Module 13: Optimizing Query Performance

Overview

- Introduction to the Query Optimizer
- Obtaining Execution Plan Information
- Using an Index to Cover a Query
- Indexing Strategies
- Overriding the Query Optimizer

Introduction to the Query Optimizer

- Function
- How It Uses Cost-Based Optimization
- How It Works
- Phases
- Caching the Execution Plan
- Setting a Cost Limit

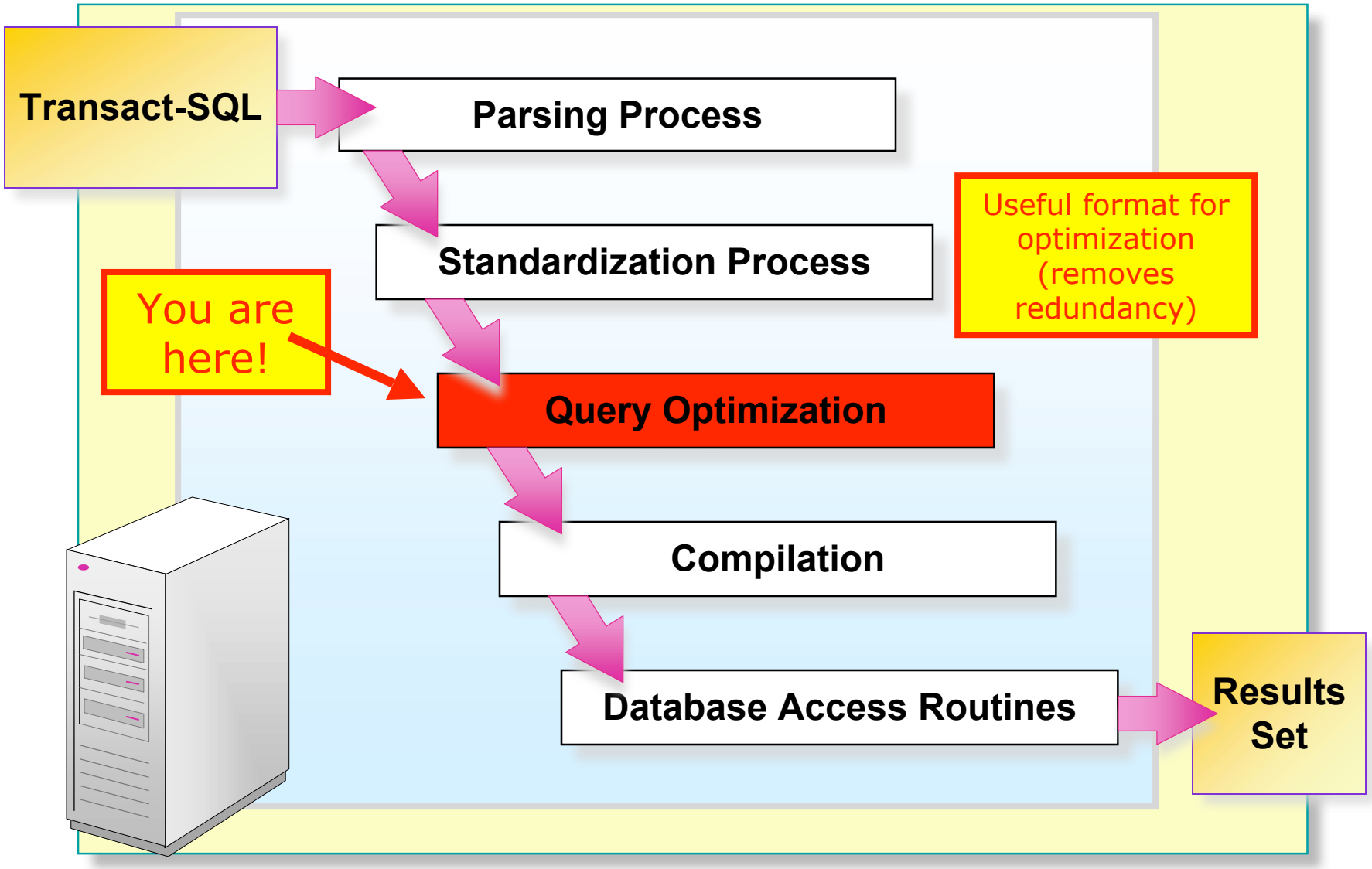
Function of the Query Optimizer

- **Determines the Most Efficient Execution Plan**
  - Determining whether indexes exist and evaluating their usefulness
  - Determining which indexes or columns can be used
  - Determining how to process joins
  - Using cost-based evaluation of alternatives
  - Creating column statistics
- **Uses Additional Information**
- **Produces an Execution Plan**

How the Query Optimizer Uses Cost-Based Optimization

- **Limits the Number of Optimization Plans to Optimize in Reasonable Amount of Time**
  - Cost is estimated in terms of I/O and CPU cost
- **Determines Query Processing Time**
  - Use of physical operators and sequence of operations
  - Use of parallel and serial processing

How the Query Optimizer Works



Query Optimization Phases

- **Query Analysis**
  - Identifies the search and join criteria of the query
- **Index Selection**
  - Determines whether an index or indexes exist
  - Assesses the usefulness of the index or indexes
- **Join Selection**
  - Evaluates which join strategy to use

Caching the Execution Plan

- **Storing a Execution Plan in Memory**
  - One copy for all serial executions
  - Another copy for all parallel executions
- **Using an Execution Context**
  - An existing execution plan is reused, if one exists
  - A new execution plan is generated, if one does not exist
- **Recompiling Execution Plans**
  - Changes can cause execution plan to be inefficient or invalid
    - For example, a large number of new rows added
    - ALTER TABLE/VIEW
    - UPDATE STATISTICS
    - Dropping an INDEX that is used
    - Explicit sp_recompile

Setting a Cost Limit

- **Specifying an Upper Limit (based on Estimated Costs)**
  - Use the query governor to prevent long-running queries from executing and consuming system resources
    - Effectively controls run-away queries
- **Specifying Connection Limits**
  - Use the **sp_configure** stored procedure
  - Execute the SET QUERY_GOVERNOR_COST_LIMIT statement
  - Specify 0 to turn off the query governor

Obtaining Execution Plan Information

- Viewing STATISTICS Statements Output
- Viewing SHOWPLAN_ALL and SHOWPLAN_TEXT Output
- Graphically Viewing the Execution Plan

Viewing STATISTICS Statements Output

Statement	Output Sample
STATISTICS TIME	SQL Server Execution Times: CPU time = 0 ms, elapsed time = 2 ms.
STATISTICS PROFILE	Rows Executes StmtText StmtId… ---------- 47 1 SELECT * FROM [charge] 16 WHERE (([charge_amt]>=@1) . . .
STATISTICS IO	Table 'member'. Scan count 1, logical reads 23, physical reads 0, read-ahead reads 0.

Viewing SHOWPLAN_ALL and SHOWPLAN_TEXT Output

- **Structure of the SHOWPLAN Statement Output**
  - Returns information as a set of rows
  - Forms a hierarchical tree
  - Represents steps taken by the query optimizer
  - Shows estimated values of how a query was optimized, not the actual execution plan
- **Details of the Execution Steps**
- **Explore:**
  - What is the difference Between SHOWPLAN_TEXT and SHOWPLAN_ALL Output

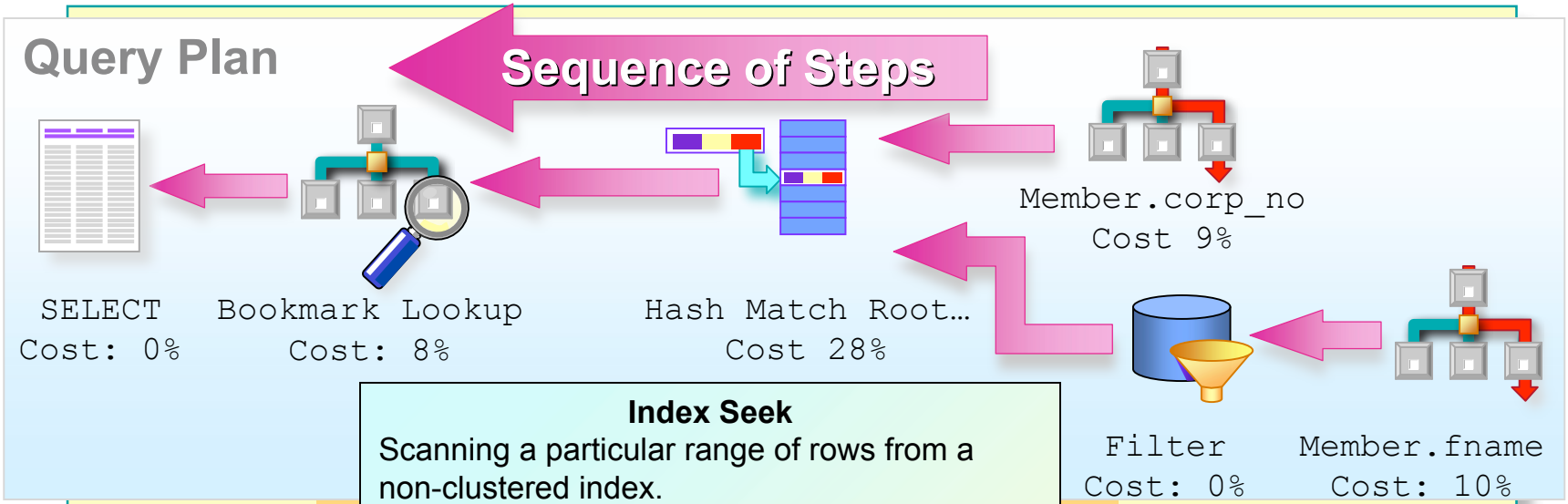
Graphically Viewing the Execution Plan

- Elements of the Graphical Execution Plan
- Reading the Graphical Execution Plan Output
- Using the Bookmark Lookup Operation

Elements of the Graphical Execution Plan

- Steps Are Units of Work to Process a Query
- Sequence of Steps Is the Order in Which the Steps Are Processed
- Logical Operators Describe Relational Algebraic Operation Used to Process a Statement
- Physical Operators Describe Physical Implementation Algorithm Used to Process a Statement

Reading Graphical Execution Plan Output



Index Seek

Scanning a particular range of rows from a non-clustered index.

Physical operation:	Index Seek
Logical operation:	Index Seek
Row count:	414
Estimated row sizes:	24
I/O cost:	0.00706
CPU cost:	0.000605
Number of executes:	1.0
Cost:	0.007675(6%)
Subtree cost:	0.00767

Argument:
OBJECT: ([credit].[dbo].[member].[fname]), SEEK: ([member],[firstname] >='Rb' AND [member],[firstname] <'T') ORDERED

Using an Index to Cover a Query

- **Covering a Query: Resolving Queries without accessing the data pages**
  - Introduction to Indexes
  - Locating Data by Using Indexes
  - Identifying Whether an Index Can Be Used
  - Determining Whether an Index Is Used
  - Guidelines for Creating Indexes

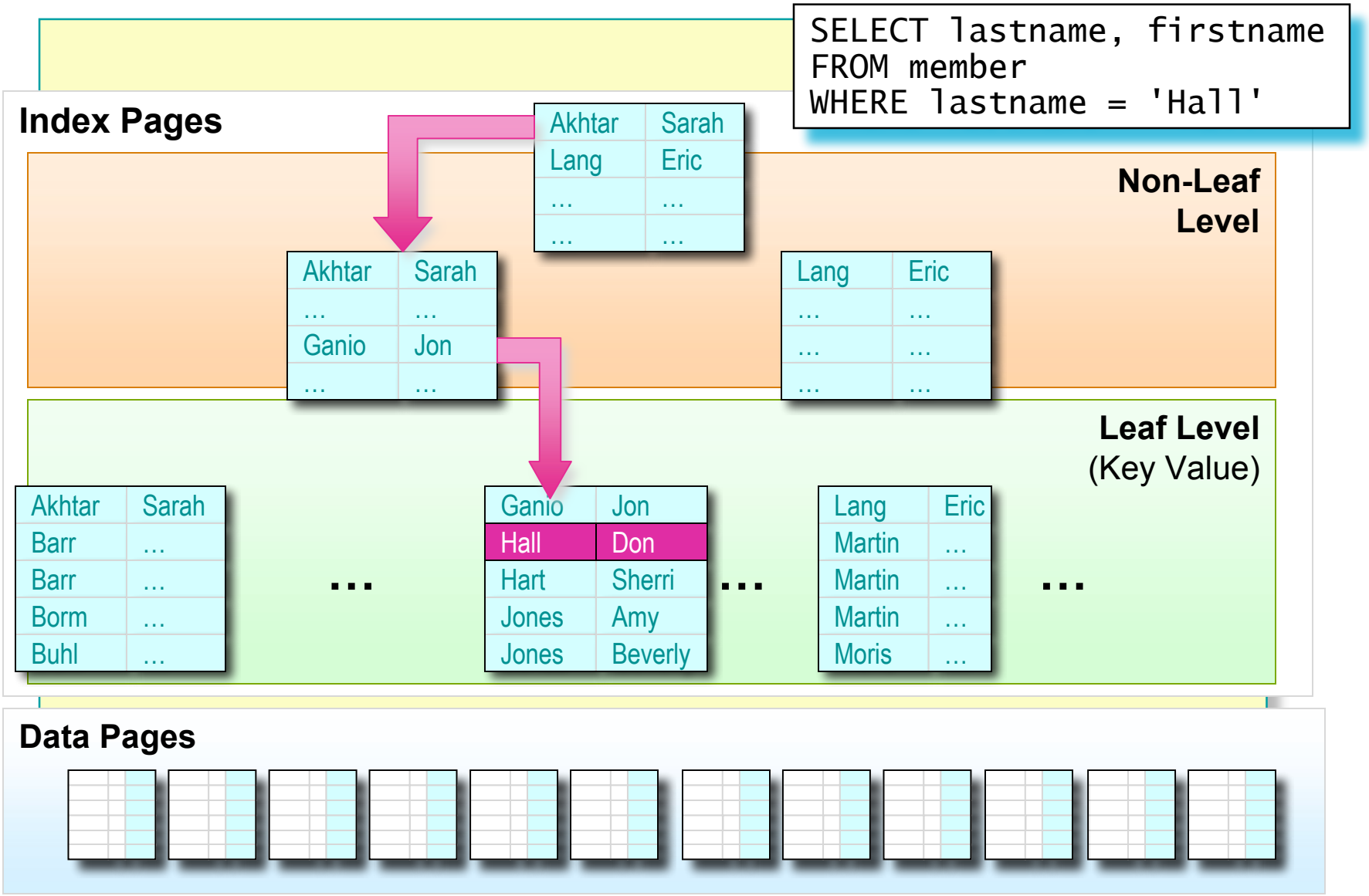
Introduction to Indexes That Cover a Query

- Indexes That Cover Queries Retrieve Data Quickly
- Only Nonclustered Indexes Cover Queries
- Indexes Must Contain All Columns Referenced in the Query
  - No Data Page Access Is Required
- Indexed Views Can Pre-Aggregate Data

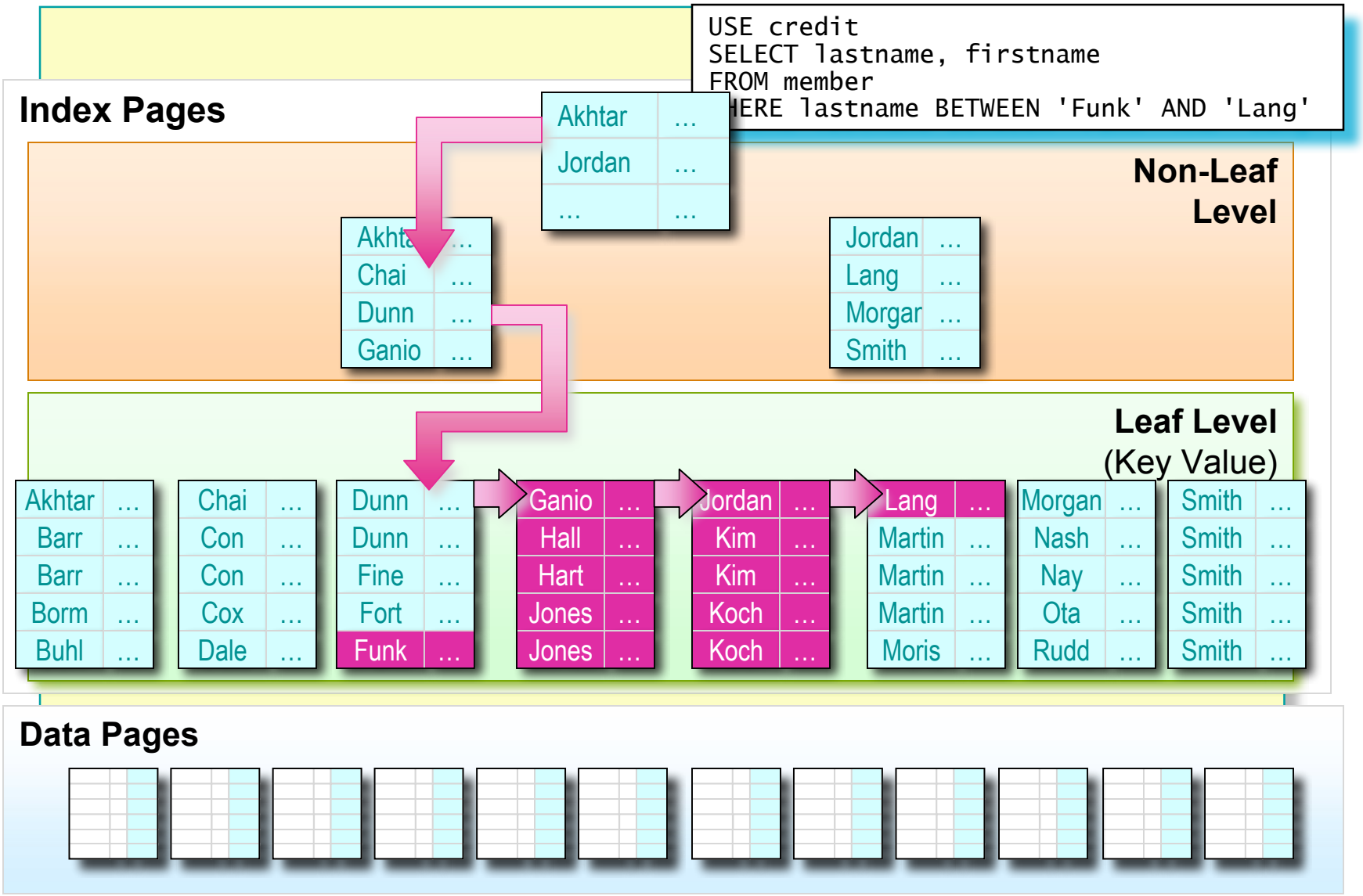
◆ Locating Data by Using Indexes That Cover a Query

- Example of Single Page Navigation
- Example of Partial Scan Navigation
- Example of Full Scan Navigation

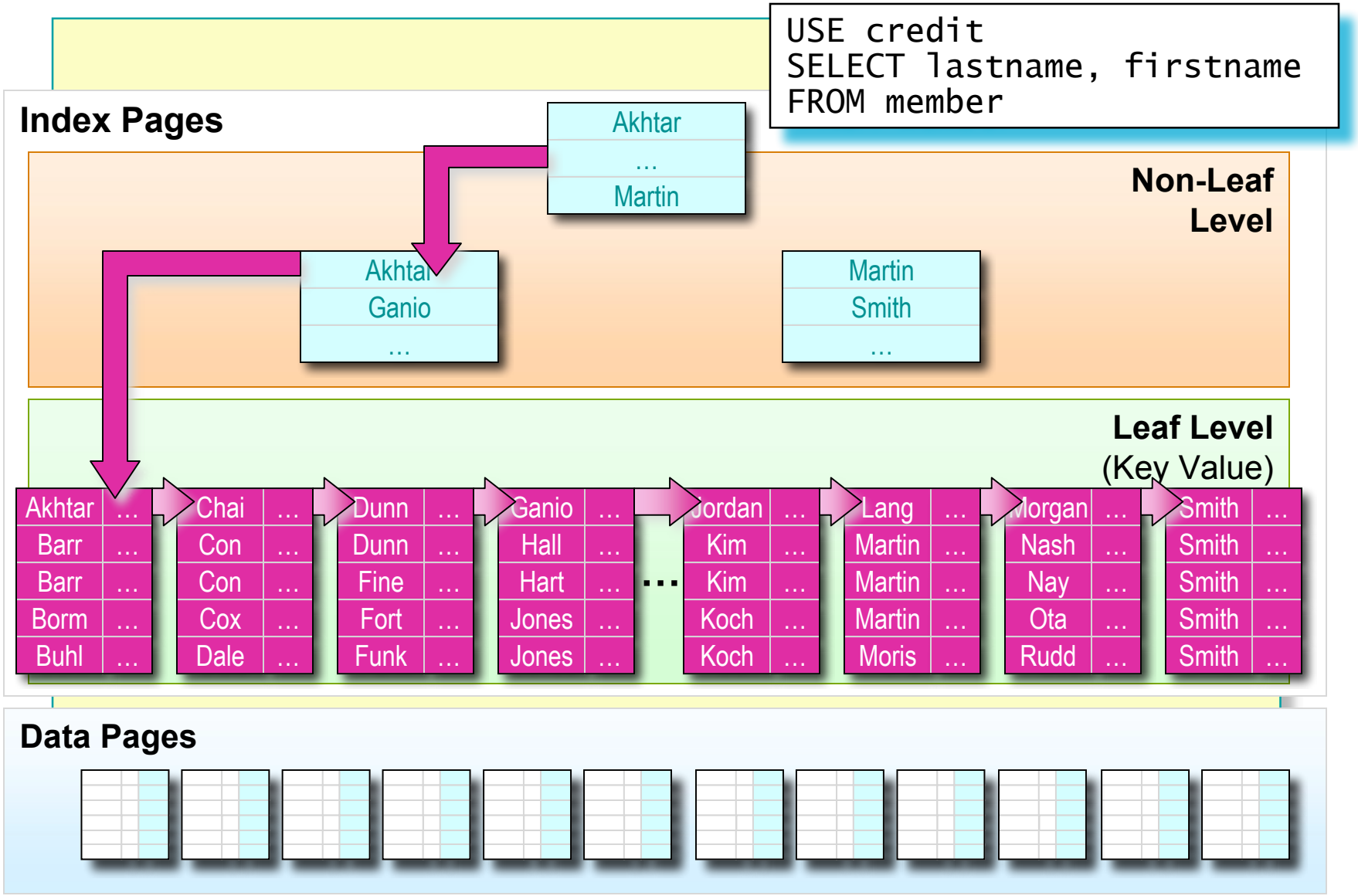
Example of Single Page Navigation



Example of Partial Scan Navigation



Example of Full Scan Navigation



Identifying Whether an Index Can Be Used to Cover a Query

- All Necessary Data Must Be in the Index
- A Composite Index Is Useful Even if the First Column Is Not Referenced
- A WHERE Is Not Necessary
- A Nonclustered Index Can Be Used if It Requires Less I/O Than a Clustered Index Containing a Column Referenced in the WHERE Clause
- Indexes Can Be Joined to Cover a Query

Determining Whether an Index Is Used to Cover a Query

- **Observing the Execution Plan Output**
  - Displays the phrase “Scanning a non-clustered index entirely or only a range”
- **Comparing I/O**
  - Nonclustered index
    - Total number of levels in the non–leaf level
    - Total number of pages that make up the leaf level
    - Total number of rows per leaf-level page
    - Total number of rows per data page
  - Total number of pages that make up the table

Guidelines for Creating Indexes That Cover a Query

- Add Columns to Indexes
- Minimize Index Key Size
- Maintain Row-to-Key Size Ratio

Indexing Strategies

- Evaluating I/O for Queries That Access a Range of Data
- Indexing for Multiple Queries
- Guidelines for Creating Indexes

Evaluating I/O for Queries That Access a Range of Data

```
SELECT charge_no
FROM charge
WHERE charge_amt BETWEEN 20 AND 30
```

Access method	*Page I/O*
Table scan	**10,417**
Clustered index on the charge_amt column	**1042**
Nonclustered index on the charge_amt column Each data page is read multiple times	**100,273**
Composite index on charge_amt, charge_no columns Covering Query	**273**

Indexing for Multiple Queries

Example 1

```
USE credit
SELECT charge_no, charge_dt, charge_amt
FROM charge
WHERE statement_no = 19000 AND member_no = 3852
```

Example 2

```
USE credit
SELECT member_no, charge_no, charge_amt
FROM charge
WHERE charge_dt between '07/30/1999'
AND '07/31/1999' AND member_no = 9331
```

Guidelines for Creating Indexes

- **Determine the Priorities of All of the Queries**
- **Determine the Selectivity for Each Portion of the WHERE Clause of Each Query**
- **Determine Whether to Create an Index**
  - Based on priority, selectivity, column width
- **Identify the Columns That Should Be Indexed**
- **Determine the Best Column Order of Composite Indexes**
- **Determine What Other Indexes Are Necessary**
- **Test the Performance of the Queries**
  - SET SHOWPLAN ON
    SET STATISTICS TIME ON
    SET STATISCTICS IO ON

Overriding the Query Optimizer

- Determining When to Override the Query Optimizer
- Using Hints and SET FORCEPLAN Statement
- Confirming Query Performance After Overriding the Query Optimizer

Determining When to Override the Query Optimizer

- **Limit Optimizer Hints**
  - Leads Optimizer in a certain direction
  - Use only if Optimizer is not doing a good job
- **Explore Other Alternatives Before Overriding the Query Optimizer by:**
  - Updating statistics
  - Recompiling stored procedures
  - Reviewing the queries or search arguments
  - Evaluating the possibility of building different indexes

Using Hints and SET FORCEPLAN Statement

- **Table Hints**
  - Forces use of an Index
- **Join Hints**
  - Forces what time of JOIN to use. E.g., MERGE-JOIN
- **Query Hints**
  - Forces a query to use a particular aspect of the plan
- **SET FORCEPLAN Statement**

Confirming Query Performance After Overriding the Query Optimizer

- Verify That Performance Improves
- Document Reasons for Using Optimizer Hints
- Retest Queries Regularly

Recommended Practices

- Use the Query Governor to Prevent Long-Running Queries from Consuming System Resources
- Have a Thorough Understanding of the Data and How Queries Gain Access to Data
- Create Indexes That Cover the Most Frequently Used Queries
- Establish Indexing Strategies for Individual and Multiple Queries
- Avoid Overriding the Query Optimizer

Review

- Introduction to the Query Optimizer
- Obtaining Query Plan Information
- Using an Index to Cover a Query
- Indexing Strategies
- Overriding the Query Optimizer