Index Internals
What You Really Need to Know!
SDV306

Kimberly L. Tripp
SQLskills.com
Kimberly@SQLskills.com

Author/Instructor: Kimberly L. Tripp

SQL skills

Microsoft MVP Most Valuable Professional

Microsoft Regional Director PROGRAM

- Consultant/Trainer/Speaker/Writer
- Founder, ***SYSolutions*, Inc.** (www.SQLskills.com)
  - e-mail: Kimberly@SQLskills.com
  - blog: http://www.SQLskills.com/blogs/Kimberly
  - Twitter: @KimberlyLTripp
- Writer/Editor for SQL Magazine www.sqlmag.com
- Author/Instructor for SQL Server courses: Designing for Performance, Indexing for Performance, Maintenance, Disaster Recovery/HA
- Author/Manager of SQL Server 2005 Launch Content, Co-author/Manager for SQL Server 2008 Jumpstart Content, Author/Speaker at TechEd, SQL Connections, SQLPASS, ITForum, Conference Co-chair for SQL Connections
- Author of several SQL Server Whitepapers on MSDN/TechNet: Partitioning, Snapshot Isolation, Manageability and SQLCLR for DBAs
- Author/Presenter for more than 25 online webcasts on MSDN and TechNet (two series and other individual webcasts)
- Co-author for *SQL Server 2008 Internals*, *SQL Server MVP Deep Dives*, and *SQL Server 2000 High Availability*
- Presenter/Technical Manager for SQL Server 2000 High Availability DVD and various 2005/2008 HOL DVDs

SQL SERVER MVP DEEP DIVES
Microsoft SQL Server 2008 Internals
Kalen Delaney
Microsoft SQL SERVER 2000 HIGH AVAILABILITY

Overview

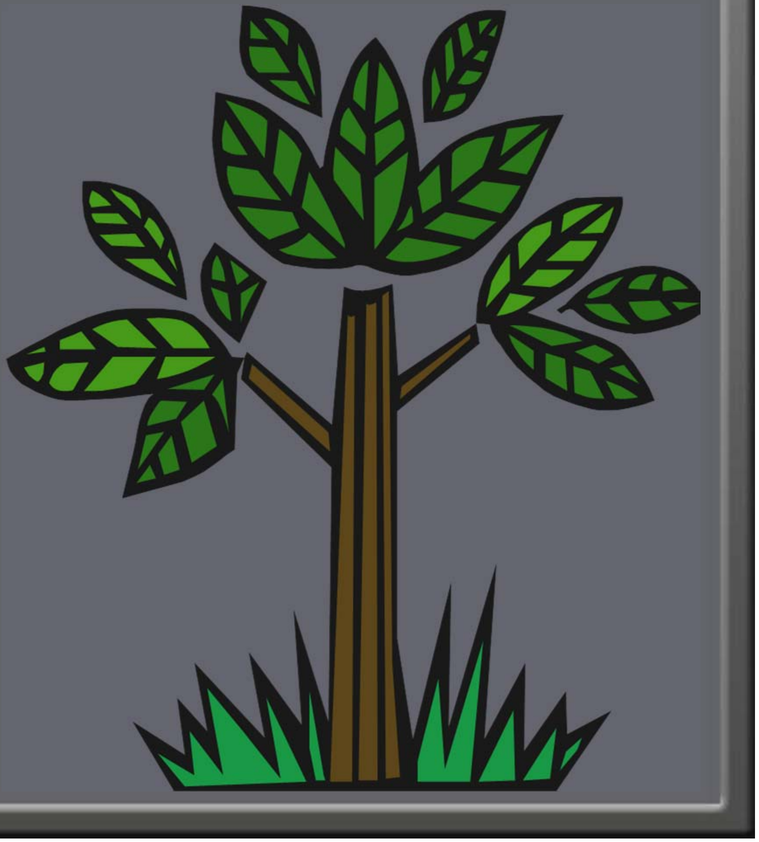
- Index Concepts
- Table Structure
- Index Internals
  - Heaps
  - Why cluster
  - Table usage
  - Employee table case study

Index Concepts – Tree Analogy

If a tree were data and you were looking for leaves with a certain property, you would have two options to find that data….

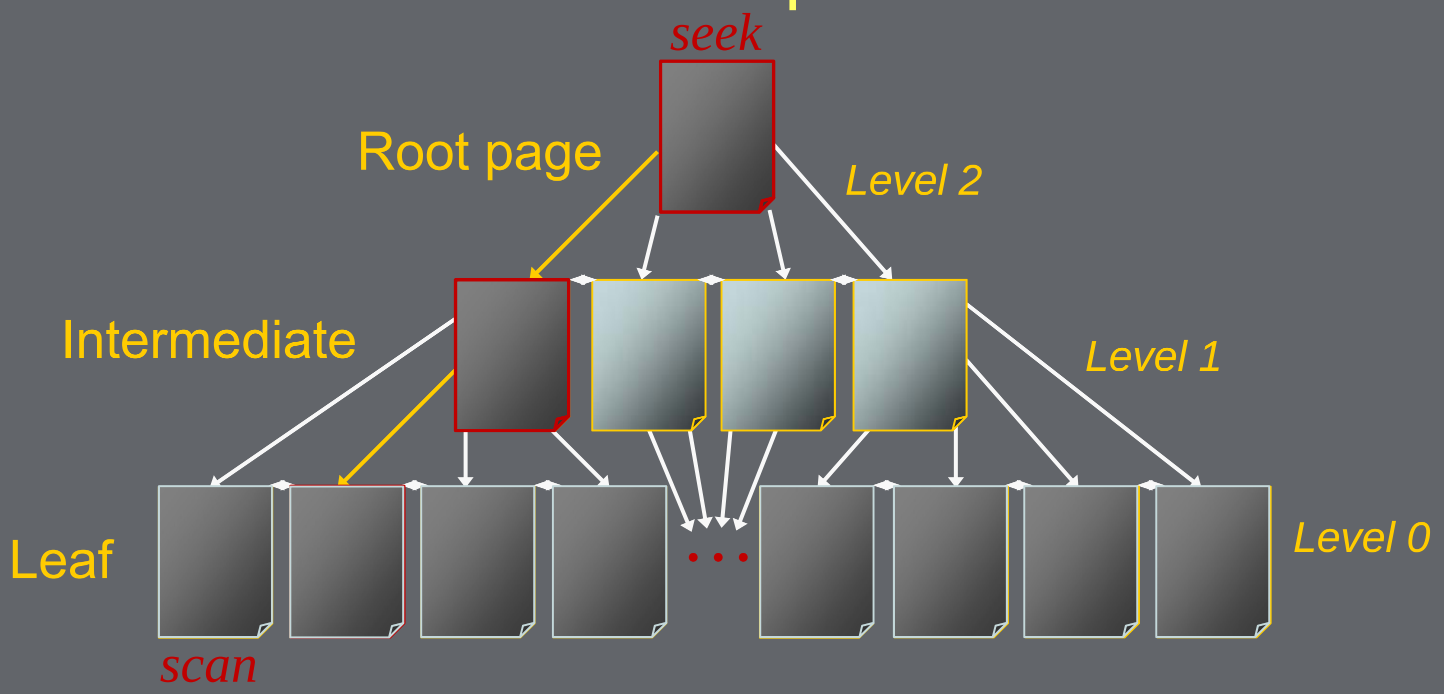
1) Touch every leaf – interrogating each one to determine if they held that property…SCAN

2) If those leaves (which had that property) were grouped such that you could start at the root, move to the branch and then directly to those leaves…SEEK



Seek vs. Scan

- Seek – starts at the root and uses the balanced structure to move from top to bottom



- Scan – moves through the leaf level from left to right (possibly right to left)

SQL Server CONNECTIONS

Table Structures

- Overview
- Heap
- Heap issues – why cluster?
- Table usage
- Clustered index review
- Key constraints create indexes
- Nonclustered index review
- What do we know?
- What should we do?

Table Structure – Overview

- HEAP – a table without a clustered index
- Clustered table – a table with a clustered index
- Nonclustered indexes DO NOT affect the base table's structure
- However, nonclustered indexes are affected by whether or not the table is clustered…
- **Hint: The nonclustered index dependency on the clustered index should impact your choice for the clustering key!**

What Type of Table is Best?

- Heaps offer excellent benefits for staging tables
  - Parallel data load and Parallel index create (after load)
  - Then switch into a PT or PV
- For OLTP/DS tables – user based modifications (not batch) – performance is better with a clustered index
- However, clustered indexes require administrative maintenance to alleviate negatives with regard to space
  - Heaps re-use space as records can be in any order
  - Clustered tables have splitting to logically make room for more rows (keeping the leaf level's linked list intact)
- Are all clustered indexes going to give the same gains?
  - NO – it's not that you should have just any clustered index…
  - For true performance gains you must have the **RIGHT** clustered index!

Clustered Index Overview

- Not required – although highly recommended
- Only one per table
- Physical order applied at creation
  - Very expensive to create (SQL makes a copy of the data)
- Logical order maintained through a doubly-linked list
- Requires ongoing and automated maintenance
  - Can be expensive to manage
- Need to choose wisely!
  - This might allow you to reduce certain maintenance requirements

Clustered Index Criteria

- Unique
  - Yes – No overhead, data takes care of this criteria
  - NO – SQL Server must “uniquify” the rows on INSERT
    - This costs time and space
    - Each duplicate has a 4-byte “uniquifier”
- Narrow
  - Yes – Keeps the NC indexes narrow
  - NO – Makes the NC indexes unnecessarily wide
- Static
  - Yes – Improves performance
  - NO – Costly to maintain during updates to the key

- In fact, an identity column that’s ever-increasing is often ideal…

Clustering on an Identity

The Good

- Naturally Unique
  - (should be combined with constraint to enforce uniqueness)
- Naturally Static
  - (should be enforced through permissions and/or trigger)
- Naturally Narrow
  - (only numeric values possible, whole numbers with scale = 0)
- Naturally creates a hot spot…
  - Needed pages for INSERT already in cache
  - Minimizes cache requirements
  - Helps reduce fragmentation due to INSERTs
  - Helps improve availability by naturally needing less defrag

Key Constraints Create Indexes

- Primary key constraint
  - Defaults to unique clustered
  - Only one per table

```
ALTER TABLE Employee
  ADD CONSTRAINT EmployeePK
    PRIMARY KEY CLUSTERED (EmployeeID)
```

- Unique key constraints
  - Default to unique nonclustered
  - Maximum of 249 in SS2005 or 999 in SS2008 per table

```
ALTER TABLE Employee
  ADD CONSTRAINT EmployeeSSNUK
    UNIQUE NONCLUSTERED (SSN)
```

Nonclustered Indexes

The Book Analogy

- Think of a book – with indexes in the back
- The book has one form of logical ordering
- For references – you use the indexes in the back… to find the data in which you are interested you look up the key
- When you find the key – you must lookup the data based on its location… i.e. a “bookmark” lookup
- The bookmark always depends on the (book) content order

Index – Species Common Name

Index – Animal by Type, Name
Bird, Mammal, Reptile, etc…

Index – Animal by Country, Name

Index – Animals by Habitat, Name
Air, Land, Water

Index – Species Scientific Name

SQL Server
CONNECTIONS

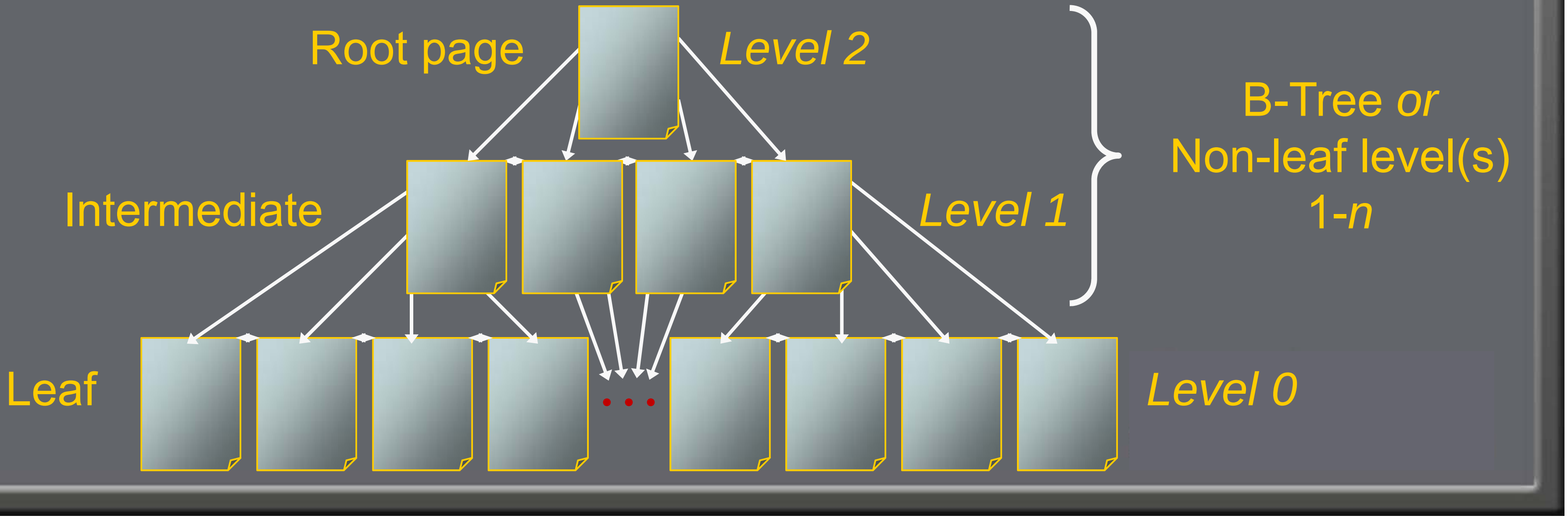
Nonclustered Index Overview

- Not required – although critical to achieving optimal performance
- Maximum of: 249 per table and increased to 999 per table in SQL Server 2008
- Leaf structure separate from base table
- Based on the heap’s fixed rid or clustering key
- Logical order of index entries maintained through a doubly-linked list
- By far – the FASTEST type of index for range queries if it covers the query!
- Don’t ask for *, limit your queries!!!

Physical Index Levels

Generic Overview

- **Leaf Level** - Contains *something* for every row of the table – in indexed order.
- **Non-leaf Level(s) or Balanced-Tree** - Contains *something* – specifically representing the FIRST value – from every page of the level below. Always at least one non-leaf level. If only one, then it's the root and only one page. Intermediate levels are not a certainty.



Employee Table

Internals Case Study

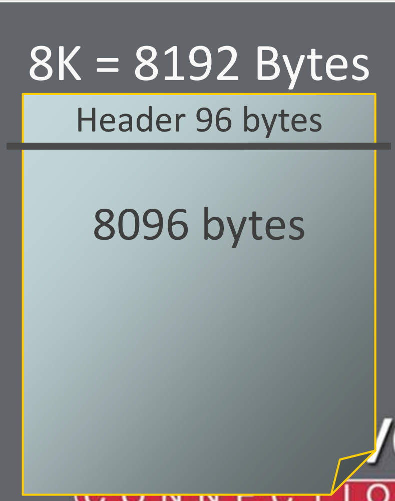
Average row size = 400 bytes/row

```
CREATE TABLE Employee
(
EmployeeID              Int             NOT NULL        Identity,
LastName                nvarchar(30)    NOT NULL,
FirstName               nvarchar(29)    NOT NULL,
MiddleInitial           nchar(1)        NULL,
SSN                     char(11)        NOT NULL,
…other columns…)
```

$$\frac{\text{8096 bytes/page}}{\text{400 bytes/row}} = \text{20 rows/page}$$

80,000 current Employees ∴ rows

$$\frac{\text{80,000 employees}}{\text{20 rows/page}} = \text{4000 pages}$$



Clustered Employee Table

Step 1 – Physically order data

Review the index level definitions…

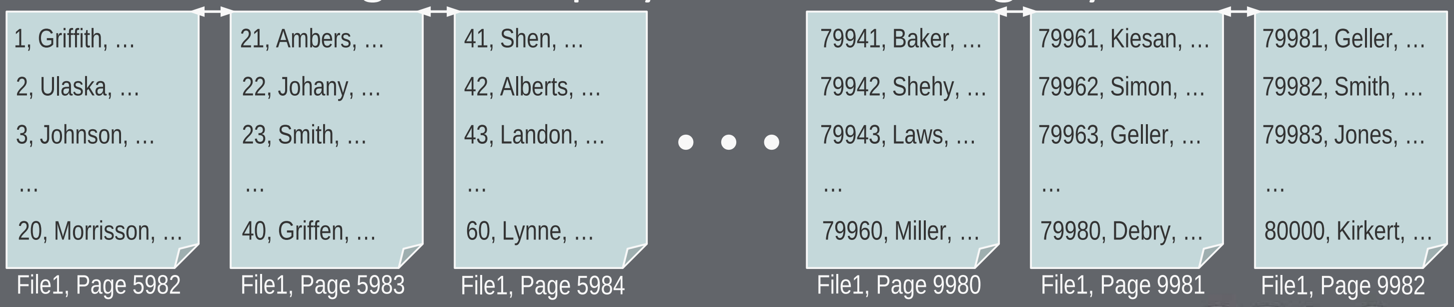
Does this seems to match one of the definitions?

Yes!

When a table is clustered

the data becomes the leaf level of the clustered index!

4000 Pages of Employees in Clustering Key Order



1, Griffith, …
2, Ulaska, …
3, Johnson, …
…
20, Morrisson, …
File1, Page 5982

21, Ambers, …
22, Johany, …
23, Smith, …
…
40, Griffen, …
File1, Page 5983

41, Shen, …
42, Alberts, …
43, Landon, …
…
60, Lynne, …
File1, Page 5984

• • •

79941, Baker, …
79942, Shehy, …
79943, Laws, …
…
79960, Miller, …
File1, Page 9980

79961, Kiesan, …
79962, Simon, …
79963, Geller, …
…
79980, Debry, …
File1, Page 9981

79981, Geller, …
79982, Smith, …
79983, Jones, …
…
80000, Kirkert, …
File1, Page 9982

Clustered Employee Table

Step 1 – Physically Order Data

Step 2 – Add the tree structure

- *starting from the leaf level and going up to a root of 1 page*
- B-tree entry = Index key value + pointer + row overhead*
  - Pointer = Page pointer of 6 bytes = 2 for FileID + 4 for PageID
  - Row overhead varies based on *many* factors
    (min of 1 byte in the row)
- Non-leaf level entry for clustered index on EmployeeID = 11
  - 4 bytes for EmployeeID (int) + 6 bytes for page pointer
    + 1 byte for row overhead

$$\frac{\text{8096 bytes/page}}{\text{11 bytes/entry + 2 bytes in slot array}} = \text{622 index entries per non-leaf level page}$$

How many entries to store? 4,000

Remember – a non-leaf level contains one entry for every PAGE of the level below.

Clustered Employee Table

- Step 1 – Physically Order Data
- Step 2 – Add the balanced tree

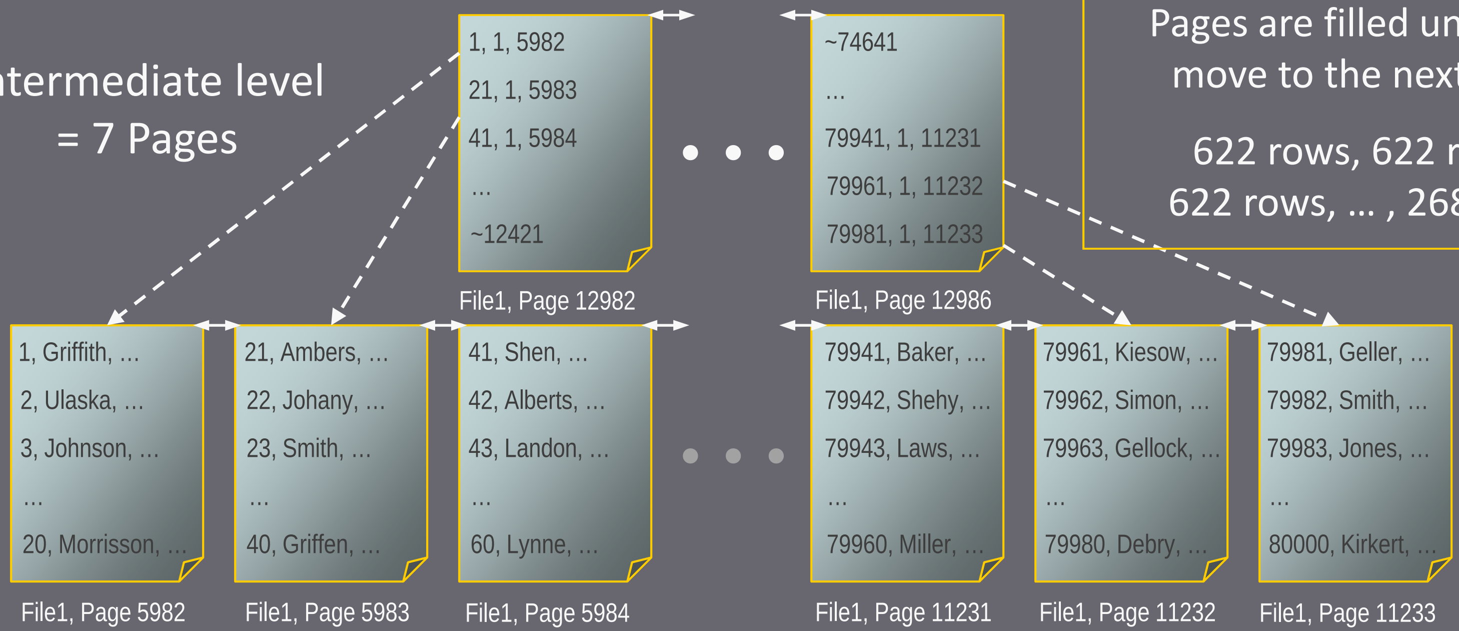
starting at the leaf level and working up to a root of 1 page

$$\frac{\text{4000 entries to store}}{\text{622 entries/page}} = \text{7 pages in the first B-tree level}$$

Intermediate level
= 7 Pages

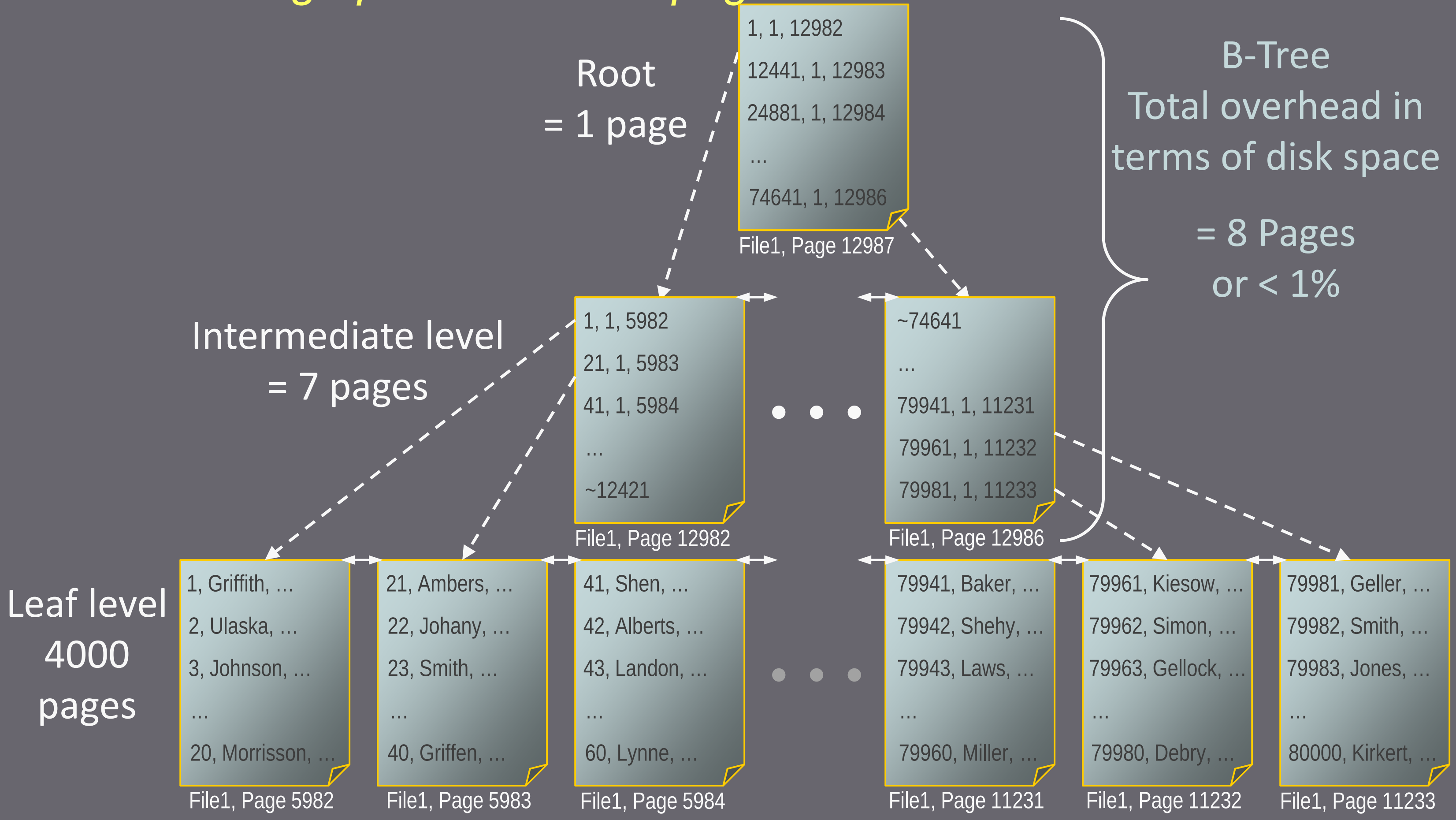
Pages are filled until they move to the next page

622 rows, 622 rows, 622 rows, … , 268 rows



Clustered Employee Table

- Step 2 – Complete the balanced tree
  continuing up to a root of 1 page



Root
= 1 page

1, 1, 12982
12441, 1, 12983
24881, 1, 12984
…
74641, 1, 12986

File1, Page 12987

B-Tree
Total overhead in
terms of disk space

= 8 Pages
or < 1%

Intermediate level
= 7 pages

1, 1, 5982
21, 1, 5983
41, 1, 5984
…
~12421

File1, Page 12982

~74641
…
79941, 1, 11231
79961, 1, 11232
79981, 1, 11233

File1, Page 12986

Leaf level
4000
pages

1, Griffith, …
2, Ulaska, …
3, Johnson, …
…
20, Morrisson, …

File1, Page 5982

21, Ambers, …
22, Johany, …
23, Smith, …
…
40, Griffen, …

File1, Page 5983

41, Shen, …
42, Alberts, …
43, Landon, …
…
60, Lynne, …

File1, Page 5984

79941, Baker, …
79942, Shehy, …
79943, Laws, …
…
79960, Miller, …

File1, Page 11231

79961, Kiesow, …
79962, Simon, …
79963, Gellock, …
…
79980, Debry, …

File1, Page 11232

79981, Geller, …
79982, Smith, …
79983, Jones, …
…
80000, Kirkert, …

File1, Page 11233

Nonclustered Index

Unique Constraint on SSN

- Leaf level entry for nonclustered index

  = NC index column(s) + Fixed RID (if Heap) *or* clustering key
  + Row overhead*

  *Row overhead (min of 1 byte in the row)

  = SSN of 11 bytes + EmployeeID (clustering key) of 4 bytes
  + 1 byte in the key

  = 16 bytes/entry + 2 bytes in the slot array

- Entries per leaf level page

$$\frac{\text{8096 bytes/page}}{\text{16 bytes/entry + 2 bytes in slot array}} = \text{449 index entries per leaf level page}$$

- Pages for leaf level

$$\frac{\text{80,000 rows}}{\text{449 rows/page}} = \text{179 pages}$$

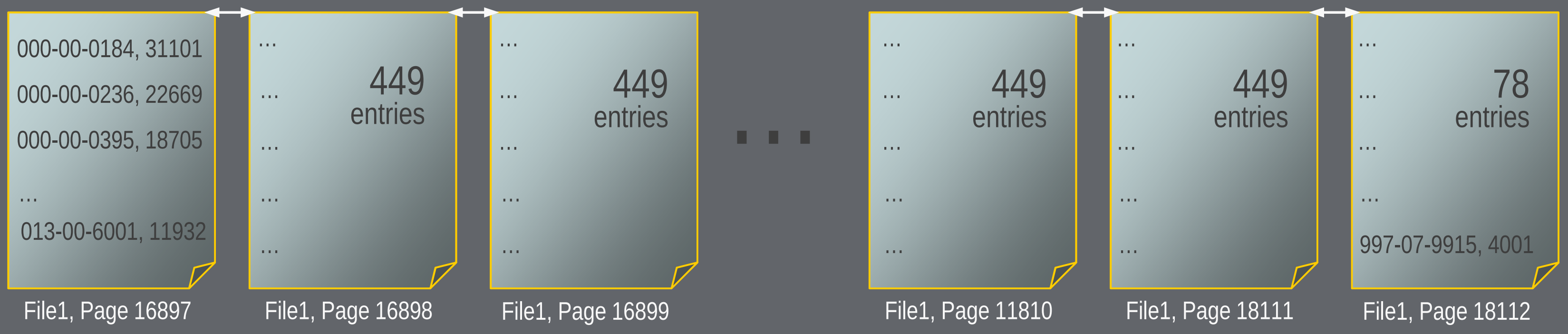
Nonclustered Index

Unique Constraint on SSN

The leaf level of the nonclustered index is built first…

- SQL Server will duplicate the SSN and EmployeeID for EVERY ROW and order it by the index definition (ascending by default).
- Every INSERT/DELETE will need to touch each nonclustered index; SQL Server will keep them up-to-date and current.

80,000 – SSN, EmployeeID Pairs = 179 Pages



000-00-0184, 31101
000-00-0236, 22669
000-00-0395, 18705
…
013-00-6001, 11932

File1, Page 16897

449 entries

File1, Page 16898

449 entries

File1, Page 16899

. . .

449 entries

File1, Page 11810

449 entries

File1, Page 18111

78 entries
…
997-07-9915, 4001

File1, Page 18112

SQL Server
CONNECTIONS

Nonclustered Index

Unique Constraint on SSN

- Non-leaf level entry for nonclustered index
  = NC index column(s) + Pointer
  + Row overhead*

  *Row overhead (min of 1 byte in the row)

  + Lookup ID** (when nonclustered is non-unique)

  ** The lookup ID (the heap's RID or the CL key) is only added to the nonleaf levels when the nonclustered is non-unique

  = SSN of 11 bytes + pointer of 6 bytes + 1 byte (row overhead)

  = 18 bytes/entry + 2 bytes in the slot array
- Entries per leaf level page

$$\frac{\text{8096 bytes/page}}{\text{18 bytes/entry + 2 bytes in slot array}} = \text{404 index entries per non-leaf level page}$$

- Pages for non-leaf le

$$\frac{\text{179 rows}}{\text{404 rows/page}} = \text{1 page} = \text{Root}$$

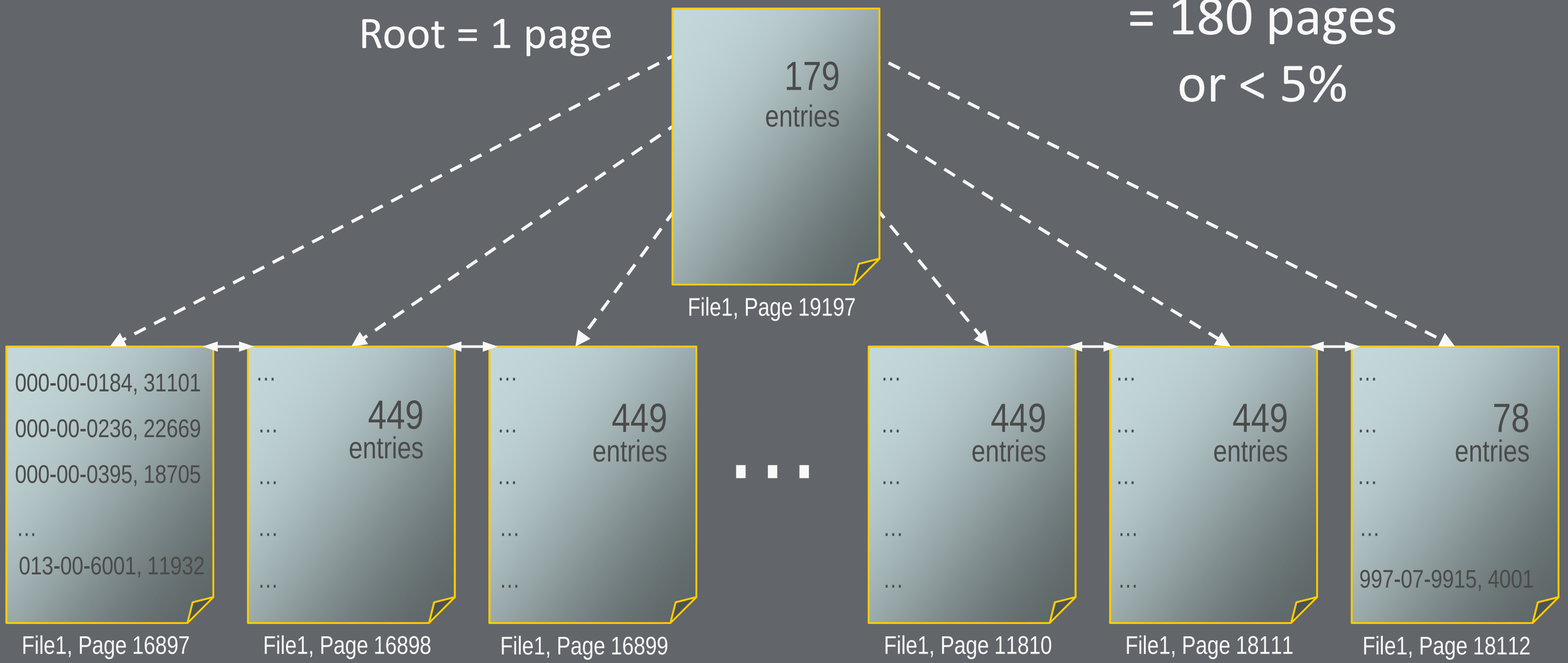
Nonclustered Index

Unique Constraint on SSN

Total overhead in terms of disk space

= 180 pages
or < 5%

Root = 1 page



179 entries

File1, Page 19197

000-00-0184, 31101
000-00-0236, 22669
000-00-0395, 18705
…
013-00-6001, 11932

File1, Page 16897

449 entries

File1, Page 16898

449 entries

File1, Page 16899

…

449 entries

File1, Page 11810

449 entries

File1, Page 18111

78 entries

997-07-9915, 4001

File1, Page 18112

Leaf level
179 pages

SQL Server
CONNECTIONS

Clustering Key Columns WHERE?

In nonclustered indexes?

- What if:
  - CREATE UNIQUE CLUSTERED INDEX IXCL
    ON *tname* (c6, c8, c2)
  - CREATE NONCLUSTERED INDEX IXNC1
    ON *tname* (c5, c2, c4)
    - KEY/btree: c5, c2, c4, c6, c8
    - Leaf: same
  - CREATE UNIQUE NONCLUSTERED INDEX IXNC1
    ON *tname* (c5, c2, c4)
    - KEY/btree: c5, c2, c4
    - Leaf: c5, c2, c4, c6, c8
- Key points:
  - Clustering key columns are added only ONCE to your nonclustered indexes
  - Where they are added (leaf only or all the way up the tree) is based on whether or not the nonclustered is nonunique. When nonunique, the CL key goes up the tree

Index Internals

What do we know?

- Clustered index leaf level IS the data
- Nonclustered index leaf level is duplicate data, in a separate structure and automatically maintained as changes occur
- B-trees are built on top of the leaf level up to a root of one page
- Nonclustered index is based on the clustered Index when the table is clustered…

Why do we need to know?

SQL Server CONNECTIONS

Index Internals

What should we do?

- OLTP tables or mixed workload tables
  - Consider a clustered index with an ever-increasing identity column
    - Creates a hot spot of activity – ensuring minimal cache requirements
    - Inserts won’t cause splits
    - The clustering key is already unique
- DSS/analysis tables
  - Will want more nonclustered indexes so you still need to be aware of the clustering key size…
- Characteristics of most/general importance:
  - Narrow, unique and static
  - Ever-increasing (reduced insertion points)

Clustering Key Suggestions

- Identity column
- Order Date, identity
  - Not date alone as that would need to be “uniquified”
- GUID
  - NO: if populated by client-side call to .NET client to generate the GUID. OK as the primary key but not as the clustering key
  - NO: if populated by server-side NEWID() function. OK as the primary key but not as the clustering key
  - ❖ Maybe: if populated by the server-side NEWSEQUENTIALID() function as it creates a more sequential pattern (and therefore less fragmentation)
- Key points: unique, static, *as narrow as possible*, and *less* prone to require maintenance – by design!

SQL Server
CONNECTIONS

Review

- Index Concepts
- Table Structure
- Index Internals
  - Heaps
  - Why cluster
  - Table usage
  - Employee table case study

If you want to take your indexing knowledge even further – consider attending the post-conference workshop on Indexing Strategies THIS Friday!

SQL Server
CONNECTIONS

Thank you!

And, please be sure to fill out your evaluation!

Index Internals
What You Really Need to Know!
SDV306

Questions?

SQL skills.com

SQL Server CONNECTIONS