to page splits (and potential additional overhead on updates and inserts).

Clustered indexes can reduce need for other indexes.