PASS 2011 SUMMIT

The Ultimate Session on Indexing

AD-302-HD

Victor Isakov
Database Architect / Trainer
SQL Server Solutions
www.sqlserversolutions.com.au
www.victorisakov.com

Abstract

You can have the best hardware possible: SSDs, TBs of RAM and 100s of processors. (Don't we all wish we had such budgets.) But if you don't have the correct indexing strategy in place your database can still perform poorly. ~~With SQL Server 11 there will be even more options available.~~ Confused about what indexing strategy to use? Then this is the session for you!

In this session Victor Isakov *(MCA, MCM, MCT, MVP)* will cover all of the indexing options that are available in SQL Server and when best to use them. Victor will cover "practical" internals that will help you understand how indexes work and why you need to maintain them. Finally he will cover what indexing strategies to use given the most important considerations: your users, their query patterns and your data.

This should prove to be a most informative and practical session that will enable you to optimise your database performance.

Speaker

Victor Isakov is a Database Architect and trainer who provides consulting and training services to various organizations in the public, private, and NGO sectors globally.

He regularly speaks at international conferences such as Microsoft Tech·Ed, SQL Connections, PASS Summit, and SQL Code Camp.

He has authored a number of books on SQL Server and worked with Microsoft to develop the SQL Server exams and certification.

In 2007, Victor was invited by Microsoft to attend the "SQL Ranger" program in Redmond, WA.

Consequently, he was one of the first IT professionals to achieve both the *Microsoft Certified Master: SQL Server* and *Microsoft Certified Architect: SQL Server* certifications globally.

MVP Microsoft® Most Valuable Professional

Microsoft® CERTIFIED Architect

Microsoft® CERTIFIED Master

Microsoft® CERTIFIED Trainer

Microsoft Certified Master

Microsoft CERTIFIED Master

Caveats

- There is always an exception
- “Things always work on the whiteboard”
  - Victor Isakov quoting himself
- Test in a DEV / UAT environment before implementing in PROD
  - Ensure your indexes are being used as intended
- The answer to every question will be “It depends”
  - But feel free to ask questions anyway! ☺

Agenda

- Heaps
- Clustered
- Nonclustered Indexes
  - Composite/Compound Indexes
    - Covering Indexes
  - Included Columns
- Initial Guidelines
- Indexed Views

Agenda

- Partitioned Indexes
- Filtered Indexes
    - Filtered Statistics
- Basic Indexing Strategies
- Fragmentation
- Maintaining Indexes
- Advanced Indexing Strategies

Why Are We Here?

- Primary concern(s)
  - Improve performance
  - Enforce primary keys (entity integrity)
- Secondary concern(s)
  - Enforce uniqueness
    - Indexes
    - Unique constraints

Another question:

- What is the most important consideration when designing your indexing strategy?
  - Queries users are running
    - Frequency of those queries
  - Batch operations
    - Frequency of those batch operations
  - Data
- Don’t forget that you can disable indexes are rebuild them as required!

Heaps

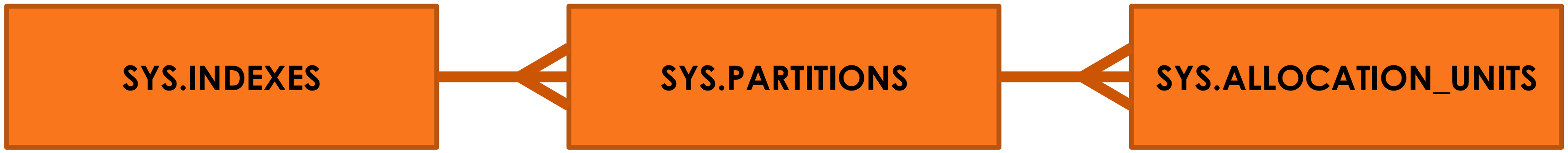
- Heaps
- Heap Structure
- Navigating a Heap
- Heap Performance

Heap

- A heap is a table without a clustered index
- Represents a “random collection of pages that make up a table”
- Free space is reused
- Index Allocation Map (IAM) page used to track allocation of pages / extents

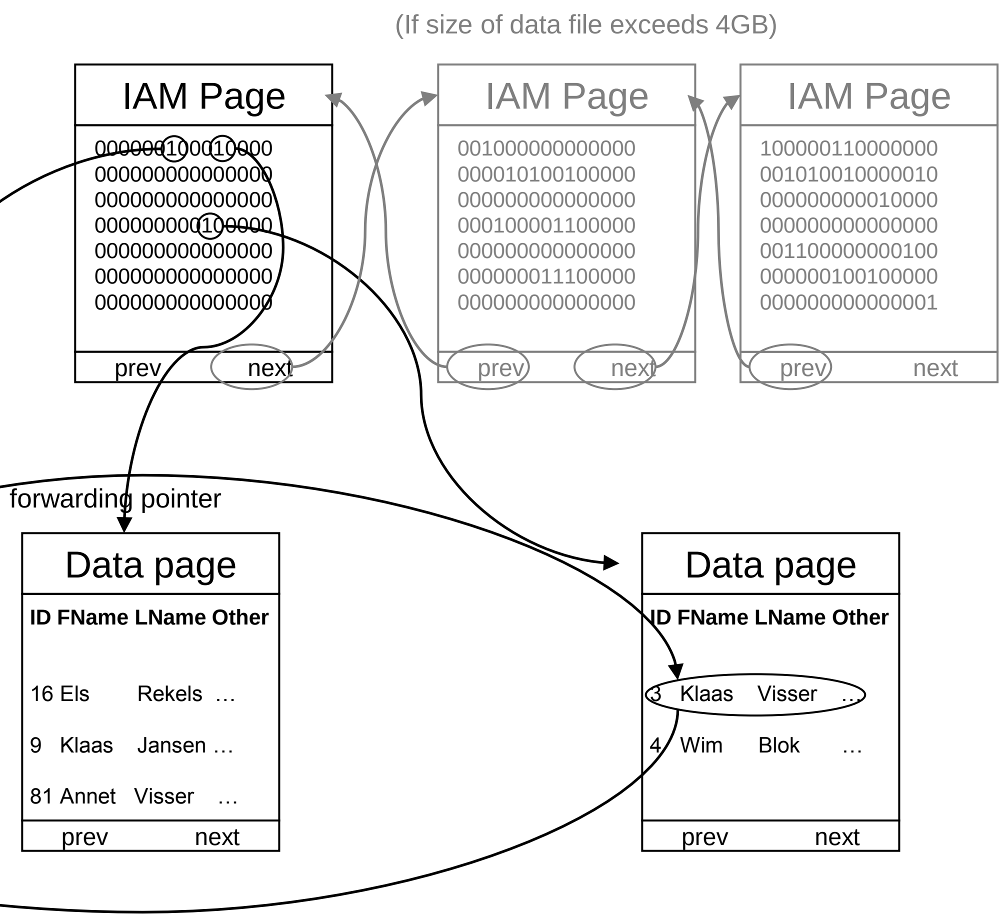
Heap

- Heaps / indexes have an entry in sys.indexs
- An “index” can be partitioned
- Each partition can store 3 different “types of pages”
  - IN_ROW_DATA
  - ROW_OVERFLOW_DATA
  - LOB_DATA



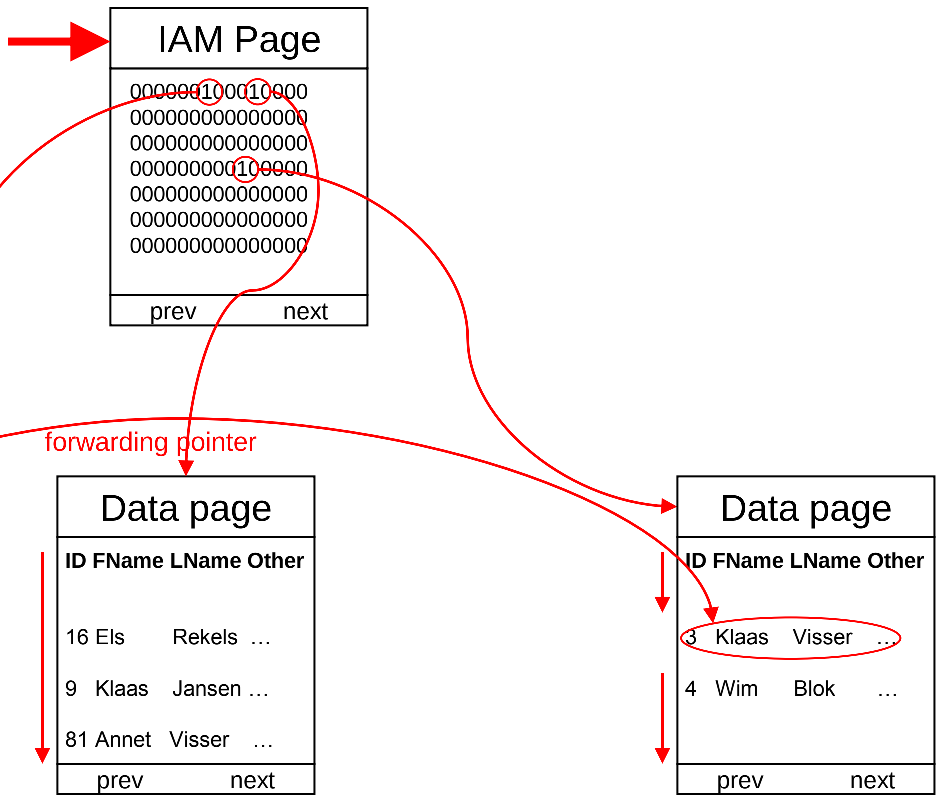
Heap Structure

- IAM Page
  - Forwarding pointers



Navigating a Heap

- Table scan



DEMO 1

Heap Performance

- Only way that SQL Server can work with a heap is to perform a table scan
- Clustered indexed bring “order” to the table
  - Give SQL Server the option of seeking the B-Tree to get the data
  - Allow SQL Server to stop when it knows that no more records can satisfy the query

Clustered Indexes

- Indexes
- SQL Server Limits
- Clustered Index
- Navigating a Clustered Index

Indexes

- Improve performance by optimizing data access path
  - Index seeks
  - Allow SQL Server to bail out earlier
- Implemented as B-Trees in SQL Server
- Two types
  - Clustered
  - Nonclustered
- Can be unique

SQL Server Limits (2008R2 BOL)

- 999 non clustered indexes per table
- **16 columns per index key**
- **900 bytes per index key**
- 30,000 statistics per non-index columns
- 1,024 columns per non-wide table
- 30,000 columns per wide table
- 1,000 partitions per partitioned table / index
- 4,096 columns per SELECT statement

Clustered Index

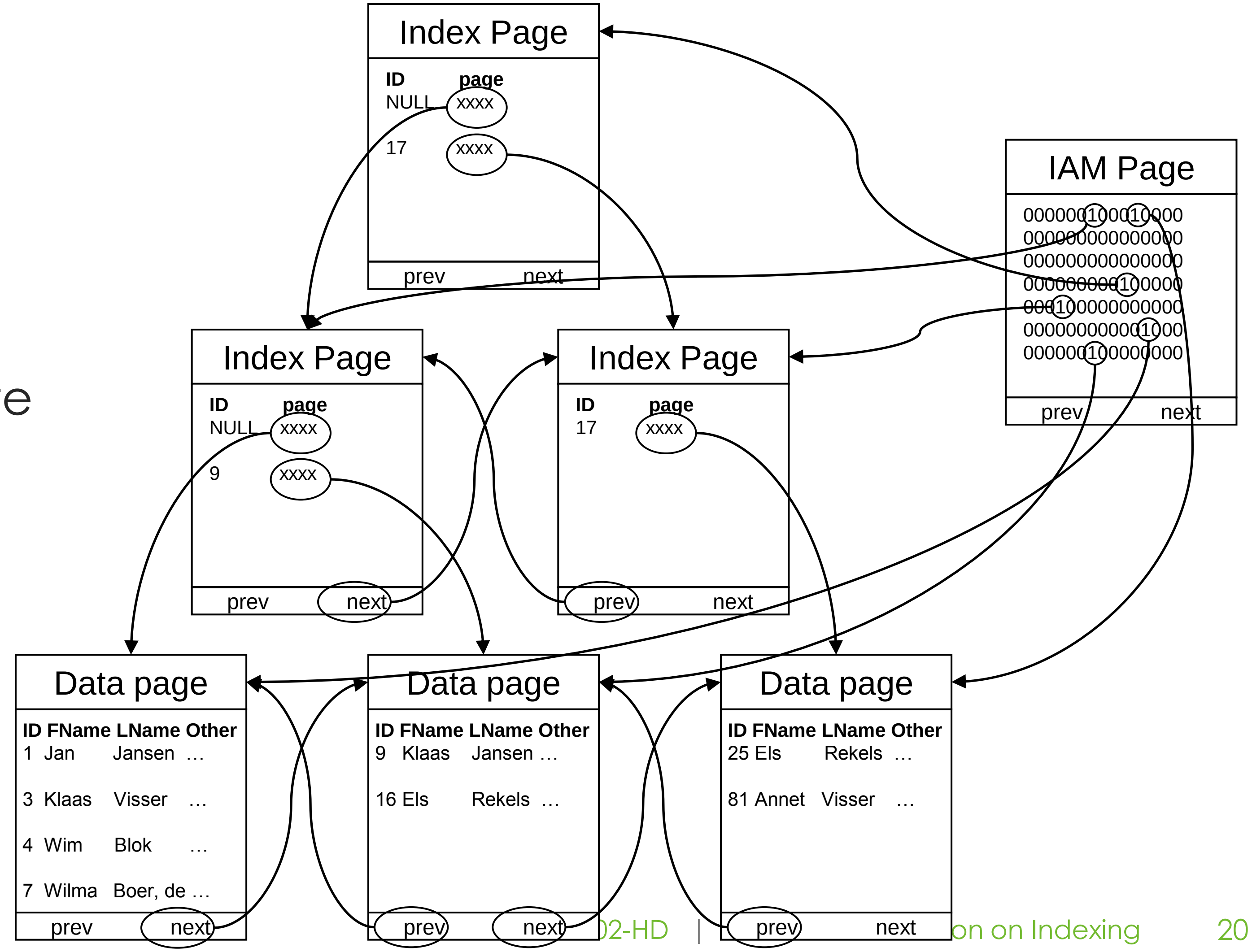
- Dictionary analogy
- Physical order == sorted order
- 1 per table
- If not created as unique, SQL Server adds 4 byte INTEGER value as required to make non-unique key values unique
  - Uniquifier

Clustered Index Structure

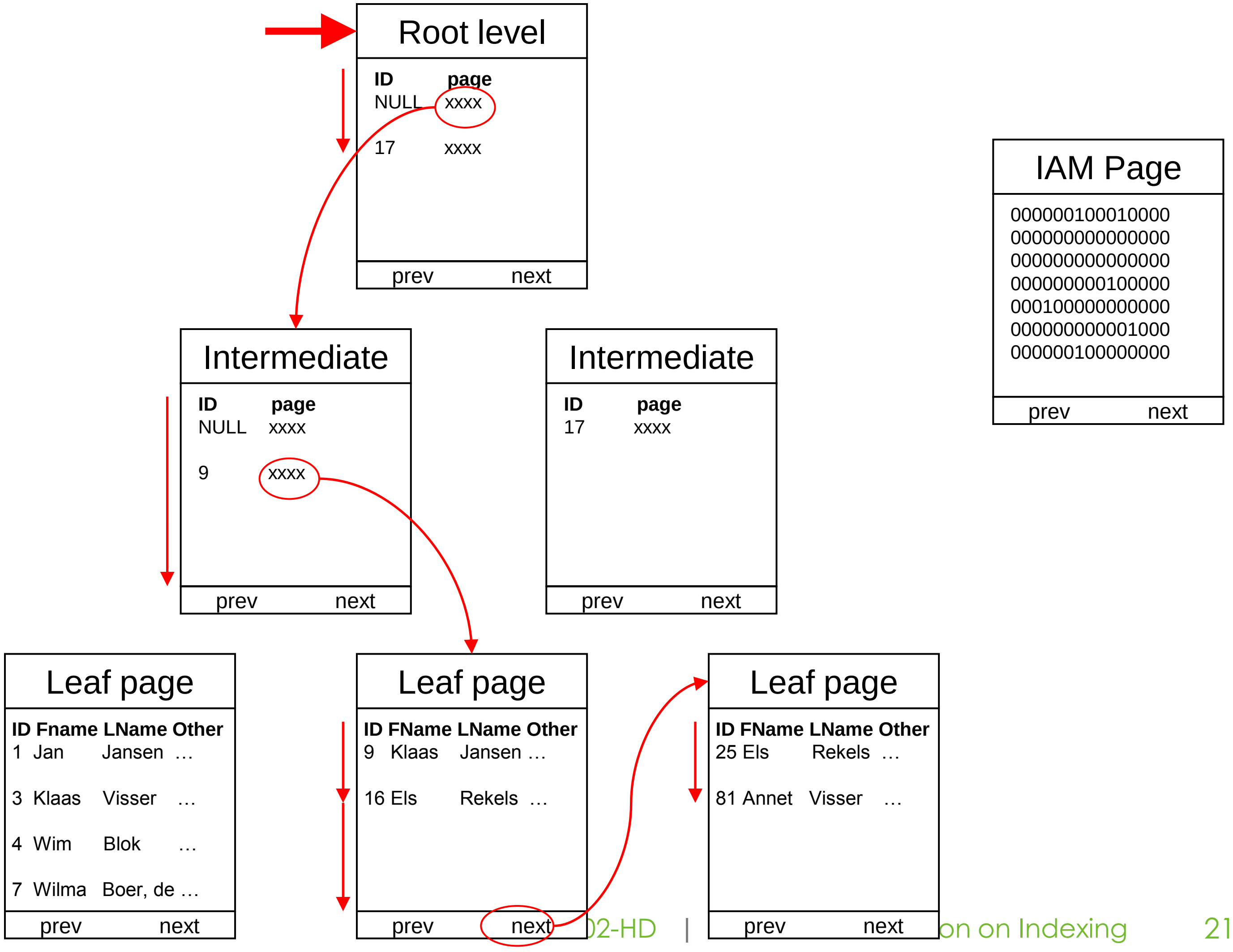
Root

Intermediate

Leaf

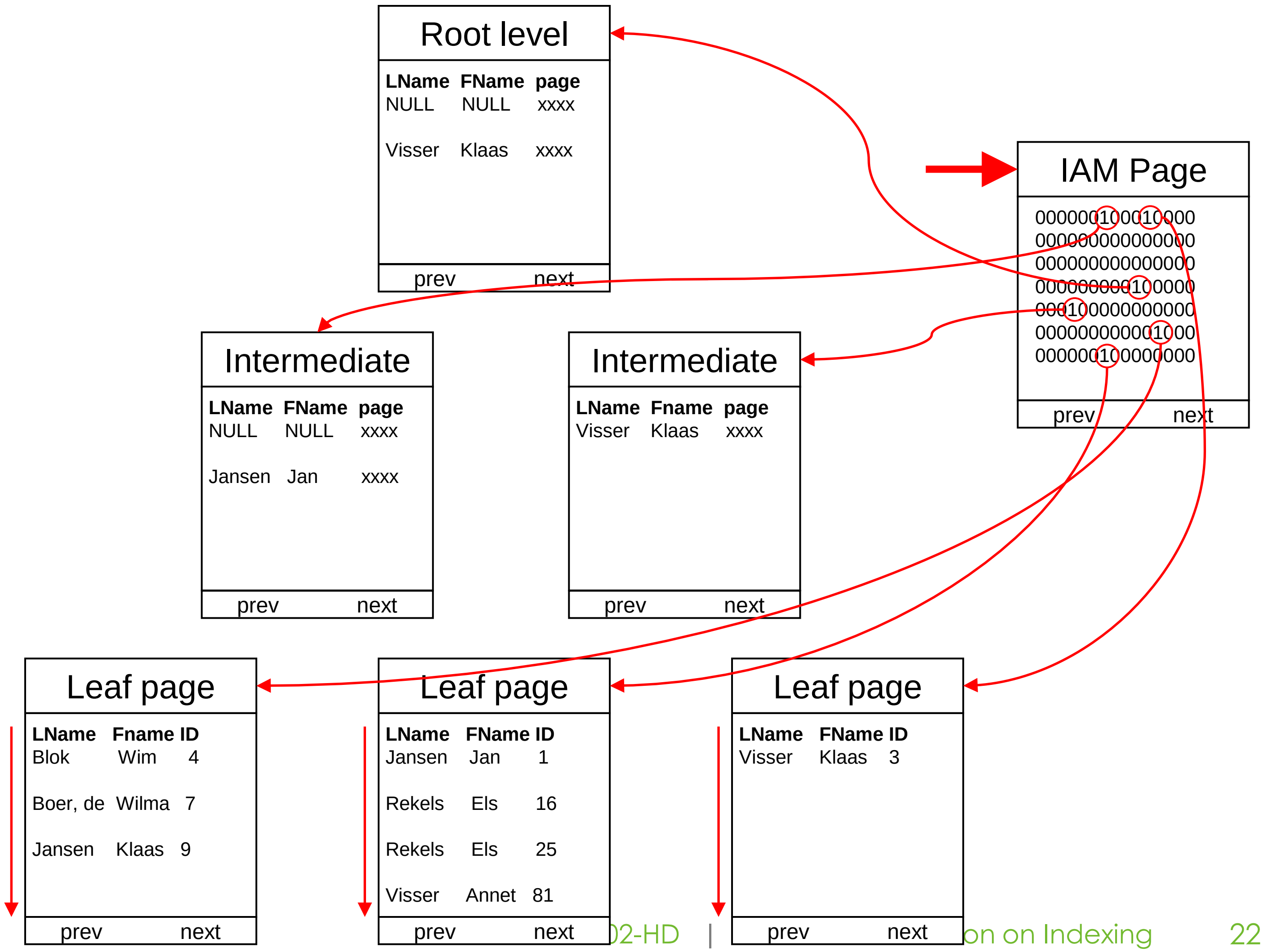


Index Seek



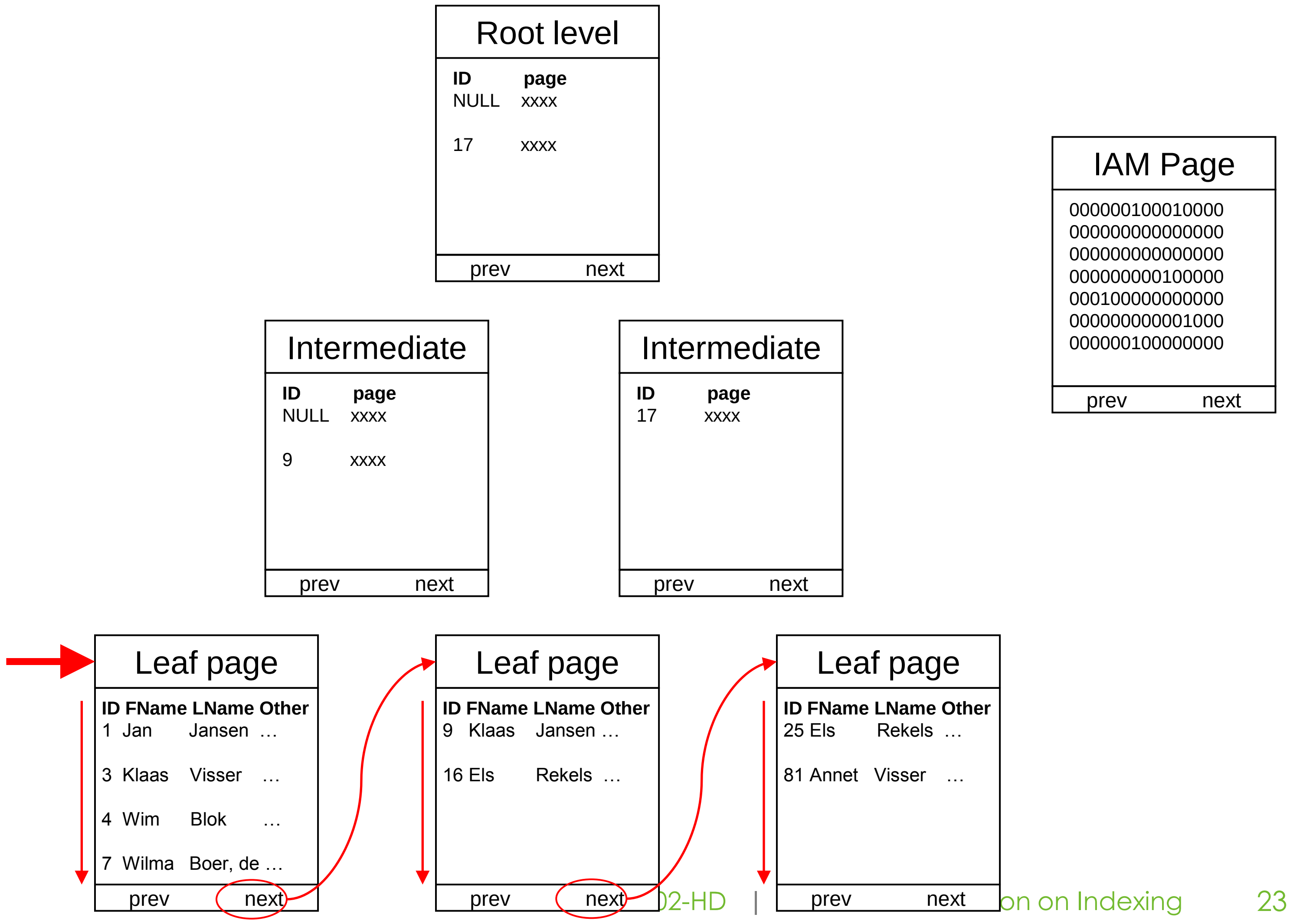
Root level
ID page
NULL xxxx
17 xxxx
prev next
Intermediate
ID page
NULL xxxx
9 xxxx
prev next
Intermediate
ID page
17 xxxx
prev next
IAM Page
000000100010000
000000000000000
000000000000000
000000000100000
000100000000000
000000000001000
000000100000000
prev next
Leaf page
ID Fname LName Other
1 Jan Jansen …
3 Klaas Visser …
4 Wim Blok …
7 Wilma Boer, de …
prev next
Leaf page
ID FName LName Other
9 Klaas Jansen …
16 Els Rekels …
prev next
Leaf page
ID FName LName Other
25 Els Rekels …
81 Annet Visser …
prev next

Index Scan (Unordered)



DEMO
2

Index Scan (Ordered)



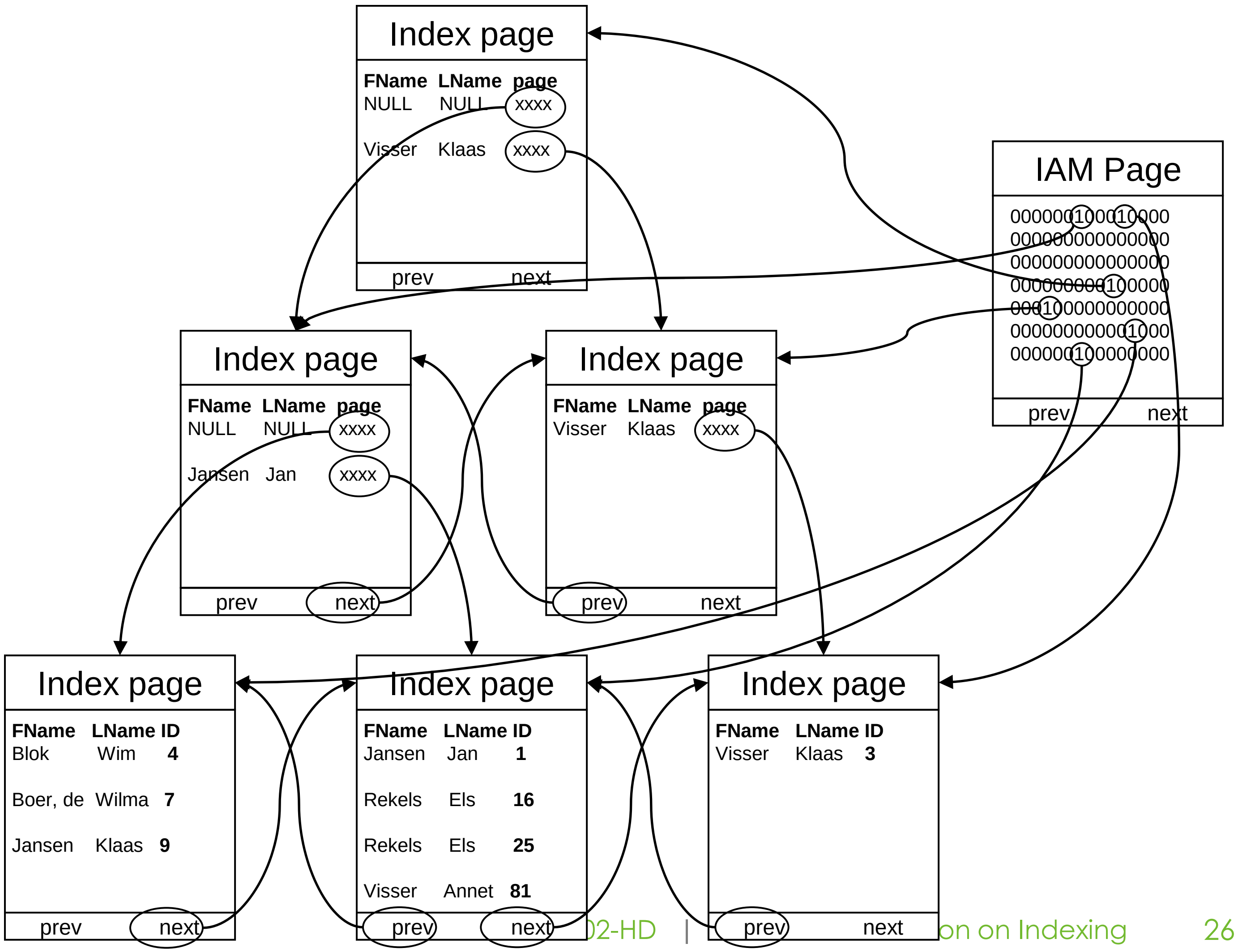
Nonclustered Indexes

- Nonclustered Index
- Nonclustered Index Structure
- Navigating a Nonclustered Index
- Composite / Compound Indexes
- Included columns
- Covering Indexs

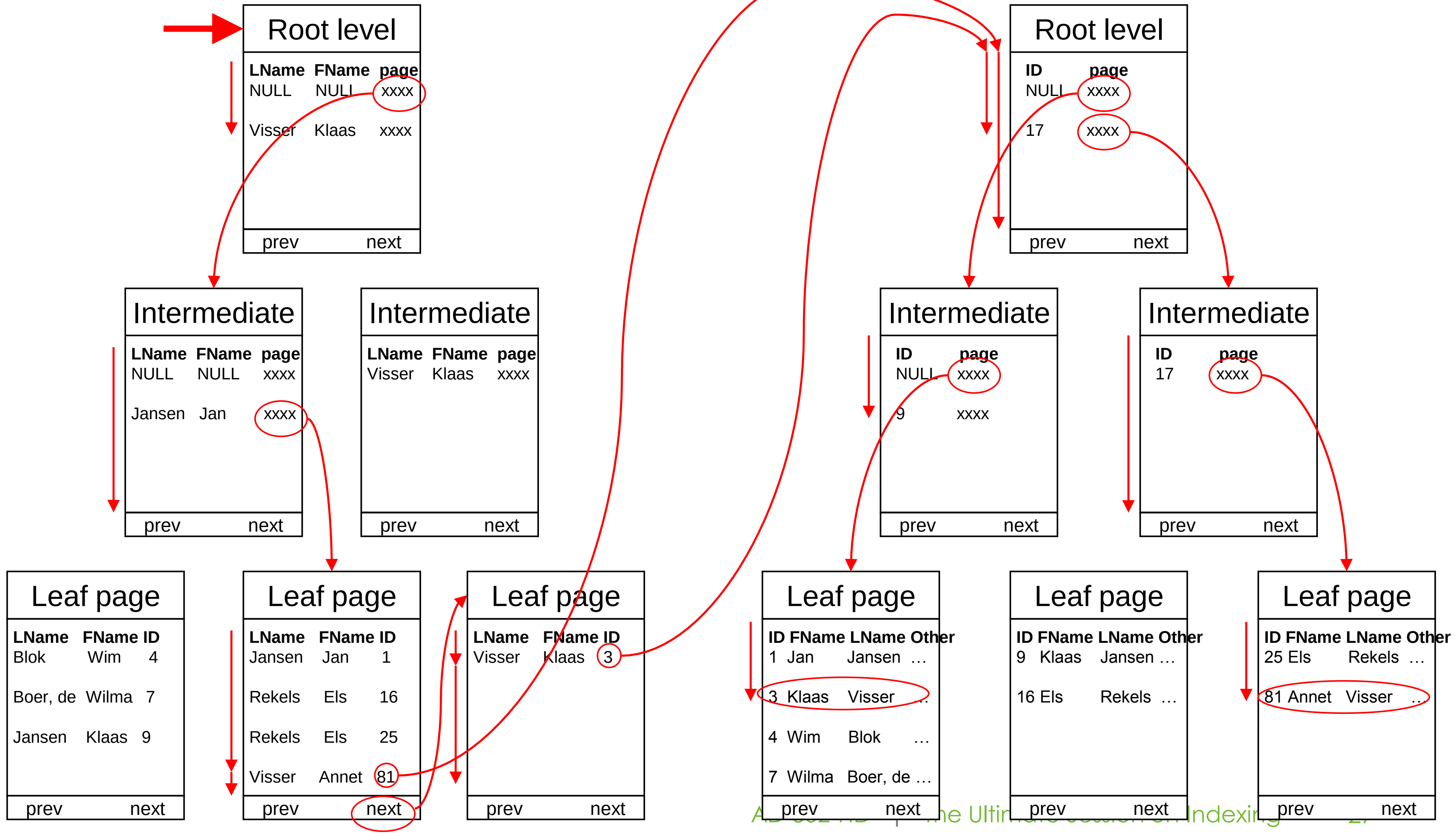
Nonclustered Index

- Index analogy
- 999 per table
- Separate B-tree structure
  - On heap (physical key)
    - RID (File ID, Page ID, Slot #)
  - On clustered index (logical key)

Nonclustered Index Structure

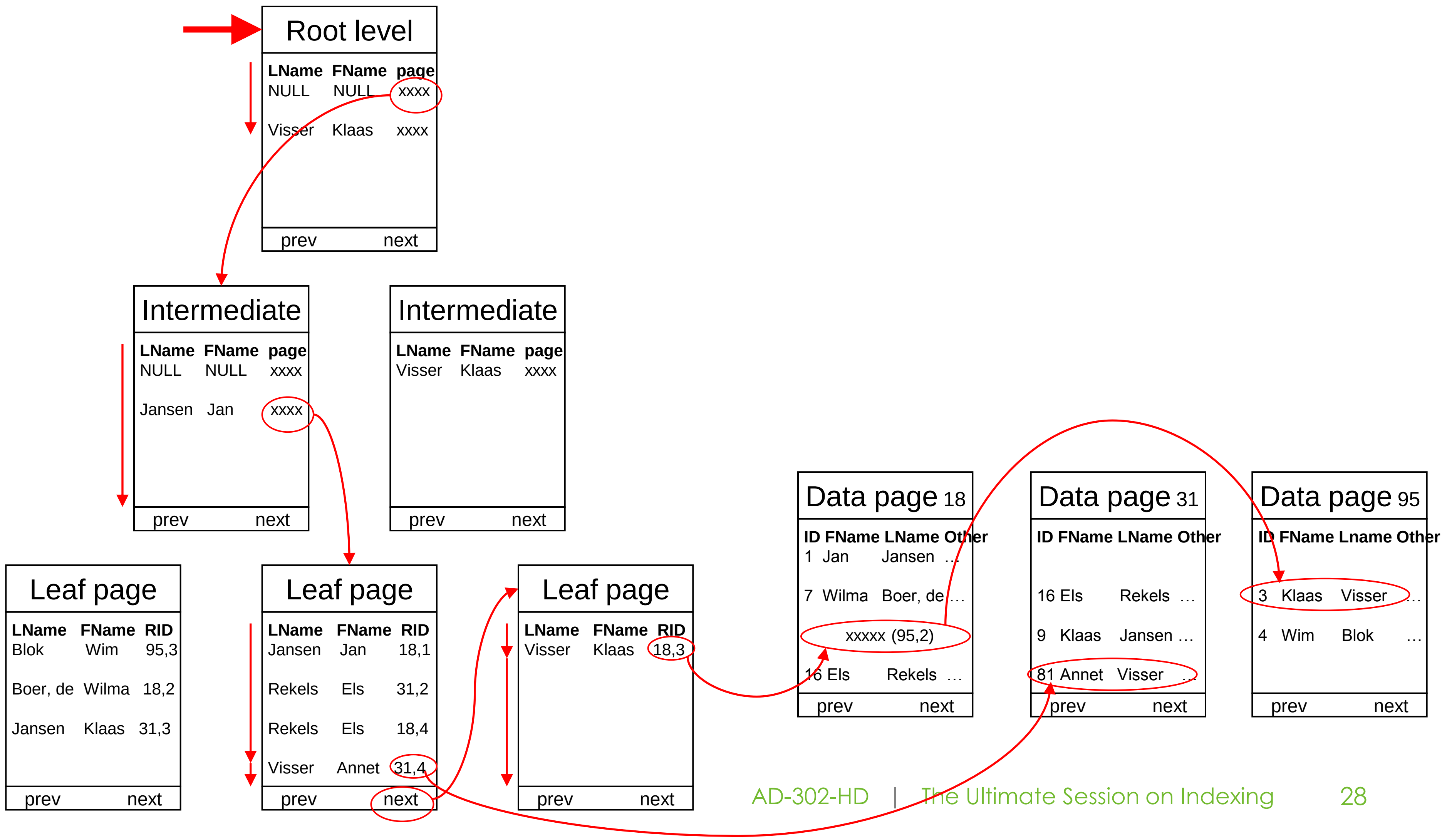


Nonclustered Index Seek (Key)



DEMO
3

Nonclustered Index Seek (RID)



Initial Guidelines

- Have a good argument not to implement a clustered index
- Ideal clustered index key
  - Narrow
  - Unique
  - Static
- If it's unique – make it unique
- Foreign key constraints do not create underlying indexes
- Try not to create multiple indexes on the same column(s) ☺

Clustered Index Considerations

- Where do clustered indexes work well?
  - Point queries
  - Range queries
  - Partial scans
  - Large number of duplicates in result set
- Where the data is clustered together!
  - Hmm… Like the foreign key

Nonclustered Index Considerations

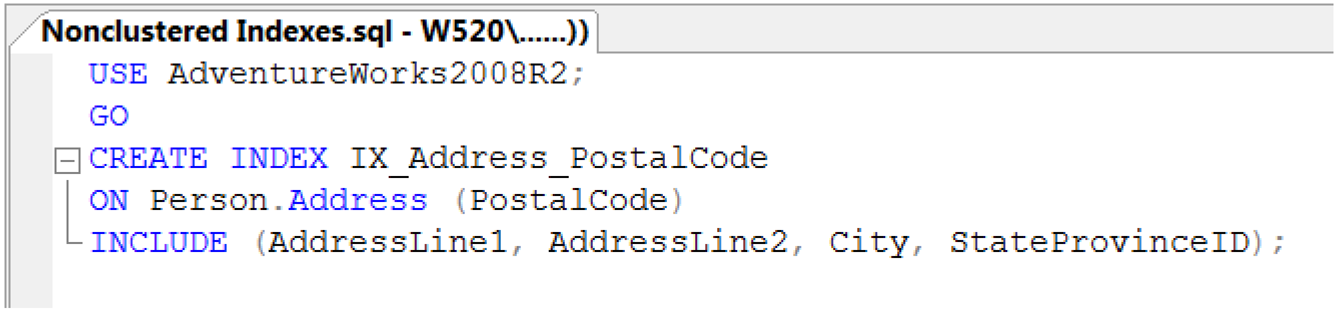
- Where do nonclustered indexes work well?
  - Point queries
- Have limited value where a large number of duplicate records / records are returned

Composite / Compound Indexes

- Multiple columns in the index’s definition
- Remember limits
  - 16 columns
  - 900 bytes
- Column order matters
  - Ideally the most selective column is first
  - But it really depends on the queries
- SQL Server only creates statistics on the high-order column

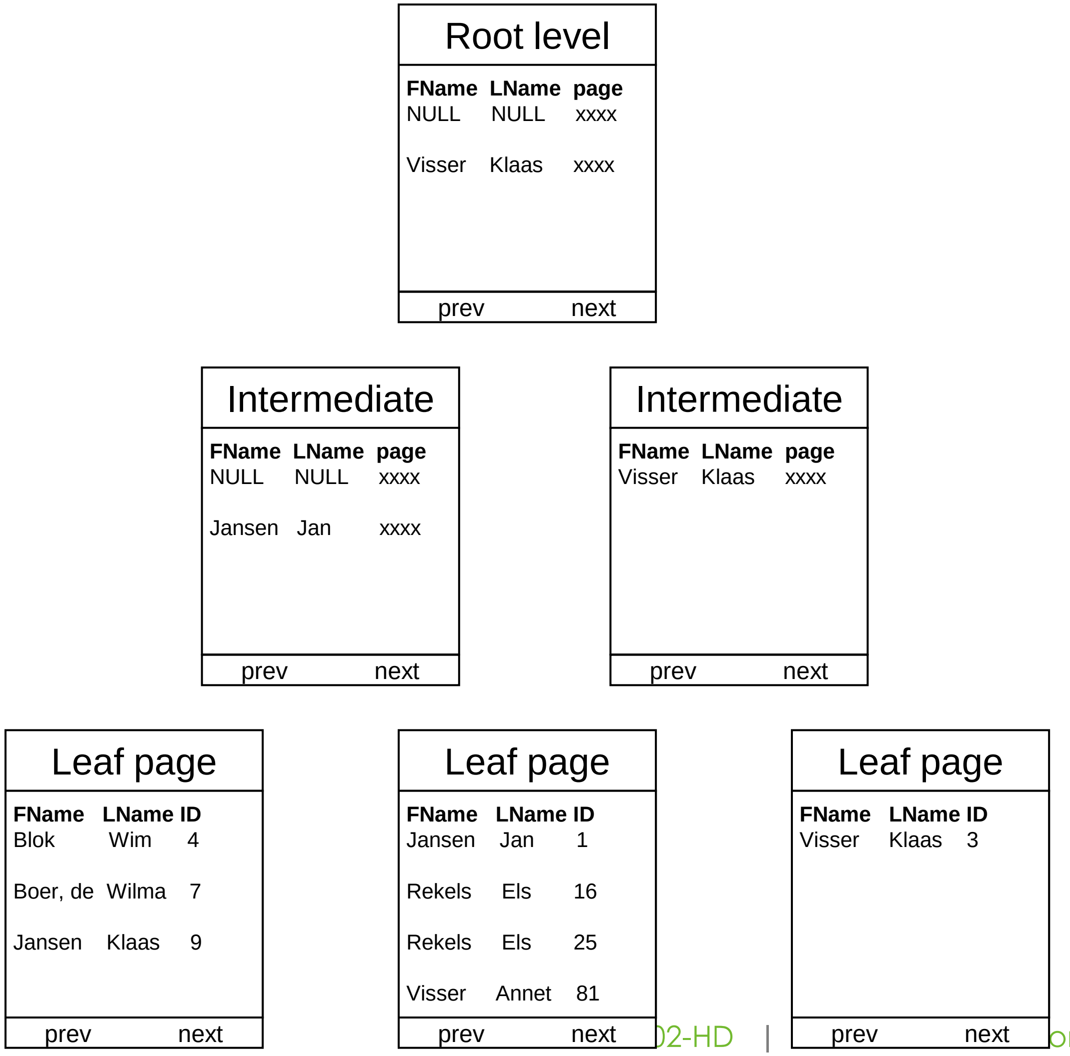
Included Columns

- Introduced in SQL Server 2005
- Columns added to leaf level only
- All data types allowed except TEXT, NTEXT and IMAGE
- Overcome 900 byte / 16 column limit…

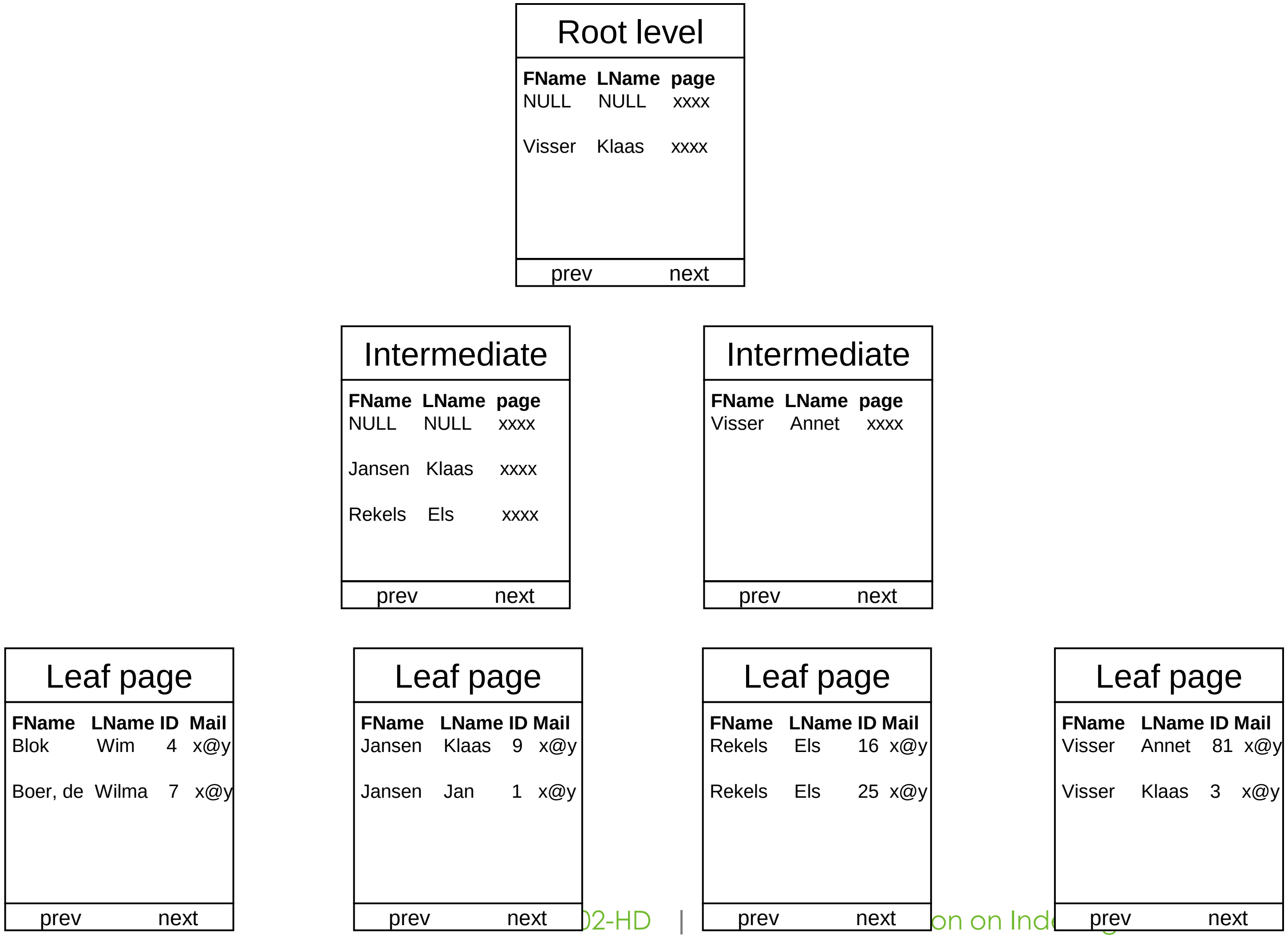

Nonclustered Indexes.sql - W520\......))

```
USE AdventureWorks2008R2;
GO
CREATE INDEX IX_Address_PostalCode
ON Person.Address (PostalCode)
INCLUDE (AddressLine1, AddressLine2, City, StateProvinceID);
```

Included Columns Impact



Included Index Impact



Included Columns Considerations

- Don’t make the nonclustered index too wide
  - SQL Server might not use it
- Exception
  - Covering index
  - Scanning the leaf level of the nonclustered index < scanning the table

DEMO
4

Covering Indexes

- The nonclustered index covers the data being requested by the query
  - No lookup required
    - No need to read the data pages
    - No locking on the data pages
- Don’t forget that SQL Server can either:
  - Seek through the root page
  - Scan through the first page
- The SARG does not have to query the highest order column

Indexed Views

- What is a View?
- Indexed Views
- Indexed View Requirements
- NOEXPAND Hint

What is a View?

```
CREATE VIEW dbo.MyView
AS SELECT FirstName, LastName
   FROM   Person.Person
   WHERE  LastName LIKE 'Isa%';
```

EXPAND

```
SELECT MAX(FirstName)
FROM   (SELECT FirstName, LastName
        FROM   Person.Person
        WHERE  LastName LIKE 'Isa%') AS v;
```

Indexed Views

- Introduced in SQL Server 2000
  - Commonly referred to as materialized / persisted views
- Ability to create indexes on a view
- Must create a unique, clustered index initially
- Benefits
  - Joins and aggregations that process many rows
  - Join and aggregation operations that are frequently performed by many queries
  - Decision support workloads

Indexed View Requirements

- "Fussy":
  - The ANSI_NULLS and QUOTED_IDENTIFIER options must have been set to ON when the CREATE VIEW statement was executed.
  - The ANSI_NULLS option must have been set to ON for the execution of all CREATE TABLE statements that create tables referenced by the view
  - The view must not reference any other views, only base tables
  - All base tables referenced by the view must be in the same database as the view and have the same owner as the view
  - The view must be created with the SCHEMABINDING option. Schema binding binds the view to the schema of the underlying base tables
  - User-defined functions referenced in the view must have been created with the SCHEMABINDING option
  - Tables and user-defined functions must be referenced by two-part names in the view. One-part, three-part, and four-part names are not allowed
  - All functions referenced by expressions in the view must be deterministic
- More in BOL!

NOEXPAND Query Hint

	Enterprise Edition	Other Editions
NOEXPAND hint used	Indexed view always used	Indexed view always used
NOEXPAND hint not used	Indexed view may be used	Indexed view never used
Indexed view not queried	Indexed view may be used	Indexed view never used

- Document, as might need to refactor!

DEMO
5

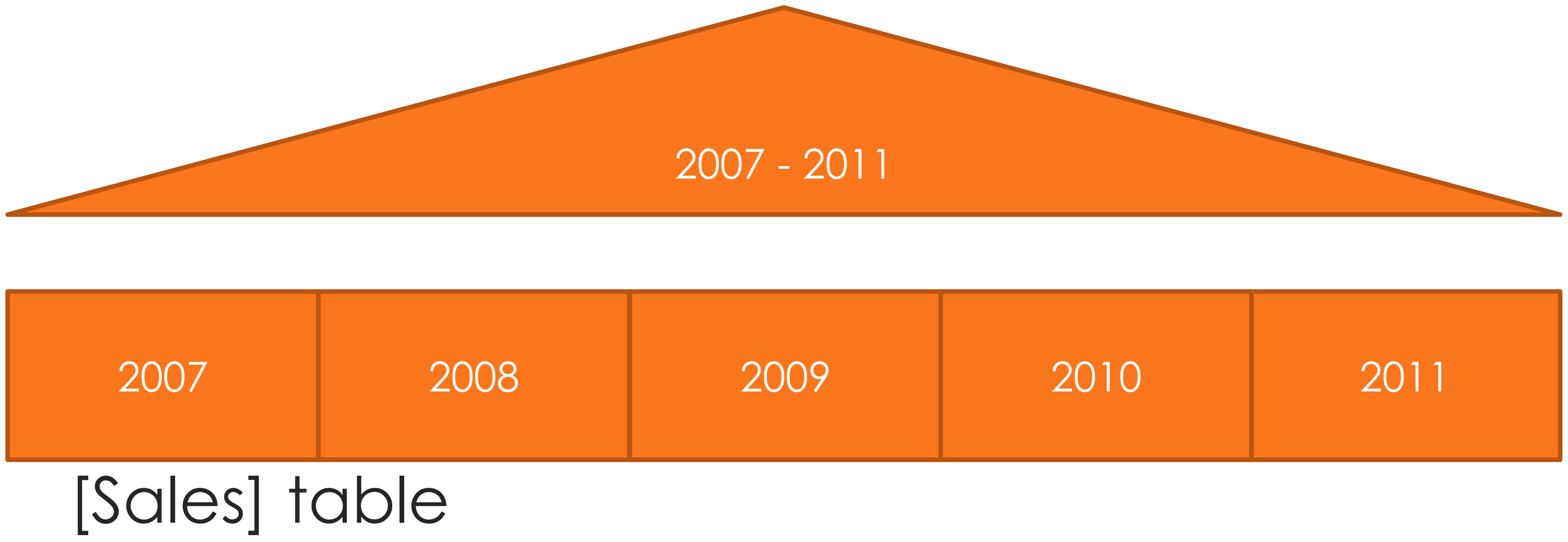
Indexed View Considerations

- Ultimately indexed views represent another form of denormalization
  - Now we have filtered indexes
  - Computed, persisted columns
- "Wider" execution plans are generated
- Not used as often now as we have other options in SQL Server

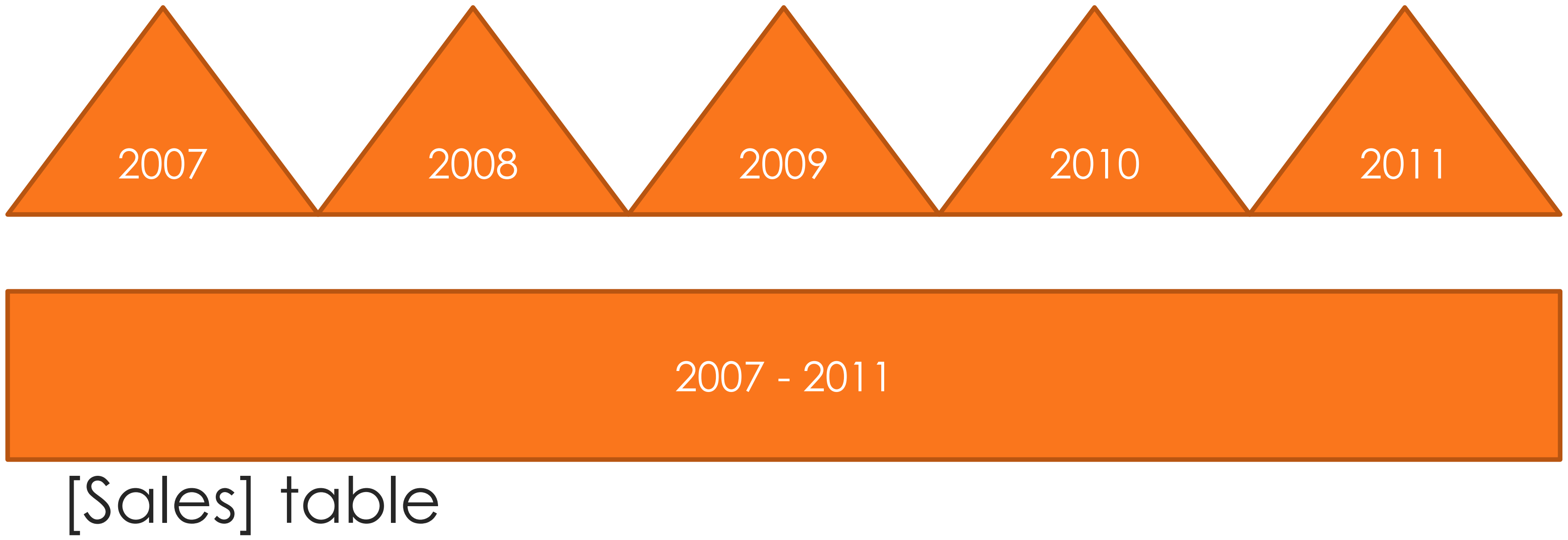
Partitioned Indexes

- Introduced in SQL Server 2005 Enterprise Edition
- Two Types
  - Aligned
  - Unaligned

Unaligned Partitioned Indexes

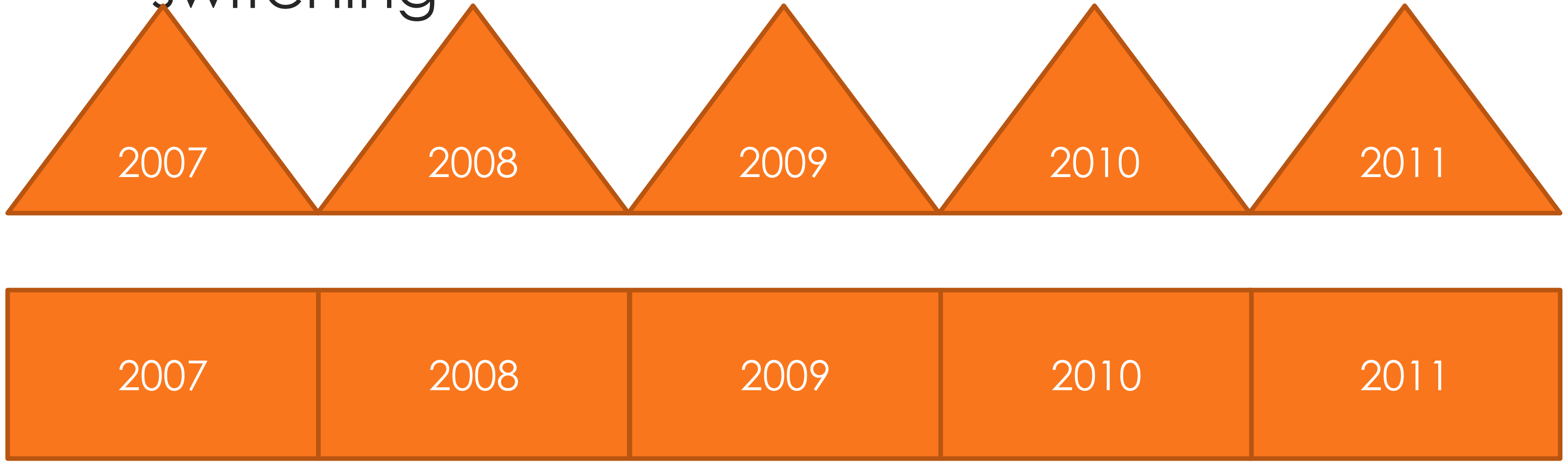


Unaligned Partitioned Indexes



Aligned Partitioned Indexes

- Need to be aligned for efficient partition switching



[Sales] table

DEMO

Partitioned Indexes Benefits

- As with tables different partitions of the index can be configured differently
  - Row compression
  - Page compression
  - Lock escalation
- Can rebuild index at partition level!
  - No need to rebuild partitions for older “static” partitions

Filtered Indexes

- Filtered Indexes
- Filtered Index Limitations

Filtered Indexes

- Introduced in SQL Server 2008
- A filtered index is an optimized nonclustered index, especially suited to cover queries that select from a well-defined subset of data
- Advantages
  - Improved query performance and plan quality
  - Reduced index maintenance costs
  - Reduced index storage costs

Filtered Index Use Cases

- Large tables with small percentage of “useful” data / data
  - 100,000s customers
    - Only 5% of customers have a pet
  - 1,000,000s sales records going back 10 years
    - 99% of table is historical
    - Historical data is hardly queries / queried differently
- Wide tables (many columns) where columns are partially related
  - 100s of columns
    - [BikeColor] is only relevant to [ProductType] = ‘Bike’

Implementing Filtered Index

- Welcome to SQL Server's fussiest feature ☺
  - Cannot create what you want
  - Can create it, but the Query Optimizer will not use it
- Feedback: various degrees of success
- My take is that sparse columns drove filtered indexes
- Important to test in DEV/UAT

Filtered Index Limitations

- Only supports very simple comparison logic in predicate
- Cannot create a filtered index on a view
- Query optimizer can benefit from a filtered index defined on a table that is referenced in a view
- **Cannot be created on computed fields**
  - **Even if they are persisted**

From BOL

- WHERE <filter_predicate>
    - <filter_predicate> ::= <conjunct> [AND <conjunct>]
    - <conjunct> ::= <disjunct> | <comparison>
    - <disjunct> ::= column_name IN (constant ,…)
    - <comparison> ::= column_name <comparison_op> constant
    - comparison_op> ::= { IS | IS NOT | | <> | ! | > | > | !> | < | < | = | != | >= | <= | !< }
- No BETWEEN, no LIKE, no subquery, no variables
- Simple and deterministic

Filtered Indexes Requirements

- Comparison work involved, so we must agree on various SET options
  - ON
    - ANSI_NULLS, ANSI_PADDING, ANSI_WARNINGS, ARITHABORT, CONCAT_NULL_YIELDS_NULL, QUOTED_IDENTIFIER
  - OFF
    - NUMERIC ROUNDABORT
- Otherwise:
  - If not set when index is created, won't create the index
  - If not set when INSERT, UPDATE, DELETE, MERGE affects the data, gives error and rolls back
  - If not set when the index might be used to optimize the query, it will not be considered

DEMO
6

Filtered Indexes *versus* Indexed Views

Allowed in expressions	Views	Filtered indexes
Computed columns	Yes	No
Joins	Yes	No
Multiple tables	Yes	No
Simple comparison logic in a predicate*	Yes	Yes
Complex logic in a predicate**	Yes	No

- Advantages over Indexed Views
  - Reduced index maintenance costs
  - Improved plan quality
  - Online index rebuilds
  - Non-unique indexes

Statistics

- Statistics
- Filtered statistics

Statistics

- The query optimizer uses statistics to create query plans that improve query performance
  - Statistical information about the distribution of values in one or more columns of a table (or indexed view)
  - The query optimizer uses these statistics to estimate the cardinality, or number of rows, in the query result
- Maximum of 200 steps

Automatic Statistics Creation

- Automatically created
- Database Options
  - AUTO_CREATE_STATISTICS
  - AUTO_UPDATE_STATISTICS
  - AUTO_UPDATE_STATISTICS_ASYNC
    - Introduced in SQL Server 2005

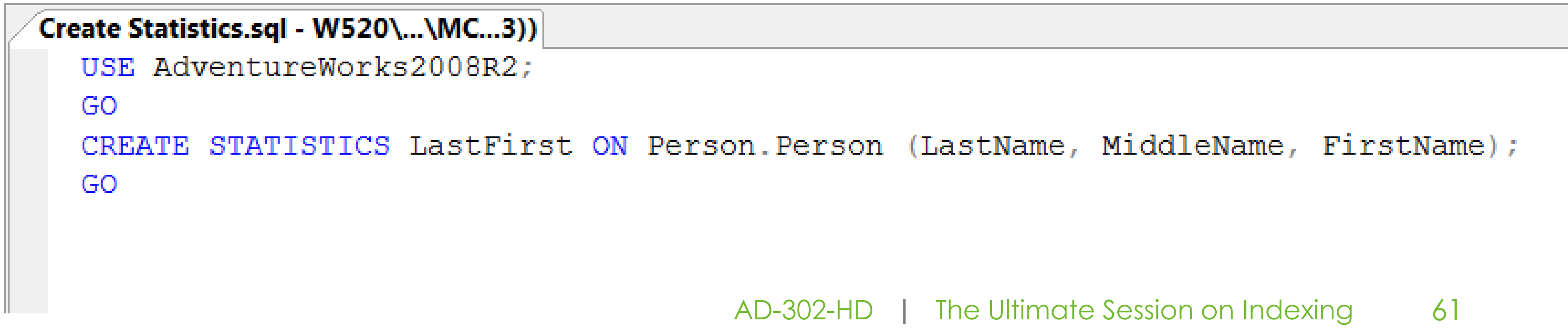
- View
  - sys.stats
  - DBCC SHOW_STATISTICS

Automatic Statistics Update

- When table row count goes from zero to not zero
- Table had less than 500 rows and there have been more than 500 changes to the leading column of the stat since the last stat update
- Table had more than 500 rows and there have been at least 500 + 20% changes to the leading column in the stat since the last update

Manual Statistics Creation

- Can create additional statistics
- These additional statistics can capture statistical correlations that the query optimizer does not account for when it creates statistics for indexes or single columns



Create Statistics.sql - W520\...\MC...3))

```
USE AdventureWorks2008R2;
GO
CREATE STATISTICS LastFirst ON Person.Person (LastName, MiddleName, FirstName);
GO
```

Filtered Statistics

- Introduced in SQL Server 2008
- Filtered statistics can improve query performance for queries that select from well-defined subsets of data
- Filtered statistics use a filter predicate to select the subset of data that is included in the statistics
- Well-designed filtered statistics can improve the query execution plan compared with full-table statistics

Filtered Statistics Use Cases

- On skewed data to assist Query Optimizer
- Potentially on very large tables
  - Create multiple statistics on same data with different filters
  - Filtered statistics will be more precise
- Partitioned tables
  - Only required on current partition
  - Required on historical partitions

Filtered Indexes Benefits

- Potentially get better execution plans for larger volumes of data
  - Better quality statistics
- Less need to update statistics
  - Older data does not change
  - Older data changes less frequently

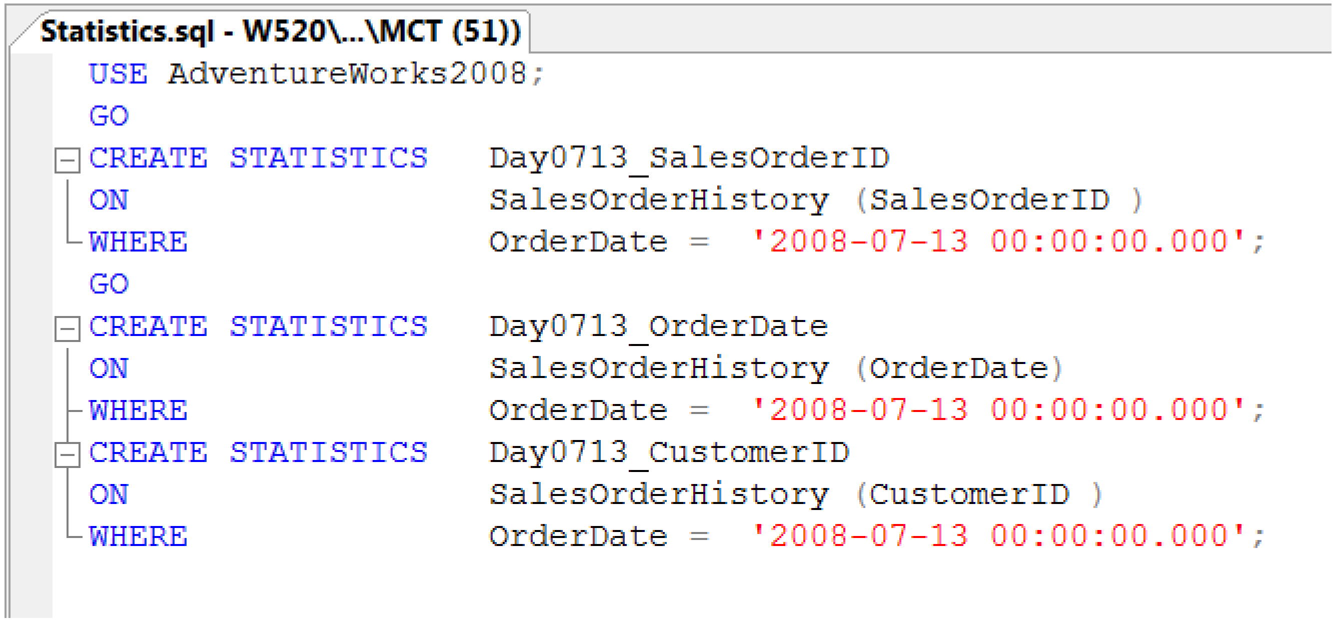
DEMO
7

Filtered Statistics

- One of the great advantages of the filtered statistics is that if we switch in new partition into the partitioned table filtered statistics built on the older partitions does not become invalid and you don't need to update it
    - You only need to create new filtered statistics for the new partition which is much faster process comparable to if you need to update or build statistics for entire table.

Filtered Statistics

```
CREATE STATISTICS …
ON …
WHERE …
```



Statistics.sql - W520\...\MCT (51))

```
USE AdventureWorks2008;
GO
CREATE STATISTICS   Day0713_SalesOrderID
ON                  SalesOrderHistory (SalesOrderID )
WHERE               OrderDate =  '2008-07-13 00:00:00.000';
GO
CREATE STATISTICS   Day0713_OrderDate
ON                  SalesOrderHistory (OrderDate)
WHERE               OrderDate =  '2008-07-13 00:00:00.000';
CREATE STATISTICS   Day0713_CustomerID
ON                  SalesOrderHistory (CustomerID )
WHERE               OrderDate =  '2008-07-13 00:00:00.000';
```

Basic Indexing Strategies

- Basic Indexing Strategies
- Missing Indexes

Basic Indexing Strategies

- FLOAT / REAL columns
  - Limited value
- BIT columns
- Encryption
- Data compression
  - Improve page density
  - Don’t forget you can implement compression at the partition level

DEMO 8

Basic Indexing Strategies

- DSS versus OLTP
  - Too many indexes might slow down DML operations unacceptably
  - Create more indexes in a DSS environment
- Multiple indexes versus composite indexes
- Index intersection
- Use filtered indexes to create unique indexes that support multiple NULL values

Fragmentation

- Fragmentation
- Analysing Fragmentation
- Page Splits
- FILLFACTOR

Fragmentation

- Over a period of time indexes will no longer be optimal and become fragmented
  - DML operations
- Page splits
- Forwarding records
- Fragmentation
  - Internal
  - External

DEMO
9

Analyzing Fragmentation

- DBCC SHOWCONTIG
  - Deprecated
- **sys.dm_db_index_physical_stats** system function

DEMO
10

Page Splits

- Clustered indexes dictate the physical order in which data is stored
- What happens if you want to insert a new record into a page that is full?
- What happens if you want to update a record and there is not enough room on the page?
- Page split

FILLFACTOR

- Purpose is to “postpone” page splits
- Only uses a certain percentage of each page during index creation and rebuilds
- PAD_INDEX Option
  - Applies fill factor to intermediate pages
- Will not always help
- Data consumes extra disk space and EXTRA RAM!
- Fine-tuning mechanism

Maintaining Indexes

- Rebuilding indexes
- Reorganizing indexes
- Maintaining statistics

Rebuilding Indexes

- Can be done offline or online
  - ONLINE option
- Uses database or tempdb
  - SORT_IN_TEMPDB option
- Temporary space required
  - Sort run
  - Mapping index
  - Version store
- Version store uses tempdb
  - sys.dm_db_task_space_usage
  - sys.dm_db_session_space_usage
  - sys.dm_db_file_space_usage

Offline Index Rebuilds

- Only option in SQL Server Standard Edition and below
  - ALTER INDEX … REBUILD
  - CREATE INDEX … DROP_EXISTING
- Alternative
  - Reorganizing indexes

REBUILD versus DROP_EXISTING

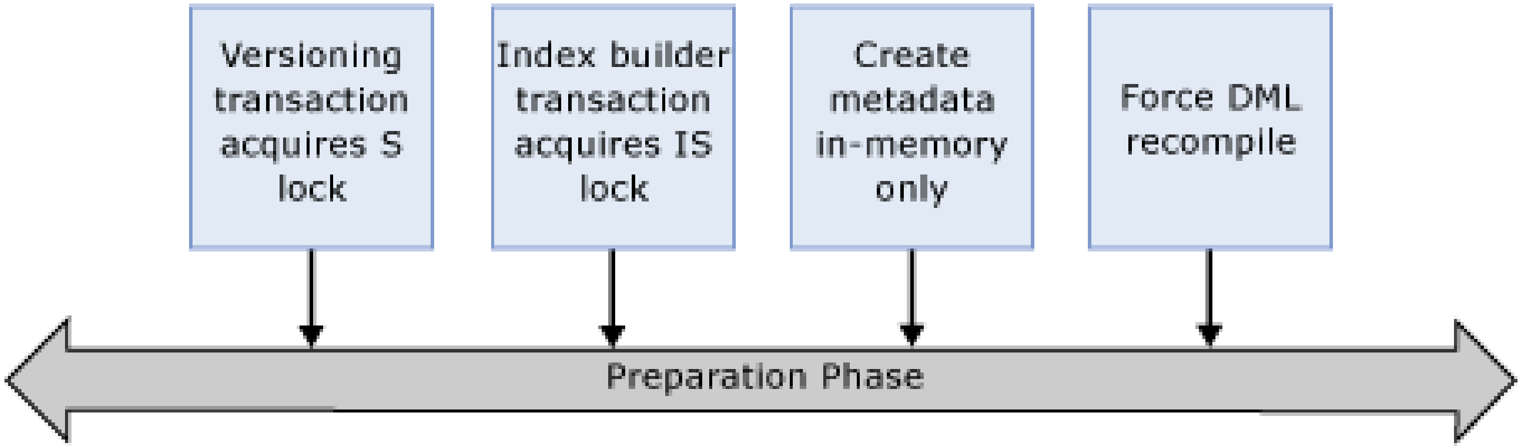
Functionality	REBUILD	DROP_ EXISTING
Index definition can be changed by adding or removing key columns, changing column order, or changing the column sort order.*	No	Yes**
Index options can be set or modified	Yes	Yes
More than one index can be rebuilt in a single transaction	Yes	No
Most index types can be rebuilt online without blocking running queries or updates	Yes	Yes
Partitioned index can be repartitioned	No	Yes
Index can be moved to another filegroup	No	Yes
Additional temporary disk space is required	Yes	Yes
Rebuilding a clustered index rebuilds associated nonclustered indexes	No***	No****
Indexes enforcing PRIMARY KEY and UNIQUE constraints can be rebuilt without dropping and re-creating the constraints	Yes	Yes
Single index partition can be rebuilt	Yes	No

Online Index Rebuilds

- Introduced in SQL Server 2005 Enterprise Edition
    - SORT_IN_TEMPDB index rebuild option
- 3 Phases
    - Preparation
    - Build
    - Finalize

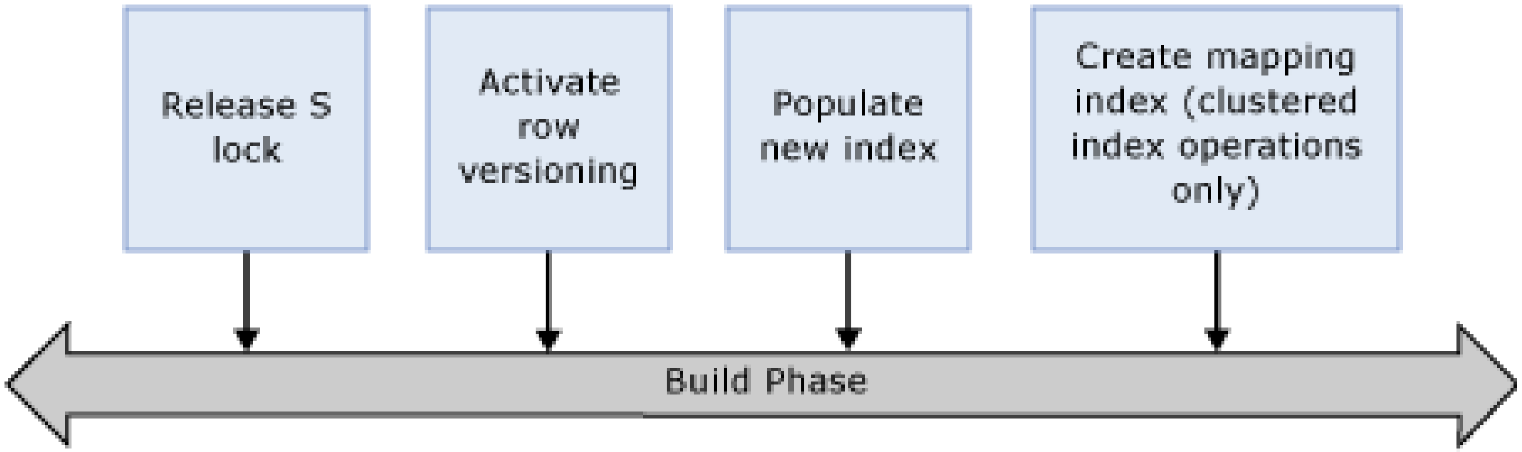
Online Index Rebuild: Phase 1

- Locks acquired
- Metadata created
- DML plans are recompiled


Versioning transaction acquires S lock
Index builder transaction acquires IS lock
Create metadata in-memory only
Force DML recompile
Preparation Phase

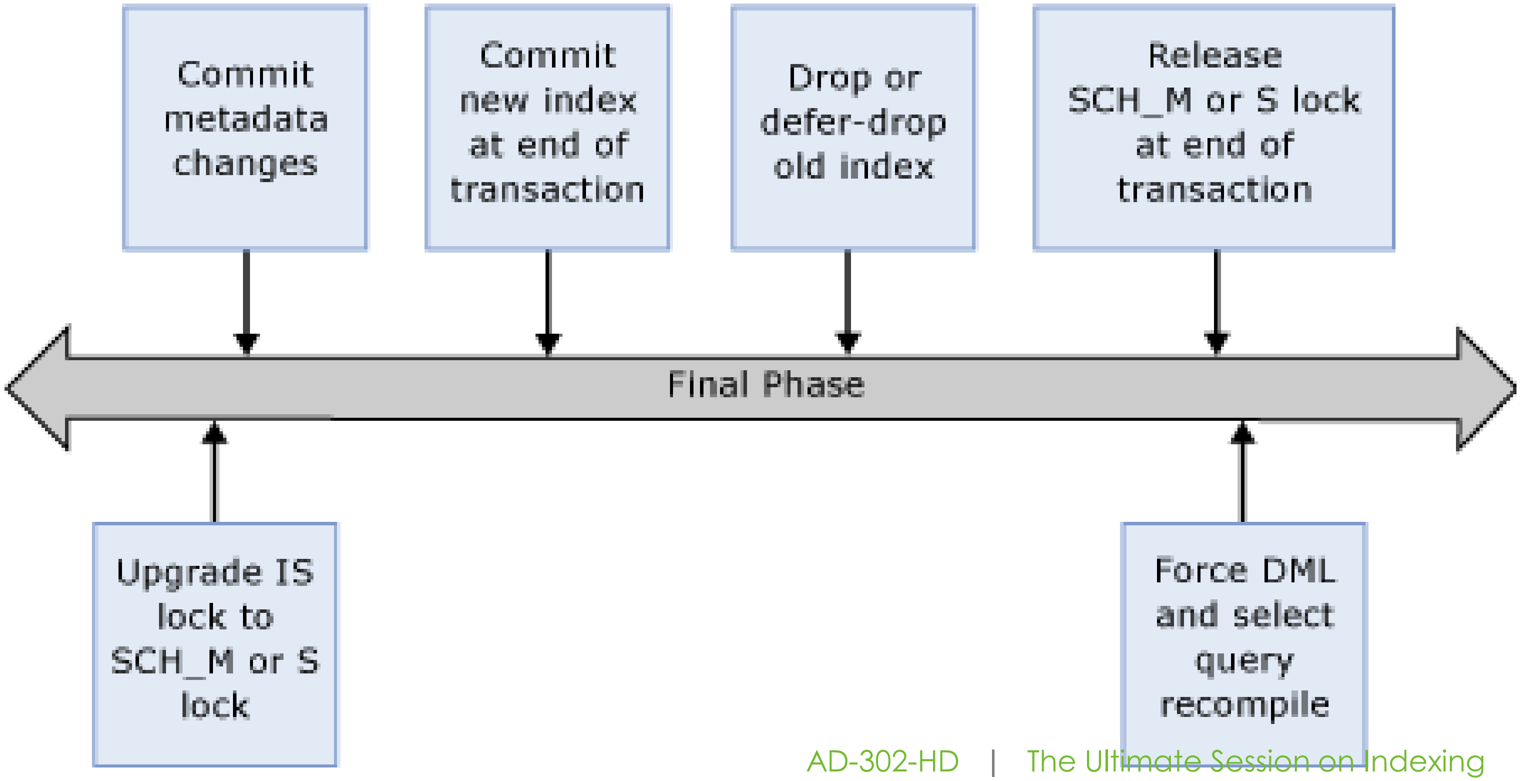
Online Index Rebuild: Phase 2

- Populate the new index with sorted data from the existing data source while allowing select and DML operations to continue
- *Mapping index* created for clustered index



Online Index Rebuild: Phase 3

- Make new index "available" and "clean up"


Commit metadata changes
Commit new index at end of transaction
Drop or defer-drop old index
Release SCH_M or S lock at end of transaction
Final Phase
Upgrade IS lock to SCH_M or S lock
Force DML and select query recompile

Reorganizing Indexes

- Reorganizing an index defragments the leaf level of clustered and nonclustered indexes on tables and views by physically reordering the leaf-level pages to match the logical order (left to right) of the leaf nodes
  - Uses existing pages
  - Additionally “compacts” pages
  - Emptied pages are “available”
  - Uses “minimal resources”
- Always online operation
- ALTER INDEX … REORGANIZE

REBUILD versus REORGANIZE

- REORGANIZE does not UPDATE STATISTICS
- Query **sys.dm_db_index_physical_stats** system function to determine level of fragmentation
  - **avg_fragmentation_in_percent** column returns the percent of logical fragmentation (out-of-order pages in the index).
- Microsoft's recommendations:

avg_fragmentation_in_percent	Statement
> 5% AND <= 30%	ALTER INDEX REORGANIZE
> 30%	ALTER INDEX REBUILD WITH (ONLINE = ON)

Victor's Alternative for Online Index Rebuilds

- Scenario:
  - 24 x 7 database solution
  - Want to take advantage of online index rebuilds
  - Have to buy Enterprise Edition
    - Enterprise Edition is expensive
      - Especially if you just want to buy it for this one feature
- Alternative:
  - Potentially can get away with REORGANIZING clustered indexes
  - Rebuild nonclustered indexes differently:
    - Create new nonclustered index on same columns
    - Drop existing nonclustered index
    - Rename new nonclustered index to match old one's name

Advanced Indexing Strategies

- Hash Index
- Helper Columns

“Helper” Columns

- Hash Index
- Helper Columns

Hash Index

- Use case:
  - You want to create an index on a column that is greater than 900 bytes
  - Might not want to leverage full text indexes
- Solution:
  - Create a computed, persisted column that “hashes” large column
    - LEFT, CHECKSUM, BINARY_CHECKSUM, HashBytes, etc
  - Create index on computed, persisted columns
  - Change SARG to include computed

"Helper" Index

- Use case:
  - You want to create an index on a column that is greater than 900 bytes
  - Might not want to leverage full text indexes
  - Data is predominantly inserted?
- Solution:
  - Create a computed, persisted bit column that searches for a pattern
    - PATINDEX, etc
  - Create index on computed, persisted bit column
  - Change SARG to include bit index
  - Remember: you cannot take advantage of filtered indexes here

Q & A

Questions?

Email	victor@sqlserversolutions.com.au
Blog	www.victorisakov.com
Twitter	@victorisakov
LinkedIn	www.linkedin.com/in/victorisakov
Website	www.sqlserversolutions.com.au
Certification	MVP Microsoft Most Valuable Professional; Microsoft CERTIFIED Architect; Microsoft CERTIFIED Master; Microsoft CERTIFIED Trainer

Complete the Evaluation Form to Win!

Win a Dell Mini Netbook – every day – just for handing in your completed form. Each session evaluation form represents a chance to win.

Pick up your evaluation form:

- In each presentation room
- Online on the PASS Summit website

Drop off your completed form:

- Near the exit of each presentation room
- At the Registration desk
- Online on the PASS Summit website

Sponsored by Dell

PASS 2011 SUMMIT

Thank you

for attending this session and the 2011 PASS Summit in Seattle