Student Guide

Volume 1

# System 10 Performance & Tuning

Version 2.2


*The Enterprise Client/Server Company™*

58164-01-0220-00-V1

# Notice



## Sybase Trademarks

SYBASE, the SYBASE logo, APT-FORMS, Data Workbench, DBA Companion, Deft, Gain*Momentum*, SA Companion, SQL Debug, SQL Solutions, SQR, Transact-SQL, and VQL are registered trademarks of Sybase, Inc. ADA Workbench, Adaptable Windowing Environment, Application Manager, Applications from Models, APT Workbench, APT-build, APT-Edit, APT-Execute, APT-Translator, APT-Library, Build *Momentum*, Camelot, Client/Server Architecture for the Online Enterprise, Client/Server for the Real World, Configurator, DataServer, Data Workbench, Database Analyzer, DB-Library, Deft Analyst, Deft Designer, Deft Educational, Deft Professional, Deft Trial, Developers Workbench, Easy SQR, Embedded SQL, Enterprise Builder, Enterprise Client/Sever, Enterprise Meta Server, Enterprise Modeler, Enterprise *Momentum*, Gain, Gain*Exposure*, Insight, Mainframe Access Products (MAP), *Momentum*, Movedb, Navigation Server, Net-Gateway, Net Library, Object *Momentum*, OmniSQL Gateway, Omni SQL *Server*, OmniSQL Access Module, Open Client, Open Gateway, Open Server, Open Solutions, PC APT-Execute, PC DB-Net, PC Net Library, Report Workbench, Report Execute, Open Server, Partnerships that Work, Replication Server, Resource Manager, RW-Display Lib, RW-Library, SA Monitor, Secure SQL Server, Secure SQL Toolset, SQL Code Checker, SQL Edit/TPU, SQL Edit, SQL Monitor, SQL Server, SQL Toolset, SQL Server/CFT, SQL Server/DBM, SQL Station, SQR Toolset, SQR Developers Kit, SQR Workbench, SYBASE Client/Server Interfaces, SYBASE Gateways, SYBASE SQL Lifecycle, Sybase Synergy Program, SYBASE Virtual Server Architecture, SYBASE User Workbench System 10, Tabular Data Stream, and The Enterprise Client/Server Company are trademarks of Sybase, Inc.

All other company and product names used herein may be the trademarks or registered trademarks of their respective companies.

## Restricted Rights Legend

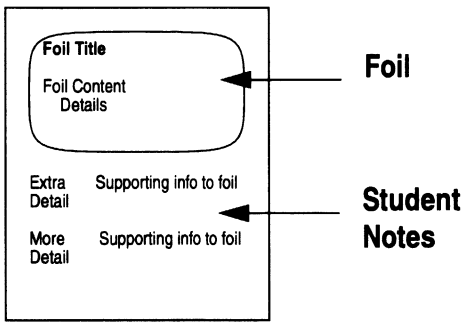
Use, duplication or disclosure by the Government is subject to restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608

# How to Use This Guide

**Student Guide**

- The pages of this Student Guide are divided into two sections: a reduced foil and a set of student notes.
- Your instructor will project the foil during class. The student notes provide additional notes to accompany the foil.




**Lab Workbook**

- In addition to the Student Guide, you are supplied with a Lab Workbook.
- The Lab Workbook includes a complete set of lab instructions and solutions for each lab.
- Each module usually includes at least one lab. The lab name and number in the Student Guide correspond to the same lab name and number in the Lab Workbook.

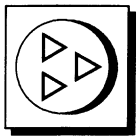
**ICONS**

Reference: Additional references are available for the current topic

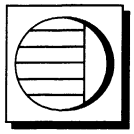
Question: This is a question you may want to ask yourself about the specific topic

Warning: This piece of information is important, and not adhering to it may cause you to run into some problems

# Diagramming Conventions



isql client



SQL Server

# Table of Contents

## Volume 1


Course Objectives ..... Intro–1
Course Roadmap ..... Intro–2

**Part 1**
**Designing Applications for Performance**

**Module 1** **Overview of Performance Issues**

Objectives ..... 1-1
What is Performance? ..... 1-2
What is Performance? ..... 1-3
Other Considerations for Performance ..... 1-4
Designing for Performance Is: ..... 1-5
Tuning For Performance Is: ..... 1-6
SQL Server Model of Operations ..... 1-7
Client/Server System Model ..... 1-8
Network/Hardware/OS Layer Issues ..... 1-9
SQL Server Layer ..... 1-10
Devices ..... 1-11
Database ..... 1-12
Application Layer ..... 1-13
System Limitations ..... 1-14
Fundamental Design & Tuning Guidelines ..... 1-15
Trade-offs ..... 1-16
SQL Server Problem Analysis Approach ..... 1-17
Lab 1 Performance Overview ..... 1-18
Case Study Introduction ..... 1-19
pubtune Database ..... 1-20
Order Entry Scenario ..... 1-21
Shipping Scenario ..... 1-22
Sales Scenario ..... 1-23
Summary ..... 1-24

## Module 2 Physical Database Design and Denormalization


Objectives ..... 2-1
Logical Database Design ..... 2-2
Physical Database Design ..... 2-3
From Logical to Physical Database Design ..... 2-4
Normalization ..... 2-5
First Normal Form ..... 2-6
Second Normal Form ..... 2-7
Third Normal Form ..... 2-8
Why Be Normal? ..... 2-9
Performance Benefits of Normalization ..... 2-10
Denormalization ..... 2-11
Performance Benefits of Denormalization ..... 2-12
Denormalization Input ..... 2-13
Denormalization Techniques ..... 2-14
Adding Redundant Columns ..... 2-15
Adding Derived Columns ..... 2-16
Collapsing Tables ..... 2-17
Duplicating Tables ..... 2-18
Splitting Tables ..... 2-19
Horizontal Partition Example ..... 2-20
Vertical Partition Example ..... 2-21
Managing Denormalized Data ..... 2-22
Using Triggers to Manage Denormalized Data ..... 2-23
Using Application Logic to Manage Denormalized Data ..... 2-24
Batch Reconciliation ..... 2-25
Sample Problem #1 ..... 2-26
Read-Only Tables ..... 2-27
Sample Problem #2 ..... 2-28
Storing Calculated Values ..... 2-29
Evaluating Benefits of Denormalization ..... 2-30
How to Group Tables ..... 2-31
Lab 2 - Physical Database Design ..... 2-32
Summary ..... 2-33


## Module 3 Table Storage


Objectives ..... 3-1
Data Pages ..... 3-2
Linked Data Pages ..... 3-3
Row Density ..... 3-4
Other Types of Pages ..... 3-5

Storage Structures: Topics ..... 3-6
Non-clustered Table Storage ..... 3-7
Operations on Non-clustered Tables ..... 3-8
Non-clustered Tables: Advantages and Disadvantages ..... 3-9
Predicting Table Size ..... 3-10
Calculating Row Size: Determine Row Size ..... 3-11
Calculating Row Length: Algorithm ..... 3-12
Calculating Row Length: Example ..... 3-13
Predicting Table Size: Number of Pages ..... 3-14
Predicting Table Size: sp_estspace ..... 3-15
Displaying Table Size: sp_spaceused ..... 3-16
Displaying Table Size: dbcc tablealloc ..... 3-17
Lab 3 – Table Storage: Size ..... 3-18
Text and Image Chains ..... 3-19
Text Page ..... 3-20
Predicting Size of Text and Image Chains ..... 3-21
Text and Image Chains: Issues and Guidelines ..... 3-22
Summary ..... 3-23

## Module 4 How Indexes Work

Objectives ..... 4–1
Why Study Indexes? ..... 4–2
Topics ..... 4–3
What Are Indexes? ..... 4–4
Why Use Indexes? ..... 4–5
Index Structure ..... 4–6
Tables and Indexes ..... 4–7
Indexes and Keys ..... 4–8
How Indexes Work: Topics ..... 4–9
Clustered Index Structure ..... 4–10
Clustered Table Storage ..... 4–11
What Happens During a Select ..... 4–12
What Happens During an Insert ..... 4–13
What Happens During a Delete ..... 4–14
Clustered Index Size ..... 4–15
Predicting Clustered Index Size ..... 4–16
Clustered/Root Index Row Size ..... 4–17
Predicting Clustered Index Size: Example ..... 4–18
Predicting Clustered Index Size (continued) ..... 4–19
Predicting Index Size Using sp_estspace ..... 4–20
Predicting Index Size Using sp_estspace (continued) ..... 4–21
Displaying Current Index Size: sp_spaceused ..... 4–22

Displaying Current Index Size: dbcc indexalloc ..... 4–23
How Indexes Work: Topics ..... 4–24
Non-clustered Index Structure ..... 4–25
Non-clustered Index Storage ..... 4–26
What Happens During a Select ..... 4–27
What Happens During an Insert ..... 4–28
What Happens During a Delete ..... 4–29
Non-clustered Intermediate Page Index Row ..... 4–30
Non-clustered Leaf Page Index Row Size ..... 4–31
Predicting Non-Clustered Index Size ..... 4–32
Predicting Non-Clustered Index Size (Continued) ..... 4–33
Predicting Non-Clustered Index Size (Continued) ..... 4–34
Predicting Index Size: sp_estspace ..... 4–35
Displaying Current Index Size: sp_spaceused ..... 4–36
Displaying Current Index Size: dbcc indexalloc ..... 4–37
Displaying Current Indexes: sp_helpindex ..... 4–38
Clustered vs. Non Clustered ..... 4–39
Lab 4a – Indexes and I/O Activity ..... 4–40
How Indexes Work: Topics ..... 4–41
Data Page Splits ..... 4–42
Clustered Index Data Page Splitting ..... 4–43
Overflow Pages (Clustered Index Data Pages Only) ..... 4–44
Clustered Index Data Page Merging ..... 4–45
Fill Factor ..... 4–46
Data Page Splits and Fill Factor ..... 4–47
Setting Fill Factor ..... 4–48
Fill Factor Guidelines ..... 4–49
Lab 4b – Fill Factor and I/O Statistics ..... 4–50
Updates ..... 4–51
Rows Inserted in an Update Operation ..... 4–52
Direct or Deferred Updates ..... 4–53
Direct Updates in Place ..... 4–54
Direct Updates in Place – Requirements ..... 4–55
Direct Updates (Not in Place) ..... 4–56
Deferred Updates ..... 4–57
Lab 4c – Where do Updated Rows Go? ..... 4–58
Summary ..... 4–59

**Module 5** **Selecting Indexes for Performance**

Objectives ..... 5-1
Topics ..... 5-2
How Indexes Can Affect Performance ..... 5-3

How Indexes Affect Performance ..... 5-4
Mechanics of Table Scans ..... 5-5
Topics ..... 5-6
Evaluating Cost of Table Scans ..... 5-7
Topics ..... 5-8
Evaluating Cost of Clustered Index Access (Point Query) ..... 5-9
Evaluating Cost of Clustered Index Access (Range Query) ..... 5-10
Evaluating Cost of Clustered Index Access (Range Query I/O) ..... 5-11
Topics ..... 5-12
Evaluating Cost of Non-clustered Index Access (Point Query) ..... 5-13
Evaluating Cost of Non-clustered Index Access (Range Query) ..... 5-14
Indexes and I/O Statistics ..... 5-15
Topics ..... 5-16
Index Covering ..... 5-17
Evaluating Cost of Covered Access ..... 5-18
Index Covering: Adding Columns ..... 5-19
Topics ..... 5-20
Choosing Indexes: Tables Without Indexes ..... 5-21
Choosing Indexes: Clustered ..... 5-22
Clustered Indexes: Guidelines ..... 5-23
Choosing Indexes: Non-clustered ..... 5-24
Choosing Indexes: Covered ..... 5-25
Lab 5 – Indexes and Performance ..... 5-26
Summary ..... 5-27

## Module 6 Query Optimization

Objectives ..... 6-1
Why Study the Optimizer? ..... 6-2
Common Symptoms of Query Trouble ..... 6-3
What Does the Query Optimizer Do? ..... 6-4
Query Plans ..... 6-5
Displaying Query Plans ..... 6-6
Displaying Query Plans (continued) ..... 6-7
Displaying Query Plans (continued) ..... 6-8
What Happens When You Run a Query? ..... 6-9
Class Exercise: showplan ..... 6-10
Query Optimization Steps ..... 6-11
Analyze Each Table ..... 6-12
Search Arguments (SARGS) ..... 6-13
SARG Equivalents ..... 6-14
Padding With SARGs ..... 6-15
Lab 6a – Search Arguments ..... 6-16

OR Clauses and Equivalents ..... 6-17
Resolution of OR Clauses ..... 6-18
OR Strategy Methodology ..... 6-19
OR Strategy Showplan ..... 6-20
Lab 6b – OR Strategies ..... 6-21
Index Covering ..... 6-22
Index Covering (continued) ..... 6-23
How Aggregate Queries Are Optimized ..... 6-24
Aggregate Query Plan Example ..... 6-25
Aggregates: Special Cases ..... 6-26
Lab 6c – Index Covering and Aggregates ..... 6-27
Review ..... 6-28
Query Optimization Steps ..... 6-29
Selecting an Index ..... 6-30
Row Estimates Using Index Access ..... 6-31
When Will the Distribution Page Not be Considered? ..... 6-32
Estimating Number of Logical Page Reads ..... 6-33
Lab 6d – Predicting Index Usage ..... 6-34
Query Optimization Steps ..... 6-35
Join Clauses ..... 6-36
Generating Join Clauses ..... 6-37
Join Clauses (continued) ..... 6-38
Nested Iterations ..... 6-39
Nested Iterations Example ..... 6-40
Nested Iterations: First Approach ..... 6-41
Nested Iterations: Second Approach ..... 6-42
Class Exercise: Which Solution Would Be Better...? ..... 6-43
Three-Way Join: First Approach ..... 6-44
Three-Way Join: Second Approach ..... 6-45
Outer Joins ..... 6-46
Row Estimates for Join Clauses ..... 6-47
What Is On a Distribution Page? ..... 6-48
Composite Index Density on Distribution Page ..... 6-49
Costing Calculations: Example ..... 6-50
Nested Iterations Performance ..... 6-51
Nested Iterations-Logical I/O Estimates ..... 6-52
Nested Iterations: I/O Calculation Examples ..... 6-53
Nested Iterations – Final Cost ..... 6-54
I/O Costing ..... 6-55
Finding the Optimal Join Strategy ..... 6-56
Lab 6e – Join Strategies ..... 6-57
Reformatting ..... 6-58
Reformatting (continued) ..... 6-59

Reformatting Query Plan ..... 6-60
Join Selection Review ..... 6-61
When Are Work Tables Used? ..... 6-62
Sample Work Tables: Order By ..... 6-63
Sample Work Tables: Group By ..... 6-64
Sample Work Tables: Distinct ..... 6-65
Lab 6f – Work Tables and Sorts ..... 6-66
Query Optimization Steps Summary ..... 6-67
Stored Procedure Optimization ..... 6-68
Stored Procedures and Cache ..... 6-69
Misleading Initial Value Example ..... 6-70
Optimizing Stored Procedures ..... 6-71
Optimizing Stored Procedures (Continued) ..... 6-72
Recompiling Stored Procedures ..... 6-73
Lab 6g – Stored Procedures ..... 6-74
Basic Questions ..... 6-75
Useful Techniques ..... 6-76
"Watch Out For" Summary ..... 6-77

**Part 2**
**Tuning Applications for Performance**

**Module 7** **Distributing Data Across Devices**

Objectives ..... 7-1
Designing & Tuning: A Course Map ..... 7-2
System Model: Object Placement ..... 7-3
Improving Performance via Object Placement ..... 7-4
Things to Check ..... 7-5
Disks and Logical Devices ..... 7-6
Device Management ..... 7-7
Spread Access Across Separate Disks: Avoid I/O Contention ..... 7-8
Isolate Server-wide I/O from Database I/O ..... 7-9
Keep Transaction Log On Separate Disk ..... 7-10
Transaction Log Storage ..... 7-11
Separating Log By Moving It to a New Device ..... 7-12
Mirroring a Device ..... 7-13
Device Mirroring: Performance Issues ..... 7-14
Using Segments to Improve Performance ..... 7-15
System-Defined Segments ..... 7-16
Adding a User-Defined Segment ..... 7-17
Adding a User-Defined Segment (continued) ..... 7-18
Dropping System & Default Segments ..... 7-19

Creating Objects On Segments ..... 7-20
Displaying Information About Segments and Devices ..... 7-21
Displaying Information About Segments ..... 7-22
Displaying Information About Objects on Segments ..... 7-23
Segment Management Tools ..... 7-24
Isolating Frequently Accessed Tables ..... 7-25
Separating Tables and Indexes ..... 7-26
Isolating Text and Image Data ..... 7-27
Spreading Table Data to Multiple Disks ..... 7-28
Splitting a Table Across Multiple Disks ..... 7-29
Sample Problem #1 ..... 7-30
Collect Data ..... 7-31
Collect Data (continued) ..... 7-32
Collect Data: Results ..... 7-33
Formulate Hypothesis ..... 7-34
Test Hypothesis ..... 7-35
Sample Problem #2 ..... 7-36
Object Placement: Questions to Ask ..... 7-37
Object Placement: Questions to Ask (continued) ..... 7-38
Lab 7 – Distributing Data Across Devices ..... 7-39
Summary ..... 7-40

# Volume 2

## Module 8 How Locking Affects Performance


Objectives ..... 8-1
Why Study Locks? ..... 8-2
Things to Check ..... 8-3
Why Are Locks Needed? ..... 8-4
Isolation Levels ..... 8-5
The "Dirty Read" Problem ..... 8-6
The "Non-repeatable Reads" Problem ..... 8-7
The "Phantom Reads" Problem ..... 8-8
SQL Server Isolation Levels ..... 8-9
Granularity of Locks ..... 8-10
Locking in SQL Server ..... 8-11
Page Locks ..... 8-12
Page Locking Examples ..... 8-13
Table Locks ..... 8-14
SQL Statements and Intent Locks ..... 8-15
Escalating Locks ..... 8-16
holdlock ..... 8-17
Checking for Blocked Processes ..... 8-18
Cursors and Locking ..... 8-19
Deadlock ..... 8-20
Avoiding Deadlock ..... 8-21
Locking and Performance ..... 8-22
Reduce Locking Techniques ..... 8-23
Reducing Locking Techniques (continued) ..... 8-24
Sample Problem ..... 8-25
Collect Data ..... 8-26
Collect Data: Results ..... 8-27
Formulate Hypothesis ..... 8-28
Test Hypothesis ..... 8-29
Implement if Confirmed ..... 8-30
Locking: Questions to Ask ..... 8-31
Lab 8 – Locking and Performance ..... 8-32
Summary ..... 8-33


## Module 9 Managing Memory


Objectives ..... 9-1
Memory Size ..... 9-2

More Memory Improves Performance ..... 9-3
How Much Memory to Give SQL Server ..... 9-4
Memory: The "Big Picture" ..... 9-5
dbcc memusage ..... 9-6
Reconfiguring Memory ..... 9-7
Displaying Configuration Values ..... 9-8
Most Options Affect Size of Server Structures ..... 9-9
Connections: Example ..... 9-10
Options Which Consume Memory: Databases, Locks, Objects, Devices 9-11
Network Memory Increases Size of Server ..... 9-12
Extent I/O Buffers Increase Size of Server ..... 9-13
Audit Queue Consumes Memory ..... 9-14
Auditing and Performance ..... 9-15
Split Remaining Memory: ..... 9-16
What Procedure Cache Is For ..... 9-17
Procedure Cache Sizing ..... 9-18
What Data Cache is For ..... 9-19
Page Aging in Data Cache ..... 9-20
Effect Of Data Cache On Retrievals ..... 9-21
Effect Of Data Cache On Updates ..... 9-22
Data Cache Performance ..... 9-23
Finding Out Cache Sizes ..... 9-24
Configuration Example ..... 9-25
Memory Allocation: Questions to Ask ..... 9-26
SQL Monitor - Cache Window ..... 9-27
Summary: Memory Size & Balance ..... 9-28
Sample Problem ..... 9-29
Collect Data: Commands ..... 9-30
Collect Data: Results ..... 9-31
Formulate Hypothesis ..... 9-32
Test Hypothesis ..... 9-33
Implement Solution ..... 9-34
Lab 9 – Memory and Performance ..... 9-35
Summary ..... 9-36

## Module 10 Maintaining High Performance

Objectives ..... 10-1
Why Study Maintenance Activities? ..... 10-2
Topics ..... 10-3
Creating Databases ..... 10-4
Creating Indexes ..... 10-5
Creating Indexes: Using Extent I/O Buffers ..... 10-6

Creating Indexes: More Suggestions ..... 10-7
Topics ..... 10-8
How Backup Server Works ..... 10-9
Backup Server: Performance Features ..... 10-10
Recovery Interval ..... 10-11
Transaction Log Dumps ..... 10-12
Topics ..... 10-13
Bulk Copy: Performance Issues ..... 10-14
Bulk Copy: Suggestions ..... 10-15
Topics ..... 10-16
Database Consistency Checker (dbcc) ..... 10-17
Lab 10 – Maintenance and Performance ..... 10-18
Summary ..... 10-19

**Module 11** **Guidelines for High-Performance Applications**

Objectives ..... 11-1
Application Development Life Cycle ..... 11-2
Iterative Application Design ..... 11-3
Initial Design Decisions Affecting Applications ..... 11-4
Client vs. Server? ..... 11-5
Interactive vs. Batch Processing ..... 11-6
Additional Batch Guidelines ..... 11-7
Application Design Summary ..... 11-8
Managing User Interaction Example ..... 11-9
System Context Model ..... 11-10
Guidelines: Network ..... 11-11
Guidelines: Hardware ..... 11-12
Guidelines: OS ..... 11-13
Guidelines: Server Configuration ..... 11-14
Guidelines: Disks/Devices ..... 11-15
Guidelines: Database ..... 11-16
Guidelines: Tables ..... 11-17
Guidelines: Indexes ..... 11-18
Guidelines: Security ..... 11-19
Lab 11 – Designing Applications for Performance ..... 11-20
Summary ..... 11-21

## Part 3
## Special Topics

## Module 12 Networks and Performance


Objectives ..... 12-1
Why Study the Network? ..... 12-2
Underlying Problems ..... 12-3
System Model ..... 12-4
Networks & Performance Topics ..... 12-5
How SQL Server Uses the Network ..... 12-6
Networks & Performance Topics ..... 12-7
SQL Server Options ..... 12-8
User Connections and Buffers ..... 12-9
Where's My Memory Going? ..... 12-10
Networks & Performance Topics ..... 12-11
Smaller vs. Larger Packet Sizes ..... 12-12
Different Packet Sizes for Individual Clients ..... 12-13
Packet Size ..... 12-14
Networks & Performance Topics ..... 12-15
Reducing Network Traffic Techniques ..... 12-16
Large Transfers ..... 12-17
Evaluation Tools with SQL Server ..... 12-18
Operating System Network Evaluation Tools ..... 12-19
Impact of Other Server Activities ..... 12-20
Networks & Performance Topics ..... 12-21
Guideline 1 ..... 12-22
Guideline 2 ..... 12-23
Guideline 3 ..... 12-24
Guideline 4 ..... 12-25
Guideline 5 ..... 12-26
Guideline 6 ..... 12-27
Guideline 7 ..... 12-28
Guideline 8 ..... 12-29
Typical Problem ..... 12-30
Collect Data: Results ..... 12-31
Test Hypothesis ..... 12-32
Networks & Performance: Questions ..... 12-33
Lab 12 – Networks and Performance ..... 12-34
Summary ..... 12-35

## Module 13 tempdb


Objectives ..... 13-1
Why Worry About tempdb? ..... 13-2
Topics ..... 13-3
What is tempdb? ..... 13-4
Used for Internal Processing ..... 13-5
Storing Temporary Tables ..... 13-6
Using Temporary Tables to Split Up Multi-table Joins ..... 13-7
Indexes on Temporary Tables ..... 13-8
Topics ..... 13-9
Initial Allocation of tempdb ..... 13-10
Sizing tempdb ..... 13-11
Input to Sizing tempdb ..... 13-12
Sizing Algorithm ..... 13-13
Sizing Algorithm: Example ..... 13-14
Topics ..... 13-15
Placing tempdb ..... 13-16
Placing tempdb: Desirable ..... 13-17
Placing tempdb: Undesirable ..... 13-18
Preventing Temporary Tables from Spanning Multiple Devices ..... 13-19
Topics ..... 13-20
Locking in tempdb ..... 13-21
Minimizing Locking in tempdb ..... 13-22
Other tempdb Performance Tips ..... 13-23
Lab 13 – tempdb ..... 13-24
Summary ..... 13-25


## Module 14 Cursors and Performance


Objectives ..... 14-1
What Is A Cursor? ..... 14-2
Set-oriented vs. Row-oriented Programming ..... 14-3
Steps in Using a Cursor ..... 14-4
Cursors: A Simple Example ..... 14-5
Cursors: More Commands ..... 14-6
Resources Required At Each Stage ..... 14-7
Two Cursor Modes: Read-only and Update ..... 14-8
Index Use and Requirements ..... 14-9
Two Global Variables ..... 14-10
Sample Stored Procedure: No Cursors ..... 14-11
Sample Stored Procedure With Cursor ..... 14-12
Sample Stored Procedure With Cursor (continued) ..... 14-13

Performance: A Comparison ..... 14-14
Performance Implications ..... 14-15
Class Demo #1: Locking With Read-only Cursors ..... 14-16
Class Demo #2: Locking With Update Cursors ..... 14-17
Lab 14 – Cursors ..... 14-18
Summary ..... 14-19

## Module 15 CPU Utilization Issues

Objectives ..... 15-1
Why Study CPU Utilization? ..... 15-2
Underlying Problems ..... 15-3
System Model ..... 15-4
Single CPU Machines ..... 15-5
CPU Utilization of SQL Server ..... 15-6
CPU Utilization of SQL Server (continued) ..... 15-7
Overall System CPU Usage ..... 15-8
From Within SQL Server ..... 15-9
Single CPU Machine Summary ..... 15-10
SMP Machines (Symmetric Multi-Processing) ..... 15-11
SMP: SYBASE Virtual Server Architecture ..... 15-12
SMP: SQL Server Task Management ..... 15-13
SMP: Choosing the Right Number of Engines ..... 15-14
SMP: Application Design Considerations ..... 15-15
SMP: Overburdened CPU ..... 15-16
Multi-CPU Machines Summary ..... 15-17
CPU: The Time Slice Story ..... 15-18
CPU: Typical Questions to Ask ..... 15-19
Sample Problem ..... 15-20
Sample Problem (continued) ..... 15-21
Sample Problem (continued) ..... 15-22
SQL Monitor - Performance Trends ..... 15-23
Lab 15 – Using CPU Resources ..... 15-24
Summary ..... 15-25

## Appendix A Distributing Data with Replication Server

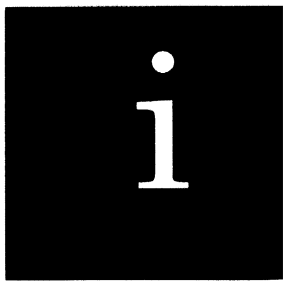
Replication Server ..... A-1
Sample Replication System ..... A-2
Replication Server: Performance Impact ..... A-3
Replication Server: Performance Impact (continued) ..... A-4

## Appendix B More About SQL Monitor


Objectives ..... B-1
SQL Monitor ..... B-2
SQL Monitor Architecture ..... B-3
Starting The SQL Monitor Client ..... B-4
SQL Monitor Main Menu ..... B-5
SQL Monitor Windows ..... B-6
Cache Window ..... B-7
Device I/O Window ..... B-8
Performance Summary Window ..... B-9
Performance Trends Window ..... B-10
Process Activity Window ..... B-11
Filtering Process Information ..... B-12
Process Detail Window ..... B-13
Process List Window ..... B-14
Transaction Activity Window ..... B-15
Summary ..... B-16
SQL Monitor Demos ..... B-17


## Appendix C Problem Analysis Walkthrough


Objectives ..... C-1
Fundamental Tuning Guidelines ..... C-2
SQL Server Problem Analysis ..... C-3
Sample Problem ..... C-4
Collect Data ..... C-5
Consider Your Options ..... C-6
Is Denormalization Appropriate? ..... C-7
How Should Objects Be Placed? ..... C-8
What Indexes Should Be In Place? ..... C-9
Is Memory Being Used Effectively? ..... C-10
Is The CPU Being Used Effectively? ..... C-11
Are There Any Concurrency Issues? ..... C-12
Lab C – Introduction to Problem Analysis ..... C-13



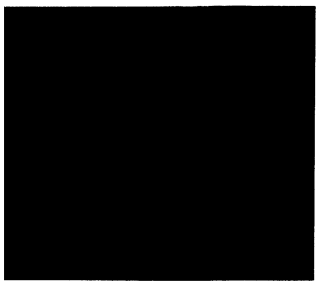
# Course Overview

## System 10 Performance & Tuning

Version 2.2




*The Enterprise Client/Server Company™*



Course Overview

# Course Objectives

- Design systems so that performance is built in up front, and so that the framework is established for successful lifecycle performance tuning
- Evaluate the performance of a system
- Analyze and solve performance problems using a systematic approach
- Minimize the performance impact of maintenance tasks
- Write queries which work effectively with the optimizer

**Prerequisites**

Experience developing applications for SQL Server or administering a SQL Server, specifically:

- Understanding of relation database design concepts
- Understanding of Sybase client/server architecture
- Proficiency in Transact-SQL
- Equivalence of four months experience administering a SQL Server (particularly for how SQL Server manages disk devices)

# Course Roadmap

## Part 1
## Designing Applications for Performance

| | |
|---|---|
| Module 1 | Overview of Performance Issues |
| Module 2 | Physical Database Design and Denormalization |
| Module 3 | Table Storage |
| Module 4 | How Indexes Work |
| Module 5 | Selecting Indexes for Performance |
| Module 6 | Query Optimization |

## Part 2
## Tuning Applications for Performance

| | |
|---|---|
| Module 7 | Distributing Data Across Devices |
| Module 8 | How Locking Affects Performance |
| Module 9 | Managing Memory |
| Module 10 | Maintaining High Performance |
| Module 11 | Guidelines for High-Performance Applications |

## Part 3
## Special Topics

| | |
|---|---|
| Module 12 | Networks and Performance |
| Module 13 | tempdb |
| Module 14 | Cursors and Performance |
| Module 15 | CPU Utilization Issues |
| Appendix A | Distributing Data with Replication Server |
| Appendix B | More About SQL Monitor |
| Appendix C | Problem Analysis Walkthrough |

# Part 1
# Designing Applications for Performance

System 10 Performance & Tuning

Version 2.2


*The Enterprise Client/Server Company™*

1

# Overview of Performance Issues

System 10 Performance & Tuning

Version 2.2



SYBASE®

*The Enterprise Client/Server Company™*

# 1

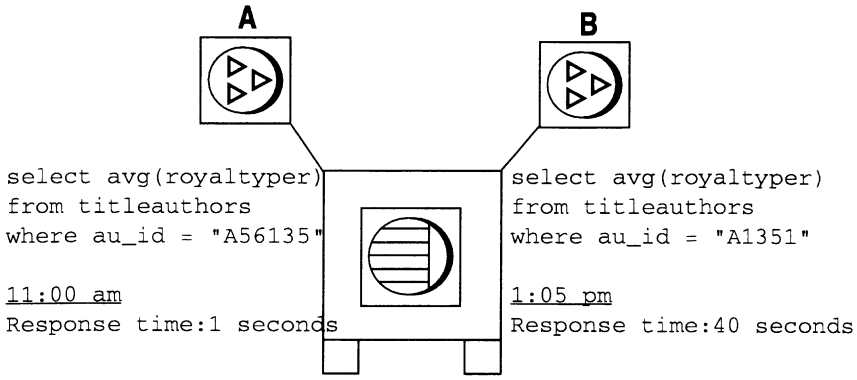
# Objectives

- Identify major issues and trade-offs for performance
  - networking
  - hardware and software architecture
  - database size and structure
  - transaction patterns
- Describe the case studies to be used throughout the class
- Introduce a problem analysis approach

**Topics**

- What is performance?
- How do we measure performance?
- Designing vs. tuning for performance
- Performance in the Client/Server model
  - architecture
  - issues
  - trade-offs
- Case studies
- Problem analysis approach

# What is Performance?




- What is happening here?
- Is performance acceptable?
- Where do you start?

# What is Performance?

Performance is the measure of efficiency of

- A computerized business application, or
- Multiple applications running in the same environment

Performance is usually measured as:

- Response time
- Throughput
- Performance & Tuning concerns *all* aspects of performance

**Response Time**

- Time it takes for a single task to complete
- Is affected by:
  - reducing contention and wait times
  - using faster components
  - reducing the amount of time the resources are needed

**Throughput**

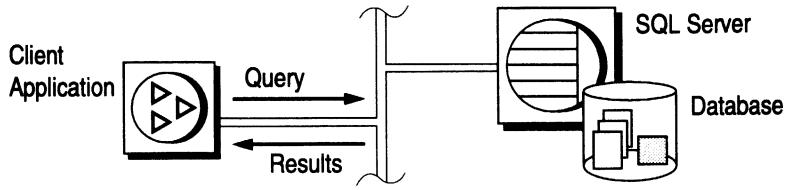
- Volume of work completed in a fixed time period
- There are two ways of thinking of throughput:
  - for a single transaction
    (for example, 5 UpdateTitle transactions per minute)
  - for the entire SQL Server
    (for example, 5 Server-wide transactions per minute).
- Is commonly measured in transactions per second (tps), but can be measured per minute, per hour, per day, etc.

# Other Considerations for Performance

- Cost/benefit (response and throughput vs. system cost)
- Ability to interface efficiently with legacy systems
- Maintenance
- Security administration
- Acceptance in the business environment
- And others

# Designing for Performance Is:

- Estimating the performance impact of various design options
  - Hardware/network design
  - Application design
  - Database design
  - Query design




- Considering these options early in the design process

## Tuning For Performance Is:

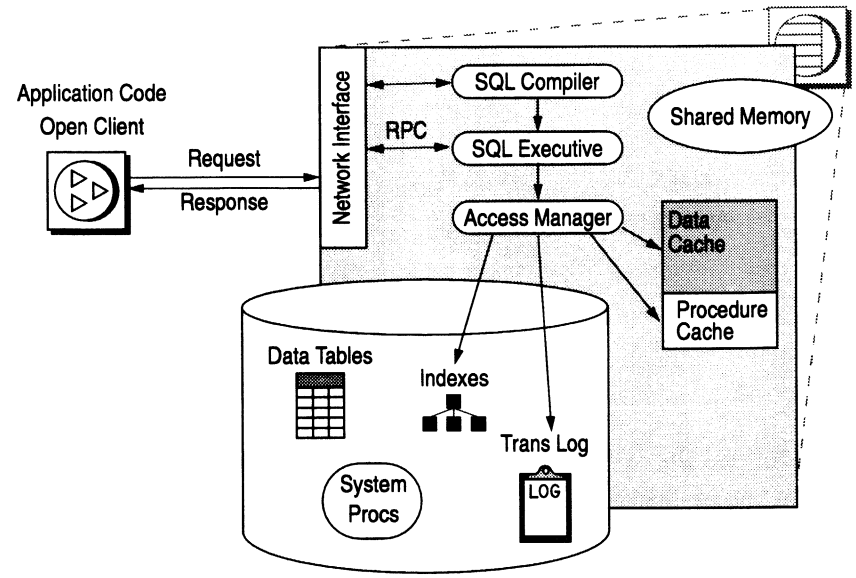
- Optimize the system to satisfy as many users as possible
- Identifying and analyzing critical transactions (shown here in gray)

| Transaction | User | Frequency | Required Response |
|---|---|---|---|
| Sales by Store | president | 5 per day per store | <3 seconds |
| Lookup Query | sales staff | 20,000 per day | <3 seconds |
| Sales Report | mailroom | weekly | <30 minutes |
| Inventory Level | manager | 10 per day | <2 minutes |

**What is a Database Transaction?**

- A logical unit of work
- One or more Transact-SQL statements, all of which must complete, or none of which get reflected in the database
- The smallest unit of recovery

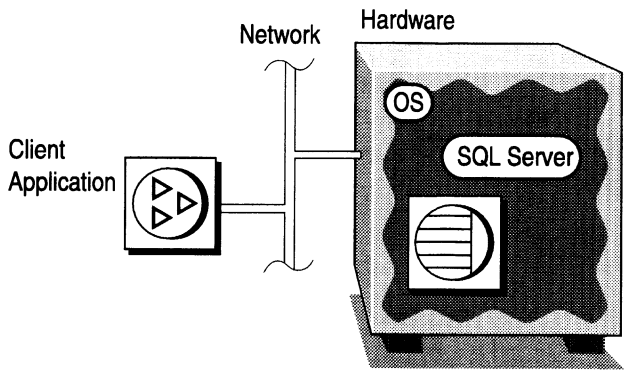
# SQL Server Model of Operations




**Model of Operations**

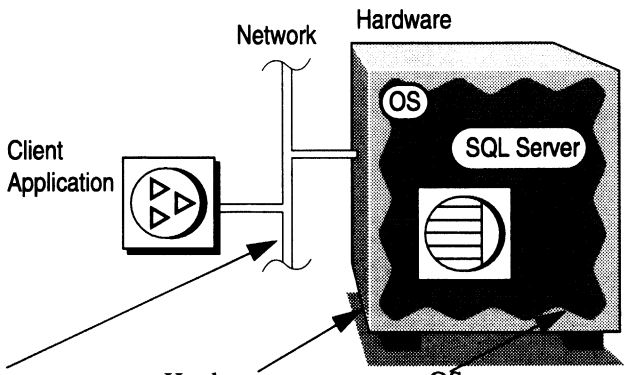
- Shows an overview of the steps SQL Server goes through in processing requests
  - language requests (Transact-SQL)
  - remote procedure calls (RPCs)
- One large disk represents all the disks that may be used by all the databases, including **master**

# Client/Server System Model




- Defines the layers of the client/server architecture
- Address performance and tuning at each layer

## Network/Hardware/OS Layer Issues




**Network**
- Network speed
- Network traffic
- Possible bottlenecks

**Hardware**
- CPU(s)
- Disks
- Memory
- Synch/asynch disks

**OS**
- Task priorities
- Memory manage
- File system management
- Network interface (connections)

**Network Options**
- Client vs server processing
- Network architecture
- Line communications speeds
- Packet sizes

**Hardware Options**
- Memory
- Slower/faster CPUs
- Multiple CPUs
- Multiple controllers
- Disk arrays (e.g., RAID)

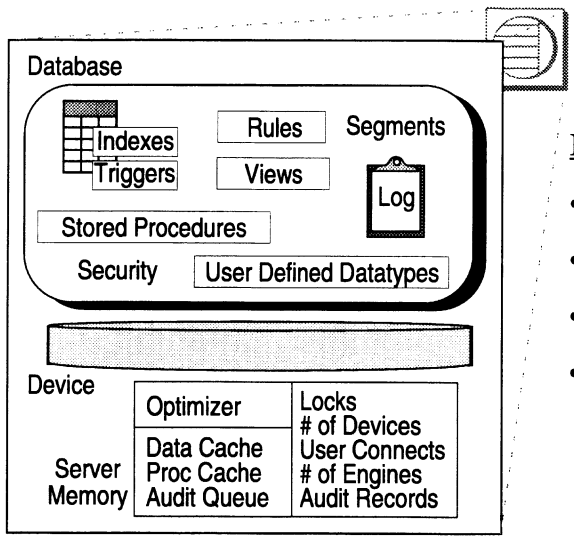
CPU > 85 %

SQL-server nach andere machine !

**Operating System Options**
- Files vs. raw partitions
- Multiple CPU utilization
- Network interface
- O/S memory size

Overview of Performance Issues

# SQL Server Layer



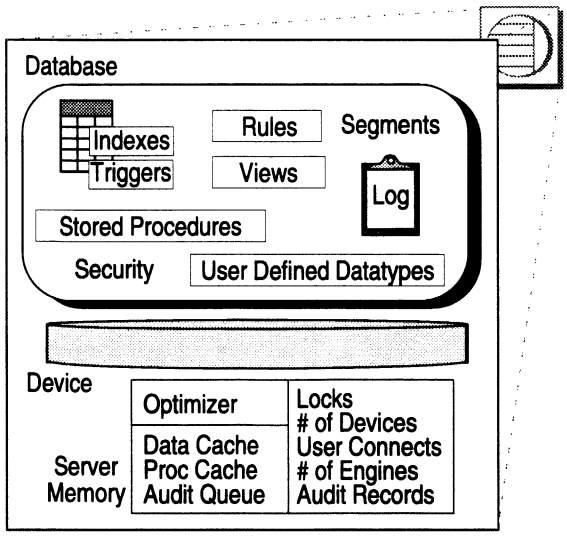

Issues

- Number of users
- Auditing
- OLTP vs. DSS
- Replication

**Options**

- Cache sizes
- Kernel/server structures
- Other configurable options
- Virtual Server architecture

# Devices




Maps databases to physical disk access

Issues

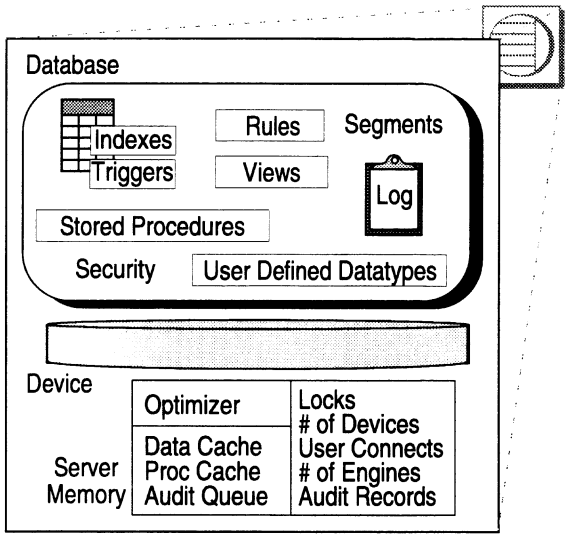
- Disks/controllers
- Fault tolerance
- File systems
- Disk array usage

**Options**

- Raw partitions vs. files
- Object placement
- Mirroring
- Disk arrays (for example, RAID)

Overview of Performance Issues

# Database




Issues

- Overall activity
- Auditing
- Backup/recover strategy
- Distributed data
- Volatility

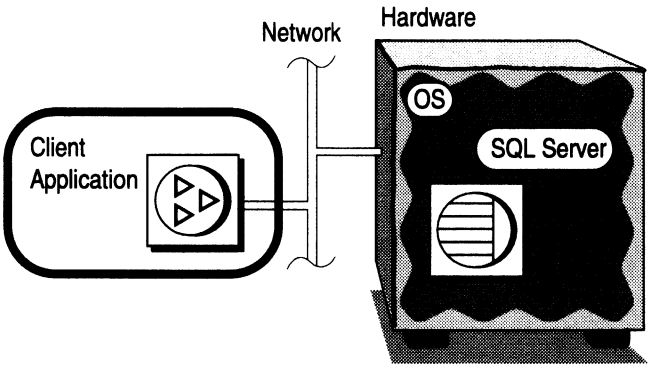
**Options**

- Object placement
- Transaction log
- Auditing
- Remote servers

**Database Contains Elements**

- Tables
- Indexes
- Triggers
- Stored Procedures
- Transaction Logs

# Application Layer




- Issues
  - Client vs.Server processing
  - Concurrency and locking
  - Decision support vs. OLTP
  - Transaction design
  - Referential integrity

**Options**

- Stored procs, views, triggers
- Rules, defaults, constraints
- Auditing & security
- Replicated processing
- Remote processing

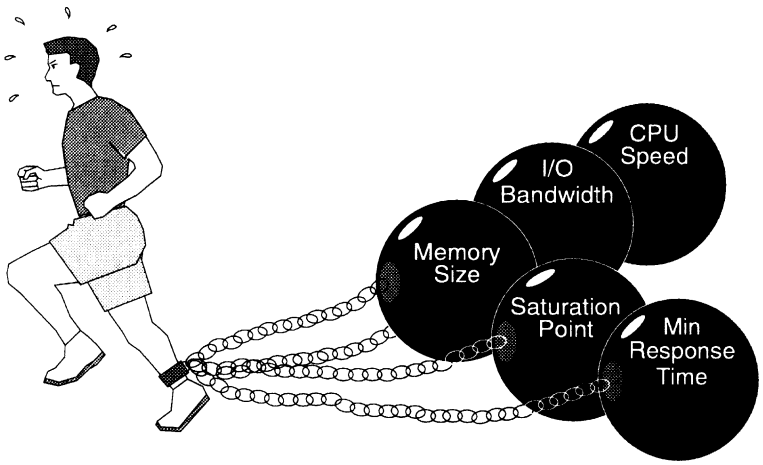
**OLTP vs. DSS**

OLTP refers to on-line transaction processing, where real-time inserts, deletes, and updates are critical.

DSS refers to decision support, which involves generating reports on the data in the database.

# System Limitations

- Limit maximum performance




**Saturation Point** — There is a point at which the system simply can't handle any more processing volume.

**Min Response Time** — For a given task, there is always a minimum amount of time it takes to perform.

**I/O Bandwidth** — Data transfer speed is limited by the capacity of the data bus.

**Memory Size** — If memory is full, pages have to be swapped out and read in again the next time they are used. This affects performance.

**CPU Speed** — The number of operations a computer can perform is strictly limited by the CPU speed.

**More Resources** — Buying resources is one way to improve performance. Sometimes it is cheaper to buy resources than to solve complex performance problems. This does not work for all performance problems, however.

# Fundamental Design & Tuning Guidelines

- Be sure design and configuration make full use of all existing resources
  - Tune queries and set configuration variables
- Increase concurrency by decreasing contention for resources
  - Especially: Reduce and distribute disk I/O
- Identify weak links (resource bottlenecks)
  - Trade-off the use of other resources to ease bottlenecks

**Reduce Contention**

If processes have to wait for resources to become free, then performance will be limited by the service time of the slowest component.

**Reduce Disk I/O**

Reducing disk I/O can improve performance dramatically.

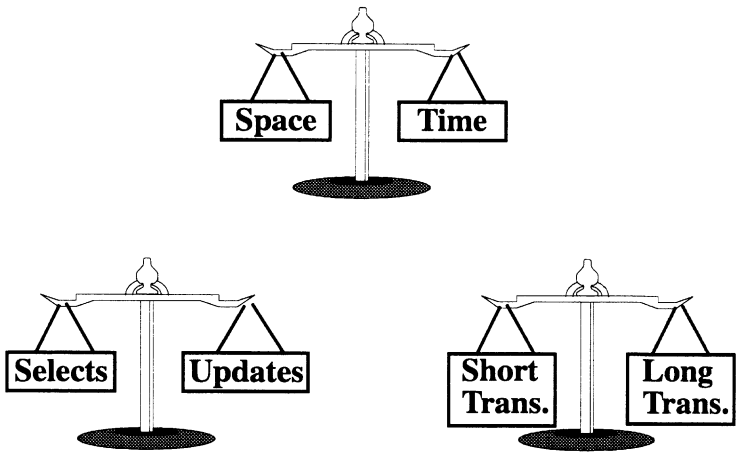
**Increase Concurrency**

The more processing that can be performed in parallel, the better performance will be.

This is the principle strategy behind symmetric multi-processors, database systems, multi-user operating systems, and RISC processors.

# Trade-offs

- Performance design and tuning involves trade-offs




**Space vs. Time**

The trade-off between space and time is a classic computer system issue.

For example, indexes take up space, but you can get to your data faster if you have the right ones.

**Selects vs. Updates**

The trade-off between selects (read only) and updates (read and/or write) becomes an important issue when determining the proper data structures for an application.

For example, indexes may speed up selects but slow down updates.

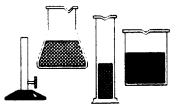
**Short vs. Long Transactions**

Multi-user database systems make use of various mechanisms to prevent users from interfering with each other as they compete for the same resources.

Such mechanisms are most appropriately selected based on how long each of the critical transactions in the system make use of resources.

# SQL Server Problem Analysis Approach

1. Collect data
   - identify the performance problem
   - define performance requirements
   - measure the current performance
   - define SQL Server environment
   - analyze application design
2. Formulate hypothesis
3. Test hypothesis
4. If confirmed, implement;
   otherwise, collect more data or formulate another hypothesis



**Identify the problem**
- Determine the symptoms.
- Determine what components of the system model are affected.

**Define the performance requirements**
- Processing need.
- Frequency of execution.
- Response time required.

**Measure the current performance**
- Get baseline measurements using the appropriate tool given the component of the system that is affected.

**Define the SQL Server environment**
- Configuration at all layers.
- Limitations at all layers.
- Analyze application design.

**Analyze application design**
- Look at tables, indexes, transactions that comprise the application design.

**Formulate a hypothesis**
- Using the data, determine the problem and possible solution.

# Lab 1 Performance Overview

- Review the concepts presented in the module
- Identify performance issues at each layer of the System Model

# Case Study Introduction

- An application for a book distribution business
- Integrated database holds all distribution information
- Three major business processes:
  - Order Entry
  - Shipping
  - Sales Management

**Introducing the Case Study**

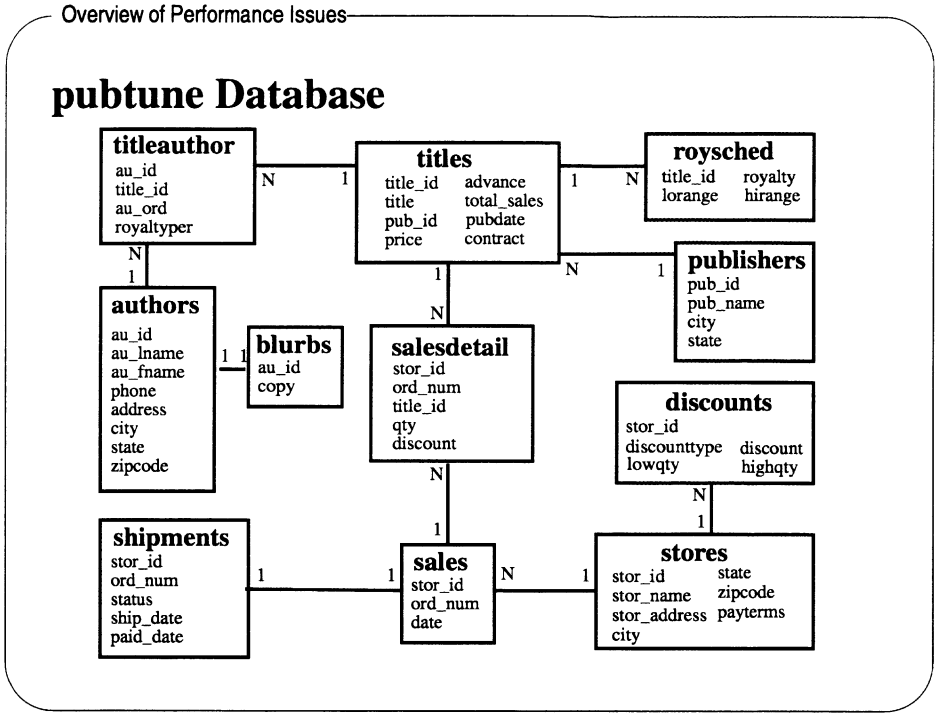
Throughout the course the examples and labs will be drawn from the operations of a fictitious publishing company. In this module, we will introduce the three major operations performed by this company, and then see how to use certain tools to investigate a performance problem.

**Company Background**

- Founded in 1985
- Over 300 stores across the country
- Gross revenues approximately $235M

**Database Users**

- CEO
- Billing Department
- Shipping Department
- Sales Manager
- Sales Clerk
- DBA




**pubtune**

This database is based on the *sample pubs2* database shipped with the Sybase product. We have made a few changes:

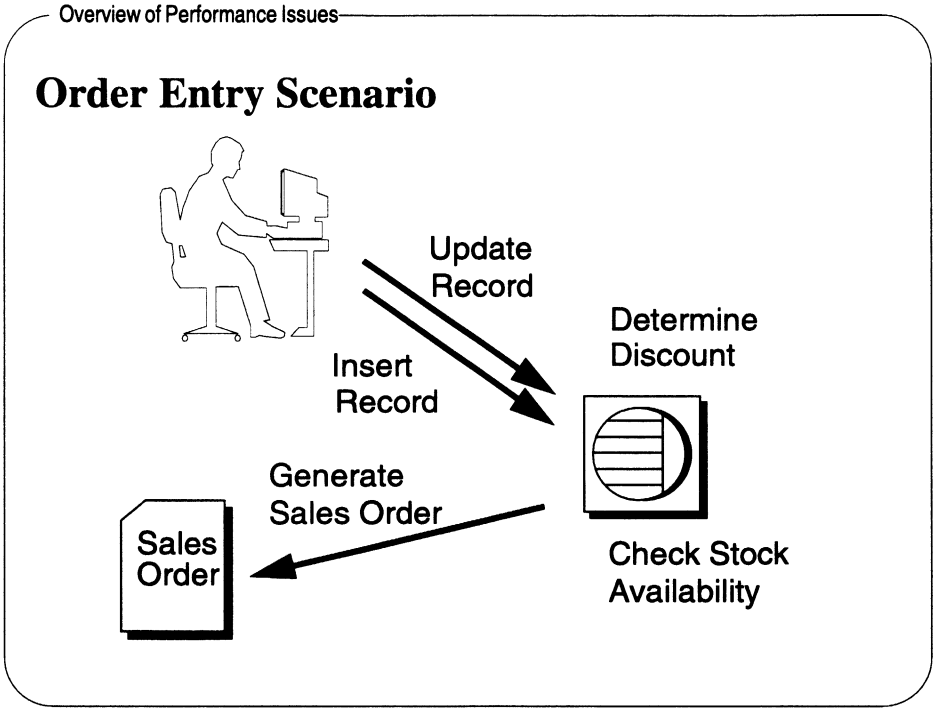
- No indexes
- Substantially more data than *pubs2*
- The *shipments* table has been added
- The *au_pix* table has been deleted

**Table variants**

To demonstrate certain concepts, we have created a number of variants on these tables. Here are some examples:

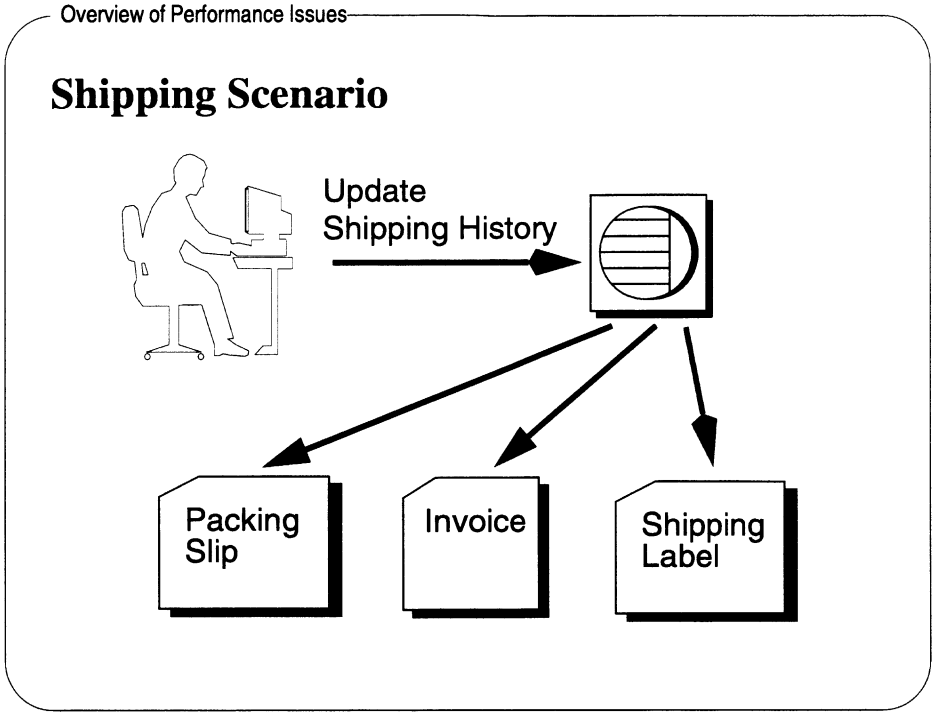
- *titles_idpr* has a clustered index on title_id and a non-clustered index on price
- *titleauthor_idid* has a clustered index on au_id and title_id
- *authors_idstate* has a non-clustered index on au_id and state
- etc.

A complete list may be found in the lab book.



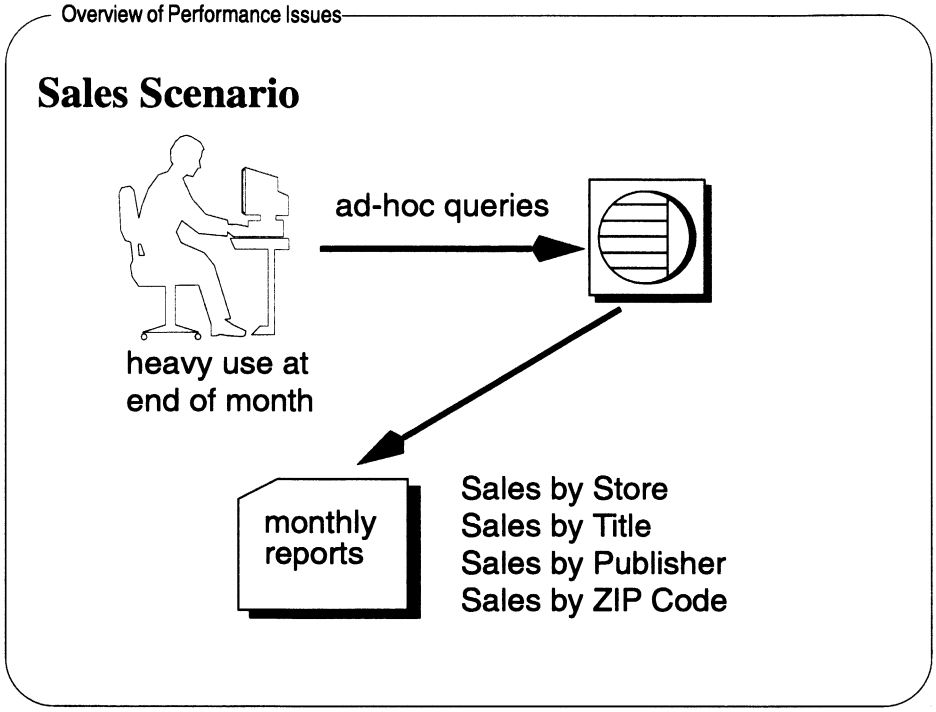

**Order Entry Scenario**

- Primarily inserts and updates
- Uses lookup tables for consistency
- Works from 8am-8pm, 365 days per year
- Tasks:
  - Enter sales order
  - Check stock availability
  - Determine discount
  - Correct sales order
  - Cancel sales order




**Shipping Scenario**

- Workload-driven
- Works from 9am-5pm, Monday—Friday
- Tasks:
  - Generate packing slip
  - Generate invoice
  - Generate shipping label
  - Update shipping history




**Sales Scenario**

- Works from 10am-3pm, Mon-Thur
- Primarily report generation
- Reports generated at close of every month
- Reports describe sales by store, title, publisher, and zip code
- Many ad-hoc queries during month

# Summary

- Performance is measured by response time and throughput
- To improve performance:
  - Verify design is good
  - Tune individual queries
  - Increase concurrency by balancing resources
  - Configure server optimally
  - Especially: Reduce & distribute disk I/O
- Address performance problems at each layer of the system model
- There are usually trade-offs
- CASE
- Problem Analysis Approach

For an overview of database performance issues, refer to *Database Tuning: A Principled Approach,* Dennis E. Shasha, Prentice Hall, 1992, ISBN 0-13-205246-6.

2

# Physical Database Design and Denormalization

System 10 Performance & Tuning

Version 2.2




*The Enterprise Client/Server Company™*

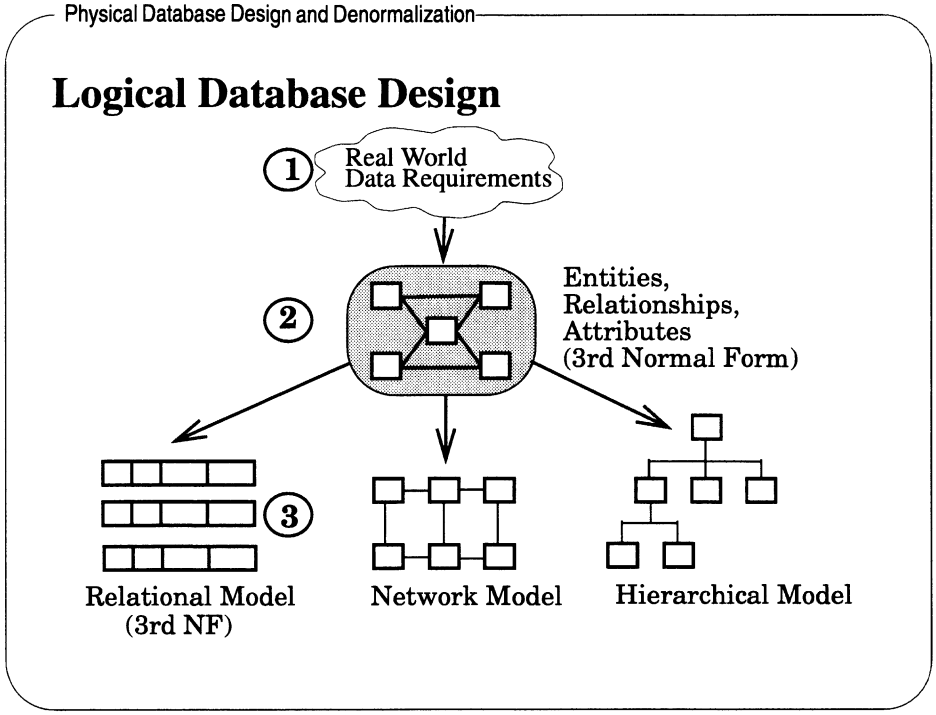
# 2

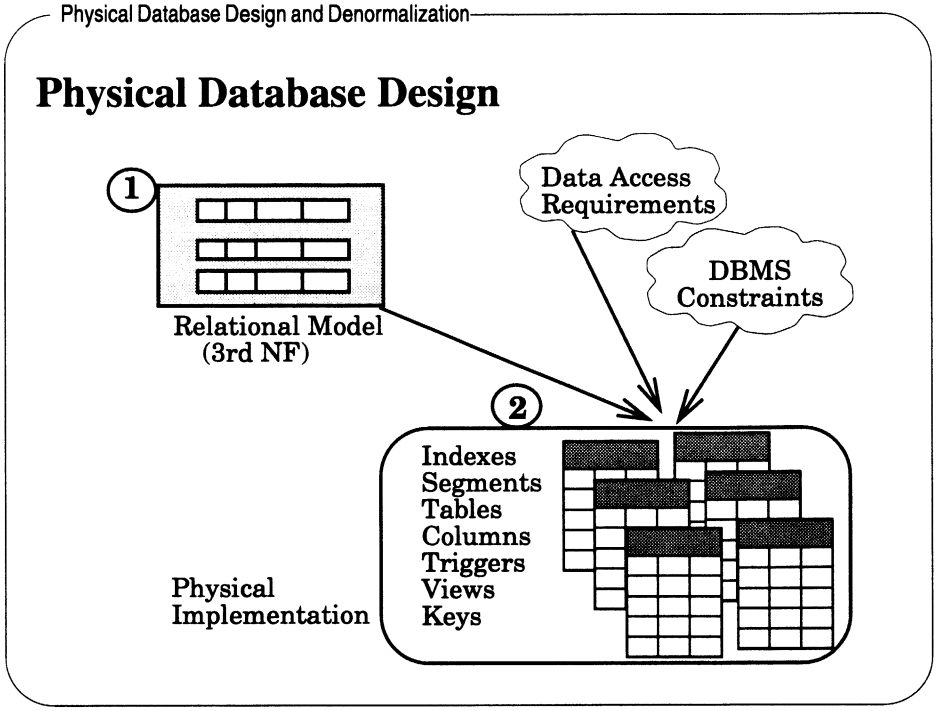
# Objectives

- Identify aspects of the physical design that have a significant impact on overall performance
- Explain how normalization affects database performance
- Develop baseline measurements
- Use denormalization techniques to improve performance
- Describe costs of denormalization
- Analyze resulting performance changes

**Topics**

- Logical & physical database design
- Normalization: what it is, and why it is done
- Denormalization: what it is, and why it is done


Logical Database Design
1
Real World
Data Requirements
2
Entities,
Relationships,
Attributes
(3rd Normal Form)
3
Relational Model
(3rd NF)
Network Model
Hierarchical Model

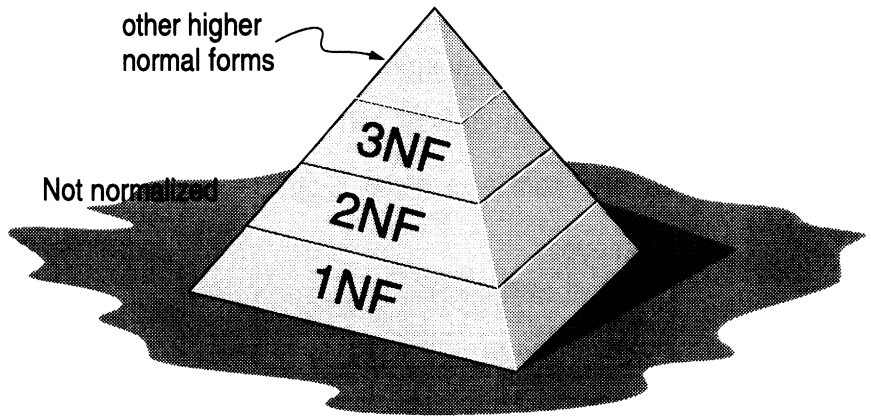

Physical Database Design
1
Relational Model
(3rd NF)
Data Access
Requirements
DBMS
Constraints
2
Indexes
Segments
Tables
Columns
Triggers
Views
Keys
Physical
Implementation

# From Logical to Physical Database Design

- Translate Entity-Relationship model (3NF) to relational model
  - relations become tables
  - attributes become columns
  - relationships become data references (PK/FK Refs)
- Based on access requirements and constraints, implement physical database design
  - denormalize where appropriate
  - partition tables where appropriate
  - group tables into databases where appropriate
  - determine use of segments
  - determine use of devices
  - implement referential integrity of constraints

# Normalization

When a table is normalized, the non-key columns in a table depend on the key, the whole key and nothing but the key




**Physical Database Design**

From a relational model point of view, it is common to have tables that are in 3rd normal form. This may not always yield the best performance, however.

**Levels of Normalization**

The next level of normalization always relies on the previous level. To conform to 2NF, you must be in 1NF, etc.

When determining if a database is in a normal form, we start with the assumption that the relation (or table) is not normalized. Then we apply the rigor of each normal form level to it.

# First Normal Form

- Rules:
  - Every column must be atomic (cannot be decomposed into two or more subcolumns)
  - You cannot have multi-valued columns (repeating groups)
  - Every row and column position can have only one value
- Example of a table that violates 1NF

  Employee (emp_num, emp_name, dept)

**Employee**

| emp_num | emp_lname | dept |
|---|---|---|
| 10052 | Jones | A10 C66 |
| 10101 | Sims | D60 |

repeating group

- Example of tables that conform to 1NF

  Employee (emp_num, emp_lname)
  Emp_dept (emp_num, dept)

**Employee**

| emp_num | emp_lname |
|---|---|
| 10052 | Jones |
| 10101 | Sims |

**Emp_dept**

| emp_num | dept |
|---|---|
| 10052 | A10 |
| 10052 | C66 |
| 10101 | D60 |

- Atomicity of a column depends on application (logical design) (emp_name) vs. (emp_fname, emp_mi, emp_lname)

## Second Normal Form

- Rule: Every non-key field must depend on the entire primary key, never on part of a composite primary key
- Violates 2NF:

```
Emp_dept (emp_num, dept, dept_name)
          \_____PK_____/  <-----------
```

- Conforms to 2NF

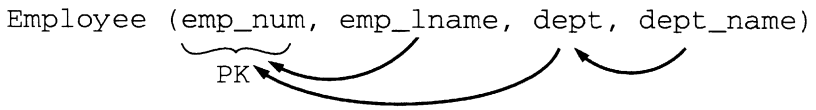
```
Emp_dept (emp_num, dept)
          \____PK____/

Dept (dept, dept_name)
      \PK/  <--------
```

**Second Normal Form**

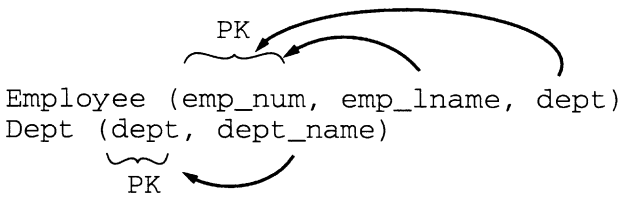
In tables that conform to Second Normal Form, there are key and non-key fields and all the non-key fields have to depend on the entire primary key and not just a portion of it. If a database has only single-field primary keys, it is automatically in second normal form.

# Third Normal Form

- Rule: A non-key field shall not depend on another non-key field
- Violates 3NF:




- Conforms to 3NF




**Third Normal Form** In tables that conform to Third Normal Form, no non-key attributes depend on other non-key attributes.

# Why Be Normal?

- Minimize data redundancy
- Reduce insert, update and delete anomalies
- Conceptually "cleaner"

# Performance Benefits of Normalization

- Reduced Redundancy
  - less data
  - less data = more efficient I/O
  - updates are faster
- Smaller Tables, Smaller Rows
  - more rows per page (less logical I/O)
  - more rows per I/O (more efficient)
  - more rows fit in cache (less physical I/O)

**Full normalization may require more joins, and these can be costly**

## Denormalization

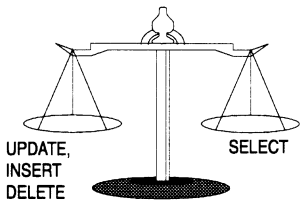
- Intentionally backing away from normalization to improve performance
- Can be done with tables or columns
- Assumes prior normalization
- Requires a knowledge of how the data is being used

**Denormalization Has Risks**

- May introduce data integrity problems
- Will usually favor some processing, at a cost to others
- Can be done only with knowledge of application

## Performance Benefits of Denormalization

- Minimizes the need for joins
- Reduces the number of foreign keys
- Reduces the number of indexes
- May reduce the number of tables
- Aggregates can be precomputed




LOW NUMBER of UPDATES +
LARGE NUMBER of QUERIES =

**DENORMALIZATION**

**Disadvantages**

- Denormalization speeds retrieval but slows modifications to the database
- Denormalization is always application specific (needs to be re-evaluated if the application changes)
- In some instances, denormalization simplifies coding; in others, it makes it more complex

# Denormalization Input

- What are the data access requirements?
  - What are the critical transactions, and what is the expected response time?
  - What tables or columns do they use? How many rows per access?
  - What is the mix of transaction types: select, insert, delete?
  - What is usual sort order?
  - How often are the transactions executed?
  - What are concurrency expectations?
- How big are the most frequently accessed tables?
- Do any processes compute summaries?
- Where is the data physically located?

# Denormalization Techniques

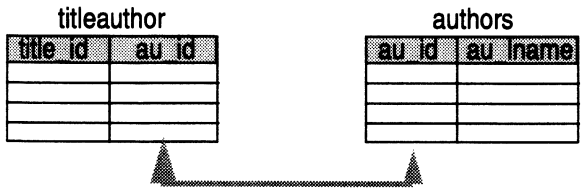
- Adding redundant columns
- Adding derived columns
- Collapsing tables
- Duplicating tables
- Splitting tables

**Denormalization Techniques**

Note that duplicating and splitting tables are not, strictly speaking, denormalization techniques. We include them here because they are done for similar purposes.

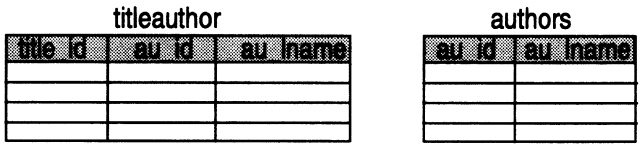
# Adding Redundant Columns

- Problem: Frequent join to access **au_lname** data




```
select ta.title_id, a.au_id,
   a.au_lname
from titleauthor ta, authors a
where ta.au_id = a.au_id
```

- Solution: Duplicate **au_lname** data in **titleauthor**

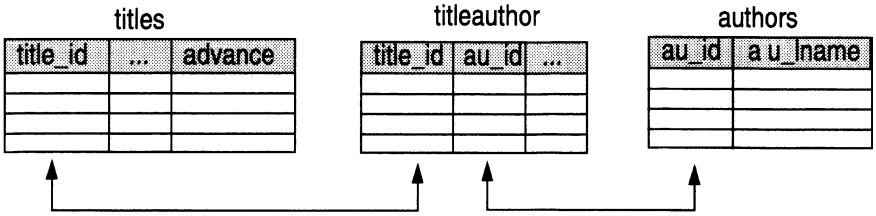



```
select title_id, au_id,
   au_lname
from titleauthor
```

- Advantage: eliminates join for many queries
- Disadvantages:
  - Requires maintenance of new column (all changes must be made twice)
  - Requires more disk space (**au_lname** is duplicated)

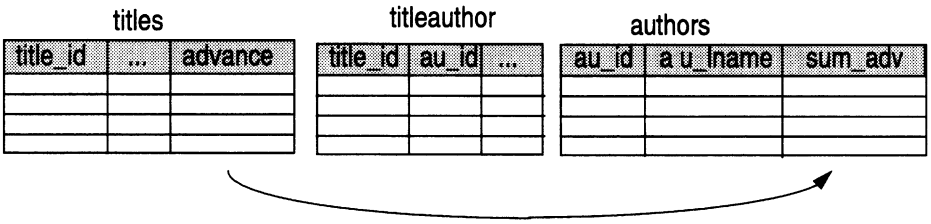
# Adding Derived Columns

- Problem: Very frequent join to get sum of advances by author




```
select a.au_lname, sum(advance)
from titles t, titleauthor ta, authors a
where t.title_id = ta.title_id
and ta.au_id = a.au_id
```

- Solution: Create and maintain derived data column in authors

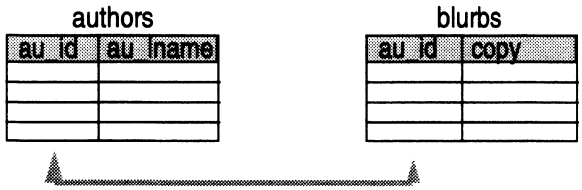



```
select au_lname,sum_adv
from authors
```

- Advantages: eliminates join, eliminates aggregate at run time
- Disadvantages
  - Increased storage
  - Requires maintenance of derived column
- How might you keep the derived column up to date?

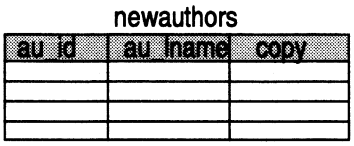
# Collapsing Tables

- Problem:
  Frequently want to query columns from multiple tables




```
select a.au_id, a.au_lname,
   b.copy
from authors a, blurbs b
where a.au_id = b.au_id
```

- Solution: Collapse the tables

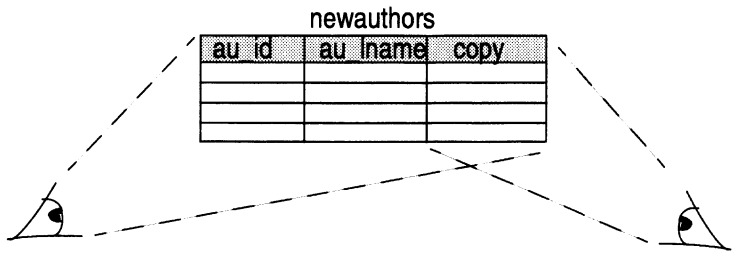



```
select * from newauthors
```

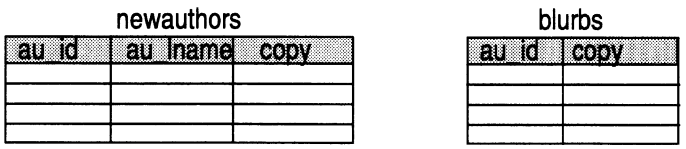
- Advantage: eliminates join
- Disadvantage: Lose conceptual separation of data
  - Could be restored with views
- Note: must be one-to-one relationship!

# Duplicating Tables

- Problem: Group of users regularly need a subset of data

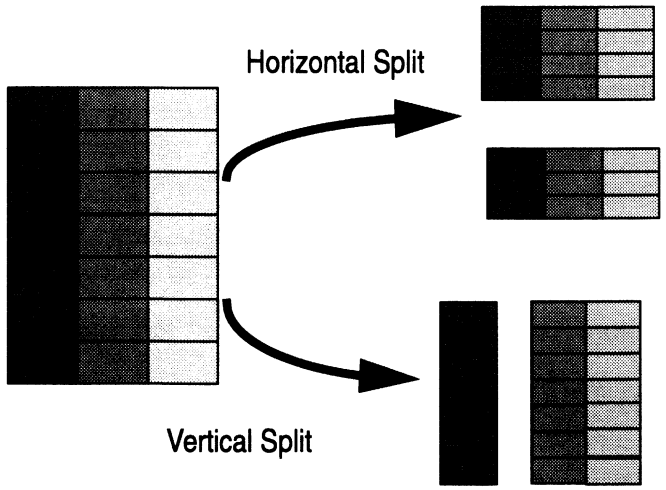



- Solution: Duplicate critical table subset for that group




- Advantages:
  - minimizes contention
  - subsets data for specific users
- Disadvantages: must manage redundancy (and possible latency)

# Splitting Tables




**Use Horizontal Splitting When**

- A table is large and reducing its size reduces the number of index pages read in a query
- The table split corresponds to a natural separation of the rows, such as different geographical sites, or historical vs. current data
- Table splitting distributes data over the physical media

**Use Vertical Splitting When**

- Some columns are accessed more frequently than other columns
- The table has wide rows, and splitting the table reduces the number of pages that need to be read

# Horizontal Partition Example

**Problem:** Usually only "active" records are accessed

| Authors | | | |
|---|---|---|---|
| active | | | |
| active | | | |
| inactive | | | |
| active | | | |
| inactive | | | |
| inactive | | | |

**Solution:** Partition horizontally into "active" and "inactive" data

| Inactive_Authors | | |
|---|---|---|
| | | |
| | | |
| | | |

| Active_Authors | | |
|---|---|---|
| | | |
| | | |
| | | |

# Vertical Partition Example

**Problem:**
Frequently access **lname** and **fname,**
infrequently access **phone** and **city**

| Authors | | | | |
|---|---|---|---|---|
| **au_id** | **lname** | **fname** | **phone** | **city** |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

**Solution:** Partition data vertically

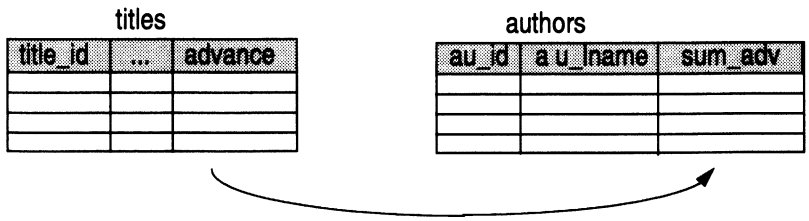
| Authors_Frequent | | |
|---|---|---|
| **au_id** | **lname** | **fname** |
| | | |
| | | |
| | | |
| | | |

| Authors_Infrequent | | |
|---|---|---|
| **au_id** | **phone** | **city** |
| | | |
| | | |
| | | |
| | | |

# Managing Denormalized Data

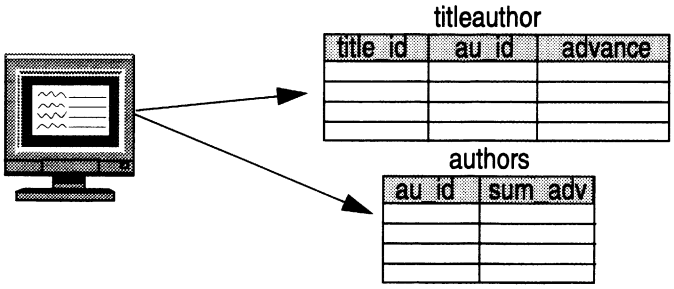
- Triggers
- Application logic
- Batch reconciliation

# Using Triggers to Manage Denormalized Data




- If you have a trigger on titles to maintain the sum_adv column in authors, what should it do
  - for inserts into title?
  - for deletes from title?
  - for updates to the advance column?

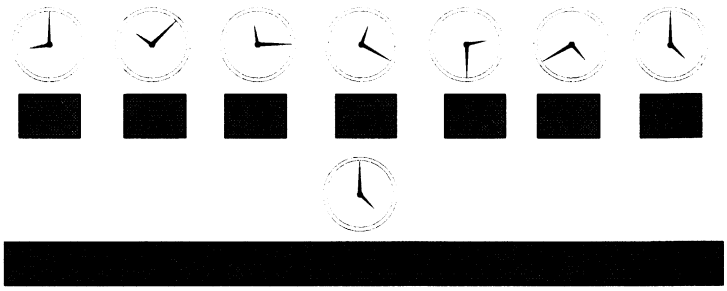
# Using Application Logic to Manage Denormalized Data




- If your application has to ensure data integrity, it will have to ensure that the inserts, deletions, or updates to both tables occur in a single transaction
- Which is better? Trigger or Application Logic?

# Batch Reconciliation

- If 100% consistency is not required at all times, a batch job (stored procedure) run during off hours could reconcile duplicate or derived data
- Could do short batches or. long batches



**Batch Reconciliation**

If you add redundant or derived columns or duplicate data, you will have to maintain these columns to be sure they are up to date.

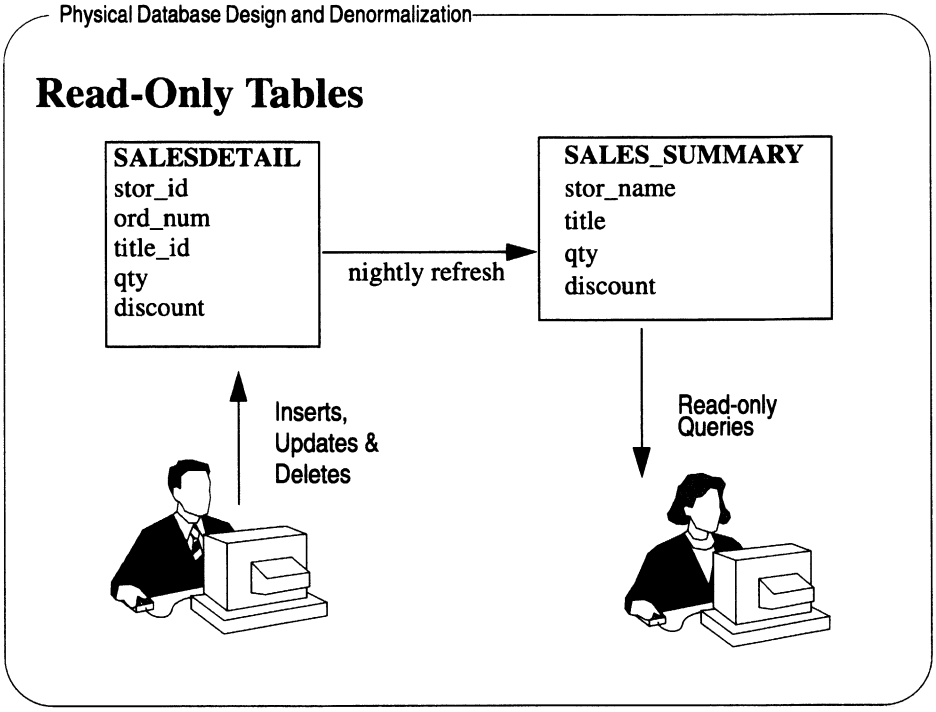
If it is not essential that the duplicate data be completely consistent with the source data, you could run a stored procedure during off-hours that checks referential integrity and updates if necessary.

# Sample Problem #1

- Problem:

  The Sales and Order Entry departments are both complaining that response time is too slow. SalesByStore and PlaceOrder are particularly problematic.

- What do you suggest?
- Hints:
  - PlaceOrder is mostly inserts and updates, and SalesByStore is read only
  - Both processes access salesdetail, sales, and stores




**Duplicating Data**

In this instance, a `sales_summary` table has been created so that the sales department can produce their sales reports without interfering with the order entry processing. The `sales_summary` table is refreshed every night from the `sales_detail` table.

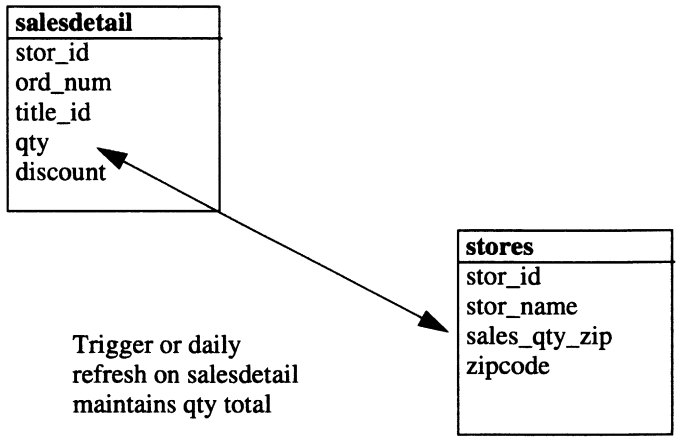
# Sample Problem #2

- Problem

  The Sales Department generates numerous reports at the end of every month. The sales clerk complains that SalesByZip, in particular, runs very slowly.

- What do you suggest?
- Hints
  - SalesByZip accesses salesdetail, sales, and stores
  - Sum transactions generally lock entire tables

# Storing Calculated Values




**Tradeoffs**

In this example, a calculated column has been created which stores the total sales by zip code.

In most systems, a sum query would be a performance problem.

In this example, the total sales are maintained with triggers.

This trades speed of data modification for speed of retrieval.

# Evaluating Benefits of Denormalization

- Measure baseline physical/logical I/O using
  `set statistics io on`
- Then, denormalize the tables
- Finally, use `showplan` and `set statistics io on` again to see if table access has changed

**Example**

```
Set showplan on
go
select * from authors
go
STEP 1
The type of query is SELECT.
FROM TABLE
authors
Nested iteration
Table Scan
```

# How to Group Tables

**(from Sybase's Database Point of View)**

- Put large tables into a database by themselves
- Put small related tables together into their own database
- Keep tables with referential integrity dependencies together in their own database
- Keep volatile data away from static data (separated into multiple databases)

# Lab 2 - Physical Database Design

- Define table access requirements for a series of transactions
- Identify potential interference if these transactions are run concurrently
- Record I/O statistics for base tables
- Record I/O statistics for denormalized tables and compare

# Summary

- Logical database design
- Physical database design
- Denormalization
  - Adding redundant columns
  - Adding derived columns
  - Collapsing tables
  - Duplicating tables
  - Splitting tables
- Tools
  - showplan
  - statistics io, statistics time

**Note** If you denormalize, document why!

3

# Table Storage

## System 10 Performance & Tuning

Version 2.2



SYBASE®

*The Enterprise Client/Server Company™*

# 3

Table Storage

# Objectives

- Identify the mechanisms of data storage for tables
- Learn how table size and other attributes affect performance

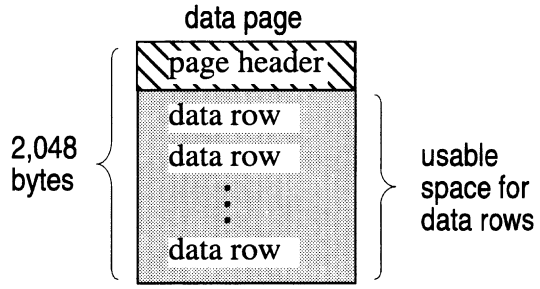
**Topics**

- Basic concepts of data storage
- Non-clustered tables

Table Storage

# Data Pages

- The basic unit of storage for SQL Server is a page:




- Rows cannot cross page boundaries (except for text and image columns)
- Each data row has at least 4 bytes of overhead, more for varchars

**Available Bytes on a Data Page**

For calculations in this module, we will be using 2,016 bytes for the available bytes in a data page.

The total amount of space made available for rows and their overhead is 2,048 minus 32 bytes, or 2,016 bytes.

Note that 2,016 is not the maximum length of a row; due to a limitation of the maximum size of a log record the maximum length of a row is 1,962 bytes. The overhead for each row in the transaction log is 54 bytes..

**Page Header**

- Page header contains some of the overhead bytes to maintain the layout of rows on the page, such as page linkage, and object and allocation information.
- There is additional overhead in the row offset table at the end of the page.

**Data Rows**

Every row on the page lives in one contiguous space.

Each row consists of the actual column data plus row details such as the row number (one byte) and the number of variable-length and null columns contained in the row (one byte).

# Linked Data Pages

- Data pages are linked to form a table
- Data pages are allocated to a table by the server in contiguous sets of 8 pages called **extents**

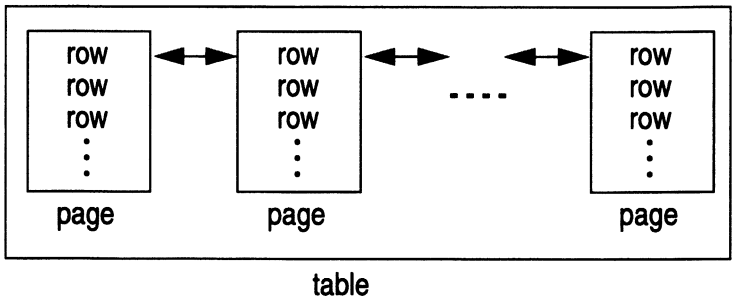
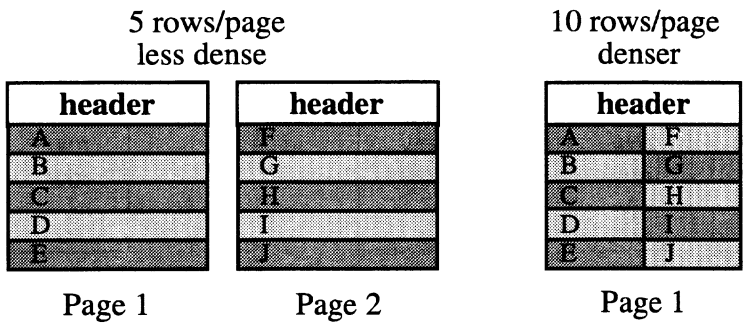


Table Storage

## Row Density

- The more rows per data page, the less total I/O for queries that ask for multiple rows




- Page size cannot be controlled, but row size can

**Row Density**

Row size can affect performance dramatically. The more rows per page, and the more pages in memory, the better the database performance.

**Actual versus Defined**

The size of a row depends on the actual size of any of its variable length field.

Consequently, row density may vary from one page to another.

When trying to predict row density accurately, it is important to estimate the average row size well, given realistic data.

**Rows per Page**

Usable space / row size.

# Other Types of Pages

- There are six other types of pages in addition to data pages:

| | |
|---|---|
| **Index Pages** | Contain index rows and pointers |
| **Distribution Pages** | Contain information about the distribution of data values in a table |
| **Overflow Pages** | Special type of clustered index pages |
| **Text/Image Pages** | Contain text or image data |
| **OAM Pages** | Contain information about object allocation |
| **GAM Pages** | Global allocation bitmaps for a database |

- We will study all pages above except allocation pages

**Overflow Pages** Overflow pages are data pages only for non-unique clustered indexes or a special type of index page for a unique clustered index.

**Allocation Pages** The first page of an extent is always an allocation page, which is a bit map of available and used pages in an extent, organized by object id.

**OAM** Object Allocation Map - Maps available pages for objects. The OAM is an index for each object into the allocation pages.

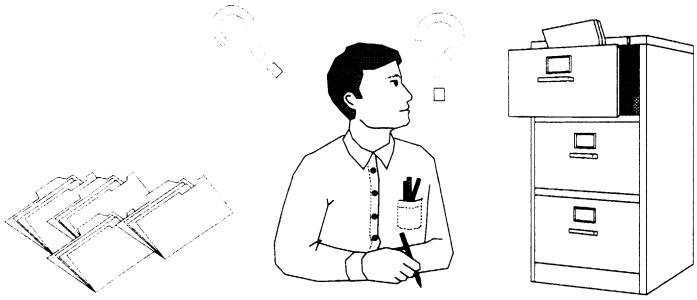
**GAM** Global Allocation Map - Maps available pages for database. The GAM is an index into the allocation units, one per database.

# Storage Structures: Topics

- Basic concepts of data storage
- Non-clustered tables 

# Non-clustered Table Storage

- When no clustered index is defined, tables are stored as non-clustered tables
- When a table has no clustered index, the data rows will not be in any particular order
- Data is always inserted on the last page



**Non-Clustered Table Storage**

Non-clustered table storage is similar to keeping a set of folders in a filing cabinet with unlabeled drawers, each contain random folders.

For a few items, such an arrangement works well, with little overhead.

It is easy to store new folders, and a quick scan of all drawers will find any folder.

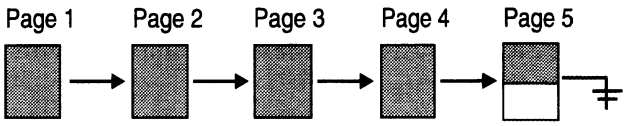
However, if the number of drawers or folders gets too large, it becomes difficult to find a particular folder.

**How Data Pages Are Stored**

Non-clustered indexes do not affect how data pages are stored.

# Operations on Non-clustered Tables

- Non-clustered tables are stored as page chains




- Possible operations on a non-clustered table
  - select
  - insert
  - delete
  - update
- Commonly called a **heap** → liever gebruiken!

**Select**

- All data pages are scanned to find any data row if no index exists.
- If a non-clustered index exists, all data pages might not be scanned in order to located the row(s).

**Insert**

- Data row is added to the last page.
- If more space is needed, an extent (of 8 pages) is allocated at the end.

**Delete**

- Data row is marked as deleted, and no space is reclaimed until an entire page is empty.
- The only way to compress non-clustered pages after a series of deletes or updates is to bulk copy the data out, delete the table, recreate it, and reload.

**Update**

- We will discuss updates in a separate section later in this module.

## Non-clustered Tables: Advantages and Disadvantages

- Sequential disk access is efficient
  - However, the entire table must be scanned if no indexes exist.
  - The entire table may not be scanned if a non-clustered index exists.
- On insert, batch inserts can do efficient sequential I/O
  - However, there is a potential bottleneck on the last page if multiple processes try to insert data concurrently
- On delete, rows never move
  - However, empty space on pages is not reusable until all rows are deleted and the page is empty
  - After all rows are deleted on a page, it can be re-allocated

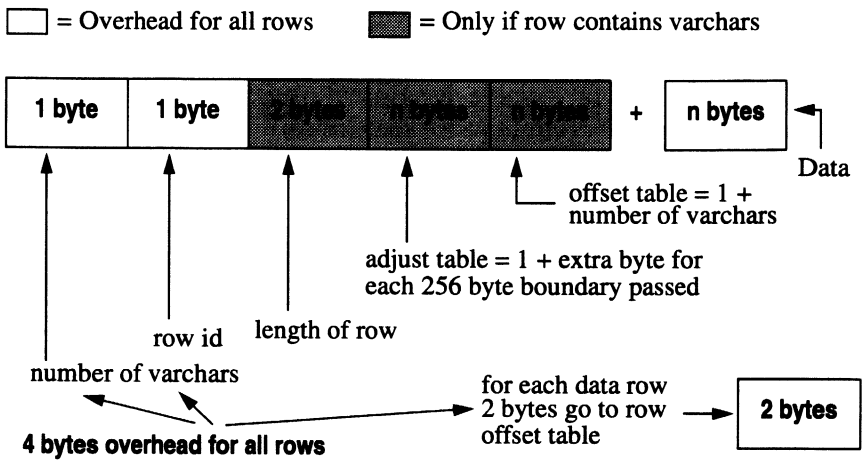
**When are Non-clustered Tables Appropriate?** The answer depends on application characteristics and the size and context of the table being accessed.

# Predicting Table Size

- To *predict* how big a table will be *before* it is created
  - Determine overhead for each row
  - Calculate its row length (includes overhead)
  - Assume an average data page efficiency (75% in examples)
  - Determine how many pages will be needed to hold all rows

# Calculating Row Size: Determine Row Size

- Data row overhead + data




**Nulls versus Non-nulls**

- Any data which allows nulls is automatically stored as a variable-length column. This means there is additional overhead in the row.
- *char* and *binary* columns which allow nulls are considered *varchars* and *varbinary* for purposes of optimization.

**Character Data**

- Columns defined as *varchar* rather than *char* add slightly more overhead to each data and index row.

**Row Offset Table**

- Bytes at the end of a page used to locate rows on that page

**Offset Table**

- Stores the offset for varchars in the datarow

**Adjust Table**

- Indicates the size of the varchar column

# Calculating Row Length: Algorithm

- Row length:
  - Tables with fixed (non-null) columns only:
    total size of all columns + 4 bytes (overhead)

  - Tables including variable-length columns:
    total size of all columns (use maximum)
    + 4 bytes overhead = subtotal

    subtotal +
    + (floor[subtotal]/256 + 1)
    + (number of variable length columns + 1)
    + 2 bytes overhead
    = total

# Calculating Row Length: Example

```
create table test_length
(colA char(10) not null,
 colB int null,
 colC money null,
 colD varchar(100) null)
```

**data row**

| colA | colB | colC | colD |
|---|---|---|---|
| **10 bytes** | **4 bytes** | **8 bytes** | **100 bytes** |

row length = 122 bytes (for the total size of columns (maximum))
+ 4 bytes overhead
+ 3 bytes + 1 (for the 3 var columns)
+ ([122+4]/256) + 1 overhead
+ 2 bytes overhead

= 133 bytes total

# Predicting Table Size: Number of Pages

- Calculate how many pages you will need
  - A page can hold 2,016 bytes of data
  - Rows per page = page size / row length
  - Number of pages = rows / (rows per page * fill factor)
- Example:
  - Assume table with 1,000,000 rows, **133** bytes per row
  - Data rows/page = (2,016)/(133) = **15** rows/page

    data & overhead
  - 1,000,000 rows / (15*0.75) = **88,889** pages

    fill factor
- Table size is **182** Mb (88,889 * 2 Kbytes)

# Predicting Table Size: sp_estspace

- By default, varchar columns are estimated as 50% full
- You can optionally specify full varchar columns
- Example:
  What will the size of the **titles** table be if it had 10,000 rows and the **title** and **notes** columns were filled?

```
sp_estspace titles, 10000, null,
   "title, notes", null, null
go

name   type idx_level Pages Kbytes Total_Mb
----   ---- --------- ----- ------ --------
titles data         0 1683    3365     3.29
```

**Full Syntax**

```
sp_estspace tableland, no_of_rows,
   [,fillfactor] [,cols_to_max] [,textbin_len]
   [,iosec]
```

**table_name** — Name of table, which must already exist in the current database

**no_of_rows** — Estimated number of rows table will contain.

**fillfactor** — Index fill factor. Default is null, meaning SQL Server will use the default fill factor.

**cols_to_max** — Comma-separated list of variable-length columns, which to use the maximum length instead of the average.

**textbin_len** — Length of all text and image columns per row.

**iosec** — Number of disk I/Os per second on host machine.

# Displaying Table Size: sp_spaceused

- Once a table is created and loaded, use `sp_spaceused` to display its size

```
1> use pubtune
2> go
1> sp_spaceused titles
2> go
name    rowtotal reserved  data
 indexes unused
------- ---------------   ---------
----------- ------
titles  5000     1248 KB    1246 KB
 0 KB         2 KB
```

**Advantage of sp_spaceused**

No performance impact; **sp_spaceused** looks only at the most recent values stored in the system tables.

**Disadvantage of sp_spaceused**

Not exact and may be slightly less accurate than **dbcc tablealloc**.

# Displaying Table Size: dbcc tablealloc

- You can also use `dbcc tablealloc`

```
1> use pubtune
2> go
1> dbcc tablealloc(titles)
2> go
TABLE: titles   OBJID = 208003772
INDID=0 FIRST=385 ROOT=1759 SORT=0
Data level: 0. 623 Data Pages in 78 extents.
...
Allow page 256 (# of extent=1 used pages=8 re
pages=8)
...
Total (# of extent=78 used pages=624 ref
pages=624) in this database
```

**INDID**

- Index id is 0 for a non-clustered table.

**Advantage of dbcc tablealloc**

- Is highly accurate.
- The space reporting is just a by-product of the true use of **dbcc tablealloc**, which reads every page and fixes any minor internal errors while checking for and reporting any serious table corruption.

**Disadvantage of dbcc tablealloc**

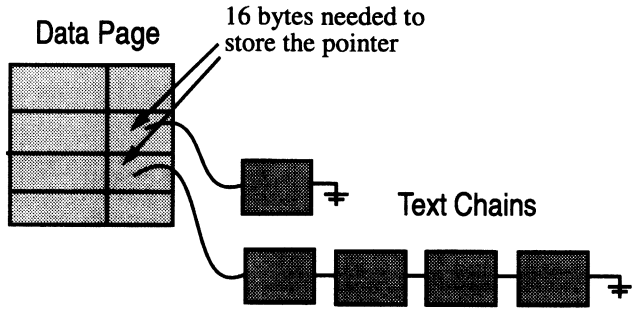
- Locks the table and reads every page, which can temporarily affect performance for other users.

# Lab 3 – Table Storage: Size

- Calculate row size
- Predict the size of two tables
- Determine the size of two tables

## Text and Image Chains

- Text and image fields are stored as separate page chains
- Sequential access of data in the text or image chain is very efficient
- Every non-NULL value in the data uses at least one 2K page




**Text Chain Size**

- Is roughly the size of the text value itself, rounded up to the nearest 2K value
- There is some overhead associated with text chains, but not much
- Overall size of all text value for a given table is the total of all non-NULL chains

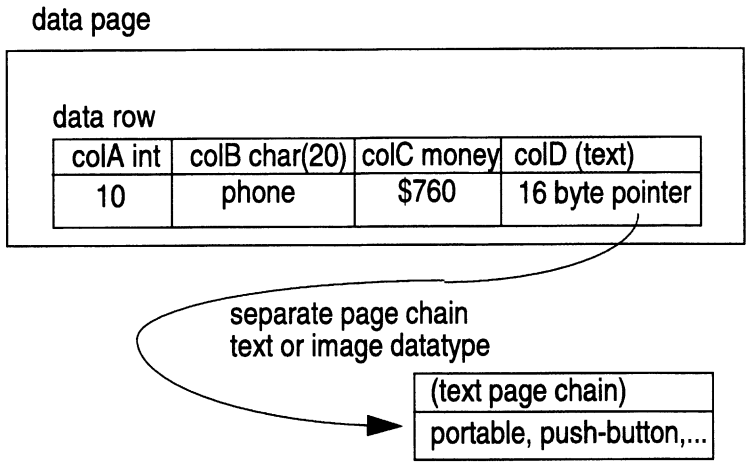
**Row Density**

- Text chains do not affect the row density of a given table or index
- Text chain pages are maintained separately from data or index pages

**Storage**

- A text or image column requires 16 bytes in the table to store the pointer and its associated 2K chain pages

# Text Page


data page
data row
colA int
colB char(20)
colC money
colD (text)
10
phone
$760
16 byte pointer
separate page chain
text or image datatype
(text page chain)
portable, push-button,...

# Predicting Size of Text and Image Chains

- Supply the average or maximum length to `sp_estspace`
- Example:

```
sp_estspace blurbs, 5000, null, null, 2000
go
name   type idx_level Pages Kbytes Total_Mb
-----  ---- --------- ----- ------ -------
blurbs text         0 10000  20000   19.68
```

**Syntax**

```
sp_estspace table_name, no_of_rows [, fill_factor
  [, cols_to_max [, textbin_len [, iosec]]]]
```

**table** blurbs

**rowcount** 5000

**fill factor** null (no fill factor)

**variable columns** null (no variable columns)

**maximum column length** 2000 bytes

# Text and Image Chains: Issues and Guidelines

- Issues
  - Is the data accessed frequently?
  - Is the text column necessary?
  - Are the text or image columns nearly empty when used?
- Guidelines
  - Define columns as nullable when values are not always present
  - If excessive I/O to the separate storage structure occurs often, consider using a short summary (varchar) column

# Summary

- Basic unit of storage for SQL Server is a page (2048 bytes)
- At least 4 bytes of overhead per row (more for rows with variable length columns)
- Data pages are allocated in 8 page extents
- Other types of pages
  - Index pages
  - Distribution pages
  - Overflow pages
  - Text/Image pages
  - Object allocation and global allocation pages
- Non-clustered tables advantages
  - Sequential disk access is efficient
  - Inserts go to the end of the chain
  - Rows never move
- Using **sp_estspace, sp_spaceused, dbcc tablealloc**
- Text and Image pages

4

# How Indexes Work

## System 10 Performance & Tuning

Version 2.2




*The Enterprise Client/Server Company™*

# 4

# Objectives

- Describe how indexes are structured and what space is required for them
- Understand how index structures and data pages react as data changes
- Describe the various types of update approaches and their performance impact

# Why Study Indexes?

- Indexes are the most important physical design element leading to high performance
- Optimal indexing demands an understanding of the best usage mix of non-clustered tables, clustered indexes, and non-clustered indexes

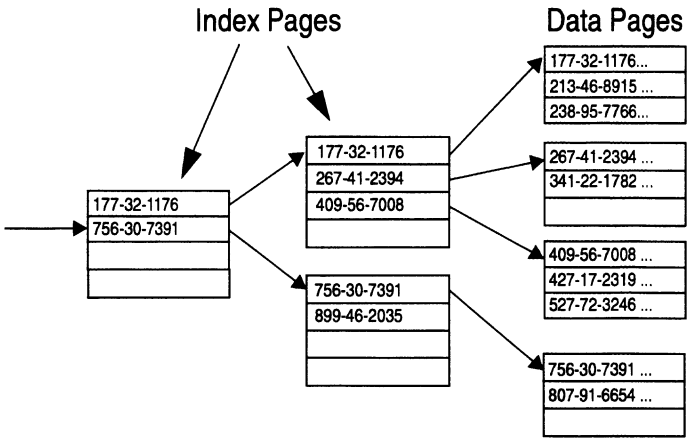
# Topics

- Indexes 
  - Clustered indexes
  - Non-clustered indexes
- Page splits/merges, fill factor
- Update efficiency

How Indexes Work

# What Are Indexes?

- Database objects that can be created for a table to allow direct access to specific data row(s)




**Indexes**

- Are a collection of index pages in addition to the data pages for a table
- Contain logical pointers to allow faster retrieval of the physical data

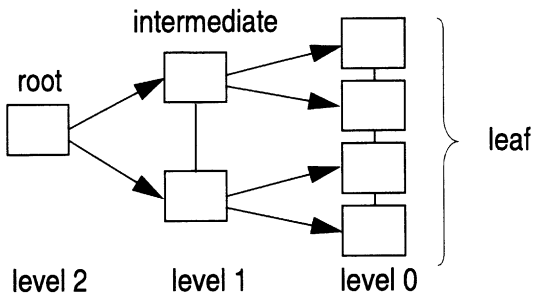
## Why Use Indexes?

- To improve performance
  - Avoid table scans when accessing data
  - Avoid table access altogether (index covering)
  - Avoid sorts
- To enforce uniqueness of data
- To randomize inserts

**Randomize Inserts** With a heap table, all inserts are added at the end of the heap.

# Index Structure

- Indexes are structured as B Trees
- Indexes have multiple levels
  - Each *level* of an index is its own page chain
  - Level numbers increase monotonically (leaf to root)




**Root**
- The highest level

**Leaf Level**
- The lowest level
- Might contain data or pointers

**Intermediate Levels**
- All levels in-between root and leaf

# Tables and Indexes

- Two types of indexes
  - Clustered
  - Non-clustered
- A table can have
  - Maximum of 1 clustered index
  - And up to 249 non-clustered indexes
- Order of data depends on indexes
  - If there is a clustered index, data is ordered by the index
  - If all indexes are non-clustered, or if there are no indexes, data is in the order in which it is inserted

# Indexes and Keys

- Two different kinds of keys:
  - **Logical** keys (primary, foreign, common)
  - **Physical** keys (single column, or composite)
- Indexes do not necessarily correspond to logical keys
  - Indexes may or may not be applied to a primary or foreign key columns at physical design time
  - Indexes may be applied to columns that are not logical keys
- Total size of physical (index) key must be less than 256 bytes

**Logical Keys**

- Are defined at the logical design level to describe how tables are related

**Physical Keys**

- Are the columns used to define the order of an index
  - Can be based on a single column
  - Can be based on multiple columns, called a *composite* key

**Composite Keys**

- Can have up to 16 columns

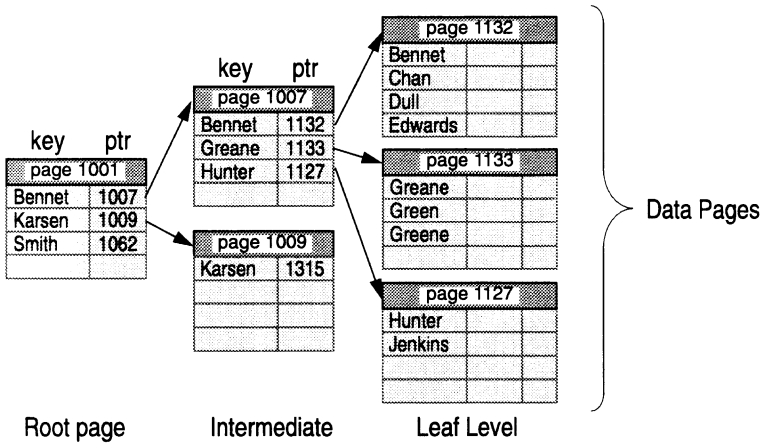
# How Indexes Work: Topics

- Indexes
  - Clustered indexes 
  - Non-clustered indexes
- Page splits/merges, fill factor
- Update efficiency

How Indexes Work

# Clustered Index Structure

- Leaf level pages are also the data pages
- Each index entry is a pointer to a page in next level

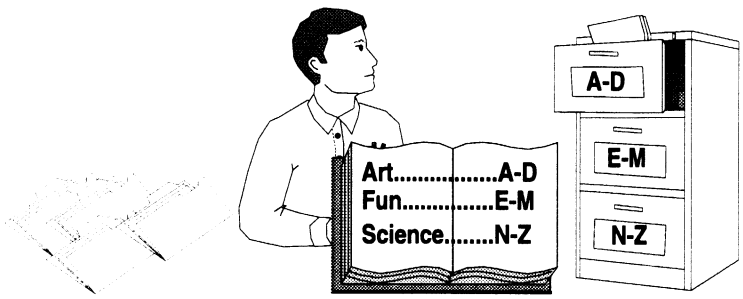



**Clustered Index Pointers**

Only one clustered index entry can point to a page in the next level

# Clustered Table Storage

- Data rows in the table are physically ordered
- User defines how data rows are ordered by specifying physical (index) key




**Clustered Index Is Like a File Cabinet**

- Data pages are like labelled file drawers that are in alphabetical order
- Higher level index pages are like a book that lists the drawer for each file
- Files are stored in order in each drawer

**Finding a Data Row**

- Look in the index book for the file, and find the drawer where it is stored
- Search the drawer for the file

**Good For Range Searches**

- Clustered indexes are best for range queries because rows will be physically together on the same data page, or they will span adjacent data pages

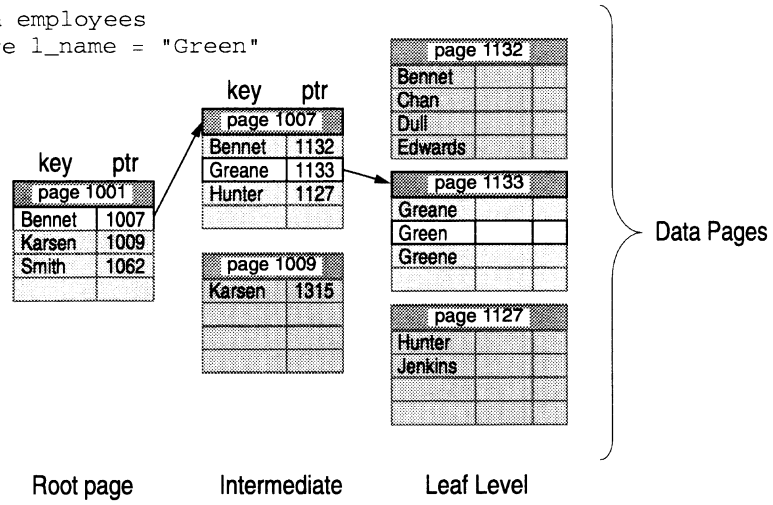
**Index Slower for Inserts**

- Files must be inserted in alphabetical order
- If adding a file to a drawer that is already full, last file in the drawer has to move to the next drawer
- Have to change the contents of the index book, keeping it all in order

How Indexes Work

# What Happens During a Select

```
select *
from employees
where l_name = "Green"
```




**With a Select**

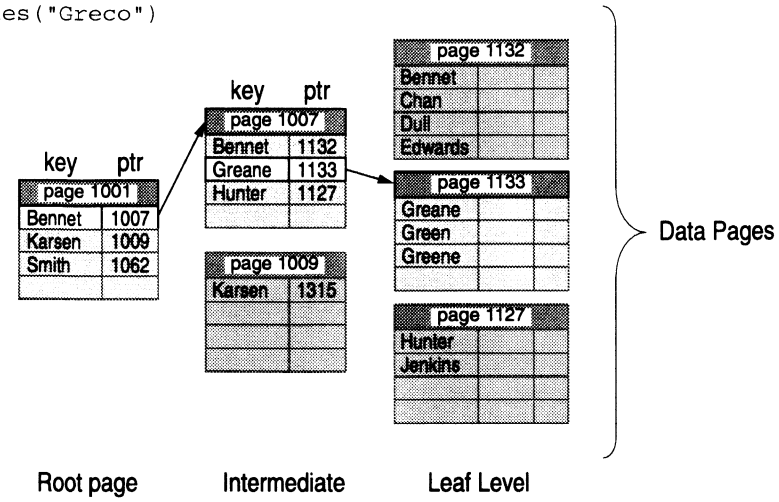
- The index pages are traversed, looking for the appropriate key entry
- Data rows are found in the leaf level data pages

**Clustered Index Performance**

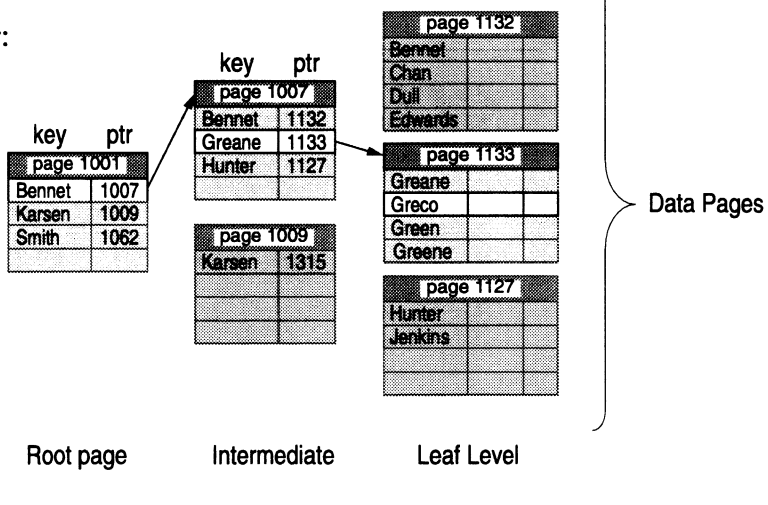
- A select using a clustered index will have the following I/O:
  - One to read root level
  - One for each intermediate level
  - One to read the data page (leaf level)
- These page reads may come from either cache or disk

# What Happens During an Insert

```
insert employees(l_name)
values("Greco")
```

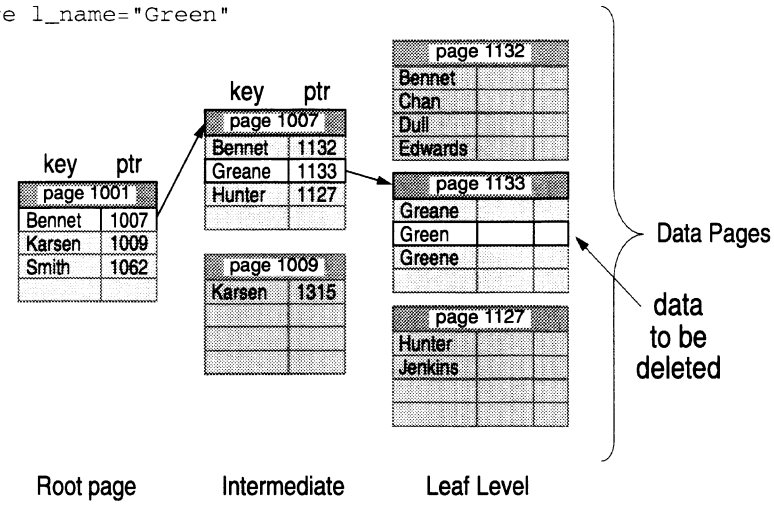



After:

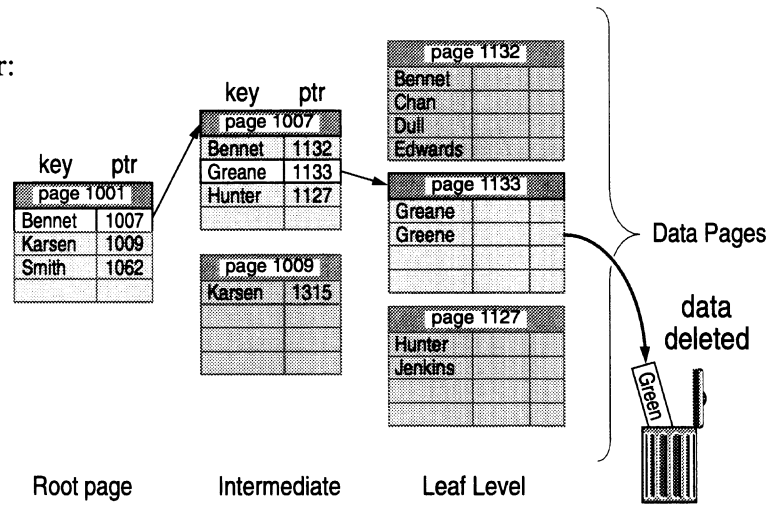


# What Happens During a Delete

```
delete employees
where l_name="Green"
```




After:



# Clustered Index Size

- To predict index size (before creation)
  - Calculate row sizes, rows per page, and number of pages by hand
- To predict index size (after creation)
  - Use `sp_estspace`
- To display current index size (index created, table loaded)
  - Use `sp_spaceused`
  - Use `dbcc indexalloc(), dbcc tablealloc()`
- To display index information on existing indexes
  - Use `sp_helpindex`

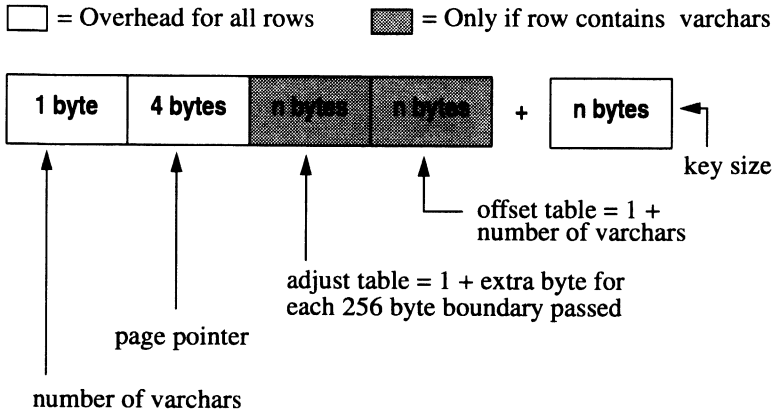
See *System Administration Guide for SYBASE SQL Server,* Chapter 10, "Predicting the Size of Tables and Indexes."

# Predicting Clustered Index Size

1. Calculate expected number of data pages for underlying table
2. Find number of index rows (**key_ptr** pairs) per index page
3. Divide number of data pages by number of index rows per page to get index pages at lowest index level
4. Repeat previous division to find number of index pages at each level
5. When division result is < 1, stop – this is the root page

# Clustered/Root Index Row Size

- Overhead for all *clustered* and all *root* index pages + key size



# Predicting Clustered Index Size: Example

- *titles* table with **1,000,000** rows of data
- Row length = **248** bytes
- Rows/page = 2016/248 bytes = **8** (floor)
- Data page efficiency = **75%**
- Data pages = 1,000,000/(8*0.75) = **166,667** pages
- Clustered index row length:

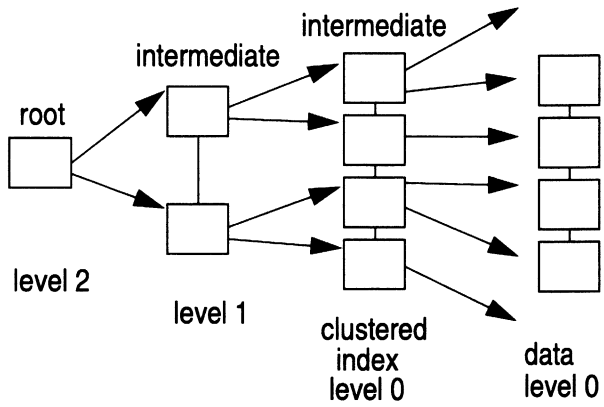
  4 bytes overhead for page pointer
  1 byte additional overhead
  6 bytes for key (title_id)
  **11** bytes total

- Clustered index rows per page = 2,016/11 = **183**

## Predicting Clustered Index Size (continued)

- Total pages for clustered index = 1,225 (2.47 Mb)

**1 + 9 + 1215 = 1225 pages**




**Level 0 Index Pages**

- Level 0: 166,667/(183*0.75) = **1215** pages (ceiling)

**Additional Levels of Index Pages**

- Level 1: 1215/(183*0.75) = 8.85 => **9** pages (ceiling)
- Level 2: 9/(183*0.75) =< 1 page
  –> Level 2 is the root page – stop

**Index Size**

- Total: 1215+9+1=**1225** (2K) pages for index B-tree pages
  - B-tree is **2.47** Mb (1225 pages * 2K)

**Data Size**

- Data size: (166,667*2016) = 336 Mb

# Predicting Index Size Using sp_estspace

- sp_estspace

```
1> sp_estspace titles, 1000000, 75, null, null, null
2> go
name         type       idx_level    pages       Kbytes
----------   ---------  --------  --------  ----------
titles       data               0    122771     245542
titleidind   clustered          0       741       1481
titleidind   clustered          1         5         11
titleidind   clustered          2         1          2

Total_Mbytes
-------------
      247.03

name         type       total_pages     time_mins
----------   ---------  ----------  -------------
titleidind   clustered       123518            614
```

**idx_level**

0 indicates the leaf level

For a clustered index, the data level is the leaf level of the index

# Predicting Index Size Using sp_estspace

**(continued)**

- Factors influencing accuracy of estimates generated by **sp_estspace**:
  - Initial fill factor
  - Average index density
  - Effects of inserts and deletes over time

# Displaying Current Index Size: sp_spaceused

```
1> sp_spaceused titles_idpr
2> go
name        rowtotal    reserved          data
index_size       unused
-------     --------    ---------     ---------
-----------       ------
titles      5000           1614 KB      1254 KB
192 KB           168 KB
```

# Displaying Current Index Size: dbcc indexalloc

```
1> dbcc indexalloc (titles_idpr, 1)
2> go
...
TABLE: titles_idpr           OBJID = 80005881
INDID=1  FIRST=3024       ROOT=3115        SORT=1
       Data level: 1. 627 Data Pages in 80 extents
       Indid     : 1.  11 Index Pages in 2 extents.
TOTAL # of extents = 82
Alloc page 3072 (# of extent=2 used pages=13 ...)
Alloc page 512 (# of extent=1 used pages=2 ...)
Alloc page 1024 (# of extent=1 used pages=8 ...)
...
Total (# of extent=82 used pages=641 ...)
```

**Syntax**

```
dbcc indexalloc
({table_name | table_id}, index_id)
```

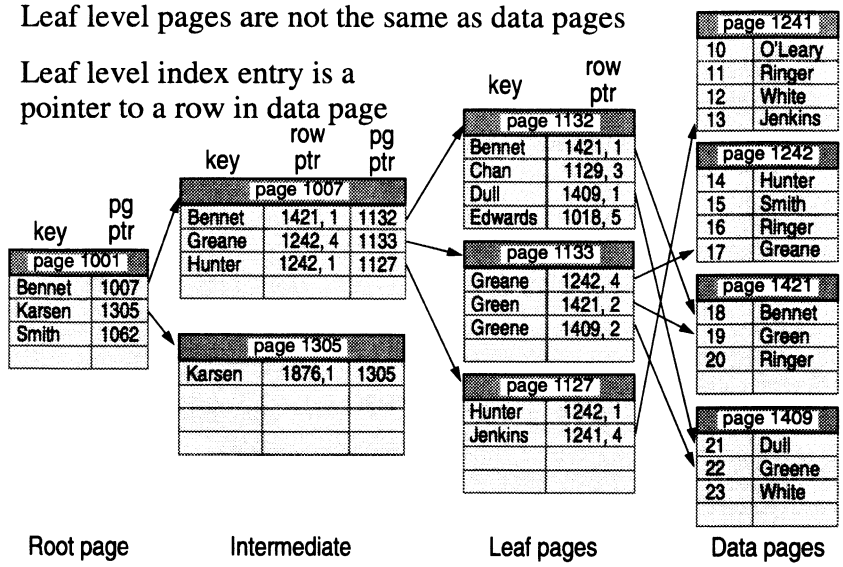
# How Indexes Work: Topics

- Indexes
  - Clustered indexes
  - Non-clustered indexes 
- Page splits/merges, fill factor
- Update efficiency

How Indexes Work

# Non-clustered Index Structure

- Leaf level pages are not the same as data pages
- Leaf level index entry is a pointer to a row in data page




**This Example**

- Non-clustered index on **l_name**

**Above the Leaf Level**

- The B-tree works similarly for non-clustered and clustered indexes

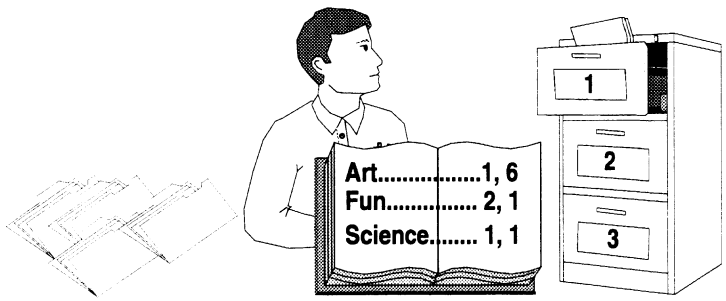
**At the Leaf Level**

- A non-clustered index is slightly different than a clustered index
  - It contains a key-pointer pair for every row in the data table in the leaf level
  - The key-pointer pairs at the leaf level are in index order which does not match ordering at the data level

How Indexes Work

# Non-clustered Index Storage

- Index keys are stored in random order at data level
- Non-clustered indexes are larger and may be less effective than clustered indexes, but we need them because we need more than one index per table




**File Cabinet Analogy**

- Data pages are like uniquely labelled file drawers in a file cabinet
- Index pages are like a book that lists the drawer for each file, and where the file is in the drawer
- The files are not in any order in the drawers

**Finding a Data Row**

- Look in the index book, find the drawer and file location
- Locate the file in the drawer

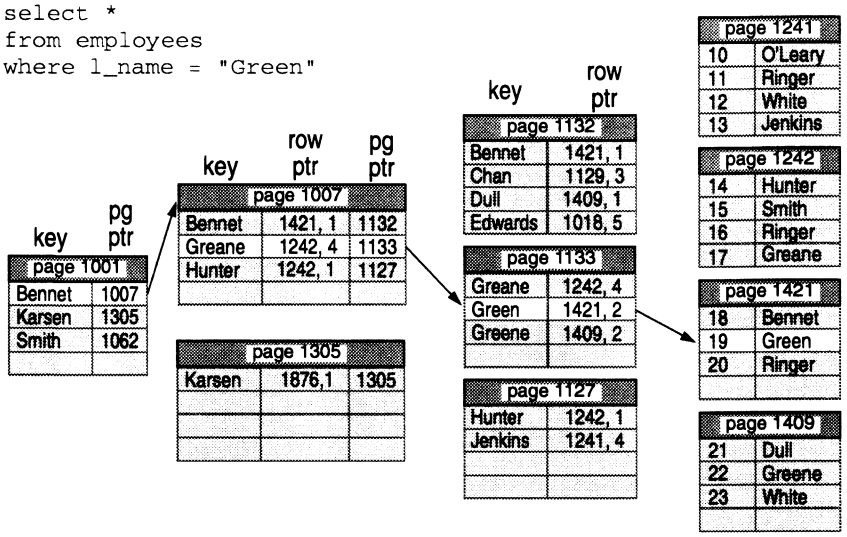
**Good For Single Row Searches**

- The data rows are not organized on a data page

How Indexes Work

# What Happens During a Select

```
select *
from employees
where l_name = "Green"
```




**Select**

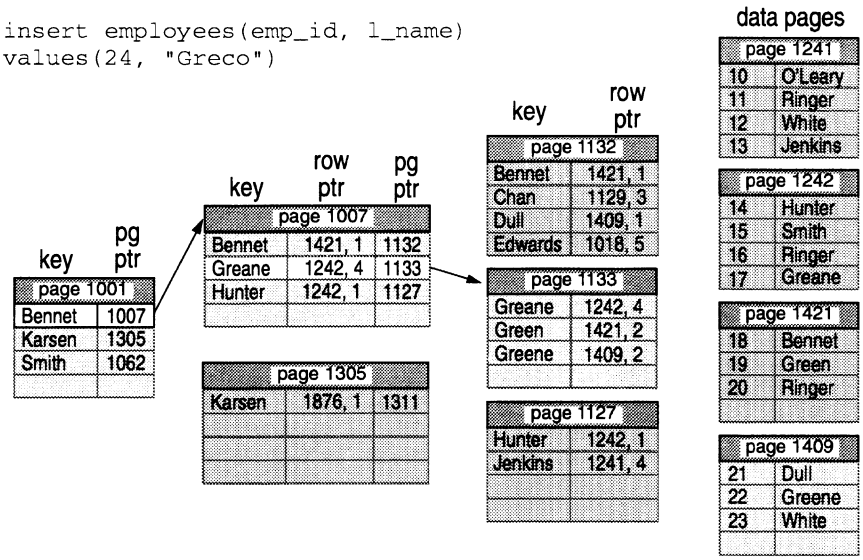
- Page pointers are used to traverse index pages to leaf level
- At leaf level, row pointer identifies location of row on data page

**Non-clustered Index Performance**

- A select using a non-clustered index will have the following I/O:
  - One I/O to read root level, one I/O for each intermediate level
  - One I/O to read the leaf level
  - One I/O to follow pointer to data page
- All of these may be physical reads, depending on what is in cache.

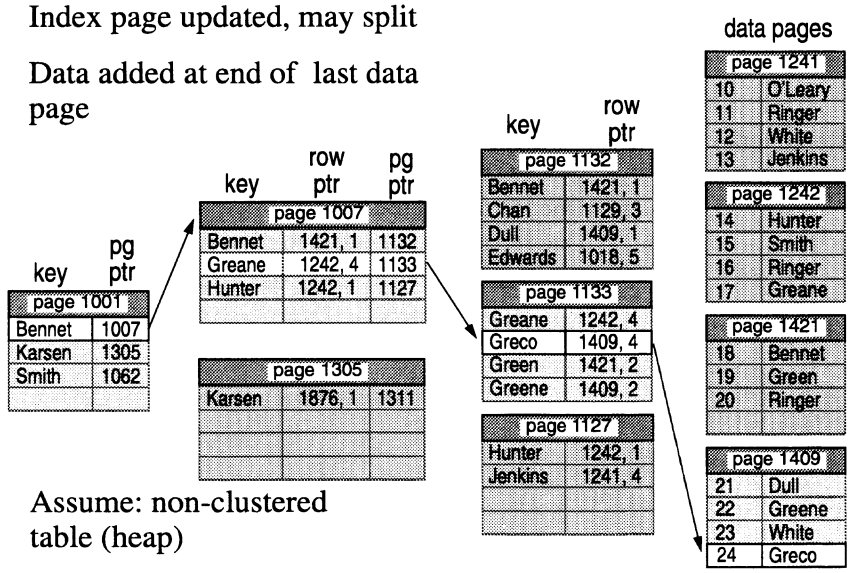
# What Happens During an Insert

```
insert employees(emp_id, l_name)
values(24, "Greco")
```




After:

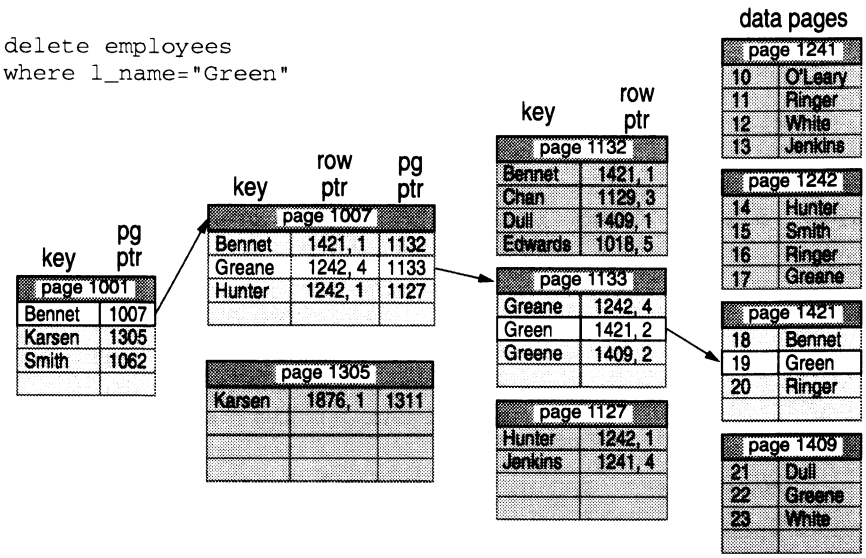
- Index page updated, may split
- Data added at end of last data page




- Assume: non-clustered table (heap)

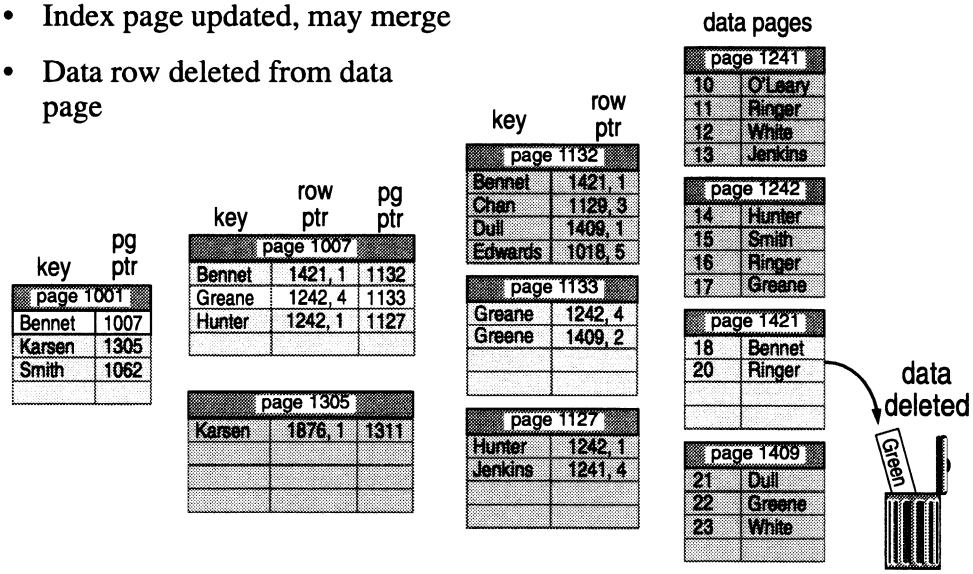
# What Happens During a Delete

```
delete employees
where l_name="Green"
```



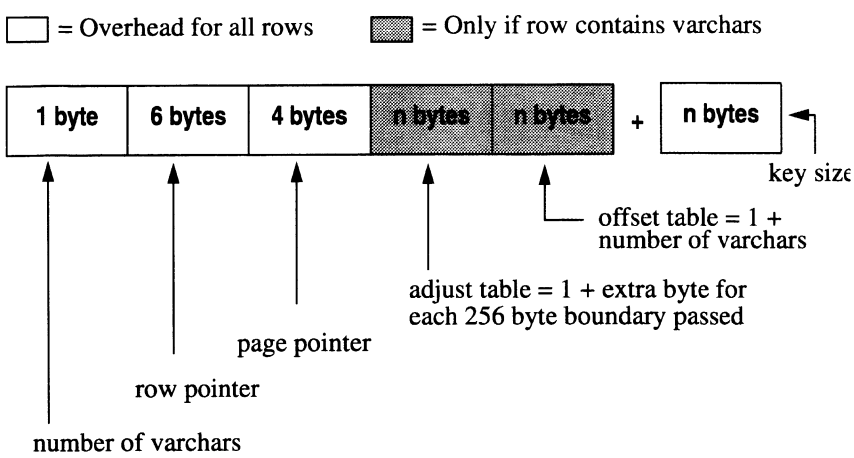

After:

- Index page updated, may merge
- Data row deleted from data page



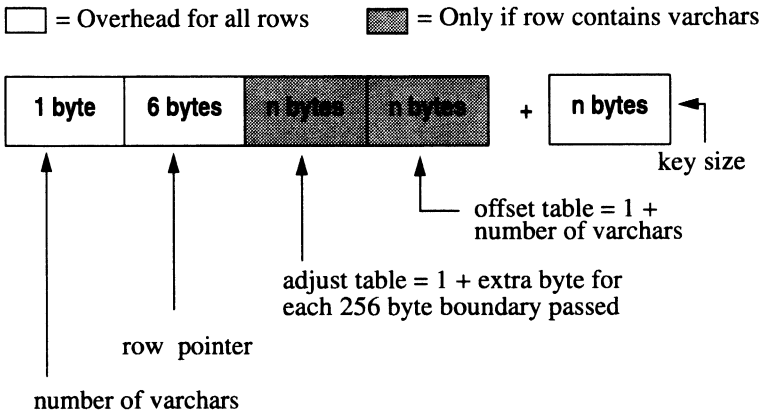
# Non-clustered Intermediate Page Index Row

- Overhead for an *intermediate* non-clustered index row + key



# Non-clustered Leaf Page Index Row Size

- Overhead for an *leaf* non-clustered index row + key size



# Predicting Non-Clustered Index Size

- Assumptions:
  - Data: *titles* table, **1,000,000** rows, **248** bytes per row
  - Index Key: **6** bytes (**title_id**)
  - Average page utilization = **75%**
- Step #1: Determine no. of data pages in table
  - Rows/page = 2016/248 bytes = **8** (floor)
  - Data pages = 1,000,000/(8*0.75) = **166,667** (ceiling)
- Step #2: Determine # of non-clustered index *leaf* pages are needed to point to 1,000,000 data rows

  1,000,000/((2016/(6+1+6))*0.75) = **8,598** (ceiling)

jaar erwan uit dat title_id geen varchar is. Klopt niet.

**Step #1 Algorithm**

Data Pages = rows in table/(rows per page * fill factor)

**Step #2 Algorithm**

Non-clustered Index Leaf Pages =
(rows in table/((page size/(index key + overhead)) * fill factor)

# Predicting Non-Clustered Index Size (Continued)

- Step #3: Determine # of rows of non-clustered *intermediate* index pages are needed to point to 8,598 non-clustered index *leaf* pages

  8,598/((2016/(6+1+4+6))*0.75) = **97** (ceiling)

- Step #4: Determine # of rows of non-clustered *intermediate* index pages are needed to point to 97 nonclustered int. pages

  97/(2016/(6+1+4+6))*0.75) = **2** (ceiling)

- Step #5: Repeat #4 until the answer < 1

  2/((2016/(6+1+4+6))*0.75) = **1** (ceiling)

**Step #3 Algoritm**

Non-clustered Intermediate Index Pages =
(non-clustered leaf pages/((page size/(index key + overhead)) * fill factor)

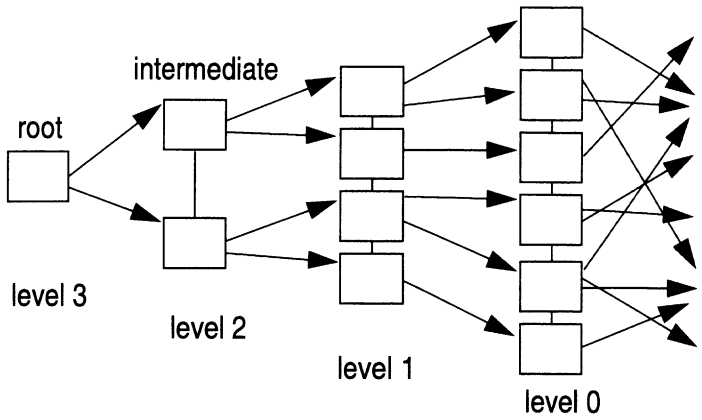
**Step #4,5,6, etc Algorithm**

Non-clustered Intermediate Index Pages =
(non-clustered int. pages/((page size/(index key + overhead)) * fill factor)

# Predicting Non-Clustered Index Size (Continued)

- Total pages for nonclustered index = 8,698 (17.81 Mb)

**1 + 2 + 97 + 8,598 = 8,698 pages**



# Predicting Index Size: sp_estspace

```
1> sp_estspace titles, 1000000, 80, null, null, null
2> go
name        type       idx_level    pages      Kbytes
----------  ---------- ----- -------- ----------
titles      data              0   122771     245542
titleidind  clustered         0      741       1481
titleidind  clustered         1        5         11
titleidind  clustered         2        1          2
titleind    nonclustered      0    33335      66670
titleind    nonclustered      1      981       1963
titleind    nonclustered      2       30         60
titleind    nonclustered      3        1          2

Total_Mbytes
-------------
      308.33
name        type       total_pages     time_mins
----------  ---------  ----------   ------------
titleidind  clustered      123518           614
titleind    nonclustered    34347           221
```

**Factors Affecting Index Size**

- Initial fill factor
- Average index density
- Effects of inserts and deletes over time

# Displaying Current Index Size: sp_spaceused

- Example:

```
1> use pubtune
2> go
1> sp_spaceused titles_idpr
2> go
name     rowtotal reserved  data
 index_size  unused
------- ---------------   ---------
-----------  ------
titles  5000      1420 KB   1242 KB
 118 KB       60 KB
```

- **sp_spaceused** reports a single figure for all indexes

**Index Size**

- Is for all indexes, but not leaf pages in clustered index

**Unused**

- Is empty data or index pages in allocated extents, not space within a page

# Displaying Current Index Size: dbcc indexalloc

```
1> dbcc indexalloc (titles_idpr, 2)
2> go
...
TABLE: titles_idpr              OBJID = 80005881
INDID=2  FIRST=576        ROOT=578         SORT=1
        Indid     : 2.  85 Index Pages in 19 extents.
TOTAL # of extents = 19
Alloc page 256 (# of extent=1 used pages=7 ...)
Alloc page 512 (# of extent=5 used pages=21 ...)
Alloc page 1792 (# of extent=5 used pages=26 ...)
...
Total (# of extent=19 used pages=86 ...)
```

**Syntax**

```
dbcc indexalloc ({table_name | table_id}, index_id)
```

**index_id**

0 = heap
1 = clustered
2-250 = non-clustered

# Displaying Current Indexes: sp_helpindex

- To see what indexes there are on a table

```
sp_helpindex titles_idpr
go
index_name  index_description             index_keys
----------  ----------------------------  ---------
idx1        clustered, unique located on... title_id
idx2         nonclustered located on ...     price
```

## Clustered vs. Non Clustered

- Clustered index size on 6 byte key = 2.49 Mbytes
- Non-clustered index size on 6 byte key = 17.81 Mbytes
- Clustered index is smaller than non-clustered index
- Clustered index calculation doesn't include leaf level pages because:
  - Leaf level *is* the data level for clustered index
- Leaf level is *not* the data level for non-clustered index
- Table data with a clustered index is ordered in key order
  - Data level needs to be re-ordered during creation
- Table data with a non-clustered index is not ordered in key order
  - Data level does *not* need to be re-ordered during creation

**Rule of Thumb** Allow 1% of data space for clustered index, 15% for non-clustered

# Lab 4a – Indexes and I/O Activity

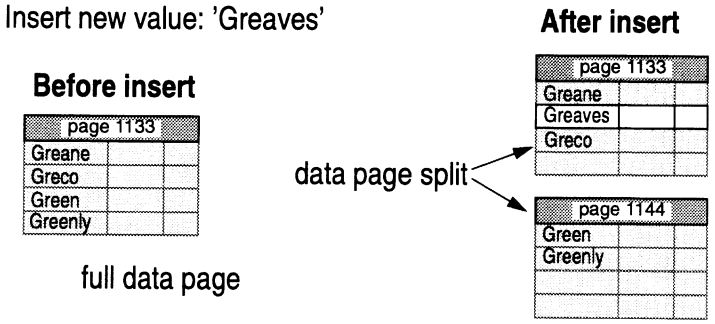
- Create a clustered index for a table, then delete and insert rows, recording I/O activity
- Add non-clustered indexes and compare I/O activity
- Display size of a table with and without a clustered index (default fill factor)

# How Indexes Work: Topics

- Indexes
  - Clustered indexes
  - Non-clustered Indexes
- Page splits/merges, fill factor 
- Update efficiency

# Data Page Splits

- When inserting data on clustered tables, if there is no room on a data page, the page is split 50-50

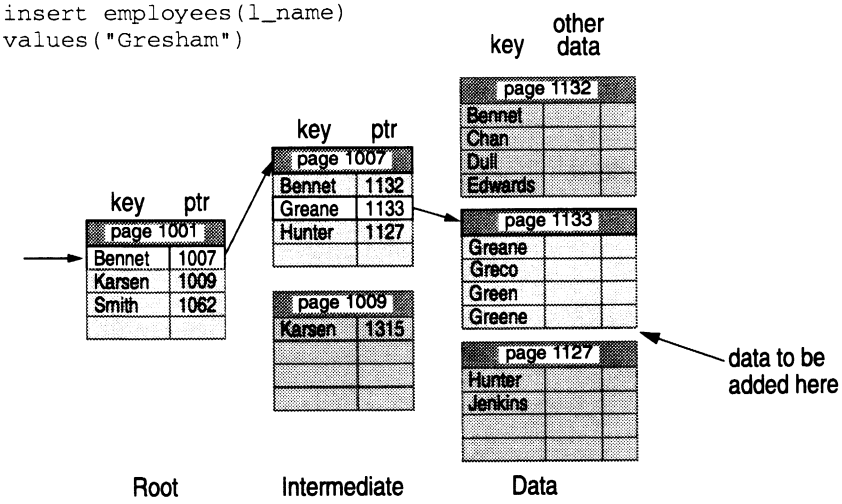



**If New Value Is Going at End of the Page** If new index entry is going at the end of an index page, the page will not split

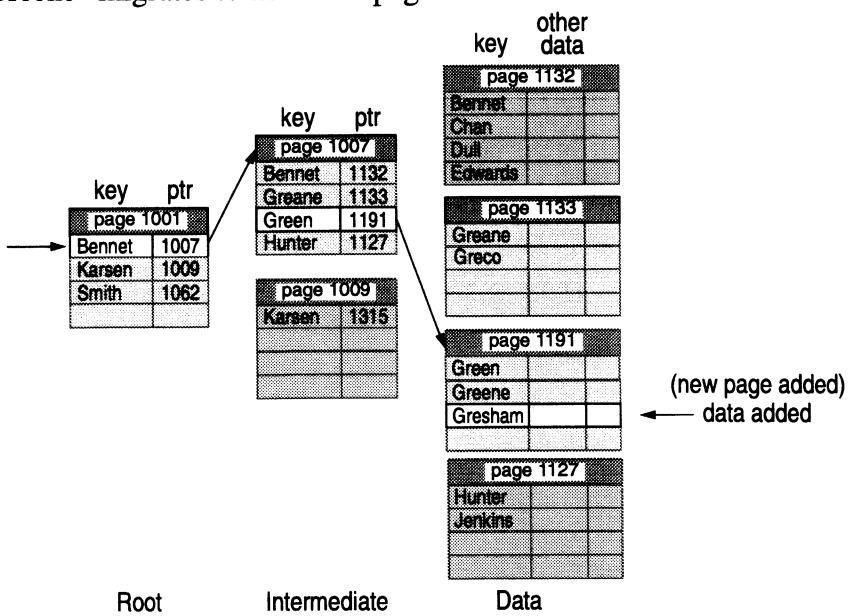
**Pointers Are Updated** Whenever an index page splits, each non-clustered index entry that points to that page must be updated

# Clustered Index Data Page Splitting

```
insert employees(l_name)
values("Gresham")
```

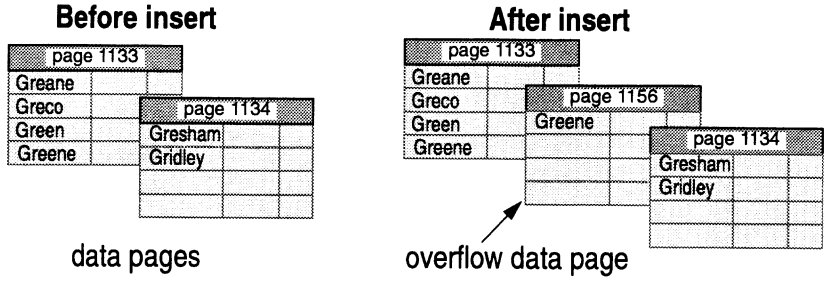



- New page added
  - "Greene" migrates to new data page



# Overflow Pages (Clustered Index Data Pages Only)

- For duplicate key inserts at end of a full page, an overflow page is allocated




- This can only happen with non-unique clustered indexes

**Overflow Pages and the Page Chain**

- Overflow data pages are inserted into the existing page
- There is no pointer from the index pages to the overflow page
- If there were many duplicates keys added, the overflow chain could be more than a single page

**Performance Impact**

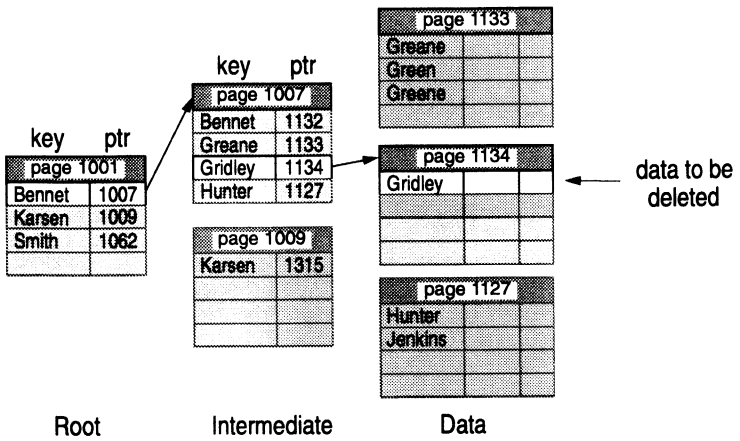
- Extra I/O to access the overflow chain – the entire overflow chain must be scanned anytime duplicated value is used

**How to Avoid**

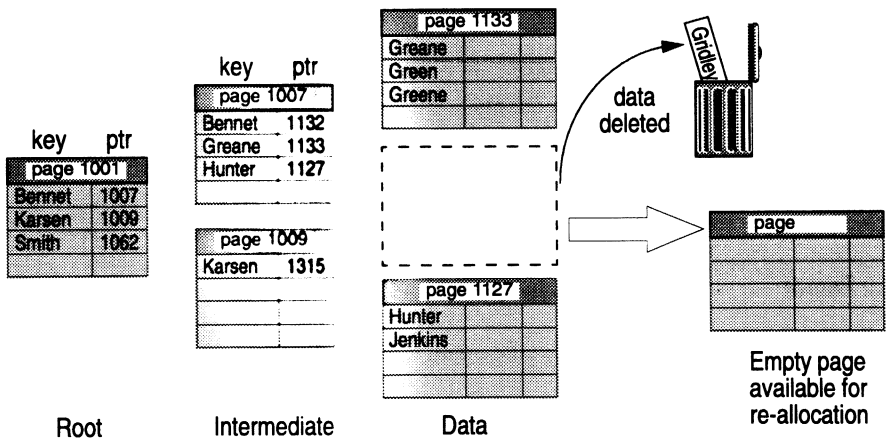
- Different indexing – can another column be added to the index?
- Should the column with the clustered index really have duplicate keys?

# Clustered Index Data Page Merging

```
delete employees
where l_name="Gridley"
```




- After delete

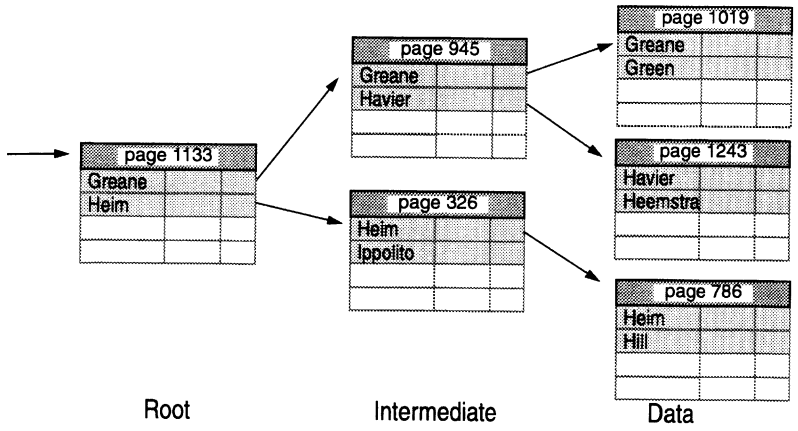



- Data pages deallocated when no rows left
- Index pages deallocated when one row left (after merging page)

How Indexes Work

# Fill Factor

- The **fillfactor** variable determines how full to make index pages when creating an index on existing data




**This Picture**

- Initially all pages would reflect the fill factor specified, for example, all may be 50% full.
- These pages are not all the same, they change over time.
- As data changes, page density will change from initial fill factor

**Low Fill Factor**

- Leaves room for rows to be added (or updated) without splitting pages
- Is better when there will be many random inserts

**High Fill Factor**

- Uses space more efficiently when data will not be changed very much
- Is better when data will be mostly added at the end of the page chain

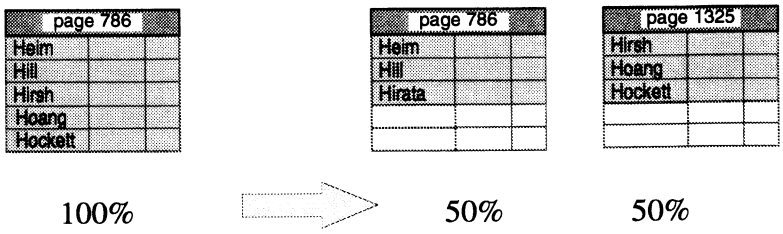
**For Clustered Indexes**

- **fillfactor** also affects leaf (data) pages

**For Non-clustered Indexes**

- **fillfactor** affects only index pages – data page inserts will always be at the end

# Data Page Splits and Fill Factor




- For clustered indexes, data pages tend to migrate toward 75% full
- This is natural and cannot be changed without recreating the index

# Setting Fill Factor

- System administrator can set fill factor system-wide

```
sp_configure "fill factor", 80
```

- Table owner can specify fill factor when creating an index

```
create index titles_ind on titles (price)
   with fillfactor = 50
```

  - Specifying fill factor overrides system-wide setting

- Default fill factor is 0, which means:
  - Full leaf (data) pages
  - Space left on intermediate levels

**sp_configure**

- Example on foil sets the default fill factor to 80%
- Any indexes created without specifying fill factor will be 80% full at index creation time

**create index...with fillfactor**

- Example on foil creates an index on *titles* with each index page 50% full initially

**Fill factor = 0**

- Useful if you cannot afford empty space in data but want to spread out index pages

**Fill factor = 100**

- Full pages at all levels except root
- Useful for static tables

# Fill Factor Guidelines

- Fill factor is valid only at index creation time
  - It is not maintained through subsequent insert/update/delete activity
  - To re-establish after significant activity, drop and recreate the index with sorted data option
- Take fill factor into account when calculating index size by hand
- Set high (100) for read-only tables
- Set low for tables that are growing rapidly with random inserts
- Take into account when next drop/create index is expected

# Lab 4b – Fill Factor and I/O Statistics

- Create clustered indexes with different fill factors and compare
  - Table size
  - I/O statistics

# Updates

- All updates occur as deletes followed by inserts (if not a direct in place update)

```
update authors au_lname = 'Jones'
   where au_id = 'A12345'
```

become (to SQL Server)

```
delete authors where au_id = 'A12345'
insert into authors
   values ('A12345', 'Jones', ...)
```

- The page where the inserted row goes depends on whether there is a clustered index or not

# Rows Inserted in an Update Operation

- In a table with a clustered index

  The inserted row goes to a location on a data page according to its physical order (based on the index)

- In a table without a clustered index (heap storage), it will go to one of these locations:
  - in the same physical location (in the same page),
  - in the same page if there is room, or
  - in the last page of the heap

# Direct or Deferred Updates

Updates are either:

- ***Direct***

  No internal intermediate processing

- ***Deferred***

  Update is done via an intermediate step

**Direct** updates can be done

- ***In place***, or
- ***Not in place***

**Direct**

- No internal intermediate processing step must be done in order to do the update – the appropriate deletes and inserts are issued directly

**Deferred**

- The transaction log is used, then the update is issued

# Direct Updates in Place

```
update authors
   set au_lname = "Jones"
where au_id = "A12345"
```

- Before

| page 1001 | |
|---|---|
| A11123 | Bennet |
| A12345 | Jackson |
| B33128 | Smith |
| | |

- After

| page 1001 | |
|---|---|
| A11123 | Bennet |
| A12345 | Jones |
| B33128 | Smith |
| | |

- A single modify record is written to the log
- Fastest, but certain conditions must be met

**Direct Updates in Place**

- Done without moving the row
- The new row goes back to the same position it came from

# Direct Updates in Place – Requirements

- Required conditions:
  - Columns being updated cannot be variable length, and cannot allow nulls
  - No index is modified
  - No joins are part of the update
  - Table must *not* have update trigger
  - Optimizer must be able to determine all affected rows at compilation time
- Multiple rows can also be updated in place if the same conditions are met
- Otherwise, update is not in place, or is a deferred update

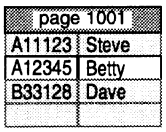
**When Replicating Data** Tables marked for replication (with Sybase Replication Server) can not have direct in place updates

## Direct Updates (Not in Place)

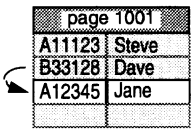
```
update authors set au_fname = "Jane"
where au_id = "A12345"
```

with an update trigger defined on table, no varchars

- Data Page Before




- Date Page After




- Two records – a delete and an insert – are written to the log

**Where the Rows Go**

- As the row is deleted, the other rows on the page move up
- For a heap table, the replacement row goes at the bottom of the page, if there is room on the page (after the delete)
- If there is not room on the page, the replacement row goes at the end of the last page in the heap
- For a table with a clustered index, rows always are inserted in sort order

**Performance Implications**

- As with inserts, all non-clustered indexes must be adjusted when a row moves to a new page
- Adjusting the non-clustered index is also done as an update (delete/insert) and can be costly

**Direct, Not in Place Updates Occur When**

- Only fixed length columns updated
- No index is modified
- No joins are part of the update
- There can be a trigger
- Multiple rows can also be direct updates, not in place

## Deferred Updates

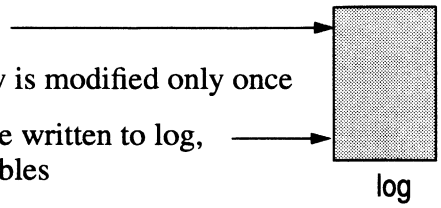
Result in a multistep process that writes, scans, and then writes to the log before updates are applied to the table

```
update authors set au_fname = "Jane"
where au_lname = "Jones"
   and au_fname = "Betty"
```

and au_fname is varchar

Three steps:

1. Row ids are written to log
2. Log is scanned to ensure row is modified only once
3. Deleted and inserted rows are written to log, and updates are applied to tables




**Deferred Updates**

- There is overhead for these log records – log is re-read to apply the changes, hence the term "deferred"
- When there is a join in the update, or other logic (such as an **or** operation), duplicate rowids can result
- Updates are written to log records before applying them to table in order to eliminate duplicate rowids in the log, and thereby minimize changes to index pages for those rowids

**Deferred Updates Occur When**

- Optimizer cannot determine the number of rows affected at compilation time (as with an **or** strategy), or
- A variable length column is updated, or
- A join is part of the update

# Lab 4c – Where do Updated Rows Go?

- Perform updates
- Determine where the updated data appears

# Summary

- Indexes help to
  - Avoid table scans
  - Avoid table access (index covering)
  - Avoid sorts
  - Enforce uniqueness
  - Randomize inserts
- Tables can have clustered or non-clustered indexes
  - Clustered index yields a table with data rows in physical order
  - Non-clustered index yields leaf pages in key physical order with pointers to the data pages
- Page splits and merges occur for clustered indexes (for both data and index pages)
- Only index page splits and merges occur for non-clustered indexes
- Fill factor plays a major role in data page splits and index page splits
  - Set higher for read-only tables
  - Set lower for volatile tables
- Updates are deletes followed by inserts, and are either:
  - Direct, in place updates
  - Direct, not in place updates
  - Deferred updates

5

# Selecting Indexes for Performance

System 10 Performance & Tuning

Version 2.2




*The Enterprise Client/Server Company™*

# 5

# Objectives

- Use SYBASE diagnostic tools to evaluate the usefulness of indexes
- Understand the significance of individual choices in indexing
- Choose the optimal indexing strategies, given anticipated access requirements

# Topics

- How indexes can affect performance 
- Evaluating cost of table scans
- Evaluating cost of clustered access
- Evaluating cost of non-clustered access
- Evaluating cost of covered access
- Choosing indexes

# How Indexes Can Affect Performance

- All access to data requires reading pages, either from disk or from cache memory
- Access to data through an index allows the Server to:
  - target specific data pages which contain specific rows (*point queries*)
  - establish upper and lower bounds for reading a contiguous range of data (*range queries*)
  - use ordered data to avoid sorts
  - avoid table access completely (*when a query is covered*)

**Performance**

- The performance of a database system is affected directly by the number of pages that must be physically read from disk
- Indexes are one way to reduce the number of pages that must be read

# How Indexes Affect Performance

- Sample query:

```
select * from titles
where title_id = "T6425"
```

- I/O for table scan:

```
Table: titles  scan count 1,  logical reads:
624,  physical reads: 0
Total writes for this command: 0
```

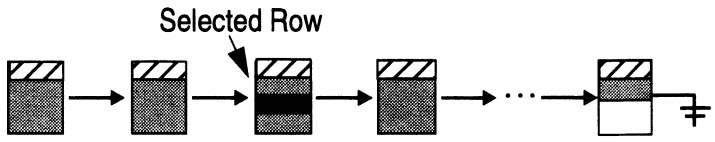
- I/O using clustered index:

```
Table: titles  scan count 1,  logical reads:
2,  physical reads: 0
Total writes for this command: 0
```

- Here, the index was 312 times faster than an equivalent table scan

# Mechanics of Table Scans

- In a table scan, every data page is read when searching for data




- Table scans occur:
  - When there is no index
  - When there is an index, but SQL Server determines that it is not useful

# Topics

- How indexes can affect performance
- Evaluating cost of table scans 
- Evaluating cost of clustered access
- Evaluating cost of non-clustered access
- Evaluating cost of covered access
- Choosing indexes

## Evaluating Cost of Table Scans

- Determine number of pages that need to be read:
  - Use `sp_spaceused, dbcc tablealloc`, or `sp_estspace`, or
  - Use `set statistics io on`
- Determine number of pages your system can read per second; divide total pages by that number
- Example:
  - `dbcc checktable` gives table size of 9,372 pages
  - System reads 80 pages per second
  - 9,372 pages/80 pages per second = 117.2 seconds, or about 2 minutes
- Speed could improve if data is in cache

**Table Scan Performance**

Depends on

- size of the table
- speed of I/O, and
- cache size
  The larger the cache, the more likely that the table pages will be in memory because of a previous read

**When Table Scans Are Used**

In general, when there are no clustered indexes and when SQL Server determines that more than 20% of the data in the table will be returned

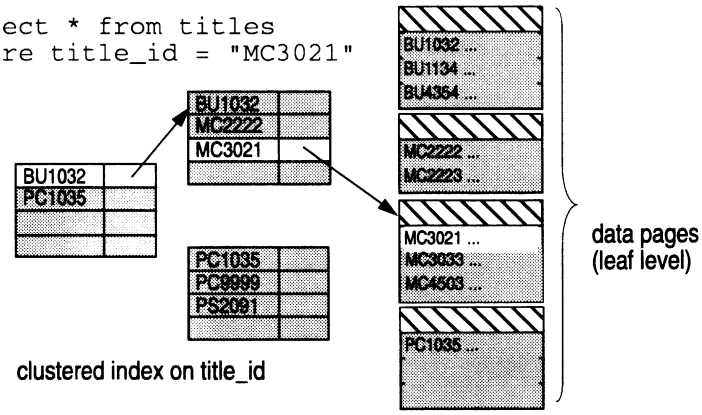
# Topics

- How indexes can affect performance
- Evaluating cost of table scans
- Evaluating cost of clustered access 
- Evaluating cost of non-clustered access
- Evaluating cost of covered access
- Choosing indexes

# Evaluating Cost of Clustered Index Access (Point Query)

Index provides address for specific page so that SQL Server does not have to scan entire table

```
select * from titles
where title_id = "MC3021"
```

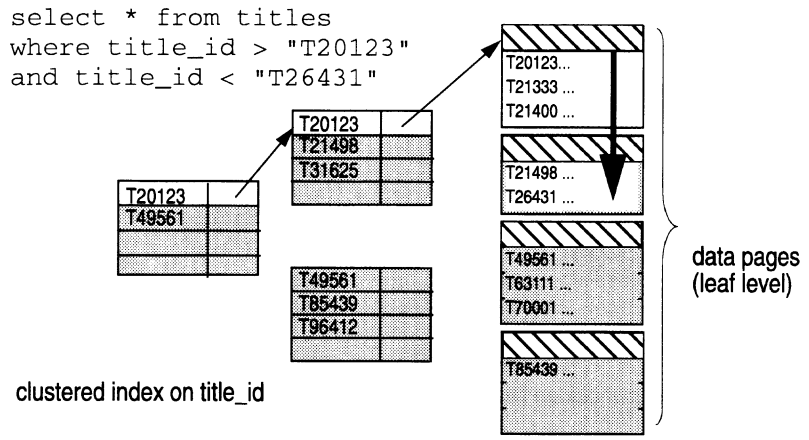



**Avoiding Table Scans**

When SQL Server determines that the number of index I/Os and data page I/Os will be less than the number required to read the entire table, it will use the index.

# Evaluating Cost of Clustered Index Access (Range Query)

Indexes can limit a table scan when a range of values is needed




**Limiting Table Scans** If the data is in order (or clustered), the index can be used to determine where to begin and end the table scan.

# Evaluating Cost of Clustered Index Access (Range Query I/O)

- Sample query:

```
1> select * from titles
2> where title_id > "T20123"
3> and title_id < "T26431"
4> go
```

- I/O for table scan:

```
Table: titles  scan count 1,  logical reads:
624,  physical reads: 0
Total writes for this command: 0
```

- I/O using clustered index:

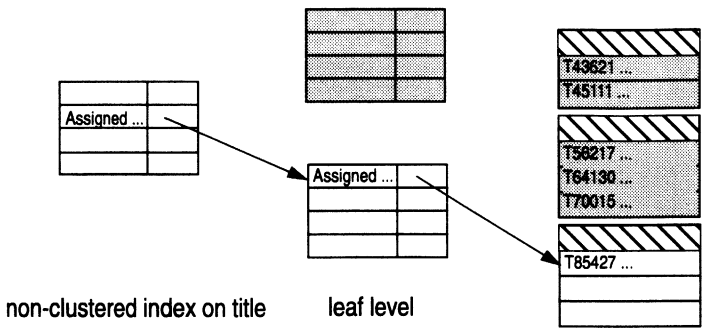
```
Table: titles  scan count 1,  logical reads: 4,
physical reads: 0
Total writes for this command: 0
```

# Topics

- How indexes can affect performance
- Evaluating cost of table scans
- Evaluating cost of clustered access
- Evaluating cost of non-clustered access 
- Evaluating cost of covered access
- Choosing indexes

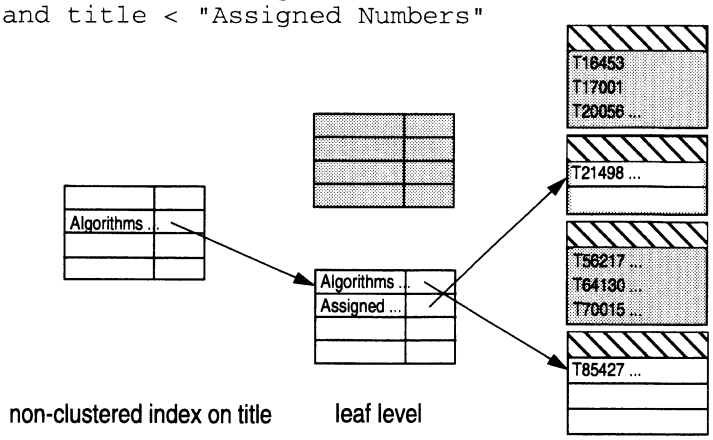
# Evaluating Cost of Non-clustered Index Access (Point Query)

```
select * from titles
where title = "Assigned Numbers"
```



# Evaluating Cost of Non-clustered Index Access (Range Query)

```
select * from titles
where title > "Algorithms in C"
and title < "Assigned Numbers"
```



# Indexes and I/O Statistics

- Sample query:

```
1> select * from titles
2> where title > "Algorithms in C"
3> and title < "Assigned Numbers"
4> go
```

- I/O for table scan:

```
Table: titles  scan count 1,  logical reads:
624,  physical reads: 0
Total writes for this command: 0
```

- I/O using non-clustered index:

```
Table: titles  scan count 1,  logical reads: 4,
physical reads: 0
Total writes for this command: 0
```

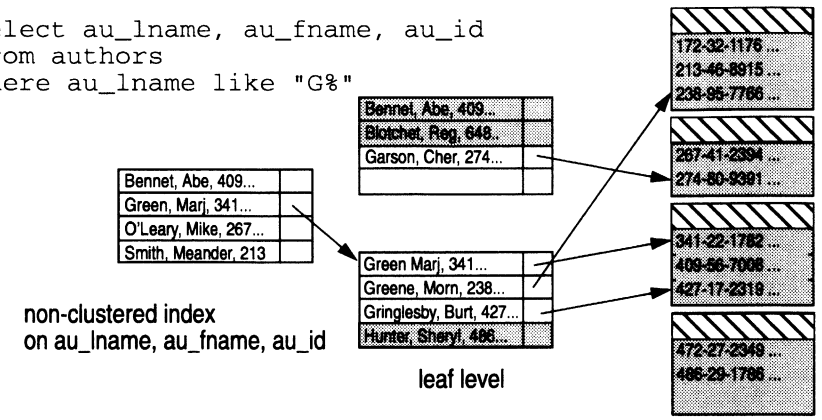
# Topics

- How indexes can affect performance
- Evaluating cost of table scans
- Evaluating cost of clustered access
- Evaluating cost of non-clustered access
- Evaluating cost of covered access 
- Choosing indexes

# Index Covering

- A query is *covered* if a non-clustered index contains all the information required by the query
- No data pages are read

```
select au_lname, au_fname, au_id
from authors
where au_lname like "G%"
```




**Index Covering**

In index covering, the index page contains all of the necessary data, and the Server does not need to access the data page.

**Clustered versus Non-clustered**

A non-clustered index will be faster than a clustered index if the non-clustered index covers the query.

# Evaluating Cost of Covered Access

- I/O results with an index that does not cover the query

```
Table: authors scan count: 1,logical reads: 4,
physical reads: 0
Total writes for this command:  0
```

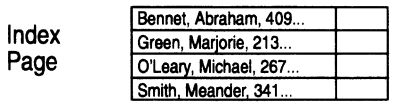
- I/O results with an index that does cover the query

```
Table: authors scan count: 1,logical reads: 1,
physical reads: 0
Total writes for this command:  0
```

# Index Covering: Adding Columns

- A non-clustered index will be faster than a clustered index if the non-clustered index covers the query
- You can add columns to non-clustered indexes to cover common queries
- Example:

```
create index au_comp
on authors(au_lname, au_fname, au_id)
```




- An index entry that is too wide, however, increases the number of levels and index pages to traverse

**Advantages**

- A dense composite index provides many opportunities for index covering.
- A composite index will probably return fewer records than a query on any single attribute.
- A composite index is a good way to enforce uniqueness of multiple attributes.

**Disadvantages**

- Composite indexes tend to have large entries, which means fewer index entries per index page and more index pages to read.
- An update to any attribute of a composite index will cause the index to be modified.
  The columns you choose should not be ones that are updated often.

# Topics

- How indexes can affect performance
- Evaluating cost of table scans
- Evaluating cost of clustered access
- Evaluating cost of non-clustered access
- Evaluating cost of covered access
- Choosing indexes 

# Choosing Indexes: Tables Without Indexes

- Do we ever want non-indexed tables?
- Useful for very small tables (1 to 5 pages)
- Useful for small- to medium-sized tables that require
  - No direct access to a single random row
  - No ordering of result sets
- Useful for tables that will have non-unique rows and the above characteristics

**Issues with Heaps**

- Will get table scans for all activity (except insert).
  If a non-clustered index exists, all data pages might not be scanned in order to located the row(s).
- May get locking and contention problems on the last page of the non-clustered table
- Consumes more storage than is necessary

**Potential Impact**

- Slow performance
- Space problems

**Reclaiming Space**

If a heap has many holes, create a clustered index for it (if there is space), then drop the index to reclaim space

# Choosing Indexes: Clustered

- Choose indexes based on the kinds of *where* clauses or joins you will be doing
- Good candidates for clustered indexes:
  - Columns that are accessed by range (*x<=column<=y*)
  - Columns with a high level of duplicates
  - Columns used by *order by* or *group by*
- Possible candidates for clustered indexes
  - Columns used in joins
  - Primary key, only if used for range query
- A column that is updated frequently may not be a good choice because of maintenance costs

**Issues**

- Does clustering create a hot spot for inserts?
- Are page splits occurring?
- Is the fill factor set appropriately to avoid splits?

# Clustered Indexes: Guidelines

- Have clustered indexes support major range processing
- If a hot spot occurs, expand the index with an artificial key or cluster on another column
- When there is a lot of insert activity, use fill factors to leave free space in data pages for future inserts

# Choosing Indexes: Non-clustered

- Considerations
  - How selective are they?
  - How much space will they take up?
  - How volatile is the candidate column?
- Because of overhead, add non-clustered indexes only when helpful
- Candidates:
  - Point queries
  - Queries accessing less than 20% of table
  - Columns used for selective join
  - Columns used for *order by* and *group by*
  - Primary Key

# Choosing Indexes: Covered

- Do we need clustered-like access to a column in a table where the clustered index is already used by another column?
  - For a range query
  - For a high-duplicates query
  - For an *order by* clause
- Do we have a read-only or read-mostly table, and plenty of space?
  - The more indexes, the better
  - Covered are the best
  - Remember the time taken to load table or create indexes

# Lab 5 – Indexes and Performance

- Try queries that use different indexes
- Observe the differences in performance

# Summary

- Evaluating the cost of table scans and comparing to index access cost
- Evaluating the cost of clustered and non-clustered index access
  - Point query
  - Range query
  - I/O statistics (set statistics i/o on)
- Covered indexes (non-clustered)
  - I/O statistics reduced due to access in the index only
- Choosing non-clustered tables (heaps)
  - Very small tables
  - Tables with no direct random access or ordering requirement
- Choosing clustered indexes
  - Range queries
  - Columns with high level of duplicates
  - Columns used by *order by* and *group by*
  - Join columns
  - Primary key columns
- Choosing non-clustered indexes
  - Point queries
  - Queries accessing less than 20% of table
  - Columns used for selective joins
  - Columns used for *order by* and *group by*
  - Columns that need to be covered

6

# Query Optimization

## System 10 Performance & Tuning

Version 2.2




*The Enterprise Client/Server Company™*

# 6

# Objectives

- Explain how the query optimizer processes queries
- Learn how to read **showplan** output
- Know how to rewrite problematic queries to improve performance
- Manage procedures with respect to optimization and recompilation

# Why Study the Optimizer?

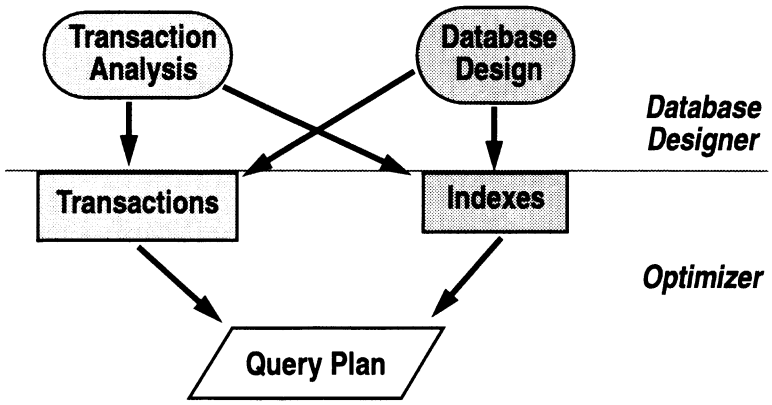
- To choose better indexes
- To write better queries
- To detect performance problems

# Common Symptoms of Query Trouble

- A query runs much slower than you predicted, based on indexes and table size
- A query runs much slower than similar queries
- A query suddenly starts running much slower
- A query processed within a stored procedure takes longer than when it is processed as an ad hoc statement
- The query plan shows the use of a table scan when a supposedly appropriate index exists

# What Does the Query Optimizer Do?

- Finds the "best" query plan (least costly in terms of time)




**What Is a Query Plan?**

The ordered set of steps required to carry out the query, complete with access methods chosen for each table.

**What Is an Optimizer?**

The optimizer selects an appropriate query plan in order to reduce the total time needed to process a query based on the **current contents** of the tables being queried.

**Why Learn About the Optimizer?**

- The goal of the database designer is to provide the optimizer with appropriate index choices **given the commands to be performed**.
- The person who write queries can sometimes improve performance by understanding the optimizer

**A Perfect Optimizer...**

- Has perfect knowledge of indexes and data
- Has an infinite amount of time to decide
- Can always choose the best query plan for the situation

# Query Plans

- Consist of retrieval tactics and order of execution
  - tables required
  - indexes used
  - order of joins
  - steps required for execution (inserts, sorts, updates...)
- Criteria used to determine query plan
  - Minimize logical page-accesses
  - Minimize physical I/Os
  - Minimize CPU time

# Displaying Query Plans

- Displaying query plans

```
set showplan on
```

- Can be used in conjunction with `set noexec on`, which prevents the SQL statements from being executed
- Example

```
set showplan on
set noexec on
go
SELECT au_lname, au_fname
   FROM   authors
   WHERE au_id = 'A1374065371'
go
```

## Displaying Query Plans (continued)

- If there is no index on authors table (or none is chosen):

```
STEP 1
The type of query is SELECT.
FROM TABLE                    <-- this table first
authors
Nested iteration
Table Scan                    <-- reads all pages
```

**STEP 1**

This simple query has only one step; some queries, such as aggregates, have multiple steps; more steps do not imply slower queries or poor performance.

**FROM TABLE**

This shows the tables to be used in the order they will be accessed.

**Nested iteration**

This means Sybase will "iterate" or loop through the table, either by reading the next row sequentially, or finding each next applicable row by using some index.

**Table Scan**

All of the pages in the table will be scanned, looking for rows that match the search criteria.

## Displaying Query Plans (continued)

- If the optimizer uses a non-clustered index on authors (au_id)

```
STEP 1
The type of query is SELECT.
FROM TABLE
authors
Nested iteration
Index: au_id_index
```

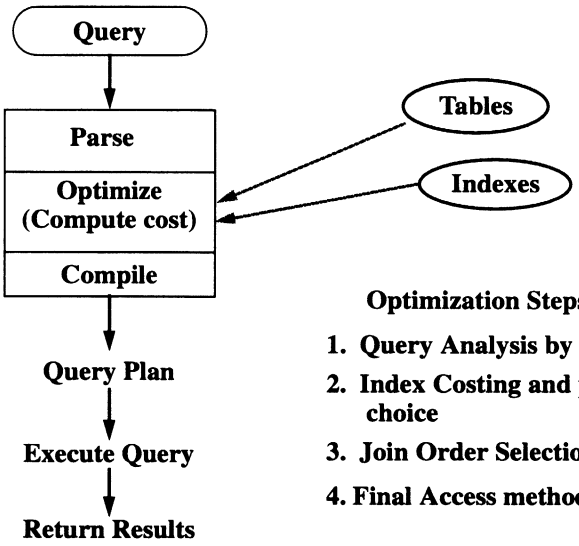
**use this (non-clustered) index**

- If the optimizer uses a clustered index on authors (au_id):

```
STEP 1
The type of query is SELECT.
FROM TABLE
authors
Nested iteration
Using Clustered Index
```

**use clustered index**

# What Happens When You Run a Query?




**Optimization Steps**

1. **Query Analysis by table**
2. **Index Costing and preliminary choice**
3. **Join Order Selection**
4. **Final Access method choice**

# Class Exercise: showplan

- For the following queries, set showplan and noexec on

```
select title, price, advance
from titles_idpr
where advance < 1000

select title, price, advance
from titles_idpr
where title_id = "T81002"

select title, price, ta.au_id, royaltyper
from titles_idpr t, titleauthor_ididtid ta
where t.title_id = ta.title_id
and price <10
```

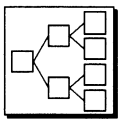
**Note**

For a multi-table query, the order in which the tables are listed in **showplan** output (top to bottom) is the order in which the tables were accessed.

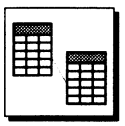
# Query Optimization Steps

1. Analyze each table


2. Select best index per table (preliminary)



3. Cost all possible join orders



4. Select best order and index combination

# Analyze Each Table

SARG?
OR?
Join?

- Find/generate SARGs
- Find OR clauses
- Find index covering possibilities
- Find/generate join clauses

# Search Arguments (SARGS)

- SARGS are expressions in the *where* clause that can potentially be resolved using an index
- Format: **<column> <operator> <constant>**,
  where *operator* is one of =, >, <, >=, <=
- Examples of SARGS

```
name = 'Smith'
30000 > salary
salary < 60000
a = 1 AND c = 7
```

- Examples which are *not* SARGS

```
salary = commission
dept != 10
salary * 12 >= 30000
a = 1 or c = 7
```

**Conjunctions**

A SARG can be part of a conjunction in which terms are linked with an AND.

Any query containing an OR clause *cannot* be a SARG (from the optimizer's perspective).

# SARG Equivalents

- Certain SQL constructs are turned into SARGs by the optimizer
- BETWEEN is evaluated as >= and <=

```
   salary between 10000 and 15000
becomes
   salary >= 10000 and salary <= 15000
```

- LIKE is evaluated as > and <

```
   name like 'Sm%'
becomes
   name >= 'Sm' and name < 'Sn'
```

- A constant expression is evaluated as a single constant

```
   salary > 3000 * 12
becomes
   salary > 36000
```

- How would these be converted?

```
1) price BETWEEN 19.95 and 29.95

2) title LIKE 'Wa%'

3) price > 2 * 15.95

4) title like '%The Sequel'

5) price < 2 and price > 4
```

# Padding With SARGs

- You can "pad" your query with redundant SARGs to give the optimizer more choices
- Give the optimizer as much information *per table* as possible

| | Query | Logical Reads | Phys Reads |
|---|---|---|---|
| 1 | SELECT t.title_id, title<br>FROM titles t, titleauthor ta<br>WHERE t.title_id =<br>ta.title_id<br>AND t.title_id = 'T81002' | 3 | 2 |
| 2 | SELECT t.title_id, title<br>FROM titles t, titleauthor ta<br>WHERE t.title_id =<br>ta.title_id<br>AND ta.title_id = 'T81002' | 6 | 3 |
| 3 | SELECT t.title_id, title<br>FROM titles t, titleauthor ta<br>WHERE t.title_id =<br>ta.title_id<br>AND t.title_id = 'T81002'<br>AND ta.title_id = 'T81002' | 3 | 2 |

- Given that titles and titleauthor are in a 1:M relationship, and title_id is unique in the titles table, Query 1 will often be better than Query 2 since it will be more selective
- Query 3 is preferable, however, because it causes the optimizer to view both tables as potential sources for satisfying the SARG, allowing for more flexibility in the upcoming optimization step of determining join order

# Lab 6a – Search Arguments

- Analyze queries to identify SARGs
- Determine if a SARG is supported by an index

# OR Clauses and Equivalents

SARG?
OR?
Join?

- The query optimizer also searches for OR clauses

  <search clause> OR <search clause> ...

  (All columns must belong to the same table)

- The optimizer generates OR clauses from an IN list
  **<column> in (constant, constant,...)** becomes
  **<column> = <constant> OR <column> = <constant> OR...**

- Two examples:

```
select * from titles
where (type = 'business' and price < 15)
or title like 'Assigned%'

select * from authors
where au_lname in ('Berry', 'Densham')
```

# Resolution of OR Clauses

- OR clauses are resolved either with a table scan or the special "OR strategy"
- Table scan is chosen unless *each* of the OR clauses can be supported by an index

SARG?
OR?
Join?

- 
- OR Strategy?
  - Index used once for each qualification and results merged together
  - Each row appears only once in the output no matter how many times it qualifies, requiring elimination of duplicates

- What (hypothetical) indexes could be used for each of these queries?

1. 
```
select * from authors
where au_lname in ("Berry", "Densham")
```

2. 
```
select * from titles
where price <15
or title like "Assigned%"
```

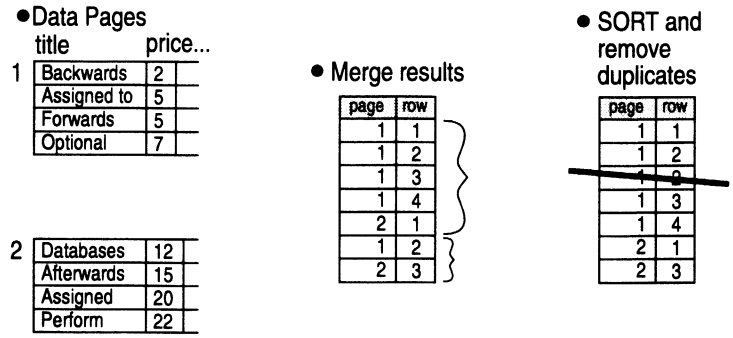
In the first example, only one index could be used, on *au_lname*. In the second case, the optimizer could use two indexes, on price and title.

Query Optimization

# OR Strategy Methodology

SARG?
OR?
Join?

```
Select * from titles_pridtitl
where price < 15 or title like "Assigned%"
```




- Use final list to retrieve data rows

**RID Retrieval** For each iteration through the data using the appropriate index, page and row ids are stored in tempdb.

**RID Manipulation** These lists of identifiers are UNIONed – they are sorted, and duplicates are removed.

**Dynamic Index** The final list is called a "dynamic index" and is used to retrieve the required data rows.

# OR Strategy Showplan

- Select * from titles_pridtitl
  where price >500.0 or title = "Assigned Numbers"
- Query Plan

```
STEP 1
The type of query is SELECT
FROM TABLE
titles_pridtitl
Nested Iteration
Using Clustered Index
FROM TABLE
titles_pridtitl
Nested Iteration
Index:idx3
FROM TABLE
titles_pridtitl
Nested Iteration
Using Dynamic Index
```

# Lab 6b – OR Strategies

- Examine and explain the query plans of certain OR queries

# Index Covering

- A query is said to be ***covered*** by an index if the index contains all the information necessary to satisfy the query
  - Applies to non-clustered indexes only (dense indexes)
  - Server will *not* access data pages if query is covered
- Consider the following query

```
SELECT au_lname, au_fname
FROM   authors
WHERE  au_lname like 'Berry%'
```

- Which index would cover the above query?

```
CREATE UNIQUE CLUSTERED INDEX au_index_1
ON authors (au_id)
```

or

```
CREATE INDEX au_index_1
ON authors (au_lname, au_fname)
```

## Index Covering (continued)

- Query:

```
select au_lname, au_fname from authors
where au_lname like 'B%'
```

- Non-clustered index on au_id, au_lname, au_fname:

```
scan count: 1, logical reads: 91, physical
reads: 0
```

  Index covered the query

- Clustered index on (au_id) - data scan needed:

```
scan count: 1, logical reads: 223, physical
reads: 0
```

  The table had to be read

**Conditions for Index Covering**

- All columns in condition clauses are in the index, **and**
- All select list items are in index

# How Aggregate Queries Are Optimized

- Processed like all other queries, except that they require a second step to present the aggregate answer to the client
- Can be optimized to use a non-clustered index instead of a table scan in many cases
- Examples

```
select count(*) from titles
```

  - Would count entries in the shortest non-clustered index's leaf level instead of counting data rows

```
select count(*) from titles
where title > 'T'
```

  - Would use a non-clustered index on title instead of scanning the entire table

# Aggregate Query Plan Example

- Assume: non-clustered index title_type_index on type
- Select count (*) from titles where type = "business"

```
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
titles
Nested iteration
Index : title_type_index
```

Use index to compute the number of rows

```
STEP 2
The type of query is SELECT
Table Scan
```

Return the result to the user

**Scalar Aggregates** This term implies each aggregate is being formed for the entire set of qualifying rows.

**Vector Aggregates** Vector Aggregates are aggregates which produce numbers for subsets of qualifying rows, as in a "group by" statement.

These will be covered later.

Covering is helpful for these also.

# Aggregates: Special Cases

- Slower performance if max and min are in the same select:

```
select max(price) from titles_idpr
select min(price) from titles_idpr
```

  - Will use index on price in either case to find highest or lowest index values

```
select min(price), max(price) from
titles_idpr
```

  - Will do a scan of the entire leaf level just to pick up first and last (but ignores index)

# Lab 6c – Index Covering and Aggregates

- Compare I/O required to execute queries that are "covered" vs. queries that are not
- Determine what indexes would cover a certain query
- Demonstrate the effect aggregates have on performance

# Review

- How would the optimizer analyze these queries?

```
1. select avg(price) from titles_idpr

2. select pub_id, avg(price)
   from titles_idpr
   group by pub_id

3. select title, price from titles_idpr
   where title_id = 'T81002' or price < $8.00

4. select title, price from titles_idpr
   where price in ($5.99, $6.99, $10.99)

5. select title, price from titles_idpr
   where title like "A%" or price < $8.00
```

**SARGs** The optimizer looks for appropriate indexes for each SARG.

**ORs** If every SARG has an index available, the optimizer combines results and eliminates duplicates.

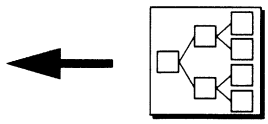
**Covering** If the leaf level of a non-clustered index contains the required columns, the optimizer always prefers scanning that leaf level rather than scanning the data pages.

**Aggregates** Aggregates always require a STEP 2 to return results

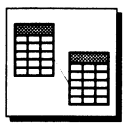
## Query Optimization Steps

1. Analyze each table


2. Select best index per table (preliminary)



3. Cost all possible join orders



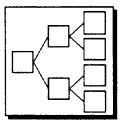
4. Select best order and index combination

**Selecting indexes**

If no joins are done, the next step (after finding SARGs, etc.), the next step is selecting indexes.

If a join is involved, the index selection is preliminary--to be determined in conjunction with join order (Step 3).

# Selecting an Index



- For each search argument (SARG), join, OR etc., the optimizer will choose the most cost-effective index and evaluate it against no index at all (a scan)
- The query optimizer evaluates the following:
  - Is there an index which matches each clause produced by query analysis?
  - How many rows will satisfy the clause?
  - How many pages will be required to find the qualifying rows?
- In the end, the optimizer chooses a scan or ONE index, even if the table has several (the OR strategy is the only exception)

# Row Estimates Using Index Access

- Optimizer estimates the number of rows to be returned by each index
- If statistics are available, the optimizer estimates using the data distribution information
- If no statistics exist, the optimizer uses default percentages for equality and range searches
  - 10% of the table for equality (=)
  - 25% of the table for a closed interval (col > y and col < z)
  - 33% of the table for open end range (>, >=, <, <=)
- If index is defined as unique, optimizer knows only one row will be returned. Distribution statistics are not needed.

**How Do Statistics Get There?**

They exist if:

- created index after table data inserted
- executed update statistics

**Details on Statistics**

- Put the most useful, most selective column first in your **create index** command statement
- If there are no statistics, the optimizer will make its own estimates based on default numbers
- If the statistics are inaccurate, the optimizer may pick the wrong indexes

# When Will the Distribution Page Not be Considered?

- Distribution statistics will not be used if data type of SARG does not match the data type of the index column, or if the SARG value is unknown at the time of optimization
- The default statistics will be used, and the optimizer may overlook a valuable index
- The type mismatch may be subtle:

```
select * from tableA where float_col = 10
```

float

int

```
create tableB(c1 char(10) null, ...)
create proc p @arg char(10)
as
select * from tableB where c1 = @arg
```

does not allow nulls

allows nulls

## Estimating Number of Logical Page Reads

- The query optimizer estimates how many pages to read, based on its estimates of the number of qualifying rows
- If no index, it estimates number of data pages in the table
- If clustered index, it estimates
  - Number of index levels + number of data pages
  - Number of data pages = number of qualifying rows / rows per data page
- If non-clustered index, it estimates
  - Number of index levels + number of leaf pages + number of qualifying rows
- If index covering, it estimates
  - Number of index levels + number of leaf pages

1 row per page does (worst case)

**Unique indexes**

If the index is unique and the query is a search for match on all parts of the index, the optimizer estimates 1 page + the number of index levels.

# Lab 6d – Predicting Index Usage

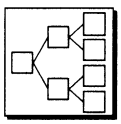
For several queries,

- Predict whether indexes will be used,
- Execute the query and observe whether indexes were used,
- And explain why certain indexes are chosen

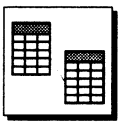
# Query Optimization Steps

1. Analyze each table


2. Select best index per table (preliminary)

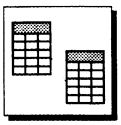


3. Cost all possible join orders



4. Select best order and index combination

# Join Clauses



- The optimizer also looks for joins

  ```
  tableA.col1 <operator> tableB.col2
  ```

- Joins always involve two tables on either side of the operator
- Join operators

  =, >, >=, <, <=, !>, !<, !=, <>, *=, =*

- Where are the join clauses in the following query?

```
SELECT title, price
FROM   titles t, publishers p
WHERE  t.pub_id = p.pub_id
AND    p.pub_name = 'Prentice-Hall'
AND    t.price BETWEEN 9.95 and 19.95
```

- Columns being joined must match in datatype and nullability for an index to be used

**Matching Datatypes** Implicit conversions of datatypes still allow indexes to be used

**Primary and Foreign Keys** Make sure primary and foreign keys match, especially with respect to nullability

# Generating Join Clauses

- Nested selects may be "unwound" to a join
- Example

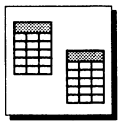
```
select title from titles
where  title_id in
   (select title_id from titleauthor
   where royaltyper > 50)
```

- Join equivalent

```
select title from titles T, titleauthor TA
where t.title_id = TA.title_id
and royaltyper > 50
```

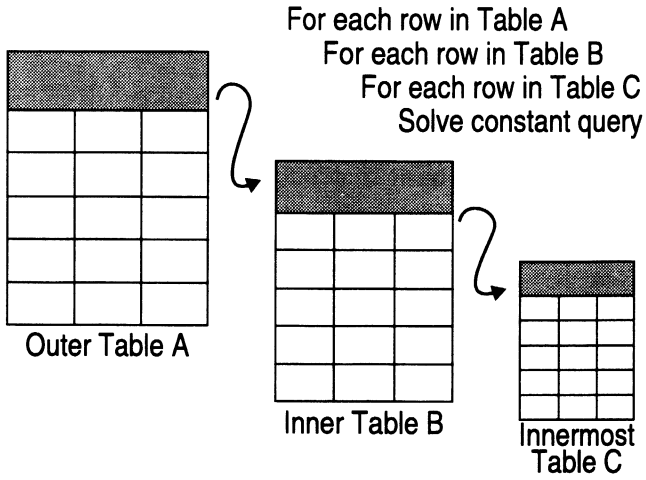
* Subqueries worden onder System-10 e.v. niet meer platgeslagen naar een join!

# Join Clauses (continued)



- Why does the optimizer look at join clauses?
  - To determine the order in which to process the tables
  - To determine the best indexes to use
  - To decide if additional optimization steps are necessary (reformatting)

## Nested Iterations




- Optimizer picks table order based on total cost estimates

**Nested Iteration**

1. Construct a set of nested loops
2. Each nesting level uses one table and only one index
3. At each level, for each row returned, probe the next table

# Nested Iterations Example

```
SELECT title
FROM   titles T, titleauthor TA
WHERE  T.title_id = TA.title_id
AND    TA.royaltyper > 50
```

- Which of the following approaches would perform better?

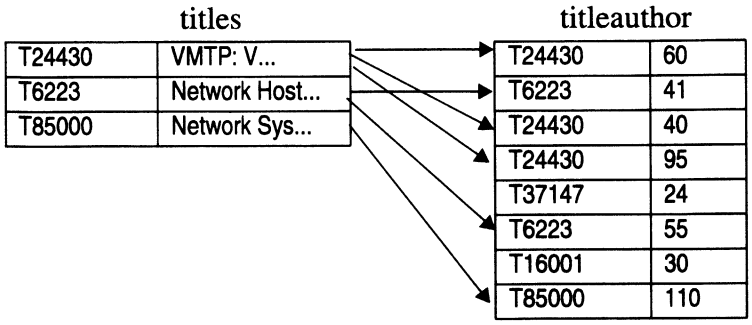
```
For each row in titles
    Get title_id and title
    Match title_id to TA.title_id
    Check TA.royaltyper > 50
```

vs.

```
For each row in TA where royaltyper > 50
    Get title_id
    Match title_id to T.title_id
    Get title
```

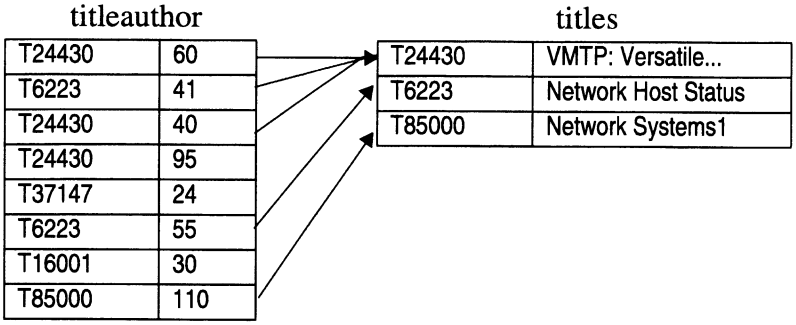
# Nested Iterations: First Approach

- Assume titles is outer table:
  - Get next row of titles; get value of title_id
  - Using index on title_id, locate each row in titleauthor
  - Compare value of royalty, print if > 50
  - Get next row of titles, etc.



# Nested Iterations: Second Approach

- Assume titleauthor is outer table:
  - Get next row of titleauthor where royaltyper > 50
  - Use index on title_id to search titles and print rows
  - Evaluate next row of titleauthor, etc.



# Class Exercise: Which Solution Would Be Better...?

1. If one table had no indexes on the join column?
2. If both tables have indexes on the join columns?
3. If both tables have indexes on the join columns, but one table has fewer rows?
4. If both tables have indexes on the join columns, and one of them also has a usable index on a SARG?

**1** Try joining titles (no index) to titles_pridtitl (non-clustered index on title_id).

**2** Join titles_pridtitl to titles_idpr which have non-clustered and clustered indexes respectively.

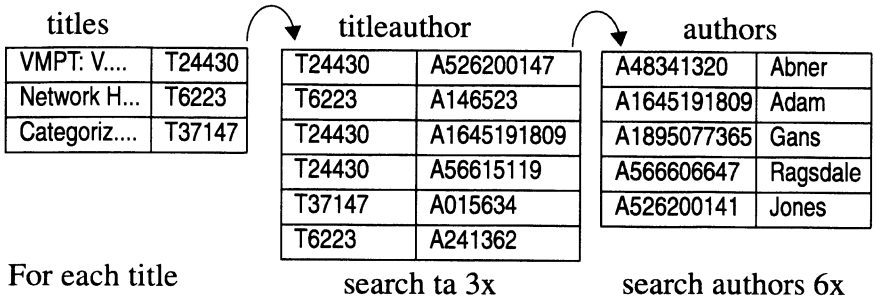
**3** Join titles_pridtitl to titleauthor_ididtid which each has non-clustered indexes on title_id.

**4** Join titles_idpr (clustered on title_id, non-clustered on price) with titleauthor_ididtid (non-clustered on title_id) where price between 6.0 and 8.0.

# Three-Way Join: First Approach

```
select t.title, a.au_lname
from titles t, authors a, titleauthor ta
where t.title_id = ta.title_id
and a.au_id = ta.au_id
and a.au_lname = 'Adam'
```

- Approach: titles, then titleauthor, then authors:



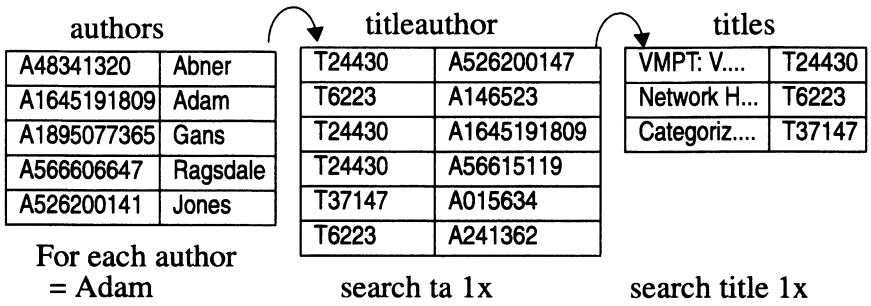

**General Nested Iteration Effect**

With multiple tables in a query, the cumulative I/O generally increases as we go left to right, or into the inner-most tables.

# Three-Way Join: Second Approach

```
select t.title, a.au_lname
from titles t, authors a, titleauthor ta
where t.title_id = ta.title_id
and a.au_id = ta.au_id
and a.au_lname = 'Adam'
```

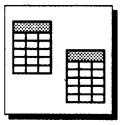
- Approach: authors, then titleauthor, then titles:




**Importance of Outer Table**

Note that in just these small tables, authors is searched 6 times in the first case and only once in the second case. This is because in the second case we have effectively reduced the number of qualifying rows in the outermost table.

# Outer Joins

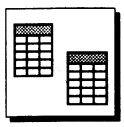


- An asterisk on either side of the join operator implies that every row on that side must be present in the final output regardless of whether it joins to inner tables.
- Example:
  There may be some authors in the authors table who have not written any books carried by us. The following will retrieve even those authors:

```
select au_lname, au_fname, title_id,
royaltyper from authors a, titleauthor ta
where a.au_id*=ta.au_id
```

- The table on the asterisked side of the operator will always be accessed first, that is be the first table in the series of nested iteration loops.

## Row Estimates for Join Clauses



- Join ***selectivity*** estimates how many rows from a particular table will join with a row from another table
- If no statistics, selectivity = 1/# rows in smaller table
- If statistics are available, join selectivity is based on the *density* of the index: the average percentage of duplicate rows
  - Unique index has low density and is highly selective
  - Index with a large number of duplicates has high density and is *not* very selective

**Selectivity Example**

Assume two tables: *employee* and *department*. 100 rows in *department*, 1000 rows in *employee*

Clause: where department.deptno = employee.deptno

Given a row in *department*, the number of rows in *employee* that join it are 1000 * .01 = 10.

Given a row in *employee*, the number of rows in *department* that join it are 100 * .01 = 1.

**Density**

Density describes the average percent of duplicates in an index.

Density is 1 if all data values are the same and 1/n if every data value is unique.

**Density Example**

Thus the estimated selectivity when joining a 500 row table with 20% duplicate records is 0.20, so SQL Server would expect to get 100 (20% of 500) rows back per scan.

# What Is On a Distribution Page?

Given:

- title_id varchar(6)
- 1000 rows in table
- create index **idxl** on titleauthor(title_id, au_id)

Distribution page

- Number of steps = available bytes in page / bytes in column

  = 2016 / 6 = 336 steps

- 1000 rows / 336 steps = **3 rows per step**

**titleauthor**

| title_id | au_id | ... |
|---|---|---|
| T137 | A120 | |
| T139 | A221 | |
| T140 | A131 | |
| T140 | A439 | |
| T140 | A190 | |
| T140 | A791 | |
| T140 | A998 | |
| T140 | A998 | |
| T141 | A101 | |
| ... | ... | |
| T999 | A101 | |

**distribution page**

| header | |
|---|---|
| **step** | **data value** |
| 0 | T137 |
| 1 | T140 |
| 2 | T140 |
| ... | ... |
| 326 | T999 |

- distinct data values for title_id = 571
- title_id density = 1/571 = **.00175**
- distinct data values for title_id + au_id = 920
- title_id + au_id density = 1/920 = **.00108**

# Composite Index Density on Distribution Page

- SQL Server stores densities for each subset of columns in a composite index
  - Density varies between 1/(no. of rows in table) and 1.0 (100%)
  - 1.0 means *all* duplicates, every row meets the given value
- Example:

```
CREATE UNIQUE INDEX stor_ord_x
ON salesdetail (store_id, ord_num, title_id)
```

| saves densities for: | sample values*: |
|---|---|
| (store_id) | 0.02 |
| (store_id,ord_num) | 0.00005 |
| (store_id,ord_num,title_id) | 0.00001 |

*Assume 50 different stores, 5 items/order, 100,000 rows in the table, 20 rows/page

**Distribution Pages**

Densities are stored in the **distribution pages,** which include the index densities and a set of buckets that represent a histogram of the data values referenced by the index (at the time the statistics were generated).

If there are composite indexes, more densities must be stored, which leaves less space to store the histogram information.

**General Formula**

density = ~~(1/distinct_values)/rows_in_table~~

(rows_in_table/distinct_values)/rows_in_table

or 1/distinct_values

# Costing Calculations: Example

- Query:

```
select
st.stor_name,sd.ord_num,sd.title_id,sd.quantity
from salesdetail sd, sales s, stores st
where st.state="CA"
and st.stor_id=s.stor_id
and s.stor_id=sd.stor_id
and s.ord_num=sd.ord_num
```

- Assume indexes on stores (stor_id), sales (stor_id, ord_num) salesdetail (stor_id, ord_num, title_id)
- If there is an index on st.state with a statistics page, optimizer will use distribution steps to estimate "California" stores. Else, optimizer will use its 10% for equality default.
- Densities are used to quickly estimate the number of orders per California store as well as the number of items per order.

**Note** Distribution steps are helpful when an index value is known. Densities are helpful in figuring I/O costing for general join work.

# Nested Iterations Performance

- Based on how many times inner queries are performed
- Order is affected by restrictions on tables (WHERE clause)
- Densities are used to estimate how many rows will be returned for each nested iteration

# Nested Iterations-Logical I/O Estimates

| Table | Rows | Row Size | Data Pages |
|---|---|---|---|
| t | 5000 | 250 avg | 620 |
| ta | 6250 | 22 | 68 |

Select t.title from titles t, titleauthor ta
where t.title_id = ta.title_id and royaltyper >50

| Order | Logical Reads |
|---|---|
| t,ta | 620 + (5000 * 2) = **10,620** |
| ta,t | 68 + (2062 * 2) = **4,192** |

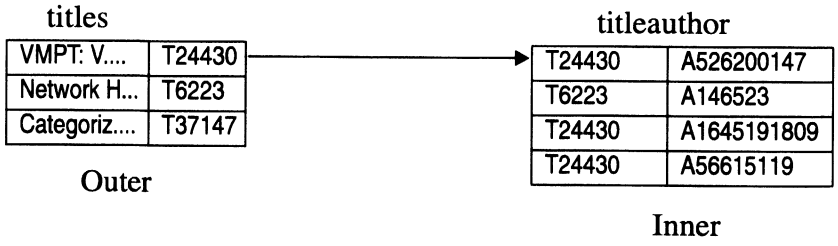
clustered index available on both join columns

**Logical I/O Formula**

To determine the number of logical reads:

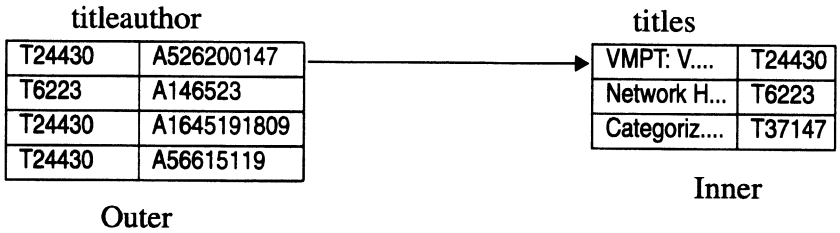
# of pages in outer table to be scanned
+ (# of qualifying rows in outer table
* # pages in inner table to be scanned)

# Nested Iterations: I/O Calculation Examples




- Clustered index on titleauthor.titleid
  Title as outer table:

  Logical I/O = 620 + (5000 * 2) = 10,620
  (estimating 2 reads of titleauthor for each row in titles - 1 index page, 1 data page)




- Clustered index on title.titleid
  Titleauthor as outer table:

  Logical I/O = 68 + (2062 * 2) = 4192
  (estimating 2 reads of titles for each row in titleauthor - 1 index page, 1 data page)

Titleauthor had 21% of its rows with royaltper >50
Without any statistical information available, the optimizer was using the default percentage of 33% for an outer range query

# Nested Iterations – Final Cost

| Table | Data Pages |
|---|---|
| t | 620 |
| ta | 68 |

| Table | Logical Reads |
|---|---|
| t,ta | 10,620 |
| ta,t | 4,192 |

| Order | Final Cost |
|---|---|
| t,ta | 10,620 (2ms) + 750 (18ms) = **34,748 ms** |
| ta,t | 4,192 (2ms) + 747 (18ms) = **21,906 ms** |

**Reads**

- Nested iteration performance is largely determined by the number of pages that must be physically read from the disk.
- Performance can be improved by either reducing the number of pages that must be read or by reading an entire table into memory.
- It is about 10 times more costly to read from disk than from memory.
- The Sybase optimizer figures 2ms for a logical read and 18ms for a physical read.
- During this final phase of query optimization the size of data cache memory is consulted.
- The optimizer makes an educated guess as to which tables might be retained in memory.
- Given our standard classroom configuration and the sizes of the above tables, the optimizer assumes both can be kept in memory.

**Final Cost Formula**

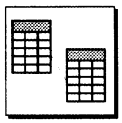
```
# logical reads * 2ms + # physical reads * 18ms
```

# I/O Costing

- Each possible table ordering is given a final cost
  - Number of logical I/Os is computed during preliminary index selection phase
  - Physical I/O is then considered during the join order permutation phase
- Costings are only estimates
  - Done ahead of actual run
  - Only as good as statistics available

update statistics !

# Finding the Optimal Join Strategy



- All possible join permutations are considered for 4 or fewer joined tables
- Optimizer takes short cuts and may not evaluate unpromising permutations

| Number of Tables | 1 | 2 | 3 | 4 | 5 | 6 | N |
|---|---|---|---|---|---|---|---|
| Number of Permutations | 1 | 2 | 6 | 24 | * | * | * |

* will be the **Best Plan** permutation

**Best Plan Permutations For More than 4 Tables (steps)**

A. Optimize each subset of 4 tables
B. Remember outer table from best plan involving 4 tables
C. Eliminate that one from tables in from clause
D. Optimize best set of 4 tables from remaining tables
E. Continue steps B through D until only 4 tables remaining

# Lab 6e – Join Strategies

- Determine join strategy optimizer will choose for a given query
- Evaluate effect of search arguments and indexes on this strategy

# Reformatting

- May be used when there are no indexes on either join column
- Reformatting creates a sorted temporary table in a work area at run time
- Worst-case join performance for two tables:

  P1 R1 P2

  Pages in table1+(Rows in table1*Pages in table2)

  - This is a scan of the inner table once for every qualifying outer row
- If P2 and R1 number in the 100's, this is slow even for small tables

# Reformatting (continued)

- Alternatives: a projected, restricted copy of T2 and possibly T1

  $\underbrace{P2 + P2\log_n P2}_{\text{sort cost}} + P1 + R1(1)$ sufficient for nested iteration

  **n** is number of elements you can merge
  minimum = 2

- Example: R1 = 300, P2 = 100, P1 = 200
  - No index: 200+300*100=30200 page accesses
  - Sort+nested iteration: $100+100*\log_2(100)+200+300*1=1300$

**Legend**

P1 = Pages accessed from Table 1

P2 = Pages accessed from Table 2

R1 = Rows in Table 1

R1(1) = Assumes 1 scan of Table 1's rows

Optimizer may then decide that it is worth using "reformatting" strategy as opposed to the other alternative.

**Sort Cost**

Sort cost of example is 100+100*log2(100) = 800

Query Optimization

## Reformatting Query Plan

```
STEP 1
The type of query is INSERT.
The update mode is direct.
Worktable created for REFORMATTING.
FROM TABLE
titleauthor
Nested iteration
Table Scan
TO TABLE
Worktable
```

Create a worktable from titleauthor

```
STEP 2
The type of query is SELECT.
FROM TABLE
titles
Nested iteration
Table Scan
FROM TABLE
Worktable
Nested iteration
Using Clustered Index
```

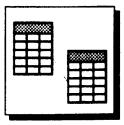
Outer table is titles, Inner table is worktable

**Query**

```
SELECT title
FROM   titles T, titleauthor TA
WHERE  T.title_id = TA.title_id
AND    price > 19.95
```

No indices defined for titles or titleauthor.

# Join Selection Review



- Steps
  - Determine best index for search clause
  - Determine best index for or clauses
  - Determine covering possibilities
  - Determine best index for join clause
  - Figure logical I/O for all above
  - Set up permutations of various table orders and various indexes
  - Estimate final costs, including taking into account physical I/O and logical CPU access times
  - Evaluate cost of reformatting
  - Pick best plan

# When Are Work Tables Used?

- In ORDER BY

| | |
|---|---|
| no index | work table created in tempdb and sorted |
| clustered index on ORDER BY column (ascending only) | no work table and no sort |
| non-clustered index applied to where clause | work table created, no sort |
| non-clustered index covering query | no work table needed, data is already sorted |

- In GROUP BY
  - work table is always created (regardless of indexing), index will be used to build the work table if one is available
- In DISTINCT
  - work table created (regardless of indexing) and sorted to find duplicates, index used if one is available

**Non-clustered Index Applied to where Clause** The work table is not sorted after being created because it is already sorted when it is created.

**Non-clustered Index Covering the Query** The `order by` statement has to be in order of the non-clustered index

# Sample Work Tables: Order By

- Query

```
select title from titles order by title
```

- Worktable generated with no index

```
STEP 1
The type of query is INSERT.
The update mode is direct.
Worktable created for ORDER BY.
FROM TABLE
titles
Nested iteration
Table Scan
TO TABLE
Worktable

STEP 2
The type of query is SELECT.
This step involves sorting.
FROM TABLE
Worktable
Using GETSORTED
Table Scan
```

- With clustered index on title, no worktable generated

```
STEP 1
The type of query is SELECT.
FROM TABLE
titles
Nested iteration
Table Scan
```

# Sample Work Tables: Group By

- Query

```
select title_id, price from titles
where title_id >= 'T810' and title_id <= 'T811'
group by title_id
```

- Worktable generated with no index

```
STEP 1
The type of query is SELECT (into a worktable).
GROUP BY
Vector Aggregate
FROM TABLE
titles
Nested iteration
Table Scan
TO TABLE
Worktable                                    <--
STEP 2
The type of query is SELECT.
FROM TABLE
titles
Nested iteration
Table Scan
FROM TABLE
Worktable
Nested iteration
Using Clustered Index
```

- With non-clustered index on title_id, worktable created, index used to build it

```
STEP 1
...(same as above)...
Vector Aggregate
FROM TABLE
titles
Nested iteration
Index : idx2                                 <--
TO TABLE
Worktable                                    <--
STEP 2
The type of query is SELECT.
FROM TABLE
Worktable
Nested iteration
Table Scan
FROM TABLE
titles
Nested iteration
Index : idx2
```

# Sample Work Tables: Distinct

- Query

```
select distinct au_id from titleauthor
where title_id like 'T8100%'
```

- Worktable generated with no index

```
STEP 1
The type of query is INSERT.
The update mode is direct.
Worktable created for DISTINCT.
FROM TABLE                          <--
titleauthor
Nested iteration
Table Scan
TO TABLE
Worktable
STEP 2
The type of query is SELECT.
This step involves sorting.
FROM TABLE
Worktable
Using GETSORTED
Table Scan
```

- With non-clustered index on title_id, worktable created, index used to build it

```
STEP 1
The type of query is INSERT.
The update mode is direct.
Worktable created for DISTINCT.     <--
FROM TABLE
titleauthor
Nested iteration
Index : idx2                        <--
TO TABLE
Worktable
STEP 2
The type of query is SELECT.
This step involves sorting.
FROM TABLE
Worktable
Using GETSORTED
Table Scan
```

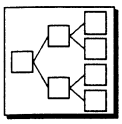
# Lab 6f – Work Tables and Sorts

- Identify queries requiring work tables, sorts, and reformatting
- Rewrite queries to change query plans
- Improve performance of queries by adding indexes
- Describe how required left outer joins limit plans

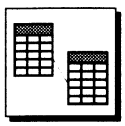
## Query Optimization Steps Summary

1. Analyze each table


2. Select best index per table (preliminary)



3. Cost all possible join orders



4. Select best order and index combination

**Selecting Best Plan**

- For each permutation, figure out which indexes and join strategies are best
- For each table, calculate logical and physical page accesses
- Take into account size of tables relative to cache
- Total cost = 2 ms * total logical reads + 18 ms * total physical reads

# Stored Procedure Optimization

- Parse trees, plans, and their execution
- Query plan confusion
- Query plan control
- Recompilation

**Parse Tree**

To produce parse tree, parser:
- Makes tokens of each word it finds
- Checks syntax
- Validates identifiers (object ids and keywords)

**Query Tree**

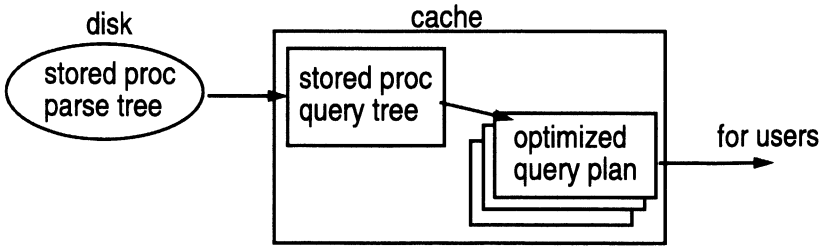
- Is the parse tree when it is brought into cache

**Query Plan**

- Is a query tree optimized for current statistics
- Is what is executed

## Stored Procedures and Cache

- The first time a procedure is run, its parse tree is loaded into the procedure cache (from sysprocedures), and the optimizer creates a query plan




- Procedure remains in cache for other users, as long as there is space
- Procedures are not re-entrant – only one user at a time
- If multiple users require the procedure simultaneously, they each get their own query plan

**Multiple Simultaneous Requests**

- If copies are not available, user gets a freshly optimized plan

**execute with recompile**

If procedure is executed **with recompile**, a freshly optimized query plan is added to the procedure cache.

# Misleading Initial Value Example

```
CREATE PROCEDURE GetBooksInPriceRange
(@low    money,
 @high   money
)
as
   SELECT title, price
   FROM   titles
   WHERE  price BETWEEN @low and @high
```

- Initial Execution:

  ```
  GetBooksInPriceRange 0.0, 100.0
  ```

  (uses table scan)

- Normal Execution:

  ```
  GetBooksInPriceRange 19.95, 29.95
  ```

  (should use different query plan)

**Problem**

The initial execution of the procedure selects a relatively large range of prices.

In this case, the optimizer may decide that a table scan is the most efficient processing approach, ignoring other indexes.

During subsequent executions, however, the range of prices selected may be significantly less and therefore the use of an index would be beneficial.

# Optimizing Stored Procedures

- By default, optimizer will optimize based on first run-time value
- Ways to change default values:

```
create procedure ... with recompile

execute procedure with recompile

sp_recompile table_name
```

**By Default**

The optimizer cannot determine actual distribution without some values, so it will optimize based on the first run-time values supplied.

**create ... with recompile**

- Re-optimizes at *every* execution
- Compilation overhead outweighs execution time for short running procedures
- Advantages outweigh disadvantages for procedures that take a long time

**execute ... with recompile**

- Redoes query plan for this execution
- Useful if you suspect plan is not optimized for current values
- May affect others who use that stored procedure. New user may get the copy left in cache

**sp_recompile**

- All stored procedures that use the table are recompiled next time they are run

# Optimizing Stored Procedures (Continued)

- Priming a Query Plan

```
CREATE PROCEDURE GetBooksInPriceRange
(@low    money,
 @high   money)
as
   SELECT title, price
   FROM   titles
   WHERE  price BETWEEN @low and @high

Execute
   GetBooksInPriceRange 19.95, 29.95
```

- Splitting a Query by Input

```
CREATE PROCEDURE GetBooksInPriceRange
(@low    money,
 @high   money)
as
   if @high - @low <5
      execute GetBooksInSmallRange @low, @high
   else
      execute GetBooksInWideRange @low, @high
```

# Recompiling Stored Procedures

- When are stored procedures recompiled automatically?
  - when any objects or indexes on any of the tables used by the query plan are dropped
- When should you run sp_recompile on a table?
  - when relevant indexes are added
  - when **update statistics** has been executed
- When should you execute with recompile?
  - when you want to optimize the stored procedure for a specific set of input parameters

# Lab 6g – Stored Procedures

- Execute stored procedures and compare the effect of using a current query plan vs. recompiling

# Basic Questions

- What is the query plan for a given query?
- Are appropriate indexes being selected?
- What is the expected number of rows selected for each table in a join?
- When was the stored procedure compiled?
- Do the parameter types match?

# Useful Techniques

- Run update statistics as needed per index
- Create indexes that exist during all or part of the execution lifetime and use them for multiple queries or processes
- Select items from a table that will yield the smallest number of items first, store in a temporary table, then iterate to find the remaining items
- Split tables by columns to increase row density

# "Watch Out For" Summary

- Statistics haven't been updated recently
- The rows that will be referenced by a given transaction don't fit the pattern reflected by the index statistics
- Queries which do not lead the optimizer to use the appropriate index
- No appropriate index exists for a critical query
- A procedure run by multiple processes uses widely varying input parameters
- A stored procedure was compiled before significant changes to the underlying tables were performed

# Part 2
# Tuning Applications for Performance

System 10 Performance & Tuning

Version 2.2


*The Enterprise Client/Server Company™*

# Course Roadmap

## Part 1
## Designing Applications for Performance

| | |
|---|---|
| Module 1 | Overview of Performance Issues |
| Module 2 | Physical Database Design and Denormalization |
| Module 3 | Table Storage |
| Module 4 | How Indexes Work |
| Module 5 | Selecting Indexes for Performance |
| Module 6 | Query Optimization |

## Part 2
## Tuning Applications for Performance

| | |
|---|---|
| Module 7 | Distributing Data Across Devices |
| Module 8 | How Locking Affects Performance |
| Module 9 | Managing Memory |
| Module 10 | Maintaining High Performance |
| Module 11 | Guidelines for High-Performance Applications |

## Part 3
## Special Topics

| | |
|---|---|
| Module 12 | Networks and Performance |
| Module 13 | tempdb |
| Module 14 | Cursors and Performance |
| Module 15 | CPU Utilization Issues |
| Appendix A | Distributing Data with Replication Server |
| Appendix B | More About SQL Monitor |
| Appendix C | Problem Analysis Walkthrough |

7

# Distributing Data Across Devices

System 10 Performance & Tuning

Version 2.2




*The Enterprise Client/Server Company™*

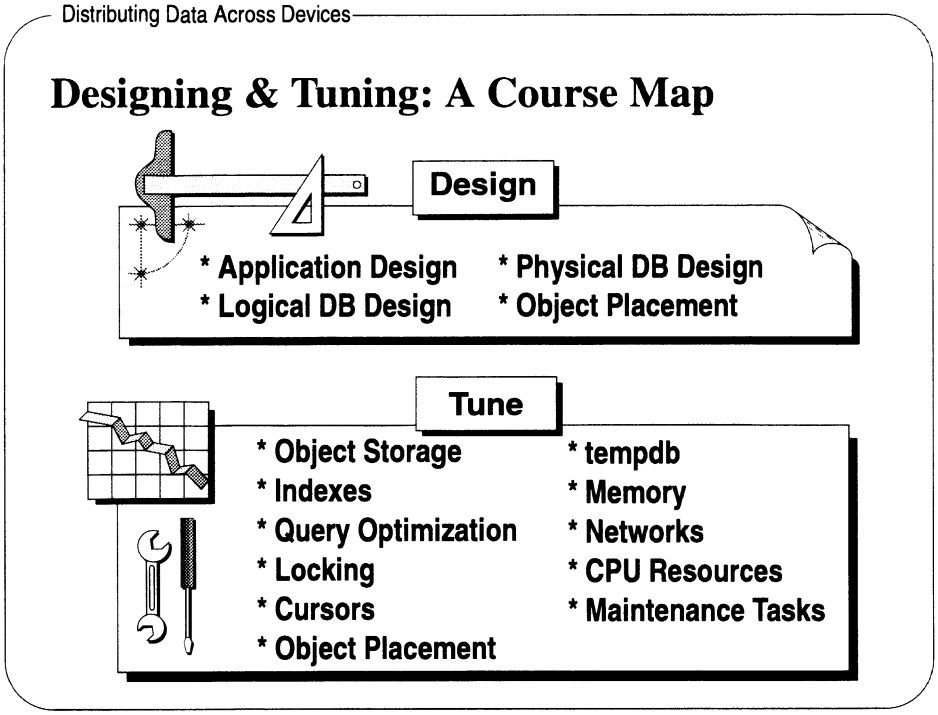
# 7

# Objectives

- Place database objects on specific physical storage devices to improve performance
- Use Sybase diagnostic tools to
  - determine current object placement
  - change object placement to improve performance
  - measure performance impact

Refer to *SYBASE SQL Server System Administration Guide*, Chapter 3: "Managing Physical Resources" for a discussion of overall resource management issues and detailed examples.

Refer to *SYBASE SQL Server System Administration Guide*, Chapter 12: "Fine-Tuning Performance" for a discussion of performance tuning issues with regard to segments.

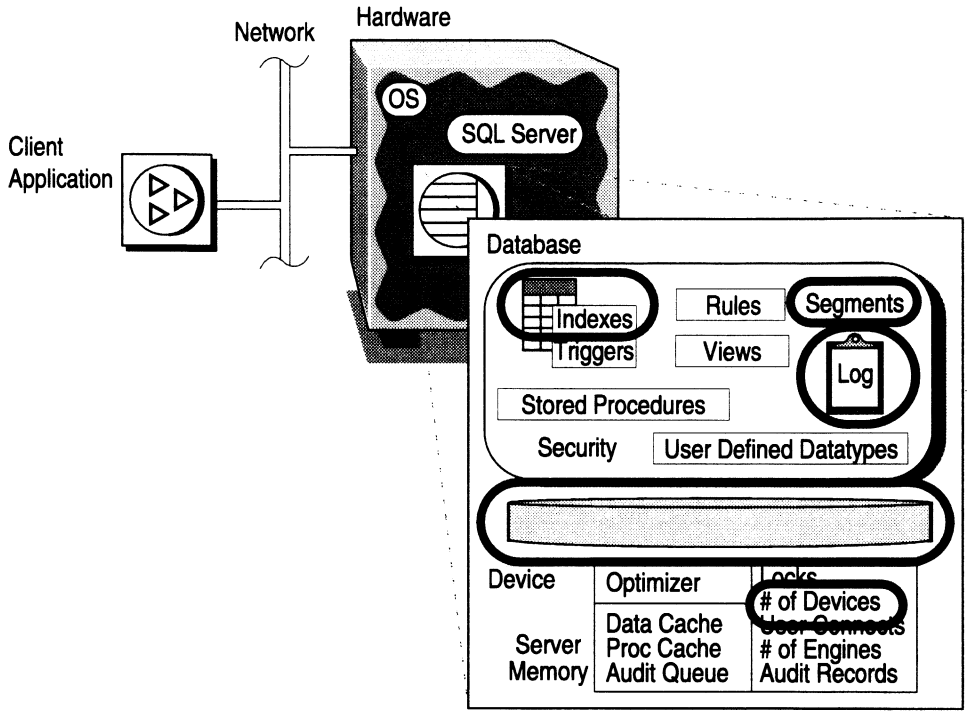



**Object Placement** Appears twice – for both designing and tuning

**Design** Careful object placement to avoid costly changes once in production

**Tuning** Once in production, need to monitor device usage, and may want to make small changes to tune performance

# System Model: Object Placement



# Improving Performance via Object Placement

Performance may be improved by better object placement when:

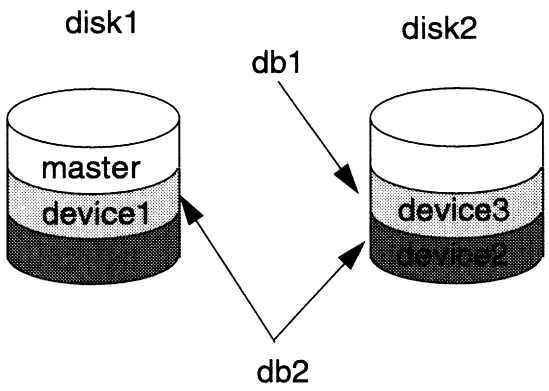
- Single user performance is OK, but response time increases significantly when multiple processes are executed
- Access to a mirrored disk takes twice as long as access to an unmirrored disk
- Query performance degrades when a lot of system table activity occurs
- Maintenance activities seem to take a long time
- Stored procedures seem to slow down as they create temp table
- Does not apply on disk arrays

**Raid** Raid devices randomize data.

# Things to Check

- Random access and serial access processes are using the same disks
- Database processes and operating system processes are using the same disks
- Serial disk mirroring is being used because of functional requirements
- Data modifications are being logged on the same disks holding the data
- tempdb activity is on the same disk as heavily used tables

## Disks and Logical Devices




**Device** A logical device is a contiguous chunk of disk storage and can be implemented as either a raw disk partition or as an operating system file.

# Device Management

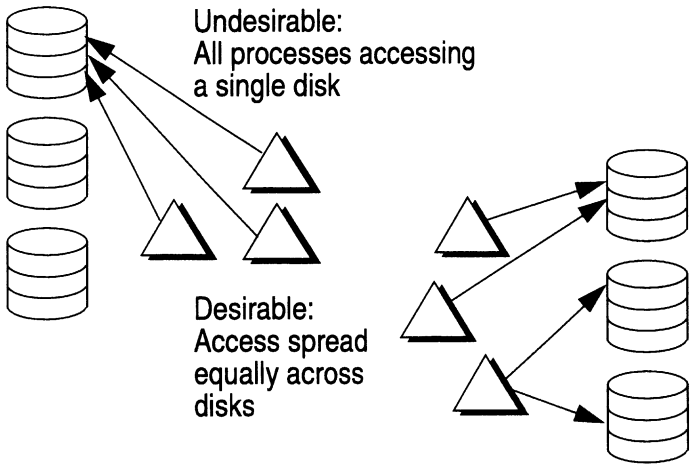
- Guidelines
  - Put heavily used tables on separate disks
  - Put frequently joined tables on separate disks
  - Use segments to place tables and indexes on their own disks
- Creating, using, & dropping databases and devices
  - **disk init** and **sp_dropdevice**
  - **create database** and **drop database**
  - **alter database**
- Displaying & investigating databases and devices
  - **sp_helpdb**
  - **sp_helpdevice**
  - **sp_helplog**

**Example**

```
use master
go

disk init
     name = "mydisk",
     physname = "/dev/sd1c",
     vdevno = 7, size = 2048
go

alter database pubtune on mydisk = 4
go
```

Creates a new device named "mydisk" that is implemented as a raw disk partition. Then, the **pubtune** database is allocated 4MBs out of the new device, which will be added to the "default" segment for **pubtune**.

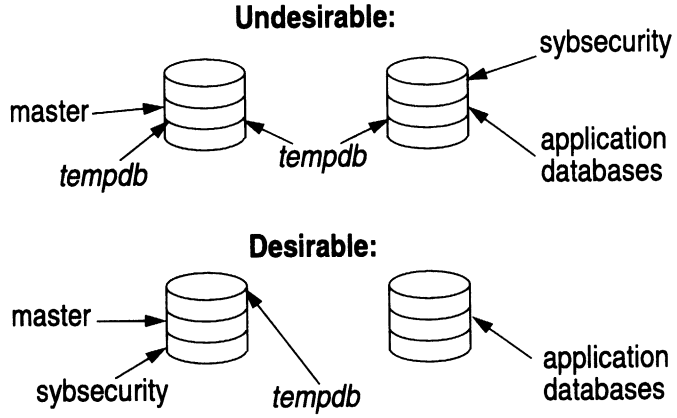
## Spread Access Across Separate Disks: Avoid I/O Contention




**Avoid Bottlenecks**

Spread access across multiple disks to avoid bottlenecks. Keep random disk I/O away from sequential disk I/O.

# Isolate Server-wide I/O from Database I/O




***sybsecurity***

The **sybsecurity** database contains the **sysaudits** table.

If a lot of auditing activity is expected, place **sybsecurity** on a disk that is not busy.

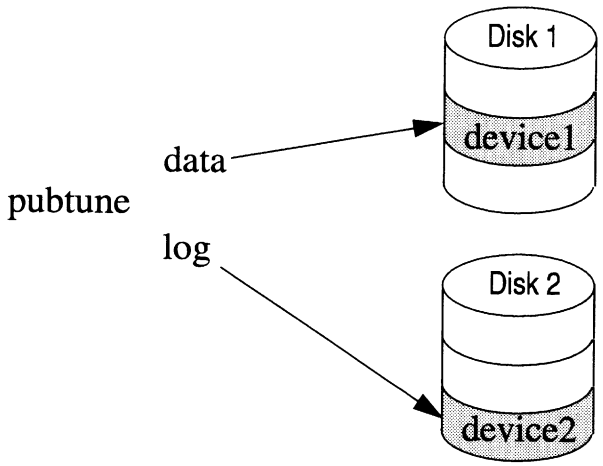
***tempdb***

**tempdb** is another database that serves all processes. Its placement and size can affect performance.

We recommend putting **tempdb** on a disk that does not have heavily-used application databases.

**tempdb** size and placement is the subject of an upcoming module.

# Keep Transaction Log On Separate Disk



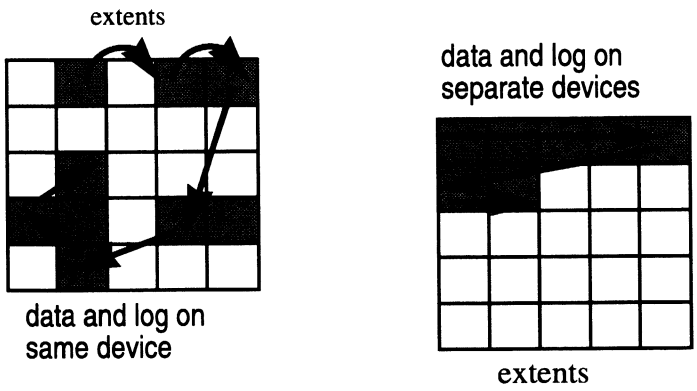

**Transaction Log**

Placing the transaction log on the same device as the data itself is such a common but dangerous reliability problem that both `create database` and `alter database` now require the use of the `with override` option if you attempt to put the transaction log on the same device as the data itself.

**Why Separate Data From Log**

- Limits log size, which keeps it from competing with other objects for space
- Allows use of threshold management techniques
- Improves performance (if on separate physical disk) because data and log do not content for same space
- Ensures full recovery in the event of hard disk crashes (if on separate physical disk) because it allows log to be dumped separately from database

## Transaction Log Storage




- Put the log on its own disk to reduce head movement and increase transfer speed

**Transaction Log Pages**

- By default, the transaction log is stored on the same disk with the data
- Are not contiguous when data and log are on same device (they are intermingled with data pages)
- Tend to be contiguous when log storage is on own device
- If a disk has nothing but the transaction log, it rarely does seeks and hence can maintain a high I/O rate

**Log Size**

- Is hard to predict
- Is based on the frequency of inserts, updates, and deletes, for all of the tables within a given database over a given period of time

# Separating Log By Moving It to a New Device

- For a database created without using the "log on" option (that is, without a separate log), take the following steps:
  - dump the log to truncate it
  - alter the database onto the new device:

    ```
    alter database dbname log on device_name
    ```
  - execute `sp_logdevice` to make that device a log device
- Perform these steps quickly to avoid log records being written to the old device (if database is active)
- Effect:
  - no new pages will be allocated on the old device
  - what is there will be deallocated as the log gets truncated
- Result: the log will grow onto the new device

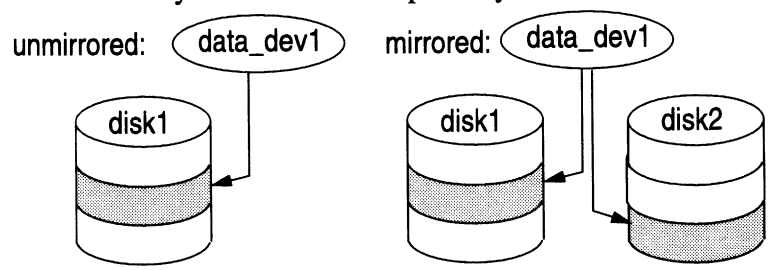
**Note on sp_logdevice**

sp_logdevice is *only* intended to be used to *separate* the log from the data, if the log and data are originally on the same device.

Refer to the System Administration Guide, Chapter 3 "Managing Physical Resources" for further information.

## Mirroring a Device

- Mirroring is a security and high availability feature which allows SQL Server to duplicate the contents of an entire database device
  - Writes go to both disks
  - Reads always come from the primary side




- Mirror on separate disks to minimize performance impact of mirroring

**mirroring devices**

- If one disk for a mirrored device fails, SQL Server notes this when it tries to read or write to that disk and continues with the other disk
- *entire devices* are mirrored, not databases
- To mirror a database, mirror all devices that database is on

- *System Administration Guide* Chapter 3, "Managing Physical Resources"
- *SYBASE SQL Server Reference Manual Volume 1*, Topics, "Disk Mirroring"
- *SYBASE Troubleshooting Guide* Chapter 5, "Disk Mirroring"

# Device Mirroring: Performance Issues

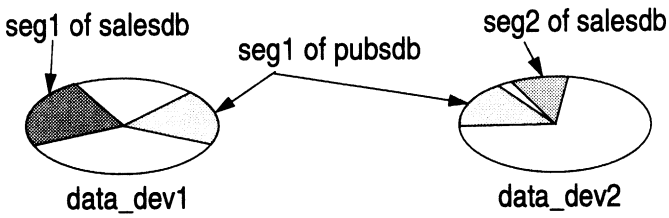
- Mirroring has no effect on the time to read data
  - SQL Server always reads from the "primary" copy
- **Noserial** mode increases time to write data
  - both writes are started at the same time
  - waits for both to complete the order to commit
  - time = $\max(W_1, W_2)$
- **Serial** mode increases time to write data even more
  - first write is started
  - waits for first write to complete, then starts second
  - waits for second write to complete
  - time = $W_1 + W_2$
- Which is more reliable?

**Serial Mode is Recommended**

Despite its performance impact, serial mode improves reliability.

# Using Segments to Improve Performance

- To improve I/O throughput, you can:
  - Spread active tables across disks
  - Separate tables and their non-clustered indexes on to different disks
- Use *segments* to place objects on fragments of disk devices
- Within a database, segments logically map device fragments to tables or indexes




**Device Fragments**

A device fragment is:

- The smallest unit of a data device on to which you can place a table or index
- Only a concept – it has no label or name and can not be used by SQL Server until defined as a segment

**Segments and Fragments**

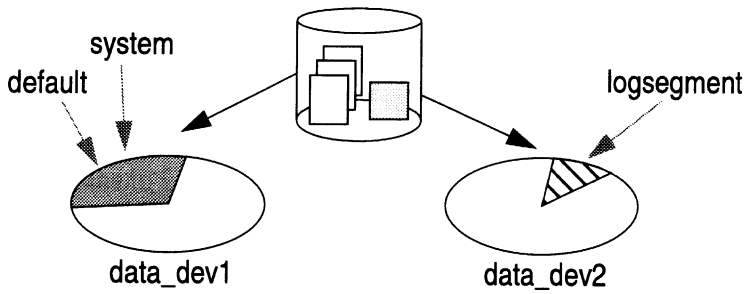
- Segments are labels you give to one or more fragments

**Segments Can Also Be Used To Control Space Usage**

- A table can never grow larger than its segment allocation; you can use segments to limit table size
- You can use the threshold manager to monitor space usage

# System-Defined Segments

- When you create a database, space allocated is labelled with three system-defined segments: *system, default, logsegment*




- In this example: data and log are on separate devices (segments)

**System**
- Holds system tables

**Default**
- Holds tables and indexes that were created without specifying a particular segment

**Logsegment**
- Holds the transaction log
- Is created even when data and log are on the same device (which is not recommended)

# Adding a User-Defined Segment

- To define a segment, execute `sp_addsegment`
- Syntax:

```
sp_addsegment segname, dbname,
   device_name
```

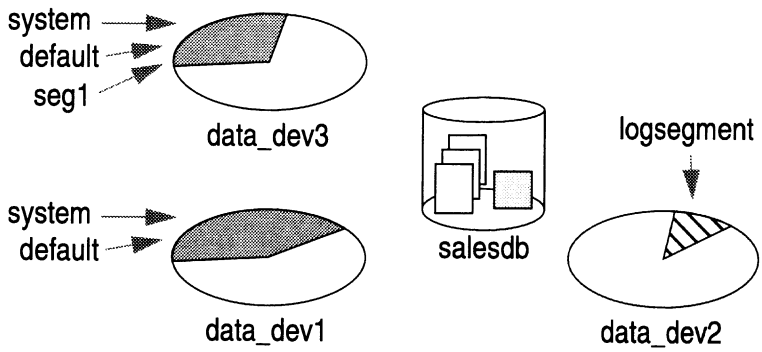
- Sample sequence:

```
disk init name = "data_dev3",...
go
alter database salesdb on data_dev3=1
go
use salesdb
go
sp_addsegment seg1, salesdb, data_dev3
```

- Defining a segment doesn't allocate additional space – it labels space that has already been allocated

**alter database** The `alter database` command creates system and default segments automatically (or a log segment if a log device is specified)

# Adding a User-Defined Segment (continued)




Notice: after `alter database` and `sp_addsegment`, *data_dev3* has *system*, *default*, and *seg1* segments of *salesdb*

**When Altering a Database Onto a Device**

- `system` and `default` segments are created
- Since those segments are on the new device, system tables and other user tables, respectively, could go onto the new device - outside of your control

**Recommendation**

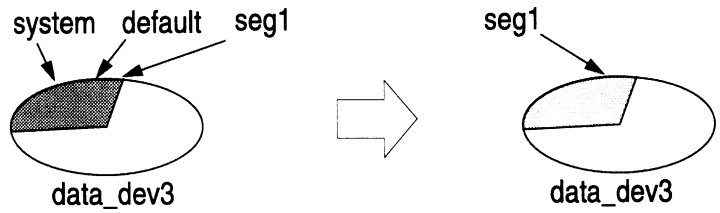
- You may want to drop `system` and `default` segments from new device

**To Put a Segment on Multiple Devices**

- Add another disk device
- Alter the database onto it
- Extend the segment onto the new device with `sp_extendsegment`

# Dropping System & Default Segments

- To ensure that objects you place on a user-defined segment do not have to compete with system tables and other objects, execute `sp_dropsegment` to drop the system and default segments from that device



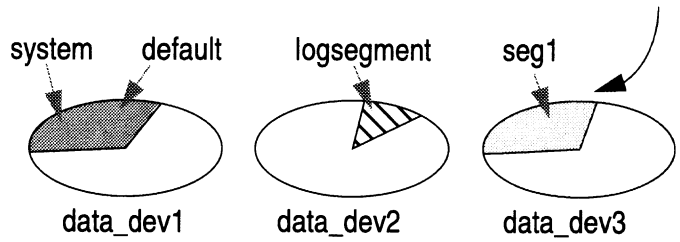

- Example:

```
sp_dropsegment system, salesdb, data_dev3
sp_dropsegment "default", salesdb,
    data_dev3
```

## Creating Objects On Segments

- Specify the segment when creating the object to place it on a specific device

```
create table tableA(x int, ...) on seg1
```




- If you do not specify a segment name, the table will be placed somewhere on the default segment (which may span devices)
- Clustered indexes and their tables are always on the same segment

**Creating Segments**

- Each database may contain up to 32 segments, including the 3 created by the system (***system***, ***logsegment***, and ***default***) when the database is created.
- Note, all of the segments defined for a given database on a given device compete for the total amount of device space that has been allocated to that database using the **create database** and **alter database** commands.
- The segment itself does not have its own space.

**Creating Object s**

- Tables and indexes are created on segments.
- If no segment is named in the **create table** or **create index** statement, then the ***default*** segment for the database is used.

**Migrating Objects**

- The **sp_placeobject** stored procedure can be used to designate the segment to be used for subsequent disk writes.
- In this way, tables and views can span multiple segments.

**Clustered Indexes and Tables**

- The clustered index is part of the table, therefore if you create a clustered index on a segment that is *not* the segment the table is on, the table moves to that new segment.

# Displaying Information About Segments and Devices

- Execute `sp_helpdb` *`databasename`* when using the database to display the segments for that database
- Sample output:

```
name db_ size owner dbid created     status
sales 4 MB    sa    5    Oct 16 1992 no options set

device_fragments  size    usage      free kbytes
dev1              2 MB    data only        1376
dev2              1 MB    log only         1008
dev3              1 MB    data only        1008

device                    segment
dev1                      default
dev1                      system
dev2                      logsegment
dev3                      seg_1
```

# Displaying Information About Segments

- Execute `sp_helpsegment` to list all the segments for the current database
- Sample output:

```
segment    name          status
    0      system        0
    1      default       1
    2      logsegment    0
    3      seg1          0
```

- Segment names are specific to the database
  - They will not be confused with similarly-named segments in other databases
- Use `sp_helpsegment [segment_name]` to monitor the space available on a segment

**Status**

- The default segment, whether or not named "default", has a status of 1.

**Segment Order**

- Segments are listed in order of their creation

# Displaying Information About Objects on Segments

- Execute `sp_helpsegment [segname]` to list the objects on the segment and show what device(s) the segment is mapped to
- Sample output:

```
sp_helpsegment seg1

segment name                status
   4 seg1                    0

device                    size          free pages
data_dev3                  1.0MB                688

table_name              index name          indid
tableA                    x_tableA            1
```

# Segment Management Tools

- Creating, using, and dropping segments
  - sp_addsegment,
  - sp_extendsegment
- Displaying segment information
  - sp_help
  - sp_helpindex
  - sp_helpsegment

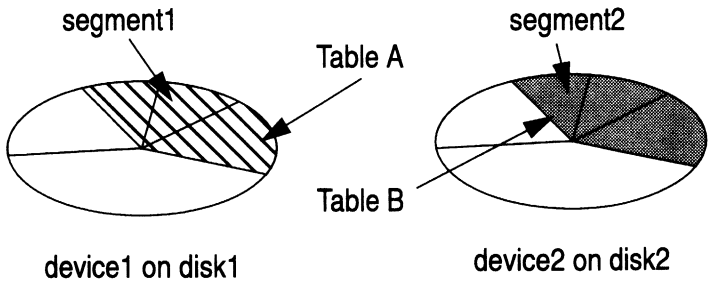
  or select from
  - syssegments, sysindexes, sysusages, sysdevices
- Monitor device I/O
  - Use SQL Monitor and OS utilities

**Segment Example**

```
use pubtune
go

sp_addsegment   mysegment, pubtune, mydisk
go

sp_dropsegment  "default", pubtune, mydisk
sp_dropsegment  "system",  pubtune, mydisk
go

create table authors (...)
on mysegment
go
```

## Isolating Frequently Accessed Tables

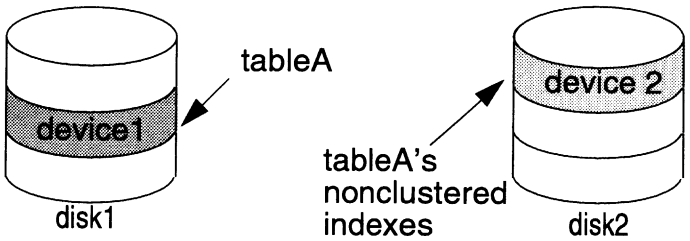
- Place heavily-accessed tables on separate disks to reduce disk contention




**Read-Only Tables** Place read-only tables on a non-mirrored device.

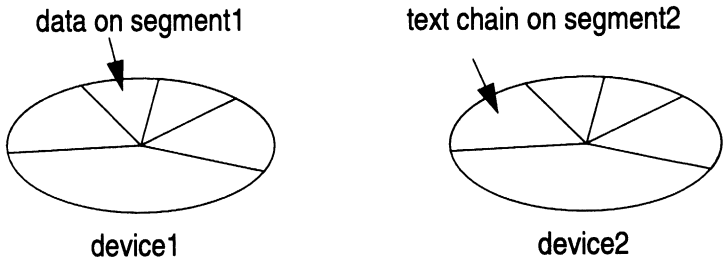
# Separating Tables and Indexes

- Use segments to put heavily-used tables on separate disks from their (equally heavily-used) nonclustered indexes



# Isolating Text and Image Data

- Isolate text and image chains to disks that are not busy with other application related table or index access




```
create table mytable
   (col1 int, col2 text) on segment1
sp_placeobject segment2,
    "mytable.tmytable"
```

**Text Chain Performance**

Text is stored in linked chains of pages.

This increases the performance of storing and retrieving large binary objects.

**Referencing Text Chains**

There is a special entry in **sysindexes** for each text chain – the table name preceded by a "t".

**Placing Text Chains on Segments**

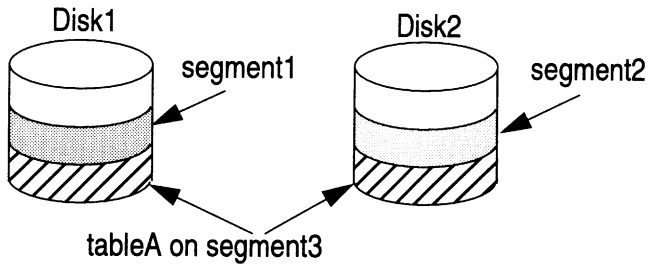
By default, a text chain is placed on the same segment as the table.

To place it on a separate segment, use the following syntax:

```
sp_placeobject segname "tablename.ttablename"
```

# Spreading Table Data to Multiple Disks

- Segments can span multiple devices and disks
- They can be used to spread data across one or more disks
- Result: balanced access to multiple disks



## Splitting a Table Across Multiple Disks

- May improve performance to frequently read tables
- Basic Steps
  - create a segment on each of two disks
  - create a third segment pointing to both disks
  - bulk copy table to file
  - drop table
  - re-create table (and clustered index) on the first segment
  - load first half of the data
  - use **sp_placeobject** to specify 2nd segment
  - load second half of data
  - use **sp_placeobject** to specify 3rd segment

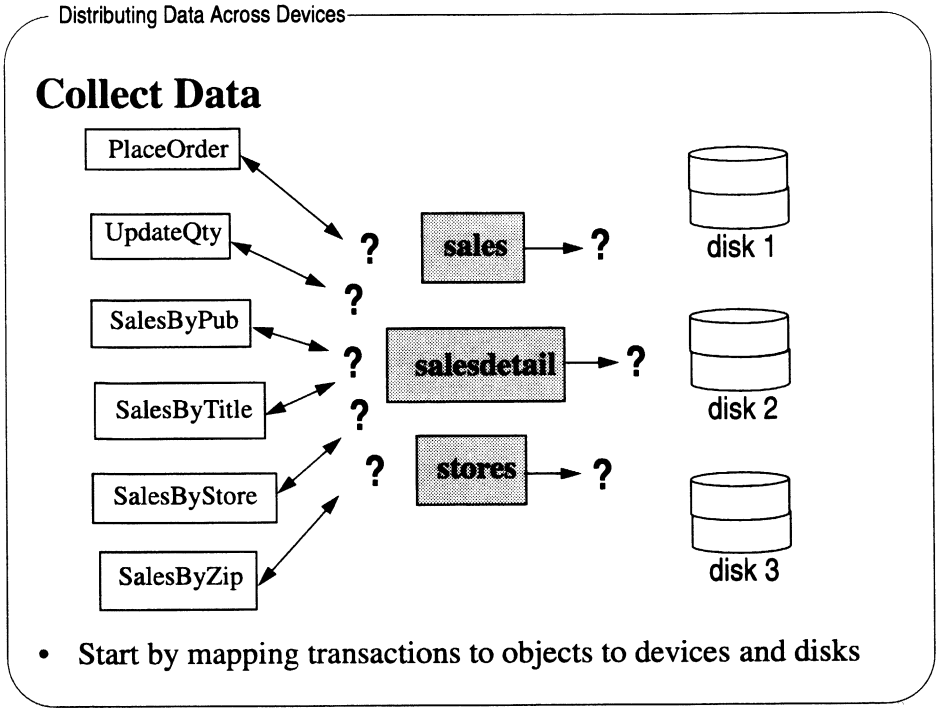
**Why Do You Need Three Segments?**

Three segments are used in the example above because we want to split the current contents of the table into two equal parts, one on each disk, and then use the third to distribute the data to both disks.

# Sample Problem #1

- Single user performance of a query is OK, but increases response time significantly when multiple processes are executed
- Query:

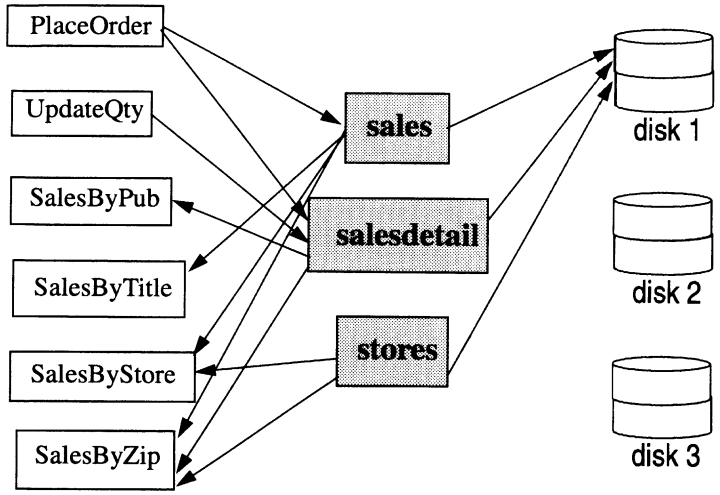
```
exec SalesByZip
```




**sp_helpsegment**

Use the **sp_helpsegment "default"** command to determine how tables map to segments (and thus devices).

## Collect Data (continued)




- These tables are all on disk 1.

# Collect Data: Results

- All data and indexes are on a single device
- Transactions accessing sales and salesdetail tables frequently, others less frequently accessed
- What do you think the problem is?

# Formulate Hypothesis

- What is wrong?
  - **sales** and **salesdetail**, which are heavily used, are on the same disk as other tables, causing contention
- What we can do?
  - Isolate **sales** and **salesdetail** to a separate segment
  - Put non-clustered indexes on a separate segment
  - May want to mirror devices critical **sales** and **salesdetail** tables are on

# Test Hypothesis

- Make changes to reflect hypothesis
  - Drop and recreate non-clustered indexes on a new device via a segment
  - Create another segment on a separate device (on a separate disk)
- Drop and recreate clustered index on **sales** and **salesdetail** table on this new segment (the tables will move automatically to the new segment)
- Rerun transactions to see if there is any improvement
- If confirmed, implement for production

# Sample Problem #2

- Single user performance of a query is OK, but it increases response time significantly when multiple processes are executed
- Query:

```
exec SalesByZip '94521'
```

- A single disk array is being used

- You can not use segments to help this, because user defined segments and object placement cannot be used to subdivide *a single* disk array
- You should keep looking for other sources of improvement

## Object Placement: Questions to Ask

- What critical transactions are performing poorly?
- How frequently are they executed?
- Which objects are being accessed?
- How are they being accessed? Read only? Update?
- Is the transaction accessing text or image data?
- Is tempdb being used heavily? Are temporary tables or work tables being created?
- Are these (or other) transactions being audited?

**Questions to Ask**

If performance is poor, here are some questions you can ask to see if there are object placement issues.

## Object Placement: Questions to Ask (continued)

- Are disk arrays being used?
- What types and sizes of disks are available?
- How many disks are available?
- How many disk controllers are there?
- Will disk drives be shared with other processes?
- Will more disks be purchased as the system evolves?
- Is disk mirroring needed for reliability?

**The Trade-Off**

Trade-off in distributed data across segments:

- More disks, less exposure to a single failure, possible parallel I/O's
- Fewer disks, less chance of a failure, but less able to isolate I/O intensive objects

# Lab 7 – Distributing Data Across Devices

- Examine database placement on devices
- Plan and implement object placement strategies to improve performance

Optional:

- Use SQL Monitor to observe device IO

# Summary

- Spread access across separate disks
- Keep translog on separate disk
- Mirror devices only when necessary
- Can isolate text and image chains
- Can split a table across multiple disks
- Trade offs: Many small disks vs. few large disks

Student Guide

Volume 2

# System 10 Performance & Tuning

Version 2.2


*The Enterprise Client/Server Company™*

58164-01-0220-00-V2

# Notice



## Sybase Trademarks

SYBASE, the SYBASE logo, APT-FORMS, Data Workbench, DBA Companion, Deft, Gain*Momentum*, SA Companion, SQL Debug, SQL Solutions, SQR, Transact-SQL, and VQL are registered trademarks of Sybase, Inc. ADA Workbench, Adaptable Windowing Environment, Application Manager, Applications from Models, APT Workbench, APT-build, APT-Edit, APT-Execute, APT-Translator, APT-Library, Build *Momentum*, Camelot, Client/Server Architecture for the Online Enterprise, Client/Server for the Real World, Configurator, DataServer, Data Workbench, Database Analyzer, DB-Library, Deft Analyst, Deft Designer, Deft Educational, Deft Professional, Deft Trial, Developers Workbench, Easy SQR, Embedded SQL, Enterprise Builder, Enterprise Client/Sever, Enterprise Meta Server, Enterprise Modeler, Enterprise *Momentum*, Gain, Gain*Exposure*, Insight, Mainframe Access Products (MAP), *Momentum*, Movedb, Navigation Server, Net-Gateway, Net Library, Object *Momentum*, OmniSQL Gateway, Omni SQL *Server*, OmniSQL Access Module, Open Client, Open Gateway, Open Server, Open Solutions, PC APT-Execute, PC DB-Net, PC Net Library, Report Workbench, Report Execute, Open Server, Partnerships that Work, Replication Server, Resource Manager, RW-Display Lib, RW-Library, SA Monitor, Secure SQL Server, Secure SQL Toolset, SQL Code Checker, SQL Edit/TPU, SQL Edit, SQL Monitor, SQL Server, SQL Toolset, SQL Server/CFT, SQL Server/DBM, SQL Station, SQR Toolset, SQR Developers Kit, SQR Workbench, SYBASE Client/Server Interfaces, SYBASE Gateways, SYBASE SQL Lifecycle, Sybase Synergy Program, SYBASE Virtual Server Architecture, SYBASE User Workbench System 10, Tabular Data Stream, and The Enterprise Client/Server Company are trademarks of Sybase, Inc.

All other company and product names used herein may be the trademarks or registered trademarks of their respective companies.

## Restricted Rights Legend

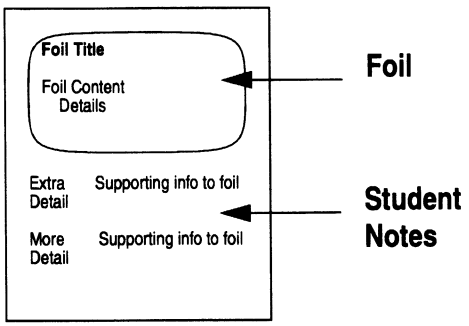
Use, duplication or disclosure by the Government is subject to restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608

# How to Use This Guide

**Student Guide**

- The pages of this Student Guide are divided into two sections: a reduced foil and a set of student notes.
- Your instructor will project the foil during class. The student notes provide additional notes to accompany the foil.




**Lab Workbook**

- In addition to the Student Guide, you are supplied with a Lab Workbook.
- The Lab Workbook includes a complete set of lab instructions and solutions for each lab.
- Each module usually includes at least one lab. The lab name and number in the Student Guide correspond to the same lab name and number in the Lab Workbook.

**ICONS**

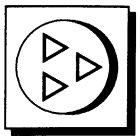
Reference: Additional references are available for the current topic

Question: This is a question you may want to ask yourself about the specific topic

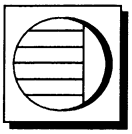
Continued: The code is continued on the following page

Warning: This piece of information is important, and not adhering to it may cause you to run into some problems

# Diagramming Conventions



isql client



SQL Server

# Table of Contents

## Volume 1


Course Objectives ..... Intro–1
Course Roadmap ..... Intro–2

**Part 1**
**Designing Applications for Performance**

**Module 1** **Overview of Performance Issues**

Objectives ..... 1-1
What is Performance? ..... 1-2
What is Performance? ..... 1-3
Other Considerations for Performance ..... 1-4
Designing for Performance Is: ..... 1-5
Tuning For Performance Is: ..... 1-6
SQL Server Model of Operations ..... 1-7
Client/Server System Model ..... 1-8
Network/Hardware/OS Layer Issues ..... 1-9
SQL Server Layer ..... 1-10
Devices ..... 1-11
Database ..... 1-12
Application Layer ..... 1-13
System Limitations ..... 1-14
Fundamental Design & Tuning Guidelines ..... 1-15
Trade-offs ..... 1-16
SQL Server Problem Analysis Approach ..... 1-17
Lab 1 Performance Overview ..... 1-18
Case Study Introduction ..... 1-19
pubtune Database ..... 1-20
Order Entry Scenario ..... 1-21
Shipping Scenario ..... 1-22
Sales Scenario ..... 1-23
Summary ..... 1-24

## Module 2 Physical Database Design and Denormalization


Objectives ..... 2-1
Logical Database Design ..... 2-2
Physical Database Design ..... 2-3
From Logical to Physical Database Design ..... 2-4
Normalization ..... 2-5
First Normal Form ..... 2-6
Second Normal Form ..... 2-7
Third Normal Form ..... 2-8
Why Be Normal? ..... 2-9
Performance Benefits of Normalization ..... 2-10
Denormalization ..... 2-11
Performance Benefits of Denormalization ..... 2-12
Denormalization Input ..... 2-13
Denormalization Techniques ..... 2-14
Adding Redundant Columns ..... 2-15
Adding Derived Columns ..... 2-16
Collapsing Tables ..... 2-17
Duplicating Tables ..... 2-18
Splitting Tables ..... 2-19
Horizontal Partition Example ..... 2-20
Vertical Partition Example ..... 2-21
Managing Denormalized Data ..... 2-22
Using Triggers to Manage Denormalized Data ..... 2-23
Using Application Logic to Manage Denormalized Data ..... 2-24
Batch Reconciliation ..... 2-25
Sample Problem #1 ..... 2-26
Read-Only Tables ..... 2-27
Sample Problem #2 ..... 2-28
Storing Calculated Values ..... 2-29
Evaluating Benefits of Denormalization ..... 2-30
How to Group Tables ..... 2-31
Lab 2 - Physical Database Design ..... 2-32
Summary ..... 2-33


## Module 3 Table Storage


Objectives ..... 3-1
Data Pages ..... 3-2
Linked Data Pages ..... 3-3
Row Density ..... 3-4
Other Types of Pages ..... 3-5

Storage Structures: Topics ..... 3-6
Non-clustered Table Storage ..... 3-7
Operations on Non-clustered Tables ..... 3-8
Non-clustered Tables: Advantages and Disadvantages ..... 3-9
Predicting Table Size ..... 3-10
Calculating Row Size: Determine Row Size ..... 3-11
Calculating Row Length: Algorithm ..... 3-12
Calculating Row Length: Example ..... 3-13
Predicting Table Size: Number of Pages ..... 3-14
Predicting Table Size: sp_estspace ..... 3-15
Displaying Table Size: sp_spaceused ..... 3-16
Displaying Table Size: dbcc tablealloc ..... 3-17
Lab 3 – Table Storage: Size ..... 3-18
Text and Image Chains ..... 3-19
Text Page ..... 3-20
Predicting Size of Text and Image Chains ..... 3-21
Text and Image Chains: Issues and Guidelines ..... 3-22
Summary ..... 3-23

## Module 4 How Indexes Work

Objectives ..... 4–1
Why Study Indexes? ..... 4–2
Topics ..... 4–3
What Are Indexes? ..... 4–4
Why Use Indexes? ..... 4–5
Index Structure ..... 4–6
Tables and Indexes ..... 4–7
Indexes and Keys ..... 4–8
How Indexes Work: Topics ..... 4–9
Clustered Index Structure ..... 4–10
Clustered Table Storage ..... 4–11
What Happens During a Select ..... 4–12
What Happens During an Insert ..... 4–13
What Happens During a Delete ..... 4–14
Clustered Index Size ..... 4–15
Predicting Clustered Index Size ..... 4–16
Clustered/Root Index Row Size ..... 4–17
Predicting Clustered Index Size: Example ..... 4–18
Predicting Clustered Index Size (continued) ..... 4–19
Predicting Index Size Using sp_estspace ..... 4–20
Predicting Index Size Using sp_estspace (continued) ..... 4–21
Displaying Current Index Size: sp_spaceused ..... 4–22

Displaying Current Index Size: dbcc indexalloc ..... 4–23
How Indexes Work: Topics ..... 4–24
Non-clustered Index Structure ..... 4–25
Non-clustered Index Storage ..... 4–26
What Happens During a Select ..... 4–27
What Happens During an Insert ..... 4–28
What Happens During a Delete ..... 4–29
Non-clustered Intermediate Page Index Row ..... 4–30
Non-clustered Leaf Page Index Row Size ..... 4–31
Predicting Non-Clustered Index Size ..... 4–32
Predicting Non-Clustered Index Size (Continued) ..... 4–33
Predicting Non-Clustered Index Size (Continued) ..... 4–34
Predicting Index Size: sp_estspace ..... 4–35
Displaying Current Index Size: sp_spaceused ..... 4–36
Displaying Current Index Size: dbcc indexalloc ..... 4–37
Displaying Current Indexes: sp_helpindex ..... 4–38
Clustered vs. Non Clustered ..... 4–39
Lab 4a – Indexes and I/O Activity ..... 4–40
How Indexes Work: Topics ..... 4–41
Data Page Splits ..... 4–42
Clustered Index Data Page Splitting ..... 4–43
Overflow Pages (Clustered Index Data Pages Only) ..... 4–44
Clustered Index Data Page Merging ..... 4–45
Fill Factor ..... 4–46
Data Page Splits and Fill Factor ..... 4–47
Setting Fill Factor ..... 4–48
Fill Factor Guidelines ..... 4–49
Lab 4b – Fill Factor and I/O Statistics ..... 4–50
Updates ..... 4–51
Rows Inserted in an Update Operation ..... 4–52
Direct or Deferred Updates ..... 4–53
Direct Updates in Place ..... 4–54
Direct Updates in Place – Requirements ..... 4–55
Direct Updates (Not in Place) ..... 4–56
Deferred Updates ..... 4–57
Lab 4c – Where do Updated Rows Go? ..... 4–58
Summary ..... 4–59

## Module 5 Selecting Indexes for Performance

Objectives ..... 5-1
Topics ..... 5-2
How Indexes Can Affect Performance ..... 5-3

How Indexes Affect Performance ..... 5-4
Mechanics of Table Scans ..... 5-5
Topics ..... 5-6
Evaluating Cost of Table Scans ..... 5-7
Topics ..... 5-8
Evaluating Cost of Clustered Index Access (Point Query) ..... 5-9
Evaluating Cost of Clustered Index Access (Range Query) ..... 5-10
Evaluating Cost of Clustered Index Access (Range Query I/O) ..... 5-11
Topics ..... 5-12
Evaluating Cost of Non-clustered Index Access (Point Query) ..... 5-13
Evaluating Cost of Non-clustered Index Access (Range Query) ..... 5-14
Indexes and I/O Statistics ..... 5-15
Topics ..... 5-16
Index Covering ..... 5-17
Evaluating Cost of Covered Access ..... 5-18
Index Covering: Adding Columns ..... 5-19
Topics ..... 5-20
Choosing Indexes: Tables Without Indexes ..... 5-21
Choosing Indexes: Clustered ..... 5-22
Clustered Indexes: Guidelines ..... 5-23
Choosing Indexes: Non-clustered ..... 5-24
Choosing Indexes: Covered ..... 5-25
Lab 5 – Indexes and Performance ..... 5-26
Summary ..... 5-27

## Module 6 Query Optimization

Objectives ..... 6-1
Why Study the Optimizer? ..... 6-2
Common Symptoms of Query Trouble ..... 6-3
What Does the Query Optimizer Do? ..... 6-4
Query Plans ..... 6-5
Displaying Query Plans ..... 6-6
Displaying Query Plans (continued) ..... 6-7
Displaying Query Plans (continued) ..... 6-8
What Happens When You Run a Query? ..... 6-9
Class Exercise: showplan ..... 6-10
Query Optimization Steps ..... 6-11
Analyze Each Table ..... 6-12
Search Arguments (SARGS) ..... 6-13
SARG Equivalents ..... 6-14
Padding With SARGs ..... 6-15
Lab 6a – Search Arguments ..... 6-16

OR Clauses and Equivalents ..... 6-17
Resolution of OR Clauses ..... 6-18
OR Strategy Methodology ..... 6-19
OR Strategy Showplan ..... 6-20
Lab 6b – OR Strategies ..... 6-21
Index Covering ..... 6-22
Index Covering (continued) ..... 6-23
How Aggregate Queries Are Optimized ..... 6-24
Aggregate Query Plan Example ..... 6-25
Aggregates: Special Cases ..... 6-26
Lab 6c – Index Covering and Aggregates ..... 6-27
Review ..... 6-28
Query Optimization Steps ..... 6-29
Selecting an Index ..... 6-30
Row Estimates Using Index Access ..... 6-31
When Will the Distribution Page Not be Considered? ..... 6-32
Estimating Number of Logical Page Reads ..... 6-33
Lab 6d – Predicting Index Usage ..... 6-34
Query Optimization Steps ..... 6-35
Join Clauses ..... 6-36
Generating Join Clauses ..... 6-37
Join Clauses (continued) ..... 6-38
Nested Iterations ..... 6-39
Nested Iterations Example ..... 6-40
Nested Iterations: First Approach ..... 6-41
Nested Iterations: Second Approach ..... 6-42
Class Exercise: Which Solution Would Be Better...? ..... 6-43
Three-Way Join: First Approach ..... 6-44
Three-Way Join: Second Approach ..... 6-45
Outer Joins ..... 6-46
Row Estimates for Join Clauses ..... 6-47
What Is On a Distribution Page? ..... 6-48
Composite Index Density on Distribution Page ..... 6-49
Costing Calculations: Example ..... 6-50
Nested Iterations Performance ..... 6-51
Nested Iterations-Logical I/O Estimates ..... 6-52
Nested Iterations: I/O Calculation Examples ..... 6-53
Nested Iterations – Final Cost ..... 6-54
I/O Costing ..... 6-55
Finding the Optimal Join Strategy ..... 6-56
Lab 6e – Join Strategies ..... 6-57
Reformatting ..... 6-58
Reformatting (continued) ..... 6-59

Reformatting Query Plan ..... 6-60
Join Selection Review ..... 6-61
When Are Work Tables Used? ..... 6-62
Sample Work Tables: Order By ..... 6-63
Sample Work Tables: Group By ..... 6-64
Sample Work Tables: Distinct ..... 6-65
Lab 6f – Work Tables and Sorts ..... 6-66
Query Optimization Steps Summary ..... 6-67
Stored Procedure Optimization ..... 6-68
Stored Procedures and Cache ..... 6-69
Misleading Initial Value Example ..... 6-70
Optimizing Stored Procedures ..... 6-71
Optimizing Stored Procedures (Continued) ..... 6-72
Recompiling Stored Procedures ..... 6-73
Lab 6g – Stored Procedures ..... 6-74
Basic Questions ..... 6-75
Useful Techniques ..... 6-76
"Watch Out For" Summary ..... 6-77

**Part 2**
**Tuning Applications for Performance**

**Module 7** **Distributing Data Across Devices**

Objectives ..... 7-1
Designing & Tuning: A Course Map ..... 7-2
System Model: Object Placement ..... 7-3
Improving Performance via Object Placement ..... 7-4
Things to Check ..... 7-5
Disks and Logical Devices ..... 7-6
Device Management ..... 7-7
Spread Access Across Separate Disks: Avoid I/O Contention ..... 7-8
Isolate Server-wide I/O from Database I/O ..... 7-9
Keep Transaction Log On Separate Disk ..... 7-10
Transaction Log Storage ..... 7-11
Separating Log By Moving It to a New Device ..... 7-12
Mirroring a Device ..... 7-13
Device Mirroring: Performance Issues ..... 7-14
Using Segments to Improve Performance ..... 7-15
System-Defined Segments ..... 7-16
Adding a User-Defined Segment ..... 7-17
Adding a User-Defined Segment (continued) ..... 7-18
Dropping System & Default Segments ..... 7-19

Creating Objects On Segments ..... 7-20
Displaying Information About Segments and Devices ..... 7-21
Displaying Information About Segments ..... 7-22
Displaying Information About Objects on Segments ..... 7-23
Segment Management Tools ..... 7-24
Isolating Frequently Accessed Tables ..... 7-25
Separating Tables and Indexes ..... 7-26
Isolating Text and Image Data ..... 7-27
Spreading Table Data to Multiple Disks ..... 7-28
Splitting a Table Across Multiple Disks ..... 7-29
Sample Problem #1 ..... 7-30
Collect Data ..... 7-31
Collect Data (continued) ..... 7-32
Collect Data: Results ..... 7-33
Formulate Hypothesis ..... 7-34
Test Hypothesis ..... 7-35
Sample Problem #2 ..... 7-36
Object Placement: Questions to Ask ..... 7-37
Object Placement: Questions to Ask (continued) ..... 7-38
Lab 7 – Distributing Data Across Devices ..... 7-39
Summary ..... 7-40

# Volume 2

## Module 8 How Locking Affects Performance


Objectives ..... 8-1
Why Study Locks? ..... 8-2
Things to Check ..... 8-3
Why Are Locks Needed? ..... 8-4
Isolation Levels ..... 8-5
The "Dirty Read" Problem ..... 8-6
The "Non-repeatable Reads" Problem ..... 8-7
The "Phantom Reads" Problem ..... 8-8
SQL Server Isolation Levels ..... 8-9
Granularity of Locks ..... 8-10
Locking in SQL Server ..... 8-11
Page Locks ..... 8-12
Page Locking Examples ..... 8-13
Table Locks ..... 8-14
SQL Statements and Intent Locks ..... 8-15
Escalating Locks ..... 8-16
holdlock ..... 8-17
Checking for Blocked Processes ..... 8-18
Cursors and Locking ..... 8-19
Deadlock ..... 8-20
Avoiding Deadlock ..... 8-21
Locking and Performance ..... 8-22
Reduce Locking Techniques ..... 8-23
Reducing Locking Techniques (continued) ..... 8-24
Sample Problem ..... 8-25
Collect Data ..... 8-26
Collect Data: Results ..... 8-27
Formulate Hypothesis ..... 8-28
Test Hypothesis ..... 8-29
Implement if Confirmed ..... 8-30
Locking: Questions to Ask ..... 8-31
Lab 8 – Locking and Performance ..... 8-32
Summary ..... 8-33


## Module 9 Managing Memory


Objectives ..... 9-1
Memory Size ..... 9-2

More Memory Improves Performance ..... 9-3
How Much Memory to Give SQL Server ..... 9-4
Memory: The "Big Picture" ..... 9-5
dbcc memusage ..... 9-6
Reconfiguring Memory ..... 9-7
Displaying Configuration Values ..... 9-8
Most Options Affect Size of Server Structures ..... 9-9
Connections: Example ..... 9-10
Options Which Consume Memory: Databases, Locks, Objects, Devices 9-11
Network Memory Increases Size of Server ..... 9-12
Extent I/O Buffers Increase Size of Server ..... 9-13
Audit Queue Consumes Memory ..... 9-14
Auditing and Performance ..... 9-15
Split Remaining Memory: ..... 9-16
What Procedure Cache Is For ..... 9-17
Procedure Cache Sizing ..... 9-18
What Data Cache is For ..... 9-19
Page Aging in Data Cache ..... 9-20
Effect Of Data Cache On Retrievals ..... 9-21
Effect Of Data Cache On Updates ..... 9-22
Data Cache Performance ..... 9-23
Finding Out Cache Sizes ..... 9-24
Configuration Example ..... 9-25
Memory Allocation: Questions to Ask ..... 9-26
SQL Monitor - Cache Window ..... 9-27
Summary: Memory Size & Balance ..... 9-28
Sample Problem ..... 9-29
Collect Data: Commands ..... 9-30
Collect Data: Results ..... 9-31
Formulate Hypothesis ..... 9-32
Test Hypothesis ..... 9-33
Implement Solution ..... 9-34
Lab 9 – Memory and Performance ..... 9-35
Summary ..... 9-36

**Module 10** **Maintaining High Performance**

Objectives ..... 10-1
Why Study Maintenance Activities? ..... 10-2
Topics ..... 10-3
Creating Databases ..... 10-4
Creating Indexes ..... 10-5
Creating Indexes: Using Extent I/O Buffers ..... 10-6

Creating Indexes: More Suggestions ..... 10-7
Topics ..... 10-8
How Backup Server Works ..... 10-9
Backup Server: Performance Features ..... 10-10
Recovery Interval ..... 10-11
Transaction Log Dumps ..... 10-12
Topics ..... 10-13
Bulk Copy: Performance Issues ..... 10-14
Bulk Copy: Suggestions ..... 10-15
Topics ..... 10-16
Database Consistency Checker (dbcc) ..... 10-17
Lab 10 – Maintenance and Performance ..... 10-18
Summary ..... 10-19

**Module 11** **Guidelines for High-Performance Applications**

Objectives ..... 11-1
Application Development Life Cycle ..... 11-2
Iterative Application Design ..... 11-3
Initial Design Decisions Affecting Applications ..... 11-4
Client vs. Server? ..... 11-5
Interactive vs. Batch Processing ..... 11-6
Additional Batch Guidelines ..... 11-7
Application Design Summary ..... 11-8
Managing User Interaction Example ..... 11-9
System Context Model ..... 11-10
Guidelines: Network ..... 11-11
Guidelines: Hardware ..... 11-12
Guidelines: OS ..... 11-13
Guidelines: Server Configuration ..... 11-14
Guidelines: Disks/Devices ..... 11-15
Guidelines: Database ..... 11-16
Guidelines: Tables ..... 11-17
Guidelines: Indexes ..... 11-18
Guidelines: Security ..... 11-19
Lab 11 – Designing Applications for Performance ..... 11-20
Summary ..... 11-21

## Part 3
## Special Topics

## Module 12 Networks and Performance


Objectives ..... 12-1
Why Study the Network? ..... 12-2
Underlying Problems ..... 12-3
System Model ..... 12-4
Networks & Performance Topics ..... 12-5
How SQL Server Uses the Network ..... 12-6
Networks & Performance Topics ..... 12-7
SQL Server Options ..... 12-8
User Connections and Buffers ..... 12-9
Where's My Memory Going? ..... 12-10
Networks & Performance Topics ..... 12-11
Smaller vs. Larger Packet Sizes ..... 12-12
Different Packet Sizes for Individual Clients ..... 12-13
Packet Size ..... 12-14
Networks & Performance Topics ..... 12-15
Reducing Network Traffic Techniques ..... 12-16
Large Transfers ..... 12-17
Evaluation Tools with SQL Server ..... 12-18
Operating System Network Evaluation Tools ..... 12-19
Impact of Other Server Activities ..... 12-20
Networks & Performance Topics ..... 12-21
Guideline 1 ..... 12-22
Guideline 2 ..... 12-23
Guideline 3 ..... 12-24
Guideline 4 ..... 12-25
Guideline 5 ..... 12-26
Guideline 6 ..... 12-27
Guideline 7 ..... 12-28
Guideline 8 ..... 12-29
Typical Problem ..... 12-30
Collect Data: Results ..... 12-31
Test Hypothesis ..... 12-32
Networks & Performance: Questions ..... 12-33
Lab 12 – Networks and Performance ..... 12-34
Summary ..... 12-35

## Module 13 tempdb


Objectives ..... 13-1
Why Worry About tempdb? ..... 13-2
Topics ..... 13-3
What is tempdb? ..... 13-4
Used for Internal Processing ..... 13-5
Storing Temporary Tables ..... 13-6
Using Temporary Tables to Split Up Multi-table Joins ..... 13-7
Indexes on Temporary Tables ..... 13-8
Topics ..... 13-9
Initial Allocation of tempdb ..... 13-10
Sizing tempdb ..... 13-11
Input to Sizing tempdb ..... 13-12
Sizing Algorithm ..... 13-13
Sizing Algorithm: Example ..... 13-14
Topics ..... 13-15
Placing tempdb ..... 13-16
Placing tempdb: Desirable ..... 13-17
Placing tempdb: Undesirable ..... 13-18
Preventing Temporary Tables from Spanning Multiple Devices ..... 13-19
Topics ..... 13-20
Locking in tempdb ..... 13-21
Minimizing Locking in tempdb ..... 13-22
Other tempdb Performance Tips ..... 13-23
Lab 13 – tempdb ..... 13-24
Summary ..... 13-25


## Module 14 Cursors and Performance


Objectives ..... 14-1
What Is A Cursor? ..... 14-2
Set-oriented vs. Row-oriented Programming ..... 14-3
Steps in Using a Cursor ..... 14-4
Cursors: A Simple Example ..... 14-5
Cursors: More Commands ..... 14-6
Resources Required At Each Stage ..... 14-7
Two Cursor Modes: Read-only and Update ..... 14-8
Index Use and Requirements ..... 14-9
Two Global Variables ..... 14-10
Sample Stored Procedure: No Cursors ..... 14-11
Sample Stored Procedure With Cursor ..... 14-12
Sample Stored Procedure With Cursor (continued) ..... 14-13

Performance: A Comparison ..... 14-14
Performance Implications ..... 14-15
Class Demo #1: Locking With Read-only Cursors ..... 14-16
Class Demo #2: Locking With Update Cursors ..... 14-17
Lab 14 – Cursors ..... 14-18
Summary ..... 14-19

## Module 15 CPU Utilization Issues

Objectives ..... 15-1
Why Study CPU Utilization? ..... 15-2
Underlying Problems ..... 15-3
System Model ..... 15-4
Single CPU Machines ..... 15-5
CPU Utilization of SQL Server ..... 15-6
CPU Utilization of SQL Server (continued) ..... 15-7
Overall System CPU Usage ..... 15-8
From Within SQL Server ..... 15-9
Single CPU Machine Summary ..... 15-10
SMP Machines (Symmetric Multi-Processing) ..... 15-11
SMP: SYBASE Virtual Server Architecture ..... 15-12
SMP: SQL Server Task Management ..... 15-13
SMP: Choosing the Right Number of Engines ..... 15-14
SMP: Application Design Considerations ..... 15-15
SMP: Overburdened CPU ..... 15-16
Multi-CPU Machines Summary ..... 15-17
CPU: The Time Slice Story ..... 15-18
CPU: Typical Questions to Ask ..... 15-19
Sample Problem ..... 15-20
Sample Problem (continued) ..... 15-21
Sample Problem (continued) ..... 15-22
SQL Monitor - Performance Trends ..... 15-23
Lab 15 – Using CPU Resources ..... 15-24
Summary ..... 15-25

## Appendix A Distributing Data with Replication Server

Replication Server ..... A-1
Sample Replication System ..... A-2
Replication Server: Performance Impact ..... A-3
Replication Server: Performance Impact (continued) ..... A-4

## Appendix B More About SQL Monitor


Objectives ..... B-1
SQL Monitor ..... B-2
SQL Monitor Architecture ..... B-3
Starting The SQL Monitor Client ..... B-4
SQL Monitor Main Menu ..... B-5
SQL Monitor Windows ..... B-6
Cache Window ..... B-7
Device I/O Window ..... B-8
Performance Summary Window ..... B-9
Performance Trends Window ..... B-10
Process Activity Window ..... B-11
Filtering Process Information ..... B-12
Process Detail Window ..... B-13
Process List Window ..... B-14
Transaction Activity Window ..... B-15
Summary ..... B-16
SQL Monitor Demos ..... B-17


## Appendix C Problem Analysis Walkthrough


Objectives ..... C-1
Fundamental Tuning Guidelines ..... C-2
SQL Server Problem Analysis ..... C-3
Sample Problem ..... C-4
Collect Data ..... C-5
Consider Your Options ..... C-6
Is Denormalization Appropriate? ..... C-7
How Should Objects Be Placed? ..... C-8
What Indexes Should Be In Place? ..... C-9
Is Memory Being Used Effectively? ..... C-10
Is The CPU Being Used Effectively? ..... C-11
Are There Any Concurrency Issues? ..... C-12
Lab C – Introduction to Problem Analysis ..... C-13

8

# How Locking Affects Performance

System 10 Performance & Tuning

Version 2.2




*The Enterprise Client/Server Company™*

# 8

# Objectives

- Explain how locking occurs and how it affects system performance
- Use Sybase tools to detect locking
- Discuss techniques to minimize locking

# Why Study Locks?

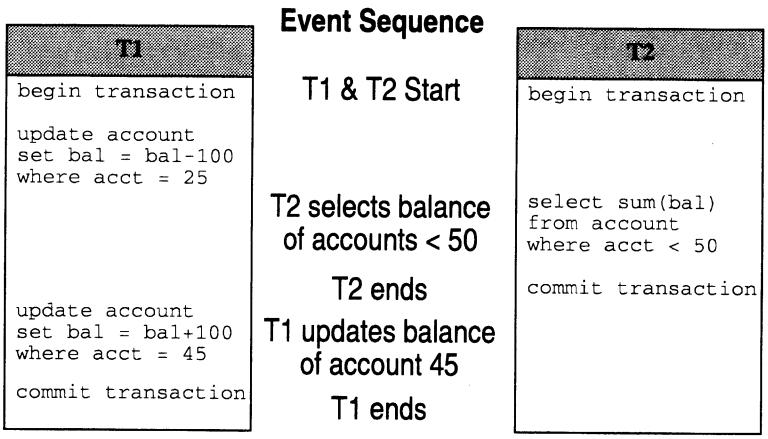
- Throughput and response time is unacceptable, even though disk I/O and CPU utilization is low
- Processes are aborted by the system on a regular basis (due to deadlock)

# Things to Check

- A small number of objects are modified by a large number of transactions
- "Hot spots" are causing contention
- Processes are acquiring locks in a deadlock-prone manner
- Processes are using BEGIN TRANS unnecessarily
- Processes are performing complex calculations inside transactions
- Processes allow user interaction within transactions
- Transactions perform multiple unrelated tasks, can be broken up
- Maintenance activities are being performed during normal processing periods

# Why Are Locks Needed?

- Enforce consistency of data in a multi-user environment




**Example**

If transaction T2 runs before, or after T1, both executions would return the same value. If T2 runs in the middle of transaction T1 (the first update) the result for transaction T2 would be off by 100.

**Locks Enforce Consistency**

Consistency means that if the transactions are re-run, the database will end up in the same state.

# Isolation Levels

- ANSI standard defines 3 levels of isolation for SQL transactions
  - Level 1 prevents **dirty reads**
  - Level 2 prevents **non-repeatable reads**
  - Level 3 prevents **phantom reads**
- The higher the isolation level, the fewer chances for an inconsistency to occur
- The higher the isolation level, the lower the concurrency

**Isolation Levels**

- Each isolation level specifies the kind of actions which are not permitted while concurrent transactions execute.
- Higher levels include the restrictions imposed by lower levels.

**Level 1**

Exclusive lock on objects being changed. Hold lock until end of transaction. No shared locks.

**Level 2**

Exclusive lock on pages being changed. Hold lock until end of transaction.

Shared lock on pages being searched. Remove lock after processing object.

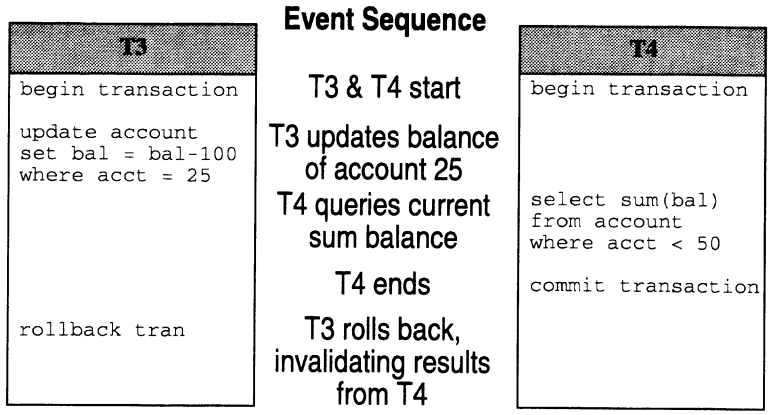
**Level 3**

Exclusive lock on pages being changed.

Shared lock on pages/table being searched.

Hold all locks until end of transaction. (Accumulate locks.)

# The "Dirty Read" Problem




- "Level 1" prevents this!
- SQL Server provides level 1 isolation by default

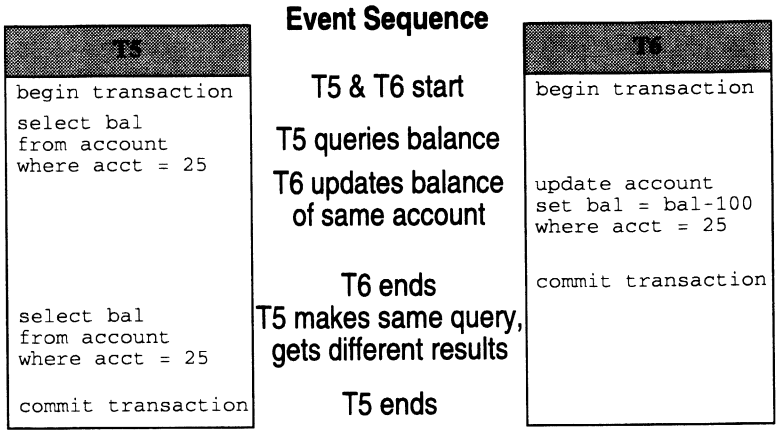
**Dirty Reads**

If transaction T4 queries the table after T3 updates it, but before it rolls back the change, the amount calculated by T4 is off by 100.

**Level 1**

Level 1 consistency prevents this problem.

## The "Non-repeatable Reads" Problem



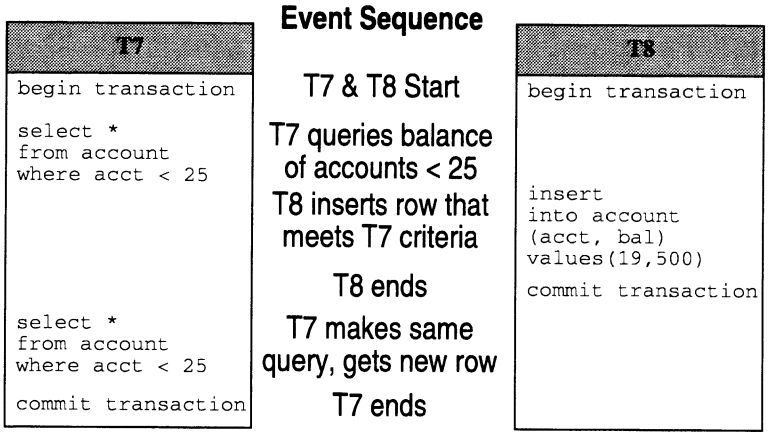

- "Level 2" prevents this!

**Non-repeatable Reads** If transaction T6 modifies and commits the changes after the first query in T5 but before the second one, the same two queries in T5 produce different results.

**Level 2** Level 2 consistency prevents this problem.

**SQL Server** Not separately addressed but is bundled with isolation level 3 support.

# The "Phantom Reads" Problem




- "Level 3" prevents this!

**Phantom Reads**

If transaction T8 inserts rows into the table that satisfy T7's search condition after T7 executes the first select, subsequent reads by T7 using the same query give different results.

**Level 3 Consistency**

Level 3 consistency prevents phantom reads.

**SQL Server**

Enforced with holdlock and/or set transaction isolation level.

# SQL Server Isolation Levels

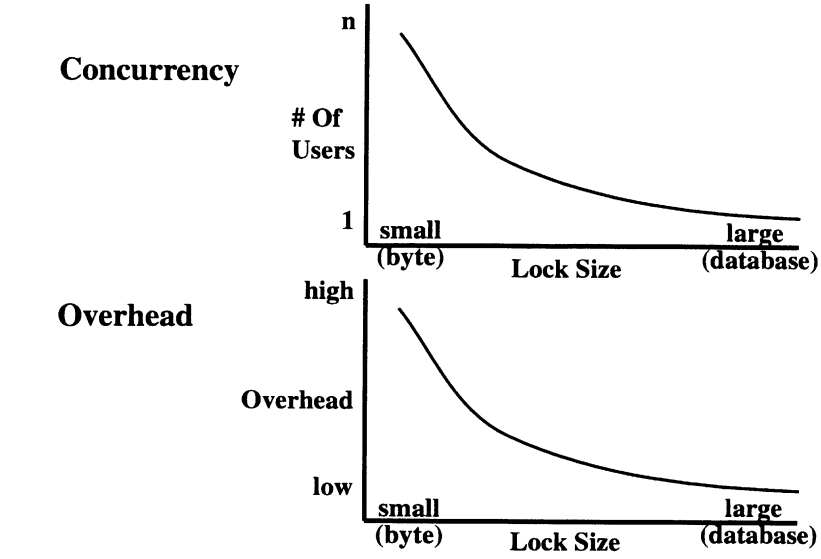
- Defaults to 1: Exclusive lock on pages being changed, held until transaction ends.
- You can enforce Level 3 with
  - select.... holdlock
  - set transaction isolation level
- No Level 2 locking specifically, but Level 3 includes Level 2

**Level 1** Exclusive lock on objects being changed. Hold lock until end of transaction.

**Level 2** Exclusive lock on pages being changed, hold until end of transaction; shared lock on pages being searched, remove lock after processing object.

**Level 3** Exclusive lock on pages being changed; shared lock on pages/table being searched; hold all locks until end of transaction (locks will accumulate).

How Locking Affects Performance




**Locking Granularity**

The granularity of locks in a database refers to how much of the data is locked at one time. In theory, a server could lock as little as a single row of data or an entire database. By increasing the lock granularity, the amount of processing required to obtain a lock becomes smaller, but large lock granularity degrades performance as users must wait until the locks are released. As lock granularity decreases, more processing is necessary to maintain and coordinate the increased number of locks.

**Concurrency**

If you lock an entire database, the per transaction overhead is trivial, but the system is single user. Large locking granularity affects performance as more users must wait for locks to be freed.

**Overhead**

If you lock a row at a time, concurrency is at a maximum, and so is lock overhead.

**Trade-off**

You must balance overhead and concurrency.

## Locking in SQL Server

- SQL Server handles all locking decisions
- SQL Server has two primary levels of locking:
  - Page Locks
  - Table Locks
- Locking strategy is determined from the query plan
- You can force the Server to use more or less restrictive locks

**Page Locks**

- Page Locks are less restrictive (smaller) than table locks.
- SQL Server attempts to use page locks as frequently as possible.

**Table Locks**

Table locks provide more efficient locking when a whole table is accessed. Once a statement accumulates more than 200 page locks, SQL Server attempts to issue a table lock.

# Page Locks

There are three types of page locks:

- Shared Locks (S)
  - Multiple transactions can lock a shared page
  - No transactions can change the page
- Exclusive Locks (X)
  - Only one transaction can lock the page
  - Other transactions wait until the exclusive lock releases
- Update(U)
  - Allows reads, but will not allow U or X locks
  - Becomes an X lock when the page is ready to be modified
  - This is an internal lock to help avoid deadlocks

**Write vs. Reads**

- In general, read operations acquire shared locks, and write operations acquire exclusive locks.

# Page Locking Examples

- What types of locks will SQL Server use for the statements below (assuming indexes are being used by the search arguments)?

```
select bal
from account
where acct = 1

insert account values (34, 500)

delete account
where bal < 0

update account
set bal = 0
where acct = 25
```

**Note** Locks are decided after the optimizer has developed the query plan. Thus the use of an index for a SARG is important. Without an index, an entire table will often be locked instead of just the pages.

**Example 1** Shared Page Lock

**Example 2** Exclusive Page Lock

**Example 3** ~~Update lock, followed by~~ exclusive lock on affected pages

**Example 4** Same as Example 3 above

## Table Locks

- Intent Locks (IS or IX)
  - indicates which types of page locks are held
  - prevents other transactions from acquiring S or X locks
- Shared Locks (S)
  - similar to shared page lock
  - example: `create nonclustered index`
- Exclusive Locks(X)
  - similar to exclusive page lock
  - example: updating salary by 10%

**Demand Locks**

Demand locks prevent any more shared locks from being set. SQL Server sets a demand lock to indicate that a transaction is next in line to lock a table or page. This avoids situations in which read transactions acquire overlapping shared locks, monopolizing a table or page, so that a write transaction waits indefinitely for an exclusive lock. Such a situation is commonly called a "live lock".

**Intent Locking**

Any time page locking is done, there is always a table Intent Lock (next page). The intent locks are actually issued as soon as the optimizer picks the plan, whereas the page locks are not acquired until execution begins.

# SQL Statements and Intent Locks

| SQL Statement | Intent Type |
|---|---|
| select | Intent Shared (Sh_intent) |
| insert | Intent Exclusive (Ex_intent) |
| delete | Intent Exclusive (Ex_intent) |
| update | Intent Exclusive (Ex_intent) |

- These are requests, not real locks, and indicate a potential to acquire locks on the table
- Establish a queue obtaining locks

**Intent Locks**

Used to indicate at the table level what type of page level locks are being acquired.

These do not lock the table: they provide a mechanism to the server to know what other table locks can be obtained.

# Escalating Locks

- SQL Server tries to satisfy requests with page locks
- If more than 200 pages are held, it upgrades to a shared or exclusive table lock
- When might this happen?
  - update with where clause that is not selective
  - bulk copy

**update with where**

SQL Server starts out with an exclusive (or update) page lock and starts updating rows which satisfy the where clause.

As it proceeds, it keeps exclusive locks on all the pages it has changed.

If the where clause is not highly selective, SQL Server may wind up holding many locks, and it still has more to read!

In that case, it would upgrade to a table lock.

**bulk copy**

SQL Server will assume it needs only a few pages.

Everything in the batch size is considered a transaction, so as the server reads rows in, it holds locks on all the pages it has changed.

When they reach a certain number, SQL Server acquires a table lock.

## holdlock

```
SELECT balance
FROM   account HOLDLOCK
WHERE  acct_number = 25
```

- Enforces ANSI isolation level 3
- Makes a shared lock more restrictive: causes server to hold shared lock until transaction is complete
  - Applies shared page lock if the search argument references indexed columns
  - Otherwise, applies shared table lock
- Use only if strictly necessary!
- If level 3 is set, use `select ... noholdlock` to reduce to level 1

**noholdlock**

The **noholdlock** option in a select statement prevents SQL Server from holding any shared locks acquired regardless of the isolation level currently in effect. This is useful in situations when the default isolation level is 3 but the query does not need to hold the shared locks until the transaction ends.

# Checking for Blocked Processes

- Use `sp_who` and `sp_lock` to determine what locks are currently being held, if any, and who is blocked by whom

```
> sp_who
> go
spid status   loginame ...blk  dbname    cmd
---- -----     ------    - ---   ------    --
...
6    sleeping  anna       ... 0  salesdb  SELECT
7    sleeping  robby      ... 6  salesdb  UPDATE

> sp_lock
> go
spid locktype      table_id    page    dbname
---- -------        --------     ----    --------
1    Sh_intent     384004399    0     master
6    Sh_table-blk  800003316    0     pubtune
```

- If blocking lasts for any length of time, investigate and/or intervene – possibly kill the offending process

**to kill a process**

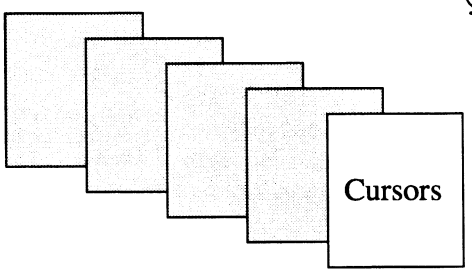
Syntax: `kill` *`spid`*

Only System Administrators can kill processes, and this permission cannot be transferred.

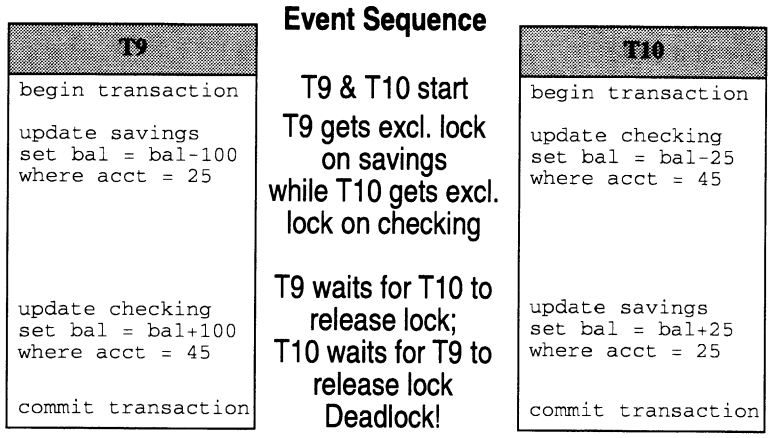
Select object_name (id) to see the object being locked.

# Cursors and Locking

- Cursors enable you to access the results of a select statement one row at a time
- While they are open, page locks may be held longer
- Cursors are discussed at length in a module of their own!



# Deadlock




- Could this have been avoided?

**Deadlock**

- Deadlock occurs when two or more transactions are holding locks or resources that the other needs to complete processing.
- SQL Server automatically detects deadlock and chooses the user whose process has accumulated the least amount of CPU time as the "victim" and aborts the process with an error message.
- Despite the automatic detection, even the "winner" in a deadlock situation sees a dramatically increased response time, waiting for the deadlock to be detected.

**Programming for Deadlock**

- Applications need to program for the possibility of deadlock by checking for error 1205.
- If deadlock occurs, an application should resubmit the transaction.

# Avoiding Deadlock

- Techniques to minimize the possibility of deadlock:
  - Have all transactions access tables in the same order
  - Use holdlock only when repeatable reads are required
  - Avoid long-running transactions
  - Avoid user interaction on the transaction length
  - Avoid many simultaneous executions of DDL commands

## Locking and Performance

- Locks affect performance when:
  - Processes wait for locks to be released
  - Transactions result in frequent deadlocks
- Tables are locked while indexes are created
  - clustered indexes get exclusive lock on entire table
  - nonclustered indexes get shared lock, and lock out updates

**Locks Affect Performance**

Any time a process "blocks" waiting for another process to complete its transaction process, the overall response time and throughput is diminished.

There is no "timeout" facility in SQL Server--processes remain blocked until the lock is released.

# Reduce Locking Techniques

- Lower fill factor to reduce contention
- Create separate, possibly redundant tables
- For small tables, pad records to reduce page density
- Avoid hot spots by choosing appropriate clustered index
- Rebuild indexes during off hours if possible

**fillfactor**

The fill factor diminishes the likelihood of needing the same page when there are random updates.

**Hot Spots**

Hot Spots occur when most access is for the same page, such as in a heap. A clustered index to distribute the data would help.

## Reducing Locking Techniques (continued)

- If possible, access tables in same order throughout application
- Maintain an accurate model of heavily used transactions
- Keep transactions short
  - no user interaction
  - no network interaction
  - minimal computation
  - single batch
  - use holdlock only when absolutely necessary

**User Interaction**

Since locks are held until a transaction is committed, a user can hold up a commit and degrade the system.

**Short Transactions**

The longer the transaction, the longer the exclusive or update locks are held, thus blocking other activity, and increasing the likelihood of more deadlock situations.

# Sample Problem

- Performance for a process degrades when it is executed with more than one other process running

# Collect Data

- See what processes are running on the system (sp_who)
- Display locks and blocking (sp_lock)
- Investigate processes that are suspect
- See what intent (select, update, etc.) these processes have
- Check length and content of transactions in suspect processes
- See what tables and indexes are accessed to see what the density is and the likelihood of more than one process needing the same page

# Collect Data: Results

- 10 processes running, and one process is blocking all the others
- Resource being held is *authors*
- There are 10 rows per page
- The clustered index on *au_id* is used for all updates
- Processes never update the same author at the same time

# Formulate Hypothesis

- What is wrong?
  - Processes are trying to update the same page
- What can you do?
  - Rebuild clustered index with lower fillfactor to spread out the data rows more and reduce possible locking
  - Create a clustered index on another column to rearange the physical location of the data

# Test Hypothesis

- Drop clustered index in test environment
- Create clustered index on authors with fillfactor = 50
- Retest with processes that had been blocking
- Result: locks reduced

# Implement if Confirmed

- Drop indexes on *authors*
- Make sure you have disk space to hold 120% of the size of the *authors* table
- Create a clustered index on *authors* with new fillfactor
- `sp_recompile authors`

# Locking: Questions to Ask

- What are the most frequently executed transactions?
- Have the transactions been grouped according to the objects they reference?
- Can each group of transactions be implemented without the need for locking?
- For those transactions that require locking, have they been implemented in a dead-lock free manner?
- What are the most heavily used objects in the system?

# Lab 8 – Locking and Performance

- Display locks that occur when multiple processes access the same data

Optional:

- Use SQL Monitor to observe locking on the SQL Server

# Summary

- Locks enforce consistency of data in a multi-user environment
- Lock isolation levels:
  - Level 1 prevents "dirty reads"
  - Level 2 prevents "non-repeatable reads"
  - Level 3 prevents "phantom reads"
- SQL Server
  - Defaults to level 1
  - Enforces level 3 with: holdlock, set transaction isolation level
  - No level 2, but level 3 includes level 2
- Deadlocks: Server will determine a winner and exterminate a loser
- Reducing locking techniques

9

# Managing Memory

## System 10 Performance & Tuning

Version 2.2



SYBASE®

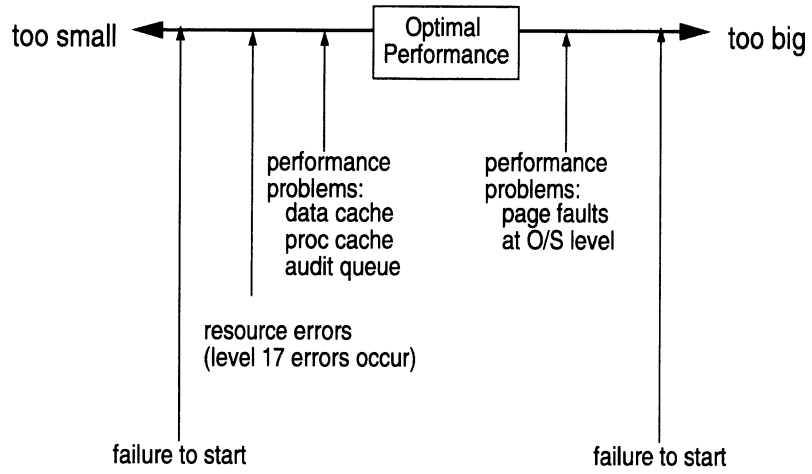
*The Enterprise Client/Server Company™*

# 9

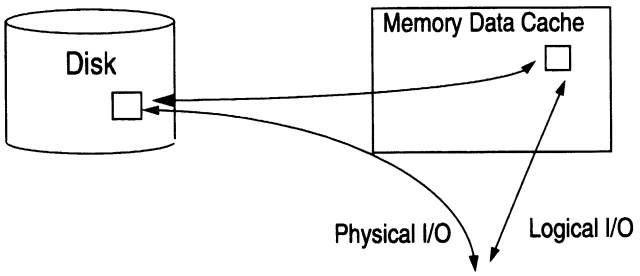
# Objectives

- Understand the relationship between memory and performance
- Estimate how to configure server memory size
- Examine and change server configurations which affect memory usage

# Memory Size


too small
Optimal Performance
too big
performance problems:
data cache
proc cache
audit queue
performance problems:
page faults
at O/S level
resource errors
(level 17 errors occur)
failure to start
failure to start

## More Memory Improves Performance



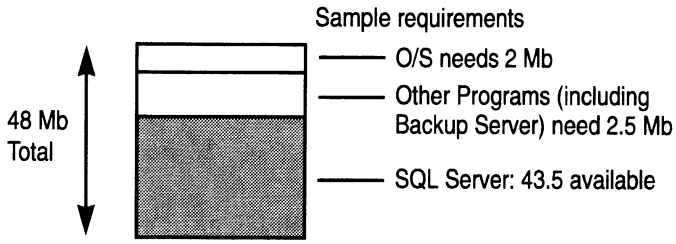

- Logical I/O is much faster than physical I/O

### SQL Server Caches

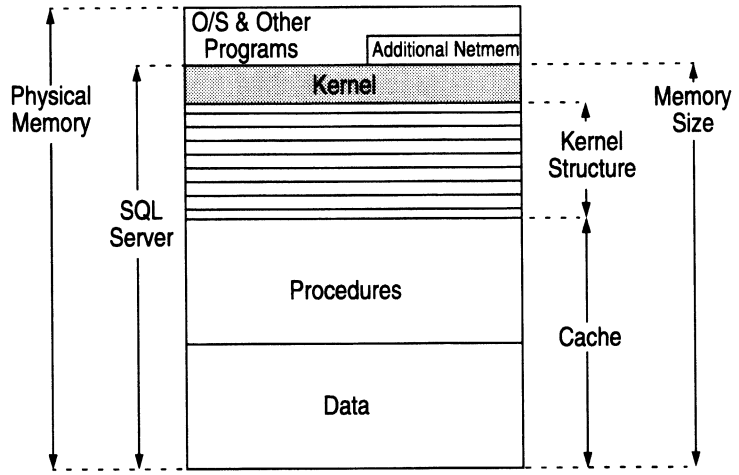
- **Data:** Both indices and tables cached for update, delete, insert and select.
- **Procedures:** Procedures are compiled when loaded into cache. They are re-run directly from cache.
- SQL Server estimates physical (disk) I/O at 18ms, and logical (in memory) I/O at 2ms.

# How Much Memory to Give SQL Server

- To estimate the size of memory for your SQL Server:
  - calculate the memory required for O/S and other programs; subtract this from the total available physical memory



# Memory: The "Big Picture"




## Memory Requests

- Request memory at install time.
- Less memory will be allocated if requested memory is greater than available memory.
- Additional netmem is allocated outside of "normal" SQL Server memory allocation, but still attached to SQL Server.

# dbcc memusage

- Output is in three parts:

Code:

| | Meg | 2K Blks | Bytes |
|---|---|---|---|
| Configured memory | 16.0000 | 8192 | 16777216 |
| Code size | 2.5686 | 1316 | 2693400 |

...

Data Cache: Top 20 Objects

| Dbid | Object Id | Index Id | 2K Buffers |
|---|---|---|---|
| 6 | 8 | 0 | 1774 |

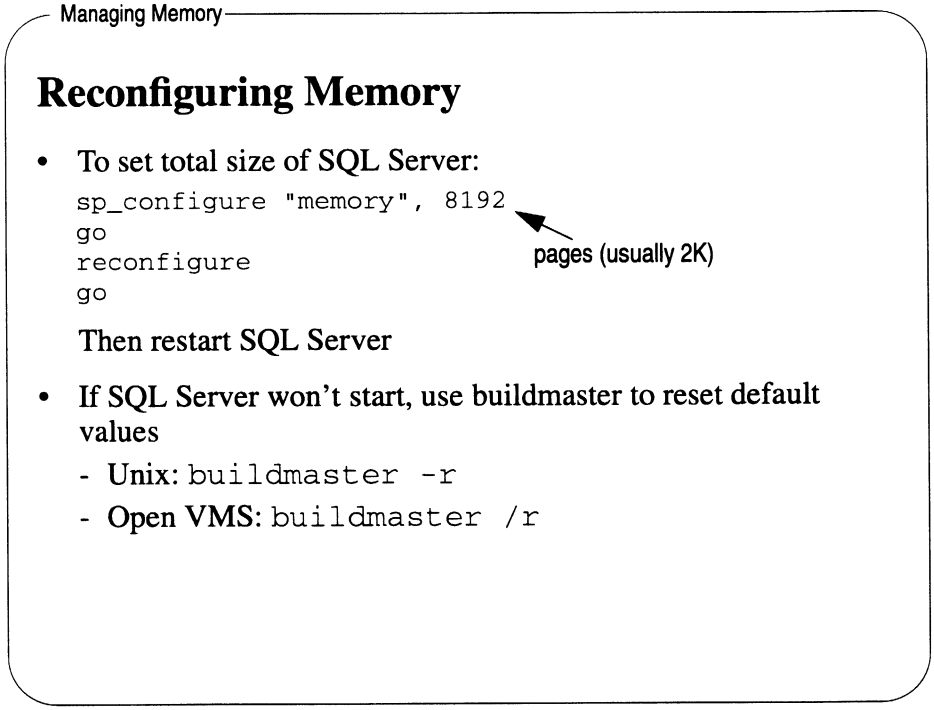
...

Procedure Cache: Top 20 Stored Procedures

Database Id: 1
Object Id: 1456008218...
Type: stored procedure...
Size of plans: 0.109215 Mb...

**dbcc memusage**

```
dbcc traceon (3604) /* redirect output to screen */
dbcc memusage
dbcc traceoff (3604) /* send output back to log */
```




**Reconfiguring Memory**

- Buildmaster -r restores all the default configurations.

**Using buildmaster option**

- If you use this option to buildmaster, the configuration block that contains all the system start-up parameters is overwritten with the default values. Be sure you have a record of what these were so you can reset them.
- Do not use this option unless SQL Server will not start!
- See Utilities manual for details.

# Displaying Configuration Values

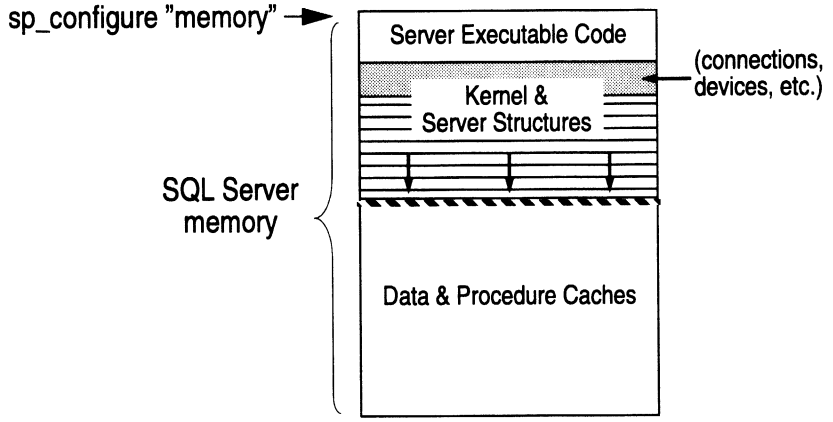
- Use sp_configure to display current configuration values

```
1> sp_configure
2> go
 name                           minimum     maximum     config_value run_value
 ------------------------------ ----------- ----------- ------------ -----------
 recovery interval                    1       32767           0           5
 allow updates                        0           1           0           0
*user connections                     5  2147483647           0          25
*memory                            3850  2147483647        8192        8192
*open databases                       5  2147483647           0          10
*locks                             5000  2147483647           0        5000
*open objects                       100  2147483647           0         500
*procedure cache                      1          99           0          20
 fill factor                          0         100           0           0
 time slice                          50        1000           0         100
 database size                        2       10000           0           2
 tape retention                       0         365           0           0
 recovery flags                       0           1           0           0
 nested triggers                      0           1           1           1
*devices                              4         256           0          10
 remote access                        0           1           1           1
*remote logins                        0  2147483647           0          20
*remote sites                         0  2147483647           0          10
*remote connections                   0  2147483647           0          20
 pre-read packets                     0  2147483647           0           3
 upgrade version                      0  2147483647        1000        1000
 default sortorder id                 0         255          50          50
 default language                     0  2147483647           0           0
 language in cache                    3         100           3           3
 max online engines                   1          32           1           1
 min online engines                   1          32           1           1
 engine adjust interval               1          32           0           0
 cpu flush                            1  2147483647         200         200
 i/o flush                            1  2147483647        1000        1000
 default character set id             0         255           1           1
*stack size                       20480  2147483647           0       28672
 password expiration interval         0       32767           0           0
*audit queue size                     1       65535         100         100
-> additional netmem                  0  2147483647           0           0
*-> default network packet size     512      524288           0         512
*-> maximum network packet size     512      524288           0         512
*extent i/o buffers(for sort)         0  2147483647           0           0
 identity burning set factor          1     9999999        5000        5000

(38 rows affected, return status = 0)
```

*marks options which impact memory usage (within "memory")
->marks options which impact memory usage (outside "memory")

# Most Options Affect Size of Server Structures

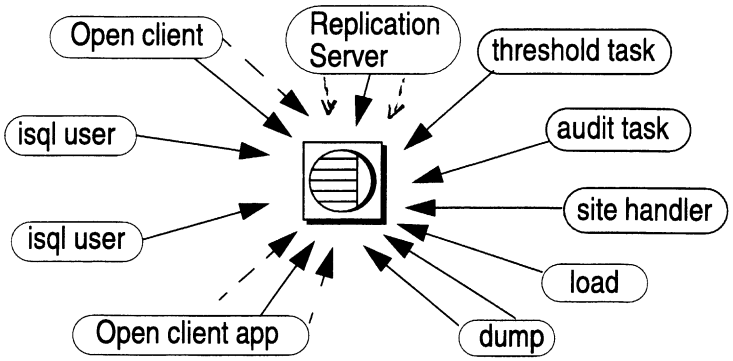
- "Memory" sets total size of server




- Increasing these options decreases memory available for data and procedure caches

| | |
|---|---|
| **User Connections** | 50 Kb each |
| **Devices** | 512 Bytes each |
| **Open Databases** | 17,408 Bytes each |
| **Open Objects** | 315 Bytes each |
| **Locks** | 72 Bytes each |

# Connections: Example




- The "user connections" option sets the maximum number of user connections that can be simultaneously connected to SQL Server

**Example**

```
sp_configure "user connections", 50
go
reconfigure
go
```

**Memory per connection** ~ 50 Kb *plus any* increase in stack size

**Total Memory Usage** 50 connections = 50 * 50 Kb = 2500 Kb
which removes 2.5 MB from your procedure and data caches

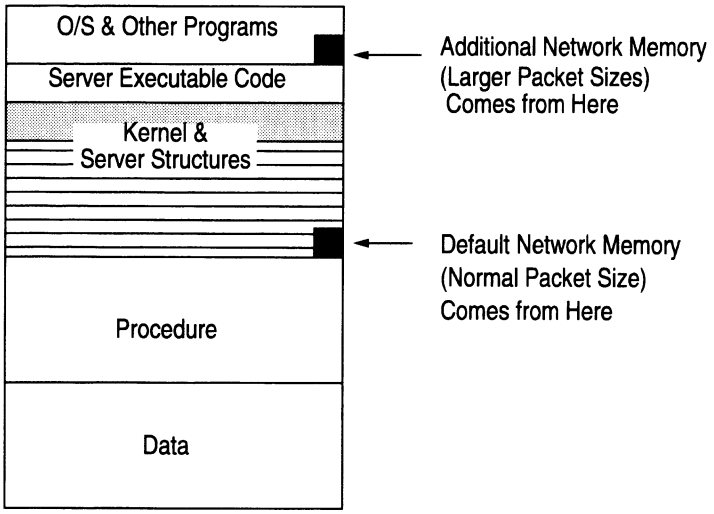
See *SYBASE Troubleshooting Guide,* "SQL Server Configuration Issues" for more information on estimating memory use for user connections and other options.

**Other Devices** Other devices also use connections to SQL Server

# Options Which Consume Memory: Databases, Locks, Objects, Devices

- Devices - The number of partitions to be allocated to the use of SQL Server (i.e., maximum vdevno)
- Open Databases - One per database
- Remote Sites - Each takes one connection; one per remote SQL Server
- Remote Logins - Total simultaneous remote users
- Open Objects, Locks - Scale proportionally with database complexity and number of users OR wait for errors and increase when encountered

## Network Memory Increases Size of Server




Extra size of packets comes from "additional netmem."

Larger packet sizes are good for bulk transfer, text and image load.

# Extent I/O Buffers Increase Size of Server

Reserves pages in data cache for exclusive use in index creation

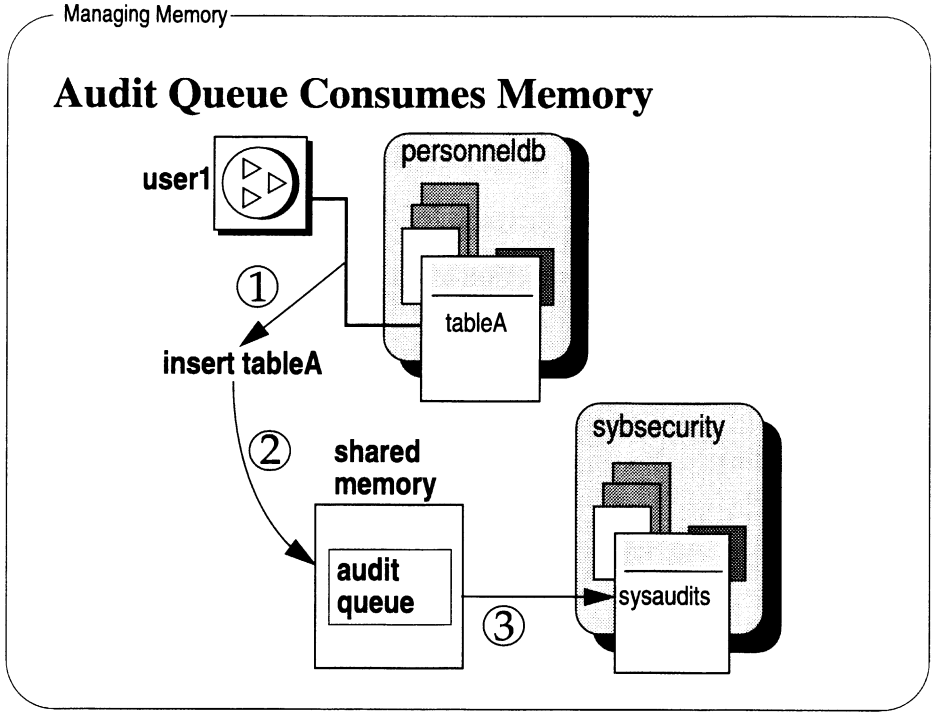
- Dramatic performance increase for "create index" statement
- Only *one* index under creation at a time gets use of "special" cache
- Each buffer takes 16Kbytes from SQL Server allocated memory
- Adjust memory allocation using sp_configure ("memory") if neccessary

**Example**

```
sp_configure "extent I/O buffers", 20
```

**Running Out of I/O Buffers**

If a index on a large table is being built, it may fail if extent i/o buffers are not large enough

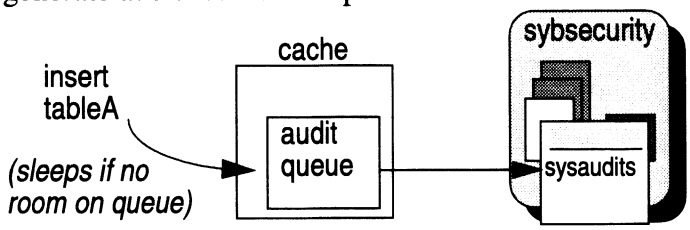



**Flow of Audit Data**

1. User tries to insert into table A, and auditing has been set up for this event.
2. After permission checks, information about this command and process is recorded in the audit queue in shared memory (separate from the data and procedure caches).
3. Rows from the audit queue get copied to disk (*sysaudits* in *sybsecurity* database) when the audit task gets its turn.

# Auditing and Performance

- Heavy auditing can affect performance
  - If the audit queue (in memory) is full, user processes that generate audit records sleep




- Performance can also suffer if *sybsecurity* is on a heavily-used disk
- SSOs can affect performance by configuring "audit queue size"
  Larger queue -> better performance
  Smaller queue -> fewer audit records vulnerable in case of failure

**Size of Audit Queue**

Default records in audit queue: 100
Size of record: 424 bytes
Size of default width queue 42K

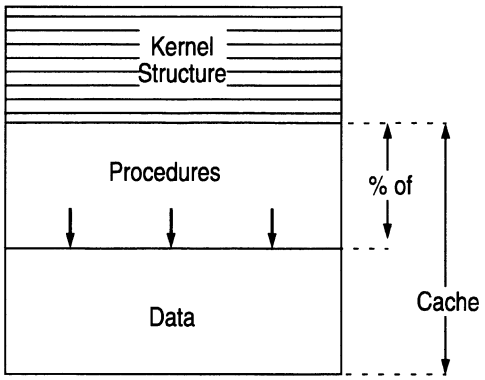
**Audit Queue Size & Performance**

Larger audit queue improves performance at increase of risk to audit data. Larger audit queue takes memory from data and procedure caches.

# Split Remaining Memory:

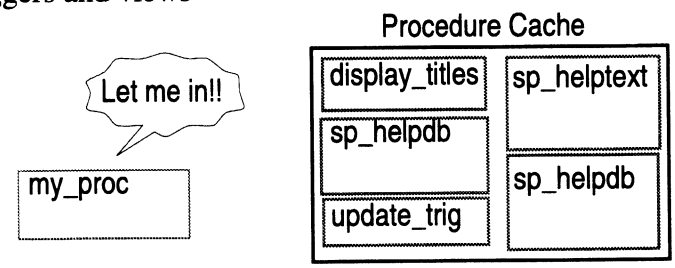
## Data & Procedure Caches

- Configure the procedure cache as a percentage of the memory remaining



# What Procedure Cache Is For

- Procedure cache holds the optimized query plans for procedures, triggers and views




- Procedures are not re-entrant, each simultaneous user gets one copy
- If procedure cache is too small, user processes may have to wait

**Faster Access to Some Objects via Procedure Cache**

- Procedures, triggers, and views are optimized when they are loaded into the procedure cache.
- If the procedure is in cache, it does not need to be re-compiled to be reused.

**If There Is Not Enough Room in Procedure Cache**

- If there is no room in the procedure cache, the procedure may not run.

**Multiple Copies of Procedures**

- Each user may have in cache multiple copies of the same query plan

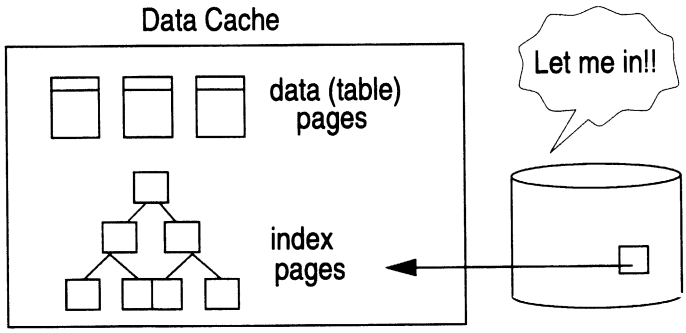
# Procedure Cache Sizing

- How big should the procedure cache be?
- The following formulas make a good starting point:

| | | |
|---|---|---|
| **Procedure Cache Size** | = | (max number of concurrent users) * (size of largest plan) * 1.25 |
| **Minimum Procedure Cache Needed** | = | (# of main procedures) * (average plan size) |

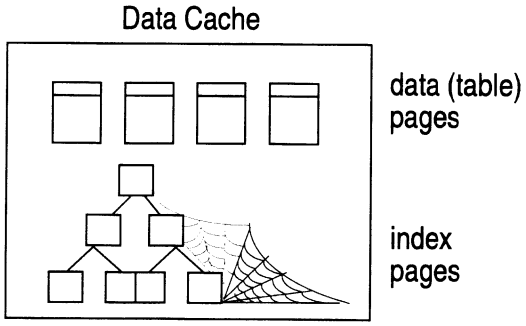
- Remember, when you increase the size of the procedure cache, you DECREASE the size of the data cache

# What Data Cache is For



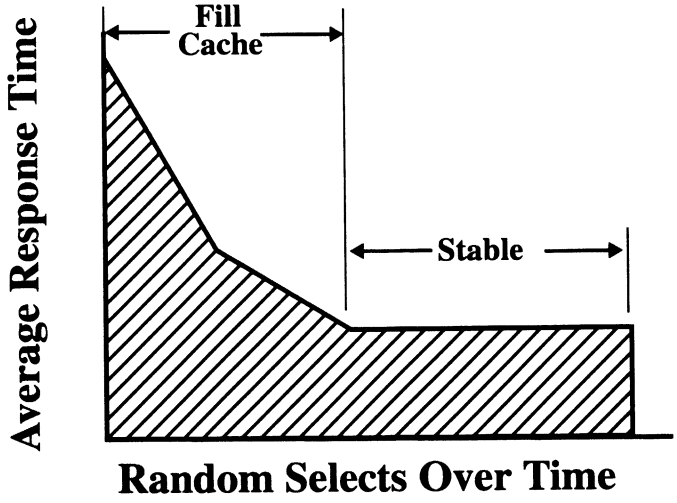

- Holds most recently used data and index pages.
- Server always looks in cache first.

# Page Aging in Data Cache

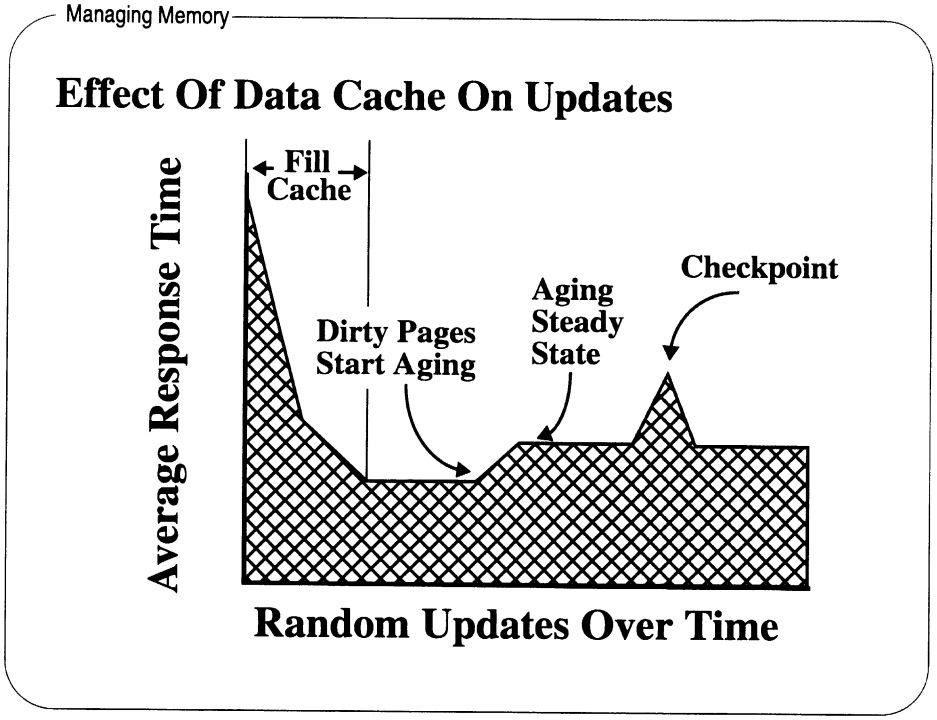



- Index pages age out of cache more slowly than data pages
- Least-recently-used (LRU) strategy used for data pages (*not* index pages)

## Effect Of Data Cache On Retrievals




As the cache stabilizes, each new request has a standard chance of finding a page already in memory.




Updates cause some pages to be changed (marked "dirty"). When the cache is full of dirty pages, a page must be written before a new page is read.

Checkpoints write all dirty pages to disk.

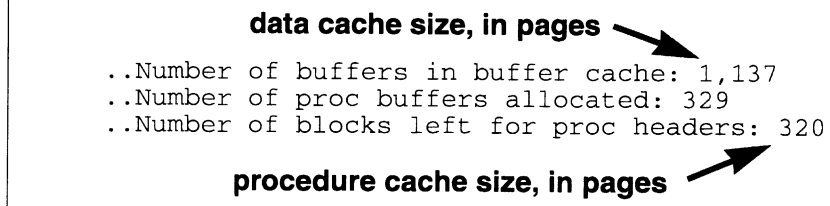
# Data Cache Performance

- Determined by examining the **cache hit ratio**
  - the percentage of page requests that are satisfied in memory
  - 100% is essentially a memory resident database
  - 5% indicates that the cache is too small
- Displaying the cache hit ratio
  - dbcc tablealloc
  - dbcc indexalloc
  - set statistics I/O

$$\frac{\text{logical}}{\text{logical + physical}} = \text{hit ratio}$$

# Finding Out Cache Sizes

- When SQL Server is booted, the **error log** states how much data and procedure cache are available




- dbcc memusage also provides this information
  - its output includes overhead that is not available for use

**buffer cache**
- Pages available for disk reads

**proc buffers**
- One per proc

**proc headers**
- Pages available for procedure
- Queue plans in memory

# Configuration Example

- Assume total system memory of 64Mb, with a dedicated SQL Server
  - What configuration values would you use for 60 connections?
- Sample calculation

| | |
|---|---|
| Total Memory | 64 Mb |
| Size of OS | -2 Mb |
| Other OS programs | -2 Mb |
| Avail. for Server | 60 Mb |
| 60 conn x 50Kb | -3 Mb |
| Avail. for Cache | 57 Mb |
| Proc Cache | -3 Mb |
| Data Cache | 54 Mb |

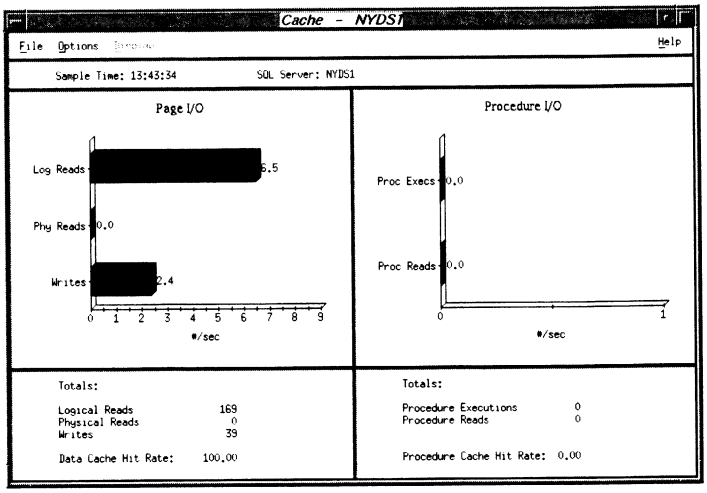
If we assume each connection uses 2 stored procedures:
60*2 plus some = 150

If we assume proc size is 15Kb:
150 procs * 15Kb = 2250Kb, round *up* to 3 Mb. Procedure cache represents 6% of total cache..

# Memory Allocation: Questions to Ask

- What kind of processing are you doing?
- How many simultaneous user connections must be supported? How many devices?
- How many databases and objects will be opened simultaneously?
- What is the maximum number of locks expected?
- How many stored procedure triggers are used simultaneously?
- What is the cache hit ratio during periods of heavy use?

# SQL Monitor - Cache Window




**Function**

Determines the efficiency of data and procedure cache

**Uses**

Data and procedure cache ratios

Memory Usage

Procedure vs. data cache distribution

**Display**

- Page I/O
  - logical reads
  - physical reads
  - writes
  - totals
  - data cache hit rate
- Procedure I/O
  - procedure executions
  - procedure reads
  - totals
  - procedure cache hit rate

# Summary: Memory Size & Balance

| | | |
|---|---|---|
| **O/S**<br>**X Windows**<br>**Etc.** | extent I/O buffers | additional netmem |
| **SQL Server Kernel** | | |
| **User Connections:** | 50 Kb * number of users | |
| **Devices:** | 512 bytes * number of devices | |
| **Open Databases:** | 7400 bytes * number of databases | |
| **Open Objects:** | 315 bytes * number of objects | |
| **Locks:** | 72 bytes * number of locks | |
| **Extent I/O Buffers:** | 8 * 2K * number of buffers | |
| **Network Buffers:** | number of users * 3 * default network packet size | |
| **Stack:** | stack size | |
| **Audit Queue:** | audit queue size | |
| **Procedure Cache** | proc cache percentage * remaining memory | |
| **Data Cache** | whatever is left over | |

**How SQL Server Uses Memory**

Balance memory between each specialized use.

# Sample Problem

- Slow running procedure:
  `exec SalesByStore`
- Takes 5 minutes to complete
- This is unacceptable

- What would you do?

# Collect Data: Commands

- show plan
- set statistics i/o on, set statistics time on
- execute query and see the logical and physical I/O and how long the query takes
- sp_help on relevant tables
- dbcc memusage
- dbcc tablealloc
- dbcc indexalloc
- sp_configure

# Collect Data: Results

- Results:
  - statistics I/O shows high physical I/O break (even with repeated execution)
  - sp_configure shows 80% procedure cache
- What do you think is going on?
- What can you do about it?

# Formulate Hypothesis

- Hypothesis:
  - Procedure cache set too high
- How should we fix it?
  - Set procedure cache to 20% (down from 80%)
  - Get data from dbcc memusage to determine what size should be
- Before we do that...
  - Find out why procedure cache was set so high--perhaps there is a reason

# Test Hypothesis

- Reconfigure and reboot
- Re-run stored procedure after setting new values
- Compare execution times
- Compare cache hit ratios

# Implement Solution

- If confirmed, implement solution on production system
  - sp_configure "procedure cache", 20
  - reconfigure
  - Reboot server
- Why might you not see immediate improvements in the performance and cache hit ratio?

# Lab 9 – Memory and Performance

- Evaluate the effectiveness of the current memory allocation
- Experiment with changing it

# Summary

- More memory improves performance
- Configurable options that affect memory (inside)
  - User connections
  - Memory
  - Open databases
  - Locks
  - Open objects
  - Procedure cache
  - Devices
  - Extent I/O buffers
- Configurable option that affect memory (outside) includes additional netmem
- Page aging:
  - Data pages (LRU)
  - Index pages (slower)
- Procedure cache sizing / data cache sizing
- Data cache hit ratio

10

# Maintaining High Performance

## System 10 Performance & Tuning

Version 2.2



SYBASE®

*The Enterprise Client/Server Company™*

# 10

# Objectives

- Explain how maintenance activities can affect system performance
- Describe how to minimize the performance impact of maintenance activities

See Chapter 12, "Fine Tuning Performance and Operations" in the *SQL Server System Administration Guide* for an overview of tuning guidelines regarding maintenance activities.

# Why Study Maintenance Activities?

- Processing is slow because maintenance activities are interfering with other processing

# Topics

- Database and index creation 
- Backup and recovery
- Bulk copy
- Database consistency checker

# Creating Databases

- Issues
  - Database creation is I/O intensive and may compete for limited memory – performance may suffer
- Guidelines
  - Create databases during off hours if possible
  - Do not create databases, do database dumps or loads, or run dbcc commands at the same time, if possible
  - Use `for load` option to `create database` if a `load database` command is going to be used to restore a database

for load = zonder initialiseren van overblijvende pages

**Extent I/O Buffers**

- There is a single 8 page extent buffer (in memory) that is used for `create database, database dump/load,` and `dbcc` commands
- The other extent i/o buffers that can be configured with `sp_configure extent i/o buffers` are used only for indexing

**System 10 vs. Earlier Releases**

- During beta version testing, it took 26 minutes to create a 2Gb TPC-B database striped across 6 devices, compared to 4.5 hours on a pre-System 10 version.

# Creating Indexes

- Issues
  - Clustered index creation locks out all table activity (exclusive table lock)
  - Non-clustered locks out update activity (shared table lock)

# Creating Indexes: Using Extent I/O Buffers

Configuring extent i/o buffers allows SQL Server to use 8-page buffers for reading and writing intermediate and final results

- Only one user at a time can use extent I/O buffers during index creation--other users will use page-at-a-time I/O
- Increasing this parameter decreases the memory available for the procedure and data caches.
- Start by setting extent I/O buffers to 10
- Settings above 100 yield only marginal benefits

**Off-hours Maintenance**

- It makes sense to schedule index maintenance for off-hours.
- Then, I/O extents can be allocated for optimum performance.
- When done, de-allocate the extra I/O extents and resume normal memory allocations.
- Note: you need to shut down and restart the server in order to change the number of extents allocated.

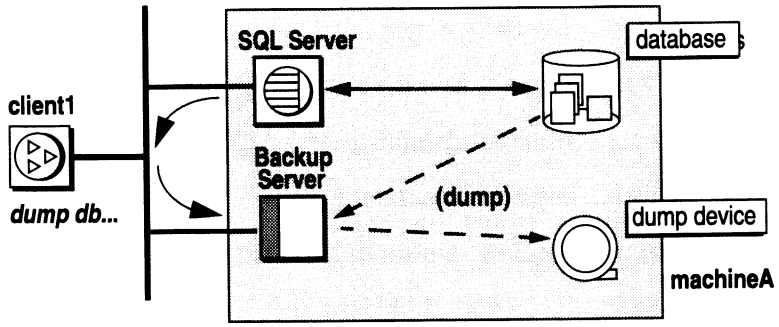
# Creating Indexes: More Suggestions

- Use fill factor to control the density of index pages
  - Only affects index creation
  - Need to rebuild indexes to control density after updates have occurred
- If using incremental dumps, a potentially lengthy recovery later
- If your data is already in the desired clustered index order, use `with sorted_data` option when creating clustered indexes

# Topics

- Database and Index Creation
- Backup and Recovery 
- Bulk Copy
- Database Consistency Checker

# How Backup Server Works




- All SQL Server backups are performed by a Backup Server
- The backup architecture uses a client/server paradigm, with SQL Servers as clients to a Backup Server

**Local Backups**

The foil illustrates a local backup, which works as follows:

Local Backup Server gets instructions from SQL Server (via remote procedure calls) on which pages to dump or load, which backup devices to use, etc., and (Backup Server) performs all the disk I/O. SQL Server does not read or send dump/load data, just instructions.

**Remote Backups**

Backup Server also supports backups to remote machines. In that case, the local Backup Server performs the disk I/O related to the database device and sends the data over the network to the remote Backup Server, which stores it on the dump device.

**Online Backups**

Backups can be done while database processing is being performed. Clearly, such processing will have an affect on other transactions, but do not be afraid to back up small, critical databases as often as necessary to satisfy the reliability requirements of the system.

See Chapter 7, "Developing a Backup and Recovery Plan" in the *SQL Server System Administration Guide* for a complete discussion of backup and recovery strategies.

# Backup Server: Performance Features

- Supports 32-way device striping
- Drives I/O devices at maximum throughput
- Performs minimal handshaking with SQL Server
- Performs backups on live databases
- Can dump to local or remote machines
- Yields faster dump and load rates, less impact on performance of other transactions
- Minimal impact on updates

# Recovery Interval

- Maximum number of minutes per database that SQL Server should use to recover, should that be necessary
  - Configurable by SA
  - Used by SQL Server to evaluate when to write changes to disk
- Provides no guarantees, but improves your chances of shorter recovery time
- Example

```
sp_configure "recovery interval", 3
go
reconfigure
go
```

## Recovery Interval

The default recovery interval (5 minutes) is sufficient for most applications and impacts performance by an acceptable amount.

Shorter intervals should be used only if functional requirements dictate a faster recovery period.

See Chapters 8 & 9, "Backing Up and Restoring User Databases" and "Backing Up and Restoring the System Databases" in the *SQL Server System Administration Guide* for a complete discussion of user and system database backup strategies.

# Transaction Log Dumps

- Use the thresholds to ensure that you don't run out of space in the log and to avoid reaching last chance threshold
- If you don't need to save the log, use the with truncate_only option
- If you are using incremental dumps
  - Use striping to improve performance

# Topics

- Database and Index Creation
- Backup and Recovery
- Bulk Copy 
- Database Consistency Checker

# Bulk Copy: Performance Issues

- Bulk copy *in* is fastest if no indexes are on a table
  - "select into/bulkcopy" option must be set for the database with sp_dboption
  - Rules and constraints not enforced; defaults *are* enforced
  - Data not logged, but page allocation *is* logged (for recovery purposes)
  - Database checkpointed on completion
- If there are indexes on a table, a slower version of bulk copy is automatically used
  - select into/bulkcopy option does not have to be set
  - Rules and constraints not enforced, triggers not fired, defaults *are* enforced
  - Data is logged, as well as page allocation
  - Database checkpointed on completion

**Reminder**

Use `sp_dboption` to set `select into/bulk copy` to *true* in order to use `bcp` to copy data into a database.

**Triggers and BCP**

Triggers are *not* invoked when using either the fast or slow version of bcp.

fast mode : dump tb nodx
na aploop

# Bulk Copy: Suggestions

- Tune batch size and set truncate log on checkpoint option
- Find optimal network packet size
- When replacing an entire table's data (large)
  - Truncate the table
  - Drop indexes
  - Load data
  - Create indexes
- When adding 10% to 20% or more
  - Drop non–clustered indexes, load data, then create non-clustered indexes
- Bulk copying large tables in or out may affect other users' response time

**Options**

Use -b option to bcp to specify batch size; use -A option to specify network packet size..

# Topics

- Database and Index Creation
- Backup and Recovery
- Bulk Copy
- Database Consistency Checker 

# Database Consistency Checker (dbcc)

- dbcc
  - checktable
  - checkdb
  - checkalloc
  - tablealloc
  - indexalloc
  - checkcatalog
  - reindex
- It is important to run these periodically, but they affect performance
  - Run after hours is possible

**Is Consistency Important?**

If you back up a corrupt database, the backup is useless.

**dbcc**

checkalloc now requires fewer I/Os (related to GAM's and OAM's implementation).

Other dbcc commands are blocked while checkalloc runs.

# Lab 10 – Maintenance and Performance

- Measure performance of bulk copy under various conditions
- Measure performance of create index under various conditions

# Summary

- Maintenance activities that can affect performance
  - Creating databases and indexes
  - Backups and recovery
  - Bulk copy
  - dbcc
- These are all important!
- Perform after hours if possible
- Tune where possible
  - Batch size, network packet size, striping

11

# Guidelines for High-Performance Applications

## System 10 Performance & Tuning

Version 2.2



SYBASE®

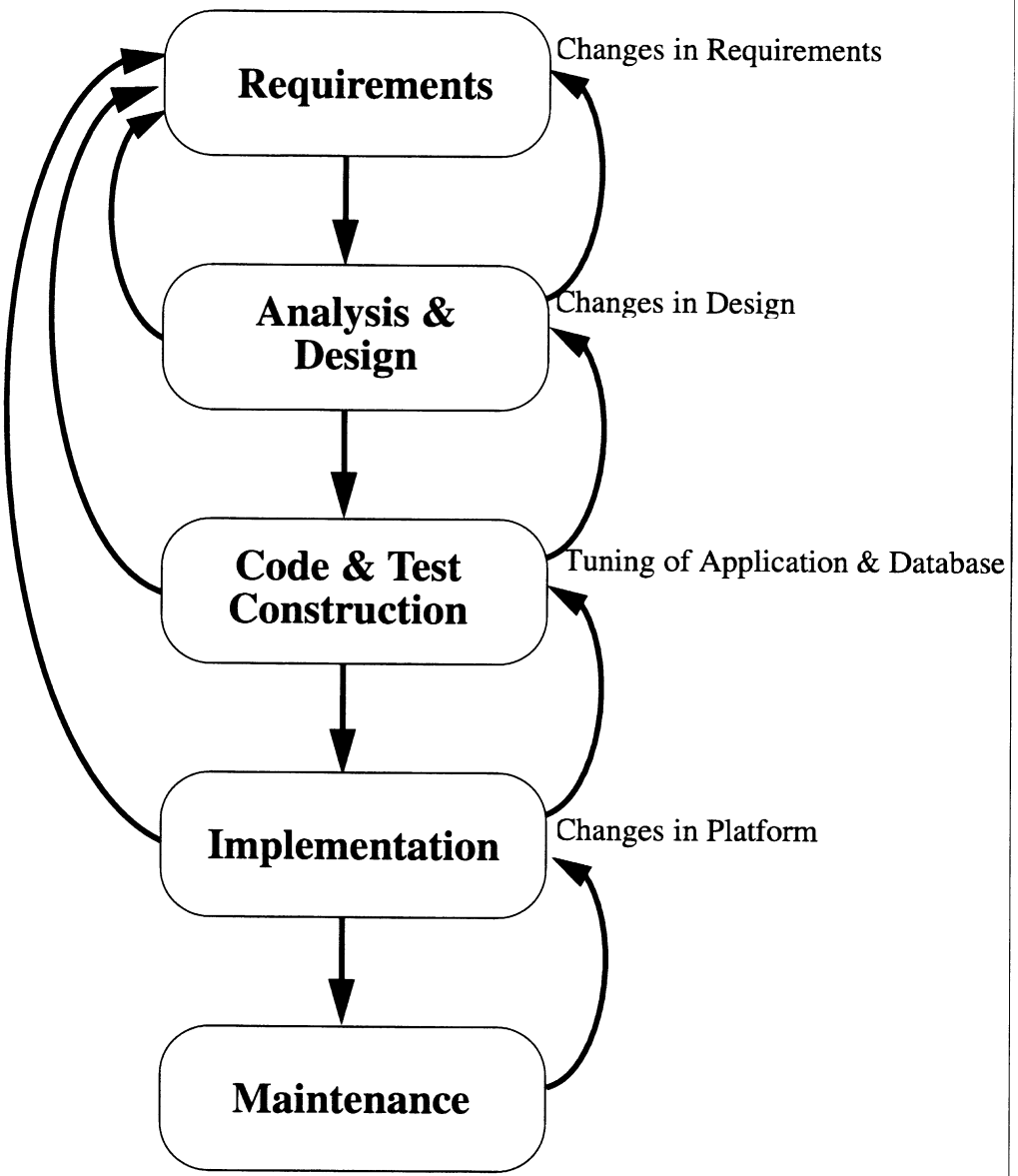
*The Enterprise Client/Server Company™*

# 11

# Objectives

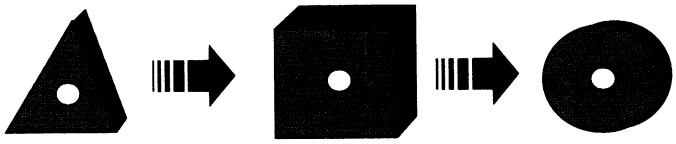
- Emphasize the points in the development life cycle affected by changes
- Summarize the initial design decisions affecting applications
- List design guidelines at each layer of the system

# Application Development Life Cycle



# Iterative Application Design

- Prototype, prototype, prototype

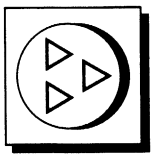


- Test different variations for critical services
- Record performance results for all variations
- Don't throw away old versions—they might be needed someday!
  - Document why they weren't used and why they might be useful in the future

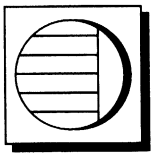
# Initial Design Decisions Affecting Applications

- Client vs. Server
- Interactive vs. Batch
- Application Design Summary
  - user interaction
  - security
  - auditing
  - common operations

# Client vs. Server?

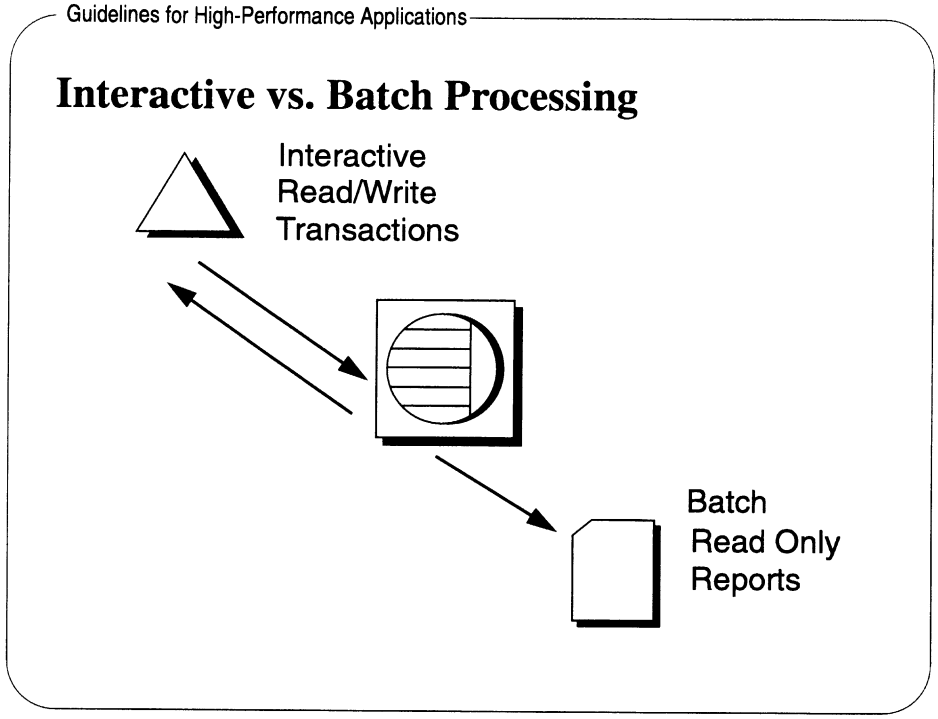


VS.



- String manipulation and formatting
- Format and range validation
- Valid values checking
- Dynamic SQL
- Ad-hoc queries
- Infrequently used queries
- Application dependent control and processing
- Number crunching

- Static, high-performance SQL
- Predefined or canned queries
- Frequently used /high-volume queries
- Application independent processing with potential for reuse
- Queries/operations which join multiple tables
- Security, Auditing
- Referential Integrity enforcement
- Client-Library supports Dynamic SQL




**Interactive Processing**

In general, interactive systems are very dynamic in their execution patterns and interact heavily with the server. Therefore, keep them from competing with batch processing, if possible.

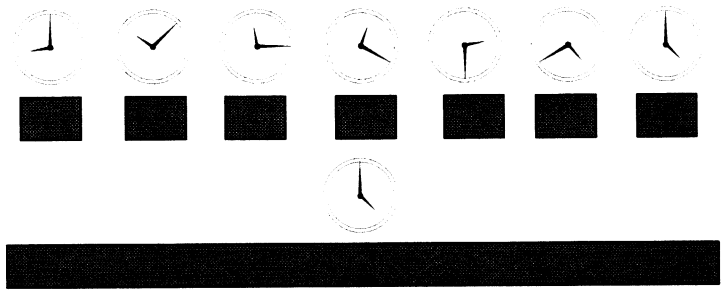
**Batch Processing**

In most systems, the work load is not constant; there is typically a peak interactive time and a peak non-interactive time. Batch usage tends to go up at the close of the month, quarter and year. Be aware of these patterns and you can use this information to improve the overall system performance.

# Additional Batch Guidelines

- Active hours vs. off-hours

- Short batches vs. long batches



**Off-hours Processing**

Example: A parts company takes on-line orders for auto parts during the day, and does not decrement inventory quantities until after the close of business. Updating inventory changes as they occur would cause system performance to plummet.

**Many Short Batches**

**Example 1:** A major hotel does a batch update every night to post the room charges. Since this batch updates every account, it acquires hundreds of locks and effectively prevents anyone from using the system while the charges are being updated. The hotel could switch the system to compute room charges when someone checks out. This might cause some performance problems during checkout periods, but that is limited by the number of available clerks and their typing speed anyway.

**Example 2:** A mutual fund firm runs a batch process to calculate the price of a fund share at the end of the day. The firm only has one hour between the close of the market and the reporting deadline. Instead of doing all the calculations at the end of the day, the price could be continually updated as trades are conducted.

# Application Design Summary

- User interaction
- Data validation
- Common operations
- Application logic
- Frequency of execution
- Intent of transaction
- Security
- Auditing

**User interaction** — Users are notorious for using systems in unexpected ways, often with unrealistic expectations.

**Data validation** — Validating data from one table with data from another table can cause extra disk accesses.

**Common Operations** — Many users doing the same thing, such as obtaining a new unique key, which can cause a bottleneck as users wait for locks to be freed.

**Application Logic** — Using the database instead of the application to do sorting or complex calculations can slow down retrievals significantly.

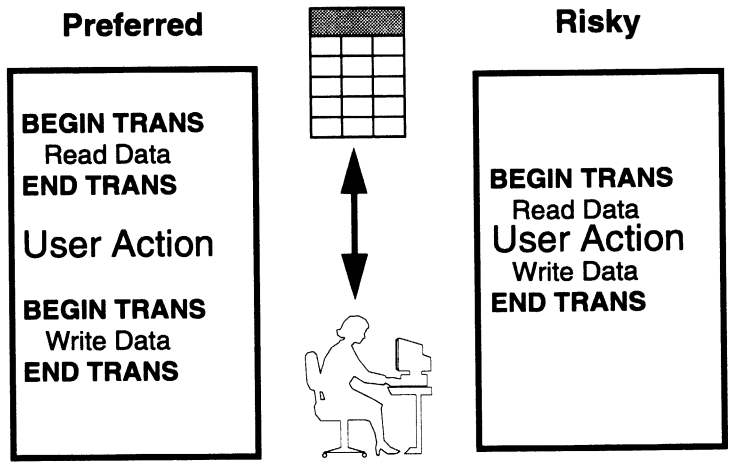
**Execution Frequency** — Transactions that execute once a month will affect performance much differently than those that execute 10,000 times an hour.

**Intent** — How the data is intended to be used will affect the way the application is designed and its impact on performance.

**Security** — Poor security implementation can degrade performance.

**Auditing** — Auditing will add overhead to each transaction.

## Managing User Interaction Example

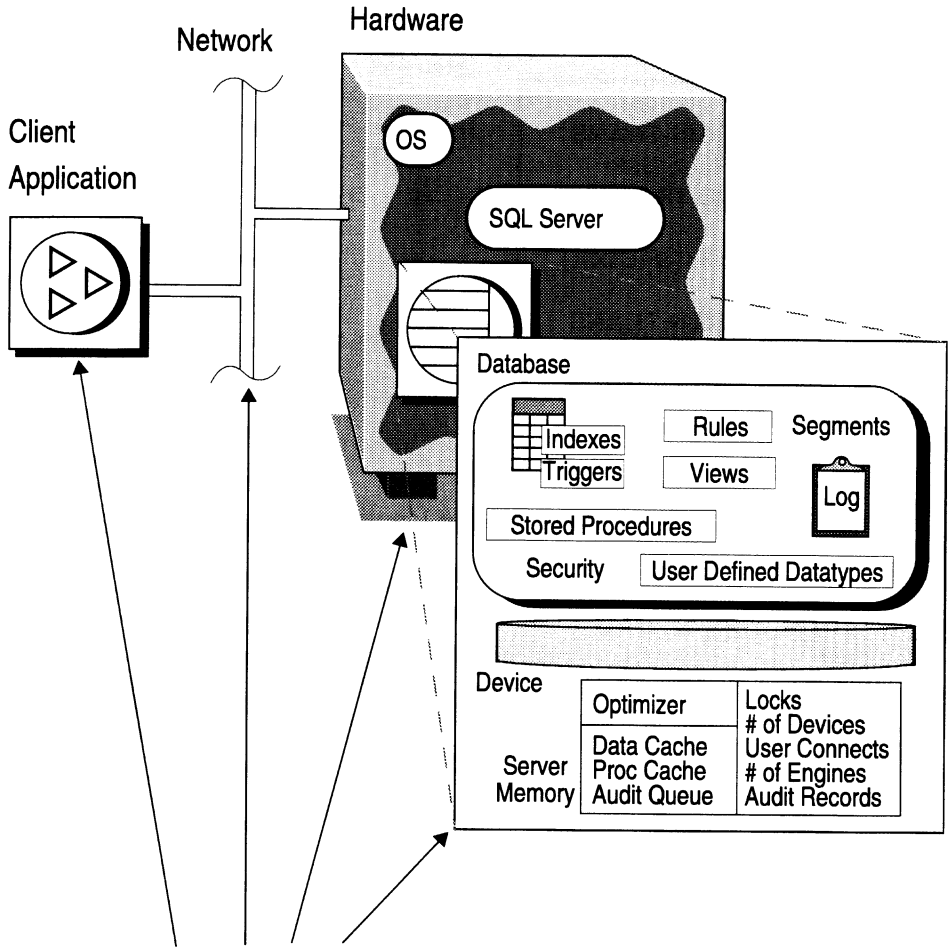



- Don't let the user control the length of the transaction!

**Embedding User Interaction Within a Transaction**

Never let the user control the length of a transaction. Someone could start a transaction, go to lunch, and not come back until hours later, leaving tables locked for the entire time.

# System Context Model




Application Guidelines Summary at each layer

# Guidelines: Network

- Specify the number of users on network
- Minimize network traffic
  - size of network packets
  - volume of network packets
- Specify server-to-server needs (RPC's) in terms of increase in network load
- Specify data replication impact in terms of increase in network load
- Connect to server using appropriate network packet size for activity

isql -optic -a

# Guidelines: Hardware

- Get as much memory as you can afford
- Use disk arrays for fault tolerant implementations
- Use multiple disks/controllers for normal implementations
- Use multiple CPUs if possible (SMP)
- Get more tape drives and more memory for backup servers

# Guidelines: OS

- Follow platform recommendations regarding device allocations
  - Unix: raw partitions
  - OpenVMS: RMS or foreign disk
- Set up *interfaces* files to allow remote access to other data sources

**Unix**

On Unix, raw partitions are preferred for production databases because Unix buffers writes and there is no way for SQL Server to know for sure whether data was copied or not.

On systems where reads and writes are direct, disk files are fine.

**OpenVMS**

Both RMS and foreign disks are fine, as reads and writes are direct.

***interfaces* File**

The *interfaces* file contains the names and network addresses of SQL Servers and Backup Servers.

# Guidelines: Server Configuration

| Configuration Item | Starting Values |
|---|---|
| User connections | +10%-30% than required |
| CPUs | 1 less than available number |
| Locks | Maximum number needed by app. |
| Devices | One more than number needed for currently allocated databases |
| Data & proc cache | Use algorithm presented |
| Audit queue size | Use default, monitor performance |
| tempdb size | Use guidelines presented |

# Guidelines: Disks/Devices

- Mirror devices for fault tolerant applications (but not for RAID)
- Keep transaction log on its own device
  - faster recovery
  - easier administration
  - reduces risk of lost database because of disk crash
- Isolate tables from non-clustered indexes to reduce disk arm contention and disk controller traffic
- Be aware that user-defined segments are ineffective to subdivide disk arrays

# Guidelines: Database

- Tightly-coupled tables belong in the same database
- Separate transaction log segments from data segments
- Separate large tables from their non-clustered indexes
- Separate volatile data from non-volatile data
- Isolate large tables to database by themselves
- Consider device striping

# Guidelines: Tables

| Consideration | Guidelines |
|---|---|
| Data access | Denormalize to improve performance |
| Data volatility | Use fill factor to leave free space |
| Critical transactions | Reduce contention with optimal table design |
| Referential integrity | Use triggers, stored procedures, and declarative ref. integrity techniques |
| Other data validation | Use stored procedures, application logic |

# Guidelines: Indexes

- Use indexes to enforce uniqueness of data
- Use clustered index to support range queries and other major accesses
- Use indexes to enforce secondary access requirements
- Use clustered indexes to eliminate hot spots and provide better distribution
- Use non-clustered indexes to cover queries
- Isolate non-clustered indexes from table (via segments)
- Use fillfactor to minimize index and data page splits

# Guidelines: Security

- Be aware of the performance costs of auditing
- Control access using
  - grant/revoke
  - views
  - stored procedures
  - roles

# Lab 11 – Designing Applications for Performance

- Determine which system issues would need to be considered for a given application

# Summary

- Where changes affect the application development cycle:
  - Requirements
  - Design
  - Tuning
  - Platform
- Client vs. Server
- Interactive vs. batch
- Application Design Summary
  - Guidelines

# Part 3
# Special Topics

System 10 Performance & Tuning

Version 2.2


*The Enterprise Client/Server Company™*

# Course Roadmap

## Part 1
## Designing Applications for Performance

| | |
|---|---|
| Module 1 | Overview of Performance Issues |
| Module 2 | Physical Database Design and Denormalization |
| Module 3 | Table Storage |
| Module 4 | How Indexes Work |
| Module 5 | Selecting Indexes for Performance |
| Module 6 | Query Optimization |

## Part 2
## Tuning Applications for Performance

| | |
|---|---|
| Module 7 | Distributing Data Across Devices |
| Module 8 | How Locking Affects Performance |
| Module 9 | Managing Memory |
| Module 10 | Maintaining High Performance |
| Module 11 | Guidelines for High-Performance Applications |

## Part 3
## Special Topics

| | |
|---|---|
| Module 12 | Networks and Performance |
| Module 13 | tempdb |
| Module 14 | Cursors and Performance |
| Module 15 | CPU Utilization Issues |
| Appendix A | Distributing Data with Replication Server |
| Appendix B | More About SQL Monitor |
| Appendix C | Problem Analysis Walkthrough |

12

# Networks and Performance

System 10 Performance & Tuning

Version 2.2



SYBASE®

*The Enterprise Client/Server Company™*

# 12

# Objectives

- Explain how the network is used by SQL Server
- Use Sybase tools to determine and evaluate network traffic in and out of SQL Server
- Apply database application and design changes to improve network usage

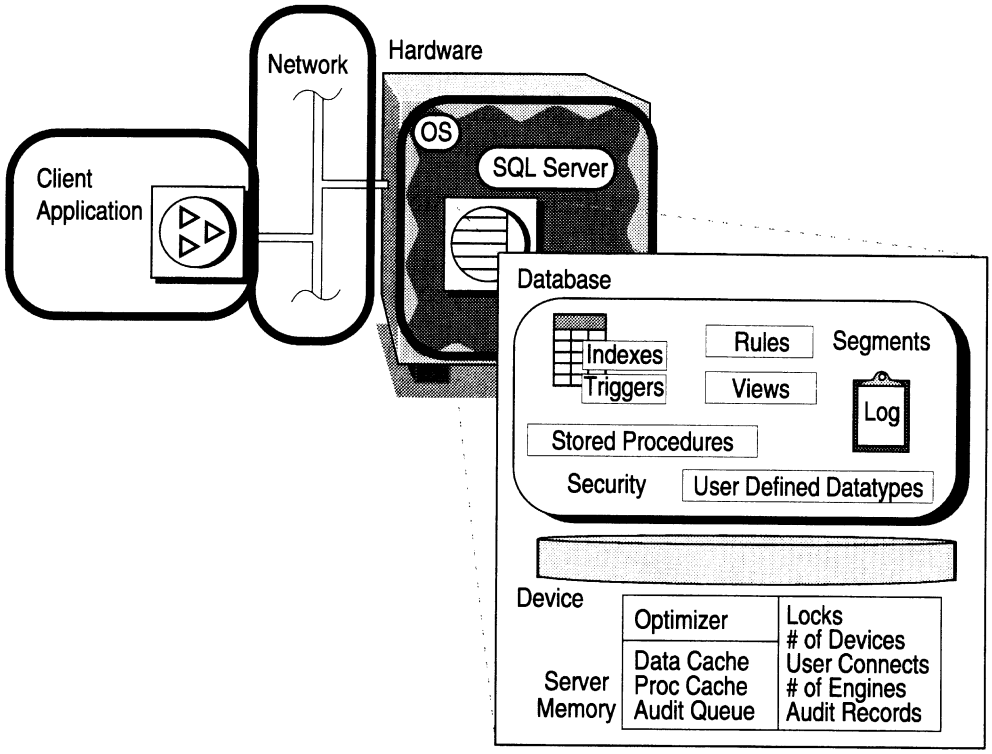
# Why Study the Network?

- Process response times vary significantly for no apparent reason
- Queries that return a large number of rows take longer than expected
- A particular client process seems to slow all other processes down

# Underlying Problems

- The physical limits of the network have been reached
- Processes are opening and closing connections too often increasing network load
- Processes are frequently submitting the same SQL transaction causing excessive and redundant network traffic
- Processes are not limiting the number of rows returned to clients putting heavy load on network
- Not enough network memory available to server
- Network packet sizes not big enough to handle type of processing needed by some clients

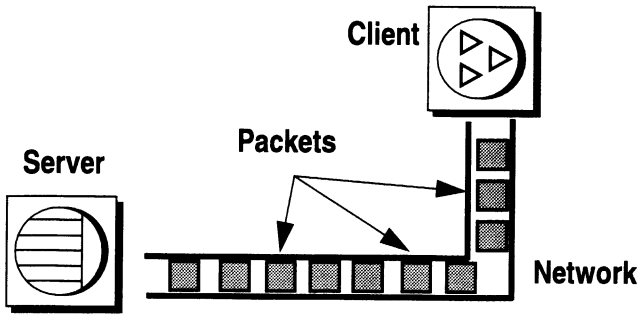
# System Model



# Networks & Performance Topics

- How SQL Server uses the network 
- SQL Server options
- Packet size and network performance
- Reducing network traffic
- Guidelines

## How SQL Server Uses the Network




- All Client/Server communications occur over a network
- All network communication is done via packets
- Packets consist of a header and routing information along with data, and can vary in size

### Background

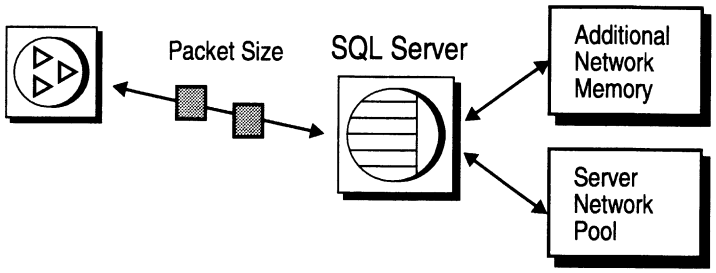
The SQL Server was one of the first database systems built around a network-based client/server architecture. Accordingly, clients initiate a **connection** to the server, which is used to send client requests and server responses. Applications can have as many connections open concurrently as is needed to perform the required task. The protocol used between the client and server is known as the **Tabular Data Stream** (TDS), which forms the basis of communication for all Sybase products.

# Networks & Performance Topics

- How SQL Server uses the network
- SQL Server options 
- Packet size and network performance
- Reducing network traffic
- Guidelines

# SQL Server Options

- Configurable options
  - default network packet size
  - maximum network packet size
- Related to network packet size
  - user connections
  - additional netmem




**default network packet size**

- The default packet size when a client connection is opened.
- Must be a multiple of 512.
- Default value is 512.

**maximum network packet size**

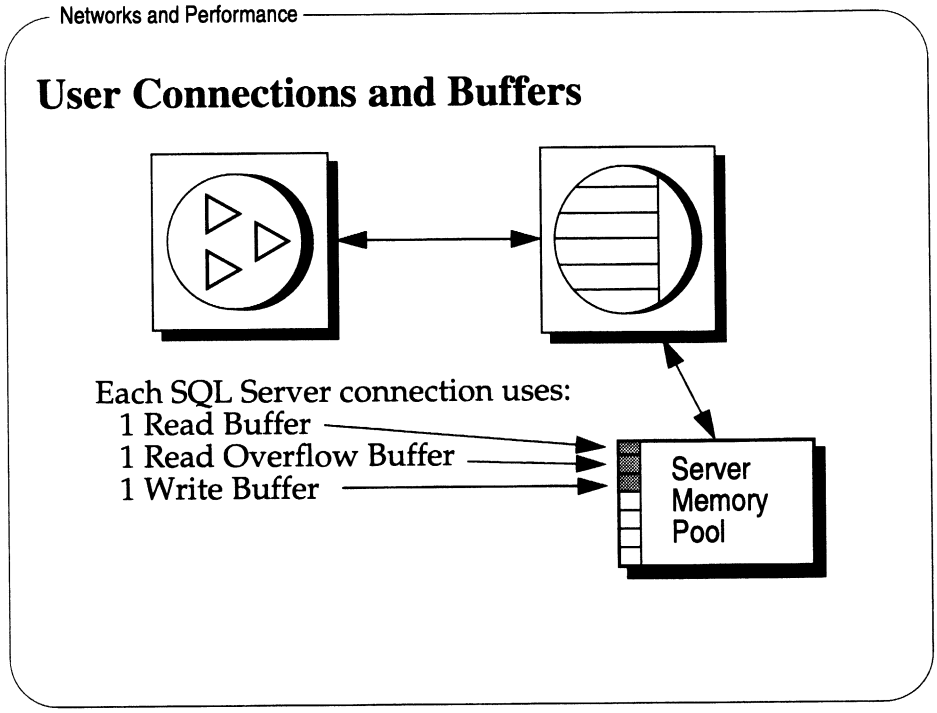
- The maximum packet size that can be used by any connection to the server.
- Packet size must be a multiple of 512 and must be greater than or equal to the default network packet size.
- The default value is 512, and the maximum value is 524,288.

**user connections**

- The maximum number of open connections to the server.
- This value is used to allocate a fixed number of network buffers from the memory allocated to the SQL Server using **sp_configure memory**.

**additional netmem**

- The amount of extra memory that should be allocated from the O/S to hold network packets that are larger than the default packet size.



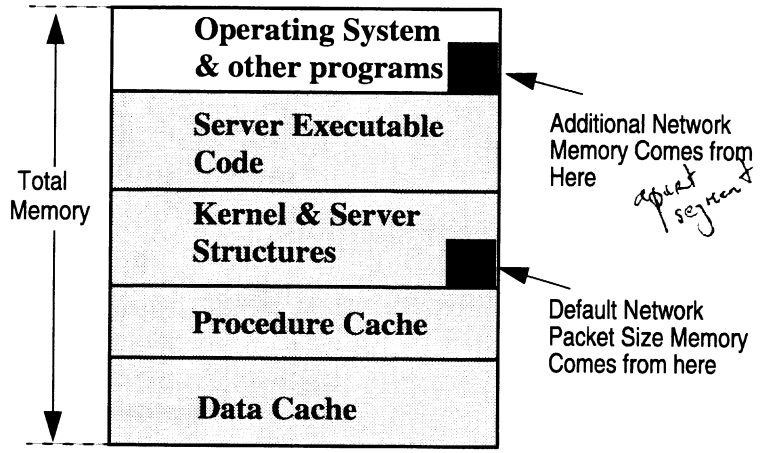

**Network Memory Allocation**

- The total amount of memory allocated for network packets is:
  user connections * 3 buffers * default network packet size
- For example, 50 users logging in with the default size will need:
  50 * 3 * 512 = 76,800 bytes

**User Packets**

- SQL Server guarantees that every user connection will be able to log in at the default network packet size.
- If you increase maximum network packet size but leave additional netmem at 0, client cannot use packet sizes that are larger than the default size.

## Where's My Memory Going?




**Network Memory**

- Connections that use bigger packet sizes (larger than the default size) get space from additional network memory.
- Connections may get a smaller packet because of space limitations, down to the default network packet size.
- Additional netmem size to request = (# connections using larger packets * large packet size * 3)

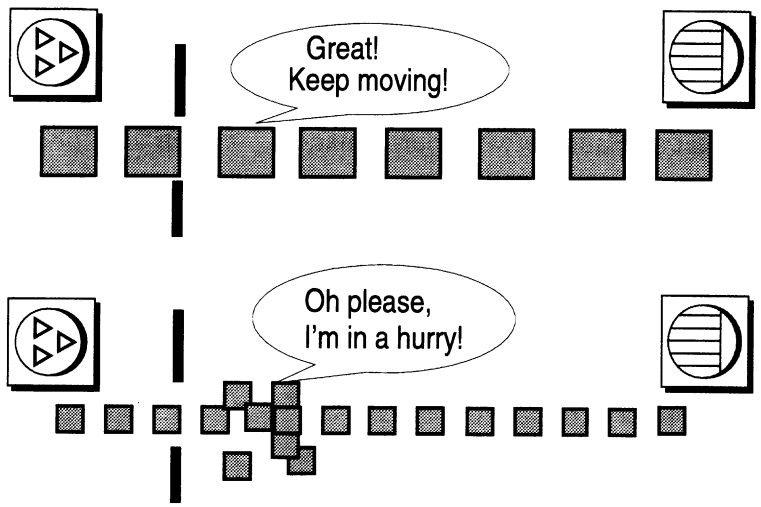
**Configuring for Additional Network Memory**

- Values should be multiples of 2048
- SQL Server will round to nearest multiple of 2048

# Networks & Performance Topics

- How SQL Server uses the network
- SQL Server options
- Packet size and network performance 
- Reducing network traffic
- Guidelines

## Smaller vs. Larger Packet Sizes



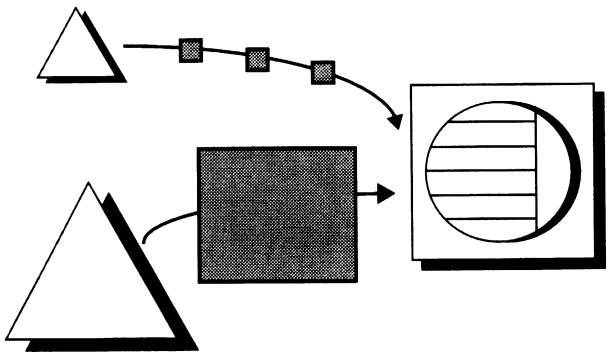

**Number and Size**

Generally, the number of packets being transferred is more important than the size of the packets. However, network performance also includes the time needed by the CPU and O/S to process a network packet, and this per-packet overhead affects performance as well. Larger packets reduce the overall overhead costs and achieve higher physical throughput, provided you have enough data that needs to be sent.

**Big Transfer Sources**

- bcp
- readtext and writetext commands
- large select statements

# Different Packet Sizes for Individual Clients

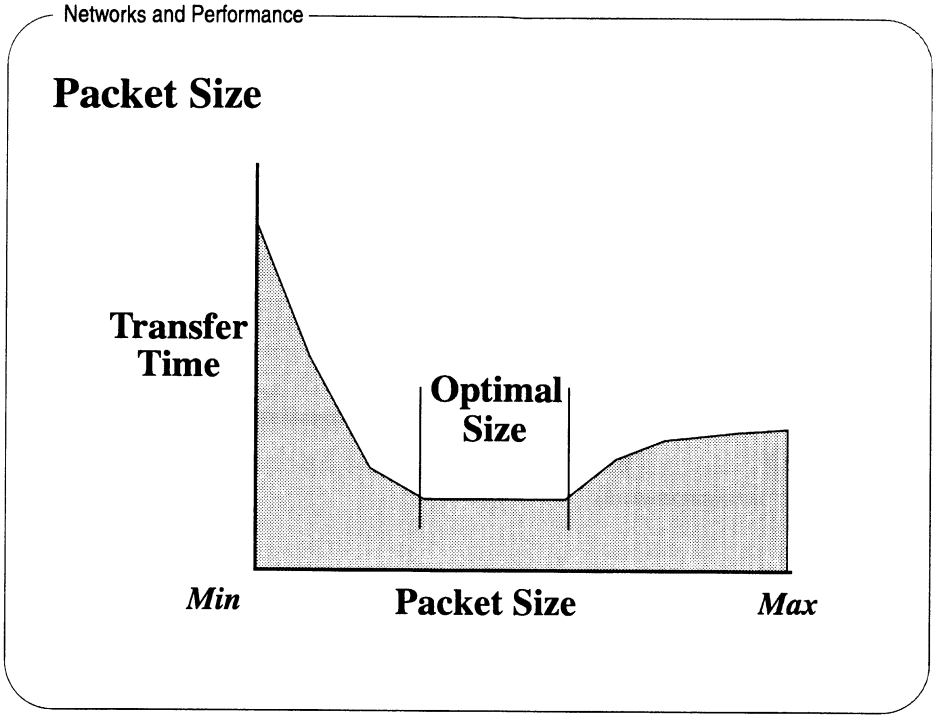


**Network Delays**

Large network packets used by one client can block or slow down other clients who are using smaller packets.

**Requesting Larger Packet Sizes**

- If clients occasionally require large amounts of data, use client request mechanisms for those special cases
  - isql -A *nnnn* (where *nnnn* is the desired packet size)
  - bcp -A *nnnn* (where *nnnn* is the desired packet size
  - dwb -A *nnnn* (where *nnnn* is the desired packet size
  - apt -A *nnnn* (where *nnnn* is the desired packet size
  - ct_con_props (connection, CS_SET, CS_PACKETSIZE, &packetsize, sizeof(packetsize), NULL)
  - DBSETL function (DB_lib)




**Packet Size**

There is always a point at which increasing the packet size will not improve performance. In fact, it may decrease performance because the packets are not always full. Although there are analytical methods for predicting that point, it is more common to vary the size experimentally and plot the results. If such experiments are conducted over a period of time and conditions, a packet size that works well for a lot of processes can be determined. However, since the packet size can be customized for every connection, specific experiments for specific processes are always beneficial.

**Monitor Packet Size vs. Performance**

Do not just set it and forget it. Monitor performance at varying packet sizes periodically to ensure it is set properly.

# Networks & Performance Topics

- How SQL Server uses the network
- SQL Server options
- Packet size and network performance
- Reducing network traffic 
- Guidelines

# Reducing Network Traffic Techniques

- Use row buffering to move large batches through the network
- Use server based T-SQL to reduce the amount of T-SQL sent across the network:
  - Stored Procedures
  - Views
  - Triggers
- Request larger packet sizes by client (we saw this already)

# Large Transfers

- Large transfers should be done during off-hour periods if possible
- If large transfers are common, consider acquiring network hardware that is suitable for such transfers:
  - token ring
  - fiber optic
  - separate network

**Token Ring**

- Token ring hardware responds better during heavy utilization periods than ethernet hardware.

**Fiber Optic**

- Fiber optic hardware provides very high bandwidth, but is usually too expensive to use throughout the entire network.

**Separate Network**

- A separate network can be used to handle network traffic between the highest volume workstations and the server.

# Evaluation Tools with SQL Server

- sp_monitor

```
sp_monitor
go
...
packets received packets sent packet err
---------------- ------------ ----------
10866(10580)    19991(19748)    0(0)
...
```

- @@pack_sent, @@pack_received, @@packet_errors:

```
SELECT Before = @@pack_sent
SELECT * FROM titles
go
SELECT After = @@pack_sent
go
```

**sp_monitor**

- Displays the packets sent and received since the server was booted
- In parentheses are the sub-totals for those counters since the last time sp_monitor was executed (by anyone).

**@@pack_sent, @@pack_received, @@packet_errors**

- Global variables that contain the total count since the server was rebooted.

# Operating System Network Evaluation Tools

- Unix
  - netstat -i

```
netstat -i

Name Mtu Net/Dest Address Ipkts Ierrs Opkts
le0  1500 indigo  ib      842892  32  51046
```

- OpenVMS
  - Monitor

**netstat -i**

This is a Unix utility that shows the total number of packets processed by the O/S for one or more interfaces. When used in a shell script, this utility provides a convenient mechanism for determining the overall network impact of one or more processes. This utility can also be used to measure the "background" network traffic while a database test is in progress. Such data can be used to determine if the database test measurements are free from "abnormal" network interference.

**Monitor**

This is an OpenVMS facility.

# Impact of Other Server Activities

- Login protocol
- Two-Phase Commit Protocol
- Replication Processing
- Backup Processing

**Many Users, Many Needs**

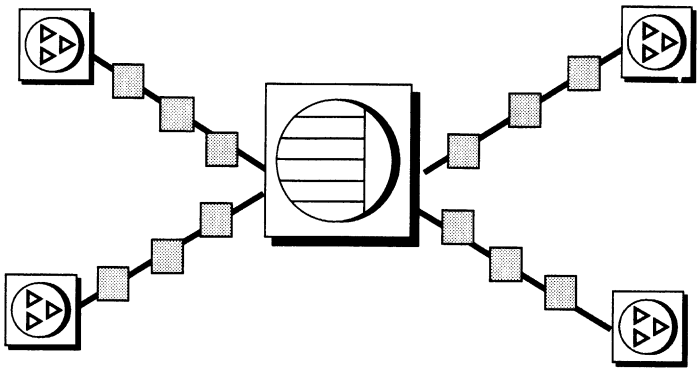
All of the activities listed above involve network communication, especially Replication Processing and the Two Phase Commit Protocol. Systems that make extensive use of these activities may see a lot of network-related problems. Accordingly, these activities should be done only as necessary. For example, a connection that is open can be shared among various modules within an application, rather than being repeatedly opened and closed. Similarly, backup and replication processing can be restricted to periods of low network activity.

# Networks & Performance Topics

- How SQL Server uses the network
- SQL Server options
- Packet size and network performance
- Reducing network traffic
- Guidelines 

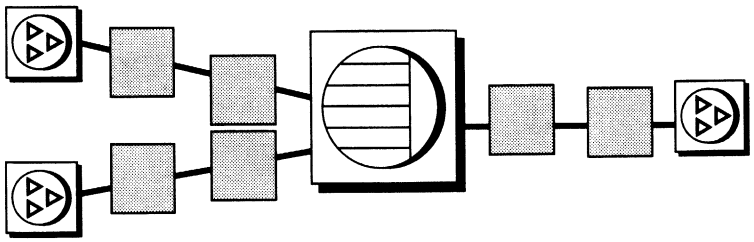
# Guideline 1

- Do you generally send and receive small amounts of data?
  - Keep *default network packet size* small (default is 512 bytes)



# Guideline 2

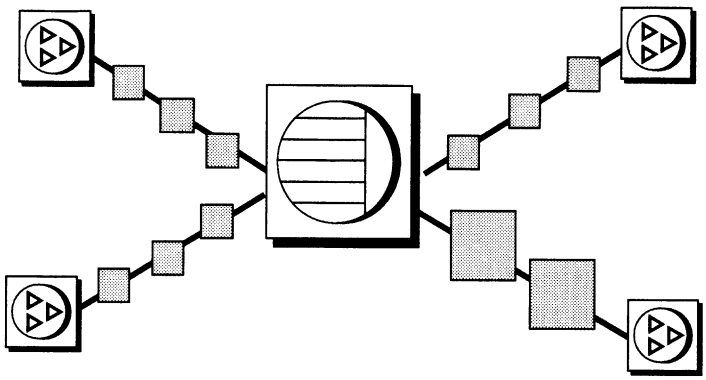
- Do most of your applications send and receive large amounts of data?
  - ~~Increase *default network packet size*~~
  - This will result in fewer transfers, with less overhead



- Increase *maximum network packet size* and additional netmem to at least the same size as the default so all processes can use larger packet sizes

# Guideline 3

- Do you have mixed needs?
  - Keep *default network packet size* small (default is 512 bytes)
  - Increase *maximum network packet size* and *additional netmem* so certain processes can use larger packet sizes

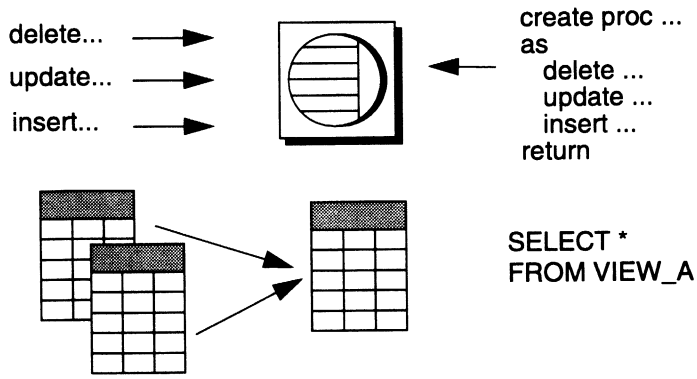


**Method**

If *maximum network packet size* is set high, and *additional netmem* is sufficient, then individual clients can request a higher packet size than the default.

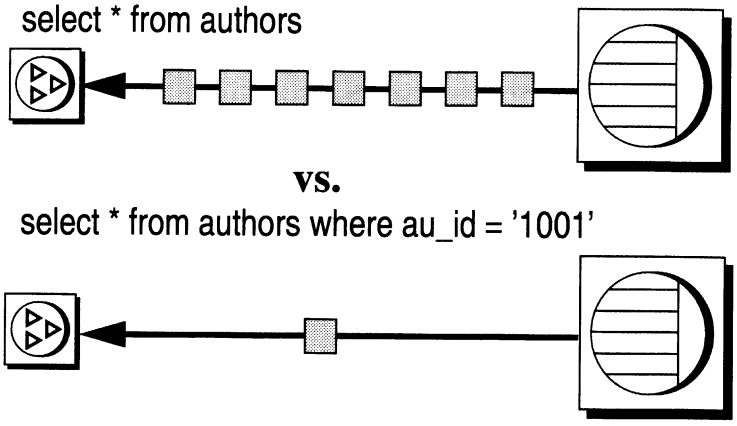
# Guideline 4

- Do you send large batches of T-SQL?
  - Convert to stored procedures or use a view



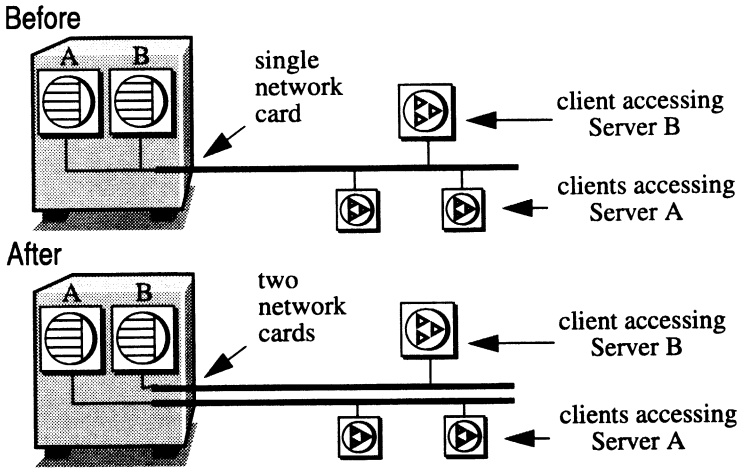
# Guideline 5

- Filter as much data as possible at the server



# Guideline 6

- Isolate heavy network users from ordinary network users




**Isolating Heavy Network Users**

Before:

- Clients accessing two different SQL Servers use one network card.
- That meant that clients accessing Servers A and B have to compete over the network and past the network card.

After:

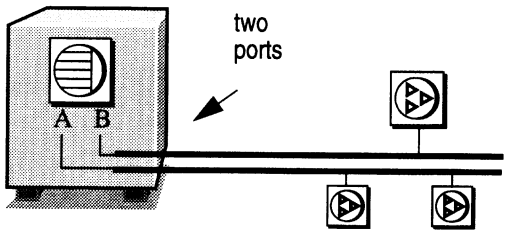
- Clients accessing Server A use one network card and clients accessing Server B use another.

Even Better:

- Put your SQL Servers on different machines!

# Guideline 7

- Two (or more) ports listening for a single SQL Server




- Front-end software may be directed to any of the configured network ports (via DSQUERY/DSLISTEN)
- Can eliminate or reduce network bottle necks (spreads out network load) thus increasing SQL Server throughput

# Guideline 8

- If clients occasionally require large amounts of data, use client request mechanisms for those special cases
  - isql -A *nnnn* (where *nnnn* is the desired packet size)
  - bcp -A *nnnn* (where *nnnn* is the desired packet size
  - dwb -A *nnnn* (where *nnnn* is the desired packet size
  - apt -A *nnnn* (where *nnnn* is the desired packet size
  - ct_con_props (connection, CS_SET, CS_PACKETSIZE, &packetsize, sizeof(packetsize), NULL)
  - DBSETL function (DB_lib)

# Typical Problem

One client process seems to be slowing down all other processes

Collect Data

- Use sp_who and sp_lock to see what processes are running and what resources they are holding
- Check activity in and out of the server
  - Run sp_monitor look at packets received and packets sent
  - Execute the problem process
  - Run sp_monitor again and compare
- Ask the user what the problem process is doing
  - huge selects? text manipulation? image processing? bcp?

**sp_monitor**

sp_monitor reports network activity for all users since the last time sp_monitor was executed by anyone. To test packets sent and received by a specific process using the method described on the foil, you have to be the only one running on the server. Otherwise, other network activity will be included in the sp_monitor report.

# Collect Data: Results

- The process is doing large amounts of text processing using default network packet size (large number of packets)
- Formulate hypothesis
  - The process should be using larger packet sizes
  - Need to:
    Configure *additional netmem* to accommodate larger packet size requests
  - Request larger packet size for that client process only

# Test Hypothesis

- Change client application to request larger packet sizes, such as 8192
- Configure *additional netmem* to new value, such as 3 * 8192
- Re-execute process
- Determine if the number of network packets has been reduced
- Results:
  - packets sent and received dropped significantly, and other processes were no longer being blocked

**new netmem value** netmem = 8192 * 3 buffers

# Networks & Performance: Questions

- Which processes usually retrieve a large amount of data?
- Are a large number of network errors occurring?
- What is the overall performance of the network?
- What is the mix of transactions being performed using SQL and stored procedures?
- Are a large number of processes using the two-phase commit protocol?
- Are replication services being performed on the network?
- How many network cards are on the machine?

# Lab 12 – Networks and Performance

- Compare packets sent and response time for a SQL script vs. a stored procedure
- Request different network packet sizes and note performance impact

Optional:

- Use SQL Monitor to observe network traffic

# Summary

- How SQL Server uses the network
- SQL Server network-related options
  - default network packet size
  - maximum network packet size
  - user connections
  - additional netmem
- Different packet sizes for individual clients
  - isql - Annnn
  - bcp - Annnn
  - others
- Network guidelines
  - small vs. large amounts of data sent/received
  - small vs. large batches of T-SQL
  - filtering data at the server
  - isolating heavy network users
  - two (or more) ports listening for a single SQL Server
  - "-A" param for client connections

# 13

# tempdb

## System 10 Performance & Tuning

Version 2.2




*The Enterprise Client/Server Company™*

# 13

# Objectives

- Size *tempdb* correctly for all server activity
- Place *tempdb* optimally to minimize contention
- Minimize locking of resources (system-wide and within *tempdb*)

# Why Worry About tempdb?

- It fills up frequently
- Sorting is slow
- Users held up (locked out) from creating temporary tables

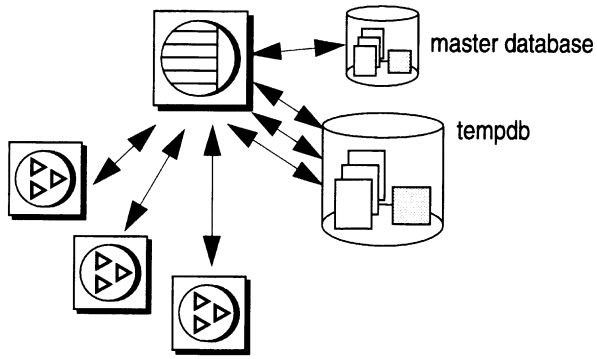
tempdb

# Topics

- Uses of *tempdb* 
- Sizing *tempdb*
- Placing *tempdb*
- Locking in *tempdb*

tempdb

## What is *tempdb*?




- Server-wide database used primarily for
  - Internal processing (sorts, working tables, reformatting, etc.)
  - Storing temporary tables/indexes created by users

*tempdb*

As we see, tempdb can be a heavily-used database, potentially used by many processes on SQL Server.

# Used for Internal Processing

- Internal working tables are created in *tempdb* for:
  - DISTINCT
  - NOT EXISTS
  - GROUP BY
  - ORDER BY
  - reformatting
  - OR strategy

# Storing Temporary Tables

- "Truly" temporary
  - Exist only for the duration of the user session or stored procedure execution
  - Created using
    ```
    create table #temptable
    ```
  - Cannot be shared
- "Permanent" temporary tables
  - Can span sessions
  - Can be shared via grants
  - Created using
    ```
    create table tempdb..temptable
    ```
  - Must be explicitly dropped by owner
  - Dropped on reboot of SQL Server

**Note**

Working tables created by SQL Server for sorts, etc. are never shared.

To make "permanent" temp tables available every time SQL Server starts up, create them in the model database.

tempdb

# Using Temporary Tables to Split Up Multi-table Joins

- Useful in the following circumstances
  - If query is joining more than four tables and doesn't choose best plan
  - Queries exceed the sixteen-table join
  - Queries are very complex
  - You would like to filter data as an intermediate step
- Using temporary tables in these instances is useful and tends to reduce I/O (fewer passes through base tables)
- Disadvantage: increased burden and space requirement on *tempdb*

**Other contexts in which temporary tables can be useful**

- Denormalize several tables into a few temporary tables
- Normalize a denormalized (i.e., repeating column) table in order to do aggregate processing

# Indexes on Temporary Tables

- You can define indexes on temporary tables!
- Restrictions
  - Optimizer will use index only if statistics have been updated
  - In stored procedure, you cannot create a temporary table, then create index, then expect the optimizer to use that index
- Guidelines
  - Update statistics in separate transaction or create the index after the table is loaded
  - Create a permanent table in *tempdb* rather than a temporary one
  - Create needed indexes on the permanent table in *tempdb*
  - For indexes on temporary tables, be sure the create cost does not exceed the benefit

**Creating an index in a stored procedure**

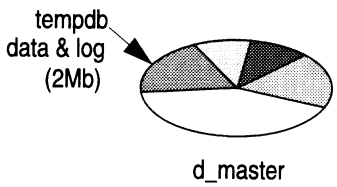
In the scenario described above, if you create a temporary table and an index on that table in a stored procedure, the optimizer cannot use that index to search the table. It *can* use the index to enforce uniqueness, however.

## Topics

- Uses of *tempdb*
- Sizing *tempdb*
- Placing *tempdb*
- Locking in *tempdb*

Locking in *tempdb*

# Initial Allocation of *tempdb*

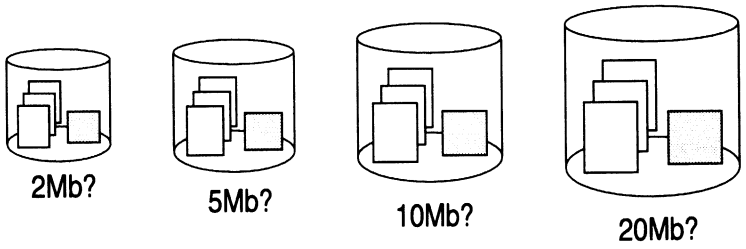



```
1> sp_helpdb tempdb
2> go
name  db_size  owner  dbid  created  status
-----------------------------------------tem
pdb 2.0 MB  sa     2     ...    select
                                into/bulkcopy
device_frag  size  usage        free kbytes
-----------------------------------------
master       2.0 MB data and log  ...
```

**Initial allocation**

When you install SQL Server, 2 Mb is allocated on the master device for *tempdb*. You may want to increase this allocation, or move it to a separate device.

# Sizing *tempdb*




- Needs to be big enough to handle (for ALL concurrent users):
  - Internal sorts
  - Other internal working tables
  - Temporary tables
  - Indexes on temporary tables
  - "Permanent" tables in *tempdb*

# Input to Sizing *tempdb*

| | |
|---|---|
| • Maximum number of concurrent users | U |
| • Size of sorts | S |
| • Working table size | W |
| • Number of steps in the query plans | Q |
| • Number of stored procedures and/or user sessions that create temporary tables and indexes | P |
| • Size of temporary tables and indexes | T, I |
| • Number of temporary tables and indexes | C, D |

**Users** — The maximum number of concurrent users, that is, proc ids.

**Sort Size** — Size requirements, in pages, of internal sorts (typical query with an ORDER BY that is not supported by an index). Stats I/O inserts will be indirect way of measuring table size.

**Working Table Size** — Size of working tables needed for reformatting, GROUP BY, etc., but not for sorting. (You can see this by looking at the number of pages listed in statistics io output.)

**Steps in Query Plans** — Typical number of steps in query plans for reformatting, GROUP BY, etc. Indicates number of work tables created.

**Stored Procedures** — Number of stored procedures (local and remote) and/or user sessions that create temporary tables and temporary indexes.

**Temporary Table and Index Size** — Size requirements, in pages, of these temporary tables (T) and indexes (I). Can display using statistics io.

**Temporary Table and Index (Number)** — Number of temporary tables (C) and indexes (D) created per stored procedure/session.

tempdb

## Sizing Algorithm

- Compute size required for usual processing
  - Sorts: U * S
  - Other: U * W
  - Subtotal * Q
- Compute size required for temporary tables and indexes
  - Temporary tables: P * T * C
  - Indexes: P * I * D
- Add these, then add 25% for padding/error/other usage

**Sizing**

- U = Maximum number of concurrent users
- S = Size of sorts
- W = Working table size
- Q = Number of steps in the query plans
- P = Number of stored procedures and/or user sessions that create temporary tables and indexes
- T, I = Size of temporary tables and indexes
- C, D = Number of temporary tables and indexes

# Sizing Algorithm: Example

- Processing requirements
  - 55 users * 15 pages = 825 pages
  - 55 users * 9 pages = 495 pages
  - Subtotal = 1320 pages
  - 1320 * 3 steps = 3960 pages, or 8.2 Mb
- Temporary table/index requirements
  - 190 procs * 10 pages * 4 tbls =7600 pages
  - 190 procs * 2 pages * .5 idxs = 190 pages
  - Total = 7790 pages, or 16 Mb
- Add together, then add 25% for padding/error
  - (8.2 MB + 16.0 Mb) * 1.25 = 30 Mb

tempdb

# Topics

- Uses of *tempdb*
- Sizing *tempdb*
- Placing *tempdb* 
- Locking in *tempdb*

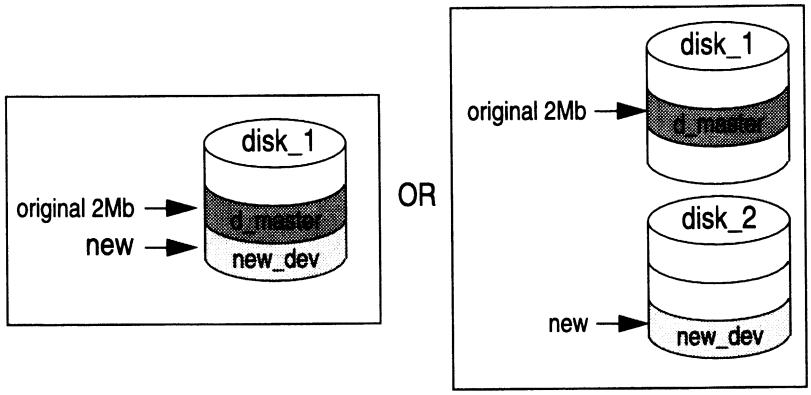
# Placing *tempdb*

- Two principles
  - Isolate *tempdb* from application databases
  - Make pages in *tempdb* as contiguous as possible

**Two Principles**

These are the principles to apply when deciding where to place *tempdb*. Note that the pages in *tempdb* should be as contiguous as possible because of its dynamic nature.

# Placing *tempdb*: Desirable



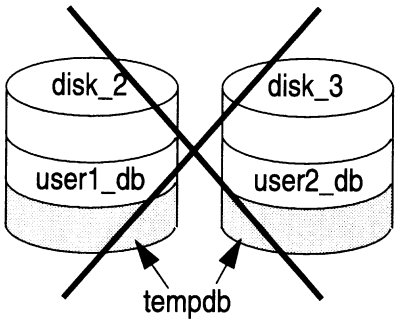

- Create new device on same disk as master or on another relatively inactive drive and limit activity on master device

**Placing tempdb**

It is desirable to isolate *tempdb* onto a separate device if it is used heavily.

Segment verwijderen van master: zie troubleshooting-guide

# Placing *tempdb*: Undesirable




- Do not expand *tempdb* across multiple disks (if possible)
- Do not put *tempdb* on disks that store your application databases

**Multiple Disks**

To minimize the possibility of a working table or temporary table having to span multiple disks, try not to expand *tempdb* across multiple disks (that does not isolate activity to one device).

**Sharing Disks with Application Databases**

To minimize disk contention, do not put *tempdb* on disks that store application data.

# Preventing Temporary Tables from Spanning Multiple Devices

- If expanding *tempdb* to a separate physical device from the master device
  - Want to avoid having worktables and temp tables span multiple devices
- Remove default and system segments for *tempdb* from master device
  - Use sp_dropsegment
  - All temporary and work tables will then be created on new device

**Moving Segments**

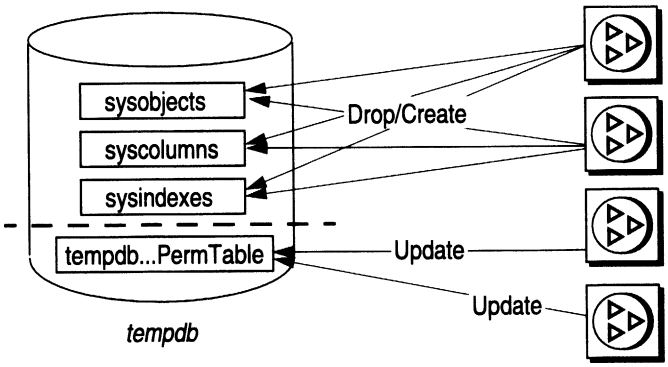
To move default and system segments off of master device for *tempdb*:

- Alter tempdb onto new device
- sp_dropsegment "default", *tempdb*, master
- sp_dropsegment "system", *tempdb*, master
- Backup master database

# Topics

- Uses of *tempdb*
- Sizing *tempdb*
- Placing *tempdb*
- Locking in *tempdb* 

# Locking in *tempdb*




**Locking**

- Locking can occur if a large number of concurrent users are:
  - Creating and dropping temporary tables and their indexes
  - Updating permanent tables in *tempdb*

**Reasons**

- Temporary table information may fall on the same pages within *tempdb* as *tempdb*'s system tables (sysobjects, syscolumns, sysindexes)
- Permanent tables in *tempdb* data pages are being held by a process for the duration of the transaction

# Minimizing Locking in *tempdb*

- Minimized by:
  - Increase I/O speed of *tempdb*
  - Minimize the system table's contention
  - Minimize update transactions of permanent tables in *tempdb*
- Techniques
  - Use RAM drives and solid state devices to hold *tempdb*
  - Remove system segment from master device and/or add dummy tables to *tempdb* that fill out the system table pages
  - Shorten transactions that update permanent tables in *tempdb* (or eliminate them)

tempdb

## Other *tempdb* Performance Tips

- Only include required columns in temp tables
  - Reduces row and table size
  - More rows per page = less I/O
- Alter *tempdb* onto an Operating System supported ram disk
- Alter *tempdb* onto a solid state storage device

Do not use select * into temp table if you only need a couple of columns.

Solid state storage devices are essentially external ram drives with battery backup. They appear to the operating system and SQL Server as a standard hard drive, but access time is much faster.

# Lab 13 – tempdb

- Display current location of and allocation for *tempdb*
- Given space requirements, describe how you would alter *tempdb* database onto an appropriate device

# Summary

- Uses of tempdb
  - Internal processing (sorts, working tables, etc.)
  - Temporary tables/indexes
- Sizing of tempdb
  - Big enough to handle all concurrent users' internal processing needs
  - (Processing Requirements + Temptable Requirements + 25%)
- Placing of tempdb
  - Isolate from application databases
  - Make pages as contiguous as possible
  - RAM resident / solid state devices
- Locking in tempdb
  - More prone as more users on server
  - Increase I/O speed
  - Minimize system table contention
  - Minimize update transaction for temp tables

14

# Cursors and Performance

System 10 Performance & Tuning

Version 2.2




*The Enterprise Client/Server Company™*

# 14

# Objectives

- Describe performance implications of cursors
- Use cursors and examine locks that occur

Refer to the *Transact-SQL User's Guide*, Chapter 15, for further information.

# What Is A Cursor?

- A cursor is a symbolic name that is associated with a select statement
  - It enables you to access the results of a select statement one row at a time

Assume: cursor with select * from authors where state = 'KY'

Programming can
- examine a row
- take an action based on row values

| | | | |
|---|---|---|---|
| A1301065096 | Jines | ... | KY |
| A1095065156 | Matheys | ... | KY |
| A1786065249 | Ensley | ... | KY |
| A978606525 | Marcello | ... | KY |
| ... | ... | ... | ... |

result set

**Cursors**

You can think of a cursor as a "handle" on the result set of a select statement. It enables you to examine and possibly manipulate one row at a time.

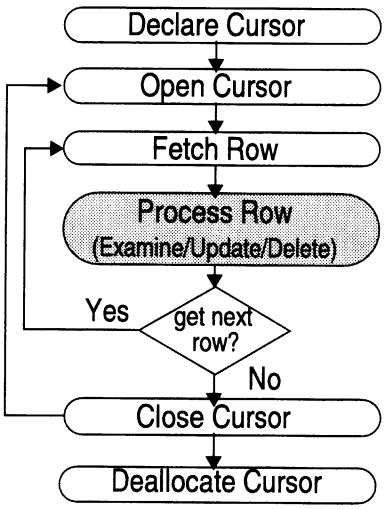
# Set-oriented vs. Row-oriented Programming

- SQL was conceived as a set-oriented language

```
update titles
   set contract = 1 where type = 'business'
```

  - Performs identical action to all rows fulfilling conditions
  - SQL Server finds most efficient way to do this
- ANSI-89 SQL requires a cursor to manipulate rows of a result set
  - Declare cursor for select statement, open cursor and fetch a row (or set of rows), process it, go on to the next row, etc.
  - May perform quite different operations based on values in the current row
  - May lose efficiency

## Steps in Using a Cursor




**Steps**

Cursors are presented in the Fast Track course. We review steps and syntax in this module.

# Cursors: A Simple Example

```
declare biz_book cursor
   for select * from titles
   where type = 'business'
go
open biz_book
go
fetch biz_book
go
/* Look at each row in turn and perform
** various tasks based on values, till
** there are no more rows
*/
close biz_book
go
deallocate cursor biz_book
go
```

**declare**

```
declare cursor_name cursor
   for select_statement
   [for {read only | update
      [of column_name_list]}]
```

**open**

```
open cursor_name
```

**fetch**

```
fetch cursor_name
   [ into fetch_target_list ]
```

**close**

```
close cursor_name
```

**deallocate**

```
deallocate cursor cursor_name
```

# Cursors: More Commands

- Fetch into variables

```
declare @book_title char(80),
        @book_id char(6)
open biz_book
fetch biz_book into @book_title, @book_id
go
```

- Deleting current row

```
delete titles where current of biz_book
```

- Update current row

```
update titles set title='The Rich
   Executive's Database Guide'
where current of biz_book
```

**fetch**

```
fetch cursor_name
   [ into fetch_target_list ]
```

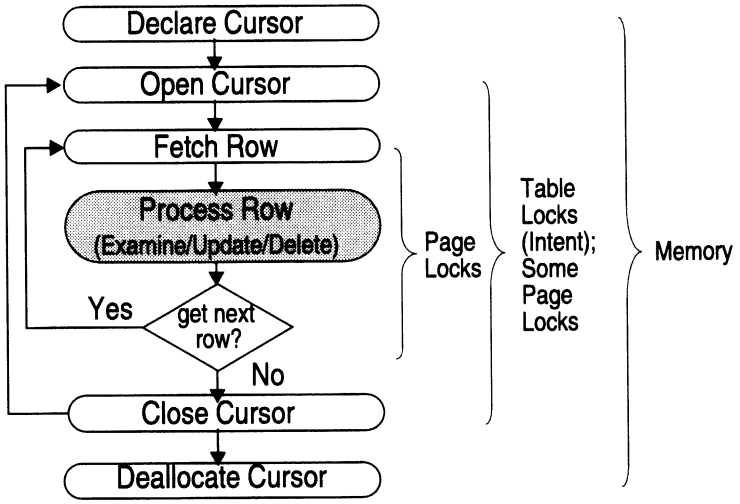
**delete**

```
delete
[from] [[database.]owner.]{table_name |
   view_name}
where current of cursor_name
```

**update**

```
update [[database.]owner.]{table_name |
   view_name}
set
[[[database.]owner.]{table_name.|view_name
.}]column_name1 =
{expression1|NULL|(select_statement)}
   [, column_name2 =
{expression2|NULL|(select_statement)}]...
where current of cursor_name
```

## Resources Required At Each Stage




**declare** When you declare a cursor, memory is allocated to the cursor and to store the query plan which is generated. (The size of the query plan depends on the select statement, but it generally ranges from 1-2 pages.)

**open** When you open a cursor, the Server starts processing the select statement. The Server optimizes the processing at this time, traverses indexes, and sets up memory variables. The Server does not access rows yet. However, it does set up the required table-level locks (intent locks) and (if there are subqueries or joins) page locks on outer table(s).

**fetch** When you execute a fetch, SQL Server sets up the required page lock, gets the row or rows required and sends it to the client. The page lock is held until the next fetch, or until the cursor is closed. This page lock is either a shared page lock or an update page lock depending on how the cursor is written.

**close** When you close a cursor, SQL Server releases all the locks but is still holding onto the memory allocation.

**deallocate** When you deallocate a cursor, SQL Server releases the memory resources used by the cursor. To reuse the cursor, redeclare it.

# Two Cursor Modes: Read-only and Update

- Read-only mode uses shared page locks
  - In effect if you specify `for read only`
  - In effect if select has `distinct`, `group by`, `union`, or aggregate functions, and in some cases `order by`
- Update mode (the default) uses update page locks
  - In effect if you specify `for update` (unless select has `distinct`, `group by`, etc.or you specify `shared`)
  - If *`column_name_list`* is specified, only those columns are updatable
- No insert mode
  - You cannot insert new rows with cursors

**Note re: declare and deallocate (previous page)**

The descriptions regarding declare and deallocate on the previous page refer to ad-hoc cursors sent using isql or CT-Lib. For cursors declared *on* stored procedures (sent using CT-Lib or the pre-compiler and known as "execute cursors"), only a small amount of memory is allocated at declare time. For cursors declared within a stored procedure, memory is already available for the stored procedure, and the declare statement does not add to this.

**Read-only vs. Update**

Specify cursor mode when you declare the cursor. Note that if the select statement includes certain options, the cursor is *not* updatable even if you declare it for update.

**Lock Compatibility**

| Lock Type | Shared | Update | Exclusive |
|---|---|---|---|
| **Shared** | Yes | Yes | No |
| **Update** | Yes | No | No |
| **Exclusive** | No | No | No |

# Index Use and Requirements

- Read-only cursors
  - Any index can be used
- Update cursors
  - If not *declared* for update, unique index preferred over table scan or nonunique index; OK if no unique index exists
  - If declared for update *without* "for update of" list, unique index required; error raised if none exists
  - If declared for update with "for update of" list, then only unique index *without* any columns from the list can be chosen
- When cursors are involved, an index which contains an identity column is considered unique even if the index is not declared that way

# Two Global Variables

- @@sqlstatus holds status information regarding a fetch:

| | |
|---|---|
| 0 | indicates successful completion of the fetch |
| 1 | indicates that fetch resulted in error |
| 2 | indicates no more data |

- Sample use:

```
fetch biz_books...
while @@sqlstatus !=2
   if @@sqlstatus = 1 raiserror...
   else ... /* Process row */
```

- @@rowcount is the number of rows fetched to the client thus far

# Sample Stored Procedure: No Cursors

```
/* Increase the prices of books in the
** titles_idpr table as follows:
**
** If current price is <= $30, increase it by 20%
** If current price is > $30 and <= $60, increase
** it by 10%
** If current price is > $60, increase it by 5%
**
** All price changes must take effect, so this is
** done in a single transaction.
*/

create procedure increase_price
as

   /* start the transaction */
   begin transaction
   /* first update prices > $60 */
   update titles_idpr
   set price = price * $1.05
   where price > $60

   /* next, prices between $30 and $60 */
   update titles_idpr
   set price = price * $1.10
   where price > $30 and price <= $60

   /* and finally prices <= $30 */
   update titles_idpr
   set price = price * $1.20
   where price <= $30

   /* commit the transaction */
   commit transaction

return
```

# Sample Stored Procedure With Cursor

```
/* Same as previous example, this time using a
** cursor.  Each update commits as it is made.
*/

create procedure increase_price_cursor
as
declare @price money

/* declare a cursor for the select from titles */
declare curs cursor
for
select price
from titles_idpr
for update of price

/* open the cursor */
open curs

/* fetch the first row */
fetch curs into @price

/* now loop, processing all the rows
** @@sqlstatus = 1 means error on previous fetch
** @@sqlstatus = 2 means end of result set reached
*/
while (@@sqlstatus != 2)
begin

   /* check for errors */
   if (@@sqlstatus = 1)
   begin
      print "Error in increase_price"
      return
   end

   /* continued on next page */
```

# Sample Stored Procedure With Cursor (continued)

```
    /* next adjust the price according to the
    ** criteria
    */
    if @price > $60
    select @price = @price * $1.05
    else
    if @price > $30 and @price <= $60
    select @price = @price * $1.10
    else
    if @price <= $30
    select @price = @price * $1.20

    /* now, update the row */
    update titles_idpr
    set price = @price
    where current of curs

    /* fetch the next row */
    fetch curs into @price
end

/* close the cursor and return */
close curs
return
```

# Performance: A Comparison

- Sample execution times against a 5000-row table:

| | | |
|---|---|---|
| increase_price | uses 3 table scans | 2 minutes |
| increase_price_cursor | uses cursor, single table scan | 5 minutes |

!!

!!

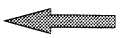
Conclusie : CURSOR geeft geweldige overhead !

# Performance Implications

- Cursors performance issues:
  - Locking (page and table-level)
  - Network resources
  - Overhead of processing instructions
- Use cursors only if strictly necessary
  - If there is a SQL programming equivalent, this is usually preferable even if it involves multiple table scans

# Class Demo #1: Locking With Read-only Cursors

- Using sp_lock, examine what locks are in place at each point:

```
declare curs1 cursor for
select au_id, au_lname, au_fname
from authors
where au_id like 'A1%'
for read only
go                          
open curs1
go                          
fetch curs1
go                          
fetch curs1
go 100                      
close curs1
go                          
```

**After declaration** — No cursor-related locks.

**After open** — Shared intent lock on authors.

**After first fetch** — Shared intent lock on authors, and shared page lock on a certain page in authors.

**After 100 fetches** — Shared intent lock on authors, and shared page lock on a certain (different) page in authors.

**After close** — No cursor-related locks.

# Class Demo #2: Locking With Update Cursors

- Open two connections to SQL Server

```
declare curs2 cursor for
select au_id, au_lname
from authors_id
where au_id like 'A1%'
for update
go
open curs2
go

fetch curs2
go
```

```
begin tran
go
select *
from authors_id
holdlock
where au_id = au_id
   fetched at left
go

sp_lock
go
```

```
delete from authors_id
where current of curs2
go

/* what happens? */

close curs2
go
```

```
delete
from authors_id
where au_id =
   same au_id

/* what happens? */
```

# Lab 14 – Cursors

- Examine locks that are set up when cursors are used

# Summary

- Cursors can downgrade performance
  - Increase chance of locking (page and table-level)
  - Increase network traffic
  - Involve considerable overhead of processing instructions
- Use only when strictly necessary!
  - Leave open as briefly as possible

15

# CPU Utilization Issues

## System 10 Performance & Tuning

Version 2.2




*The Enterprise Client/Server Company™*

# 15

# Objectives

- Explain how CPUs are utilized by SQL Server
- Use Sybase tools to determine and evaluate CPU utilization
- Describe techniques to apply CPU resources effectively

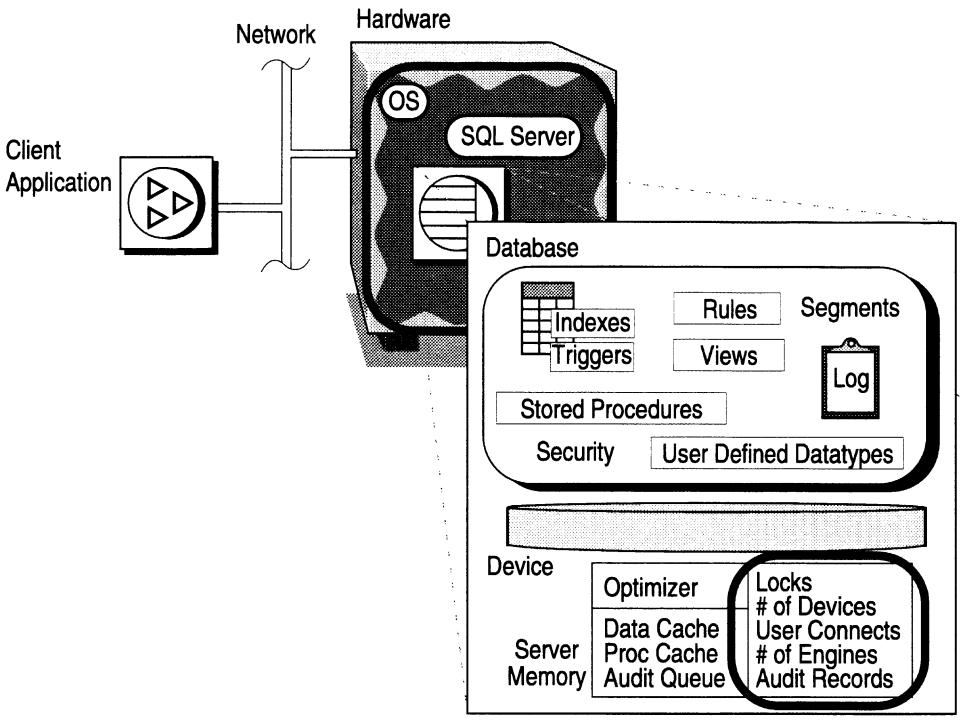
# Why Study CPU Utilization?

- Response for all O/S processes are slow
- All SQL Server processing response time increases significantly when a specific process is executed

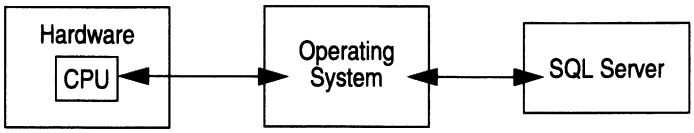
# Underlying Problems

- The CPU(s) supporting the SQL Server are overburdened
- The SQL Server and O/S compete for CPU resources
- SQL Server time slice is too small to support processes that perform complex calculations

# System Model



# Single CPU Machines




- No correspondence between operating system CPU utilization and SQL Server CPU busy information
- Use operating system utilities to see CPU utilization of SQL Server
  - ps command (UNIX)
  - vmstat command (UNIX)
  - showserver (UNIX)
- Use sp_monitor or SQL Monitor to see the percentage of time the SQL Server was using the CPU during an elapsed time interval

# CPU Utilization of SQL Server

1. Identify the process id of the data server(s) (like "SERVER1001A"

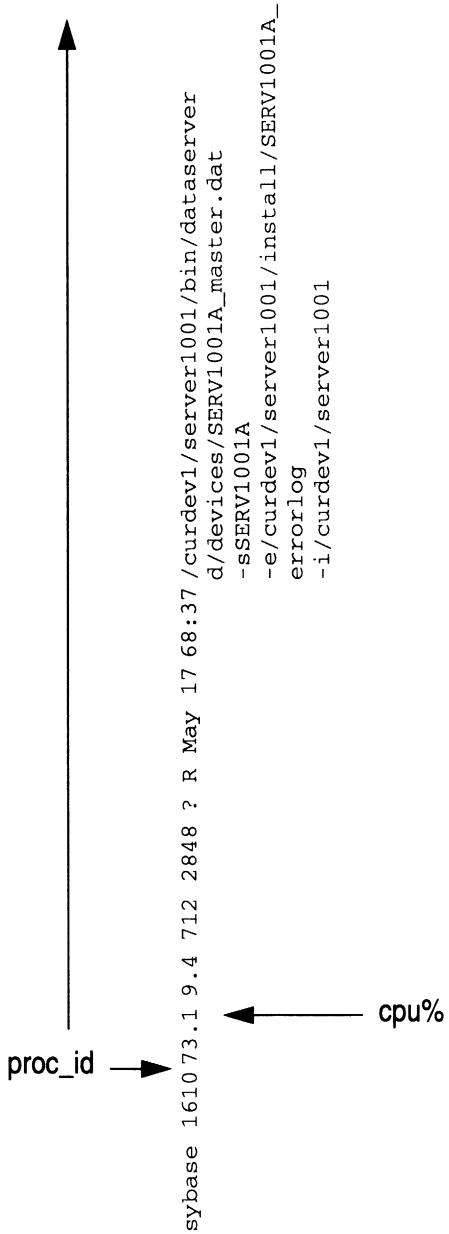
   `ps -auxww|grep dataserver` or `showserver`

```
USER       PID %CPU %MEM    SZ   RSS TT  STAT STARTED      TIME COMMAND
sybase   17425  1.6  3.7   700  1112 ?   S    Apr 27 533:07 /curdev1/server10ga/bin/dataserver -d/dev/rsd0e
                                                              -sD10GA
                                                              -e/curdev1/server10ga/install/D10GA_errorlog
                                                              -i/curdev1/server10ga
sybase   16350  1.6  3.3   700   996 ?   S    Apr 21 599:53 -M/curdev1/server10ga/install
                                                              /curdev1/server10ga/bin/dataserver
                                                              -d/devices/PAUL10_master.dat -sPAUL10
                                                              -e/curdev1/server10ga/install/PAUL10_errorlog
                                                              -i/curdev1/server10ga
sybase    5241  0.8  2.5   532   772 ?   S    Mar  7 1599:51 -M/curdev1/server10ga/install
                                                              /curdev1/server492/bin/dataserver -d/dev/rsd3e
                                                              -e/curdev1/server492/install/errorlog_GODZILLA4
                                                              92
sybase    1610  0.0  3.8   712  1164 ?   S    May 17 75:27  /curdev1/server1001/bin/dataserver
                                                              -d/devices/SERV1001A_master.dat -sSERV1001A
                                                              -e/curdev1/server1001/install/SERV1001A_errorlo
                                                              g -i/curdev1/server1001
sybase    5236  0.0  0.0  3524  0   ?   IW   Mar  7  0:00  /curdev1/server10ga/bin/backupserver
                                                              -SD10GA_BKUP
                                                              -e/curdev1/server10ga/install/D10GA_BKUP.log
                                                              -I/curdev1/server10ga/interfaces
                                                              -M/curdev1/server10ga/bin/sybmultbuf
sybase    1621  0.0  0.0  3552  0   ?   IW   May 17  0:00  /curdev1/server1001/bin/backupserver
                                                              -SSERV1001A_BKUP
                                                              -e/curdev1/server1001/install/SERV1001A_BKUP.lo
                                                              g -I/curdev1/server1001/interfaces
                                                              -M/curdev1/server1001/bin/sybmultbuf
                                                              -Lus_english -Jiso_1
```

# CPU Utilization of SQL Server (continued)

2. Show the process status (ps) for the data server

```
ps -auxww|grep 1610
```



# Overall System CPU Usage

3. Determine O/S and SQL Server utilization trend

```
vmstat 5
```

```
procs     memory          page                    disk     faults         CPU
r b w avm fre  re   at pi po fr de sr s0 s1 s  s3  in   sy   cs   us sy id
                                                  2
4 0 0 0   284  0    8  0  0  0  0  1  1  0  1  1   42  230  356  4  9  88
3 0 0 0   260  0    2  0  0  0  0  0  0  0  0  0   133 512  401  88 12 0
3 0 0 0   256  0    0  0  0  0  16 0  0  0  1  0   151 558  372  84 16 0
3 0 0 0   256  0    20 0  0  40 0  27 2  0  2  5   166 462  346  82 18 0
3 4 0 0   252  0    56 0  16 36 0  9  0  24 0  0   318 424  623  66 34 0
3 0 0 0   252  0286 0  36 48 0  3  0  17 0  1  304 866 495  77   23 0
3 0 0 0   252  0    73 0  4  8  0  0  0  0  0  0   165 942  358  86 14 0
```

Running Processes

Blocked, waiting for something

% User Processes

% for System Process

No idle time of CPU

# From Within SQL Server

4. Determine the percentage of time the SQL Server was using the CPU (that it has been given) during an elapsed time interval (cpu_busy)

```
sp_monitor
```


```
last_run                    current_run                 seconds
 -------------------------- -------------------------- ----------
       May 20 1994 11:57AM        May 20 1994 11:59AM         105

 cpu_busy                   io_busy                    idle
 -------------------------- -------------------------- --------------------
 1566(99)-94%               0(0)-0%                    244801(1)-0%

 packets_received           packets_sent               packet_errors
 -------------------------- -------------------------- --------------------
 5297(4)                    28583(4466)                30(0)

 total_read        total_write       total_errors       connections
 ----------------- ----------------- ------------------ ------------
 2336(0)           94299(14)          0(0)              136(1)

(return status = 0)
```

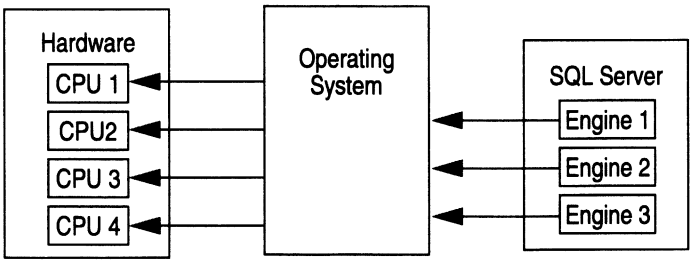
- First number is amount since last SQL Server reboot – is cumulative
- Second number is amount during the elapsed time listed

Note: This does not reflect the CPU utilization for SQL Server!

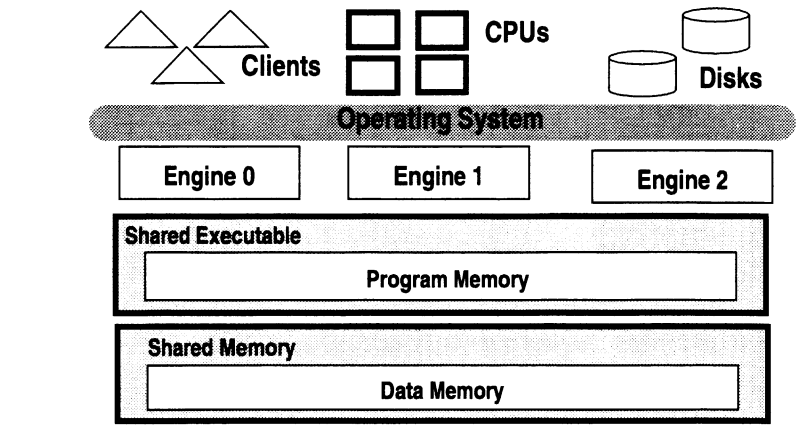
# Single CPU Machine Summary

- Measure the CPU utilization for SQL Server using O/S utilities
- Measure the percentage of time SQL Server was using CPU during an elapsed time interval using sp_monitor or SQL Monitor
- If CPU utilization is above 85% most of the time, consider multi-CPU environment or off loading work to another server machine
- Low CPU_busy % usually indicates idle SQL Server and/or I/O intensive processing

# SMP Machines (Symmetric Multi-Processing)




- O/S distributes its work evenly across all CPUs
- SQL Server (via max online engine) requests one or more CPU's processing power from the O/S
- That request will be distributed (balanced) among the available CPUs




**SQL Server**

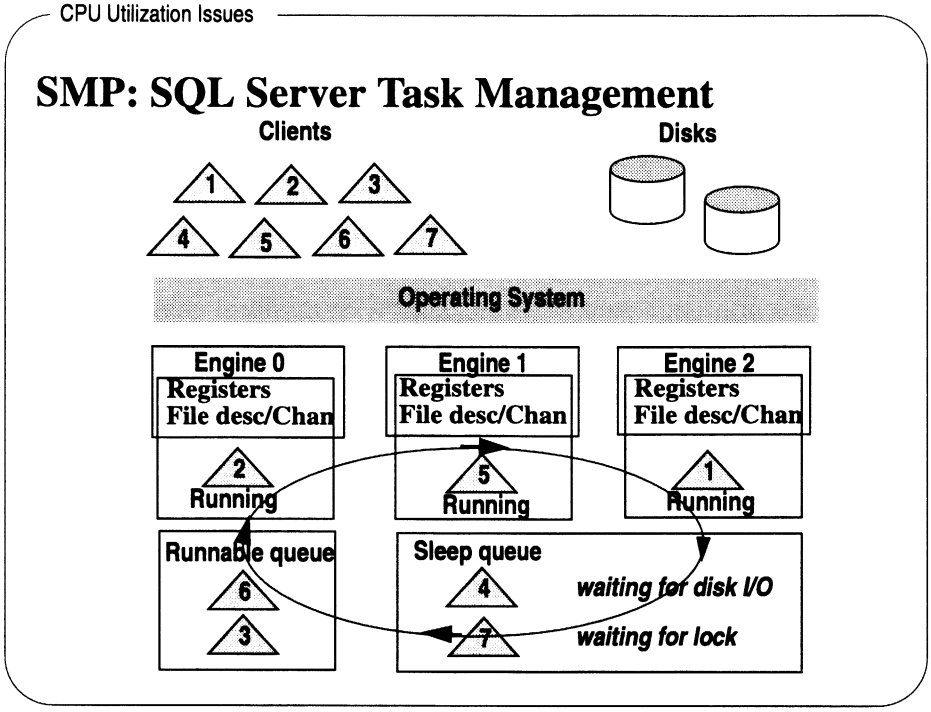
- Consists of one or more cooperating processes (called engines) all running in parallel.

**All Engines Are Peers**

- Any engine can handle any task and use any CPU.
- Exception: engine 0 handles network management.
- Engines communicate via shared memory.

**Engines Run Tasks**

- O/S schedules engine processes onto physical processors.
- Any available CPU is used for any engine.




**1** A client application issues a login request. SQL Server creates a user task to handle work from the client.

**2** The client presents SQL Server with work to do, for example a series of Transact-SQL commands.

**3** SQL Server adds the client's user task to the runnable task queue. The server engines compete for the user task at the head of the task queue.

**4** The server engine that takes the user task from the queue converts the Transact-SQL commands into low level steps, such as disk I/O.

**5** The engine executes each step until the task completes or blocks while waiting for I/O or locking. When the task blocks, it yields the server engine to run other user tasks. Once the block is resolved (i.e., disk I/O completes or a lock is acquired), the user task is again added to the runnable task queue.

**6** After the task blocks for the last time, it continues executing until finished. At that time the user task yields the server engine and moves to the sleeping task queue until the client presents the server with more work.

## SMP: Choosing the Right Number of Engines

- Monitor CPU usage with appropriate operating system utility
- For a dedicated machine, optimal number of engines is number of CPUs minus 1
- For a non-dedicated machine, start with minimal engines (like 1) and then add additional ones based on CPU utilization measurements

**Usable CPUs**

- Remember that other processes may require one or more of the CPUs.
- One engine per CPU may be excessive if the operating system or other non-server processes take up more than one of the CPUs.

**Too Many**

- If you have too many CPUs, the cost of Sybase scheduling offsets the performance gain.

## SMP: Application Design Considerations

- Beware of multiple indexes on a single table
  - Increased throughput of SMP may result in increased lock contention
- Lower the fillfactor
  - Because of added throughput, a lower fillfactor may be needed to reduce contention for data pages
- Keep transactions short
  - Long transactions on SMP have an even greater chance of lock contention

**SMP Compatibility**

SMP SQL Server is compatible with the single processor SQL Server. Applications that run on a single processor server can be run on a multi-processor server without modification.

**Multiple Indexes**

Because of the high throughputs possible on multi-processor servers, it is important to remember that locking problems increase as the number of transactions being processed increases. Therefore, keep in mind all of the performance suggestions regarding lock management and, most importantly, use no more than two or three indexes on a heavily updated table. This will prevent a large number of transactions from interfering with each other as the indexes are updated.

# SMP: Overburdened CPU

- If another major process (another SQL Server, a network server, etc.) is competing for CPU cycles, isolate it to another machine
- On platforms where affinity is possible, you can bind engines to processors

**Note**

For SMP machine, reduce the max online engines parameter to accommodate the other process.

# Multi-CPU Machines Summary

- Measure the CPU utilization for SQL Server using O/S utilities
- The percent of time SQL Server is using CPU during an elapsed time interval is now a reflection of a multiple CPU power processing request
- If CPU utilization is at or near 100% most of the time, consider adding more CPUs to the hardware configuration
- CPU_busy % indicates SQL Server CPU processing during a time interval for the number of engines configured

  It is not a direct reflection of CPU utilization!

# CPU: The Time Slice Story

## Time Slice

- Sets the number of milliseconds SQL Server's scheduler allows a user process to run

```
default: 100
```

- If you anticipate short periods where queries will have
  - Long execution time, or
  - Large result sets

  Such as quarterly or end-of-year processing, increase time slice for that period; example:

```
sp_configure "time slice", 300
```

- Don't forget to reset after this special processing period

# CPU: Typical Questions to Ask

- What is the CPU utilization during normal and off-hours processing periods?
- What is the CPU utilization when the SQL Server is not running?
- How many CPUs can be devoted to the SQL Server?
- Can a faster CPU or more CPUs be added as the system evolves?
- Is the timeslice set appropriately for the processing that is being done?

# Sample Problem

**Response times of all processes go up when a new process is added**

COLLECT DATA

- How many CPUs available?
- CPU utilization?
- sp_monitor measurements?
- Type of processing?
  - CPU intensive?
  - I/O intensive?
- sp_configure settings?

# Sample Problem (continued)

COLLECT DATA RESULTS

- 4 CPUs available
- CPU utilization 24.5%
- @@CPU_busy = 94%
- max_online_engines = 1
- No other major processes are running that are competing for CPUs

FORMULATE HYPOTHESIS

- Not enough engines configured for SQL Server
- Increase the maximum online engines to 2

# Sample Problem (continued)

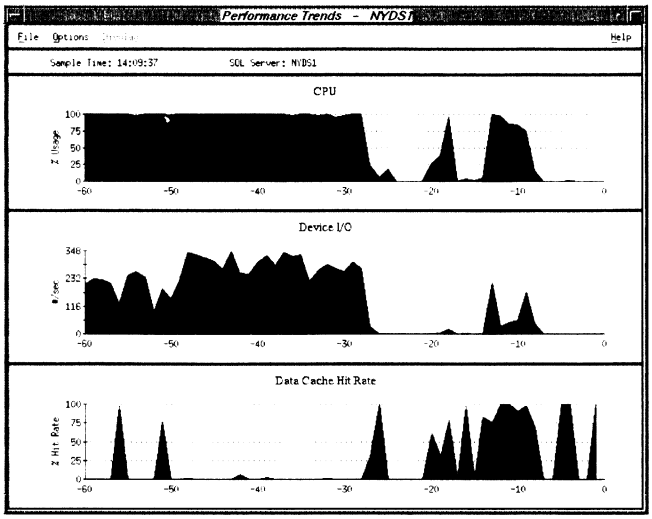
TEST HYPOTHESIS

- sp_configure "max online engines", 2
- Reboot the Server
- Resume the processing load
- sp_monitor now showing @@CPU_busy = 51%
- O/S utilities still showing CPU utilization = 24.5%

IMPLEMENT IN PRODUCTION IF CONFIRMED

- Monitor regularly

# SQL Monitor - Performance Trends




**Function**

Shows changes in SQL Server performance over time

**Uses**

Another good starting place for investigation

Shows performance over multiple sample interval

**Display**

CPU Utilization

Data cache hit rate

Network traffic

Lock hit rate

Lock request rate

Procedure cache hit rate

**Instructor Guide**

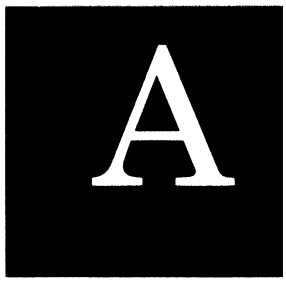
Here we have displayed three windows. You can display up to six trends. Add or delete graphs with Display -> Add Graph or Display -> Delete Graph.

# Lab 15 – Using CPU Resources

- Determine the CPU statistics for a given procedure, then find which of the procedures it calls uses the CPU most intensively
- Determine the overall CPU utilization of SQL Server during these executions

# Summary

- Single CPU machines
  - Use OS utilities to see CPU utilization
  - sp_monitor to see percent of time the SQL Server was using the CPU
- SMP machines (multi-CPU)
  - OS distributes work evenly across all available CPUs
  - SQL Server "max online engines" request
  - Right number of engines?
  - Application design considerations
- CPU: Time slice



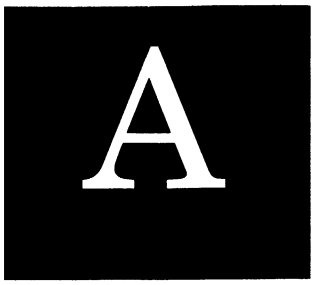
# Distributing Data with Replication Server

## System 10 Performance & Tuning

Version 2.2



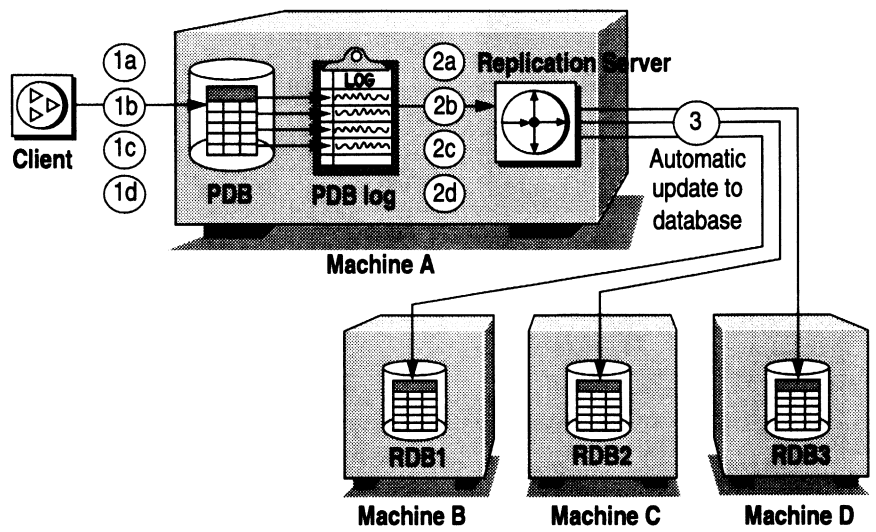

*The Enterprise Client/Server Company™*


A

Distributing Data with Replication Server

# Replication Server

- Continuous, asynchronous transfer of log records to replicate dbs




**1. Transactions**

Client continuously requests transactions that change database contents

Transactions are recorded in a transaction log for each database

**2. Replication Server Captures Transactions**

Without locking tables, Replication Server captures transactions (from log)

**3. Transactions Written into Replicate Databases**

Transactions are asynchronously but automatically written into replicate databases on other machines

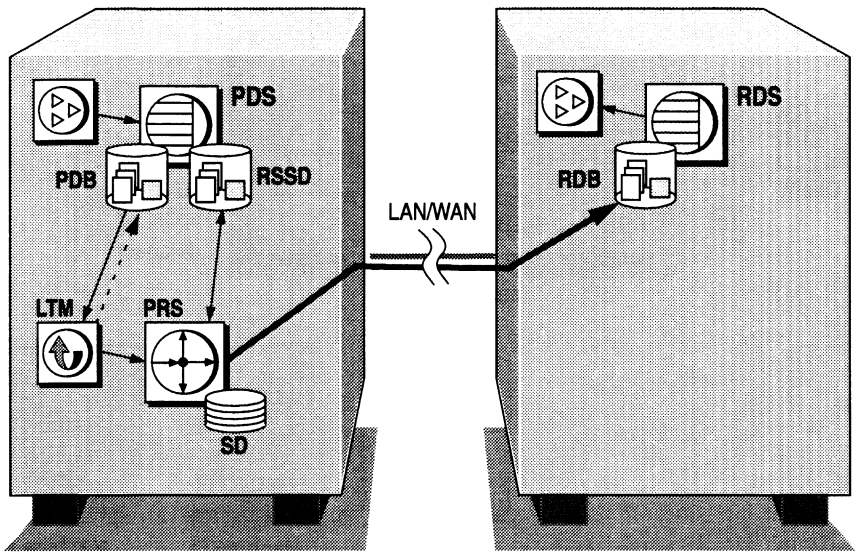
**Transaction Level**

Transactions in the primary database are applied consistently in all replicate databases

**Drawbacks**

Some latency

## Sample Replication System




**LTM Forwards Log Records**
- To the PRS

**PRS Stores Log Records on SD**
- While PRS processes them

**PRS Uses RSSD**
- To process log records
- By connecting to PDS that hold RSSD

**PRS Writes to RDB**
- It actually connects to the RDS

**Acronyms**
- See next page.

# Replication Server: Performance Impact

- Primary Data Server (PDS)
  - Potential for CPU and I/O sharing/contention with RS
  - Potential high use of network if RS is on separate machine
  - Frequent access of Replication Server System Database
  - Frequents scans by Log Transfer Manager
- Replication Server (RS)
  - Potential for CPU and I/O sharing/contention with PDS
  - Potential high use of network if PDS is on separate machine
  - Stable device usage may be high
  - High network traffic to forward transactions to Replicate Data Servers

| | |
|---|---|
| **PDB** | Primary Data Base |
| **SD** | Stable Device: Storage mechanism where Replication Server writes transactions. These queue up waiting to be delivered. |
| **PRS** | Primary Replication Server |
| **RSSD** | Replication Server System Database: Stores everything Replications Server has to deal with--the meta data, not the data itself. |
| **LTM** | Log Transfer Manager |
| **RDS** | Replicate Data Server |
| **RDB** | Replicate Data Base |
| **PDS** | Primary Data Server |

# Replication Server: Performance Impact (continued)

- Replicate Data Server (RDS)
  - Potentially processes massive amounts of transactions!
- Issues
  - How many transactions
  - Which tables accessed
  - Others

B

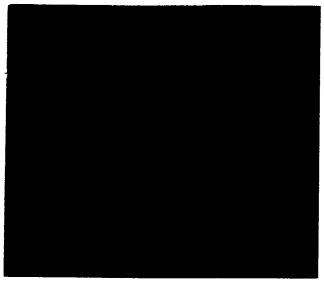
# More About SQL Monitor

System 10 Performance & Tuning

Version 2.2




*The Enterprise Client/Server Company™*

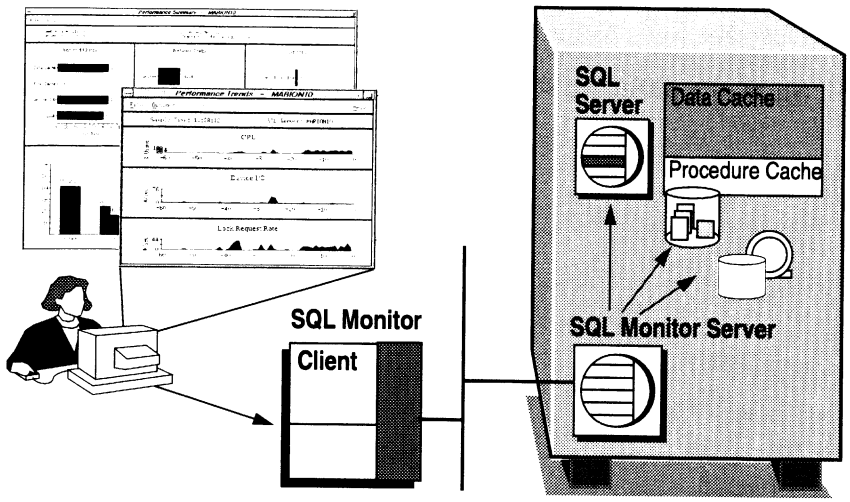


# Objectives

- Describe SQL Monitor features
- Describe SQL Monitor architecture
- Show the different SQL Monitor performance statistics graphs

# SQL Monitor

Performance monitoring tool for SQL Server




**SQL Monitor**

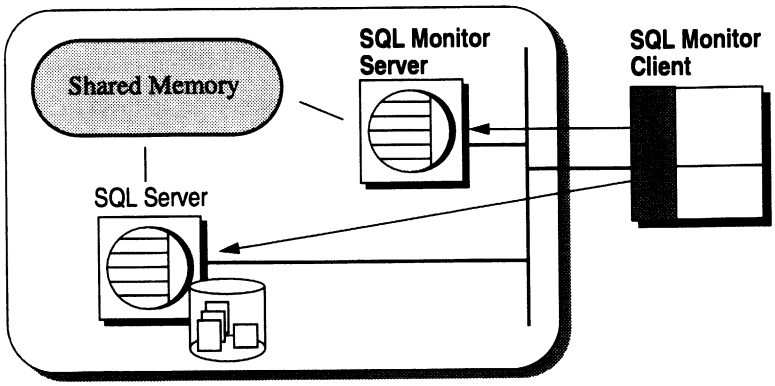
- Graphical User Interface (GUI)
- Feedback on the effect of tuning activities
  - Before and after image captures
- Filtering options
- Display on-line help

**Monitors**

- Data and procedure cache effectiveness
- Database device activity
- Lock activity
- Memory allocation
- Network traffic
- User processes
- Transaction activity

# SQL Monitor Architecture

- Accesses proprietary shared memory area of SQL Server
- Allows for minimal impact on SQL Server performance




**SQL Monitor Architecture**

- SQL Monitor Server, an Open Server application
- SQL Monitor Client, a GUI client application

**Shared Memory**

- SQL Server saves performance data in a shared memory area
- Sharing memory minimizes impact on SQL Server performance

**SQL Monitor Server**

- Obtains performance data in a unobtrusive way
- Reads the shared memory area at specified intervals
- Sends, when requested, data to SQL Monitor Client
- Supplies most of the performance data used by SQL Monitor Client

**SQL Monitor Client**

- Focusses on display of performance data in graphical form
- Relies on SQL Monitor Server to collect most performance data
- For some performance data items SQL Monitor Client connects directly to the SQL Server

# Starting The SQL Monitor Client

sqlmon [options]

| flag | argument | Description |
|---|---|---|
| -I | *filename* | Interfaces file name |
| -M | *server_name* | Monitor server to connect to |
| -P | *password* | password for user name described with -U option |
| -t | *timeout_sec* | timeout interval |
| -U | *username* | SQL Server user name to use when connecting to SQL server |
| -nomem | none | Specifies main window does not display memory information |

**Example**

```
sqlmon -U sa -P secret -M MONITOR_NYDS1
```

**Other Useful Flags**

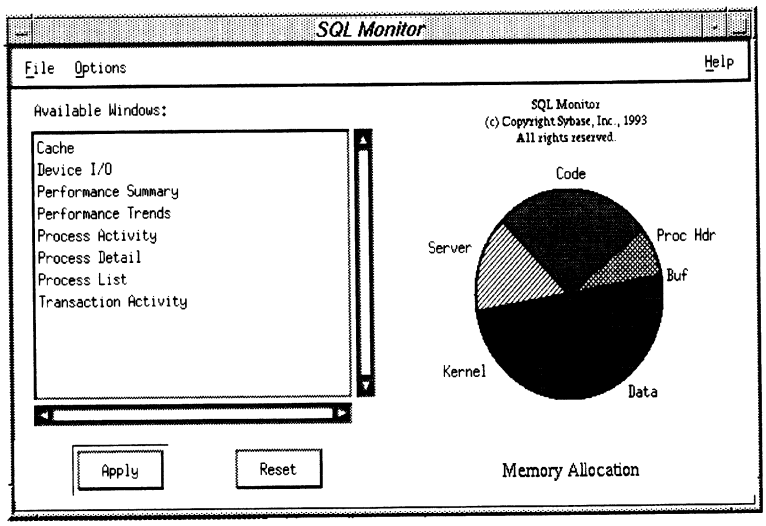
-h Displays a list of valid commands and returns

-sv Displays SQL Monitor software version

-v Displays software version number and copyright banner

All of these commands display the requested information and return to the operating system prompt.

# SQL Monitor Main Menu




**Function**

First window to appear

**Uses**

- Gain access to other windows
- See current allocation of data and procedure cache

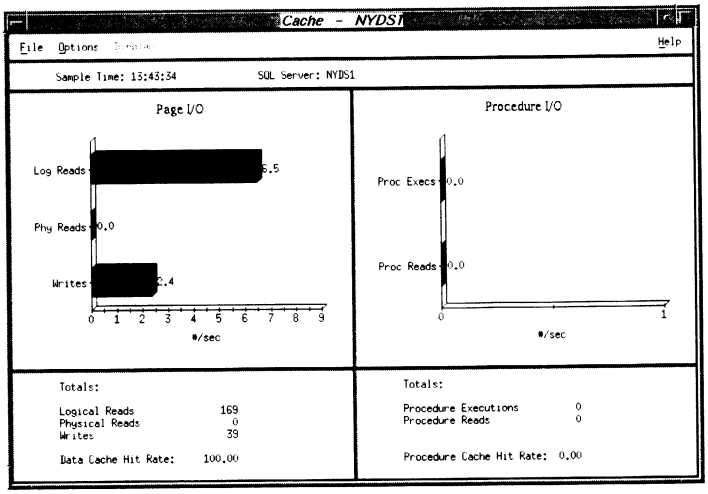
**Display**

- Lists other windows
- Displays memory allocation chart

# SQL Monitor Windows

- Cache
- Device I/O
- Performance Summary
- Performance Trends
- Process Activity
- Process Detail
- Process List
- Transaction Activity

# Cache Window




**Function**

Determines the efficiency of data and procedure cache

**Uses**

Data and procedure cache ratios

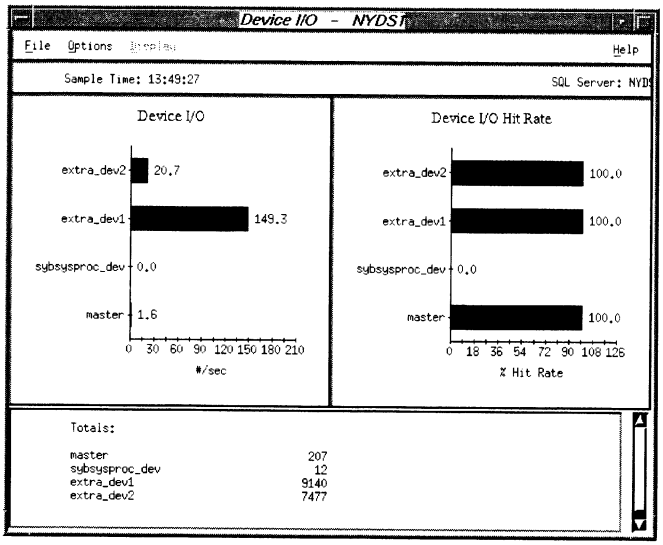
Memory Usage

Procedure vs. data cache distribution

**Display**

- Page I/O
  - logical reads
  - physical reads
  - writes
  - totals
  - data cache hit rate
- Procedure I/O
  - procedure executions
  - procedure reads
  - totals
  - procedure cache hit rate

# Device I/O Window




**Function**

Shows I/O activity occurring on defined devices

**Uses**

Level of activity on each device
Distribution of load across devices
Level of contention for device resources

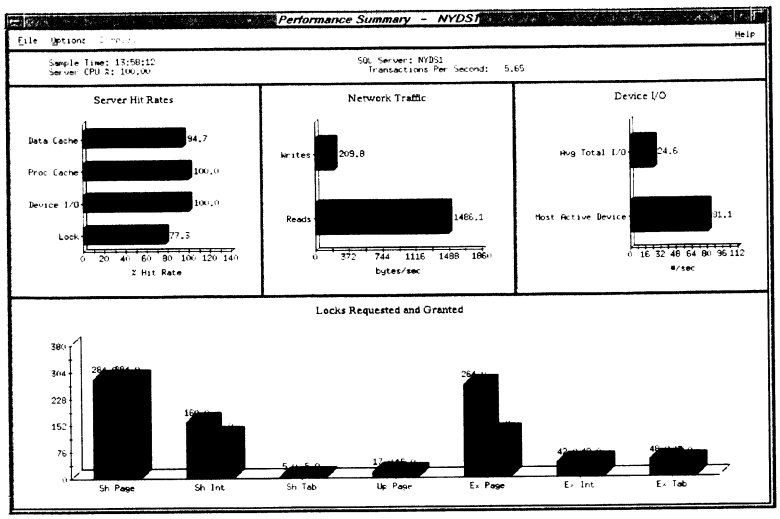
**Display**

Name of each device defined
Writes to each device
Reads to each device
Total reads and writes since the window was opened

**Summary and Detail Views**

To switch between the summary and detail views, select Display -> Summary View On/Off from the window menu bar.

# Performance Summary Window




**Function**

Summarizes performance metrics

**Uses**

Good starting point for investigation

Points to trouble areas

**Display**

Transactions per second

CPU busy percentage

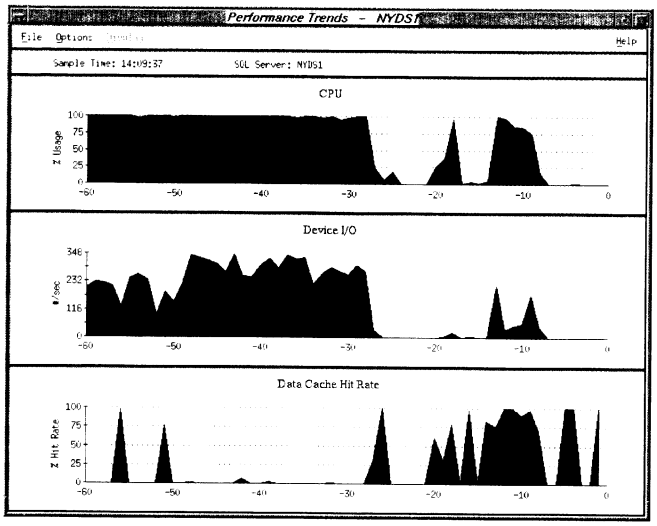
Server hit rates

Network traffic

Device I/O

Locks requested and granted

# Performance Trends Window




**Function**

Shows changes in SQL Server performance over time

**Uses**

Another good starting place for investigation

Shows performance over multiple sample intervals

**Display**

CPU Utilization

Data cache hit rate

Network traffic

Lock hit rate

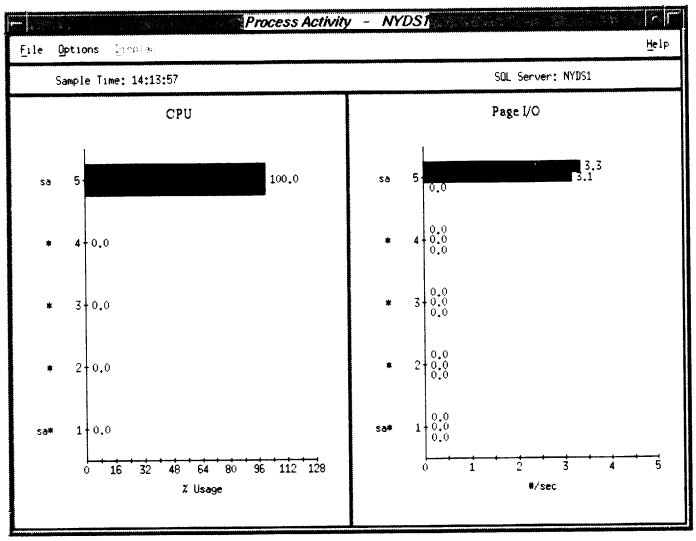
Lock request rate

Procedure cache hit rate

**Instructor Guide**

Here we have displayed three windows. You can display up to six trends. Add or delete graphs with Display -> Add Graph or Display -> Delete Graph.

# Process Activity Window




**Function**

Shows CPU rate and page I/O rate for as many as 20 user processes

**Uses**

Visually compare the way processes use server resources

Provides flexibility in how you determine which processes to monitor

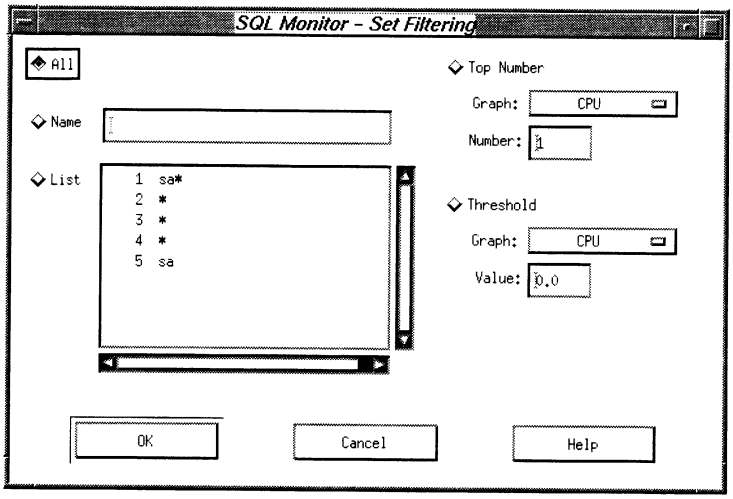
**Display**

CPU percentage for each process selected

Page I/O

- logical reads
- physical reads
- writes

# Filtering Process Information




**How To Open Window**

Select Options -> Set Filtering... from the Process Activity menu bar.

**Set Filtering**

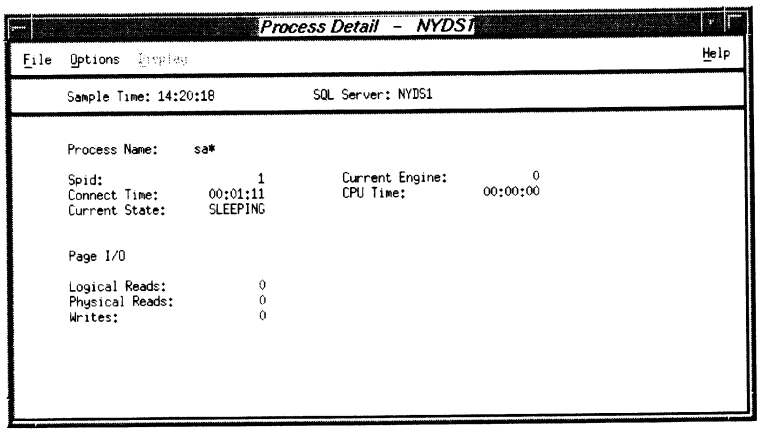
To set filtering, do one of the following:

Select the *All* radio button to monitor all processes currently active

Enter a text string in the input field next to the *Name* radio button to select a particular named process. SQL Monitor accepts a percent sign at the end of the text string as a wildcard character.

Highlight the desired processes on the selection list next to the *List* radio button.

# Process Detail Window




**Function**

Provides detailed information on a single process.

**Uses**

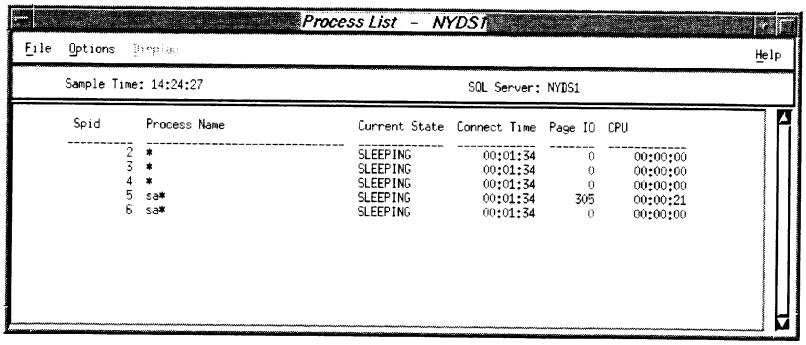
Shows how much work a selected process is doing.

Shows which resources it is using most intensively.

**Display**

- Process Details
  - Process name
  - SQL server process id
  - Connect time
  - Current State (RUNNABLE, SLEEPING, TERMINATING, YIELDING, INFECTED)
  - Current engine
  - CPU Time
- Page I/O
  - logical reads
  - physical reads
  - writes

# Process List Window




**Function**

Lists all user processes currently active in SQL Server.

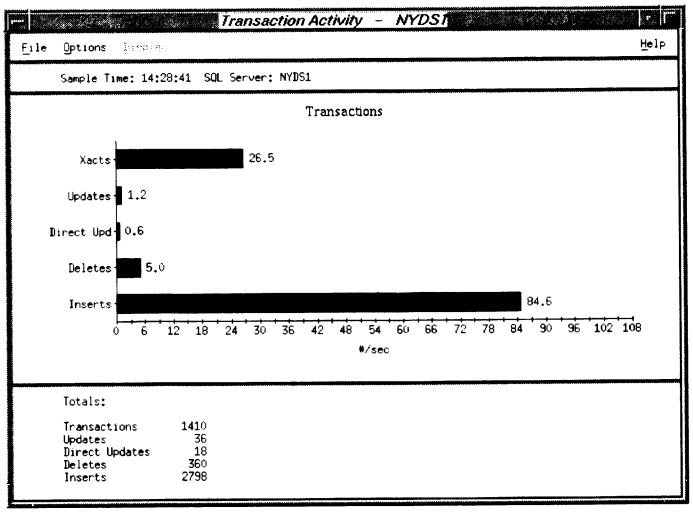
**Uses**

Provides a quick overview of current process activity.

**Display**

- Spid -- SQL Server process identification code
- Process Name -- Name of the process
- Current State -- Current Process State:
  - RUNNABLE Waiting to run according to priority
  - RUNNING Executing
  - TERMINATING In the process of being killed
  - YIELDING Voluntarily yielding execution
  - INFECTED Tagged by SQL Server as unprocessable
- Connect Time -- Time elapsed since the process was created
- Page I/O -- Total number of page reads and writes completed by the process since the process was created
- CPU -- Amount of SQL Server CPU ticks the process has used.

# Transaction Activity Window




**Function**

Summarizes Transact-SQL transactions executed by SQL Server. Activity is reported by transaction type.

**Uses**

Provides a rough measure of how well SQL Server is performing.

Breaks down database activity by transaction type.

**Display**

Shows the transaction rate per unit of time that occurred during the sample interval:

- Xacts Total number of Transact SQL statement blocks delimited by a begin transaction and commit transaction statement
- Updates Updates to a database table
- Direct Upd Updates in place to a database table
- Deletes Deletions from a database table
- Inserts Insertions to a database table

# Summary

- SQL Monitor features
  - Presents performance information visually and quantitatively
  - Has a point-and-click interface
  - Displays can be customized
- Use to
  - Establish a performance baseline
  - Find trouble spots and bottlenecks

# SQL Monitor Demos

- Allow the instructor to set up the system

Demos:

1. The "Giant Select" Demo
   - Observe cache hit ratio
2. The "install pubs 1000 times over" Demo
   - Where is the bottleneck?
3. The "Giant Select" & "install pubs 1000 times over" Demo
   - Where is the bottleneck now?

# SQL Monitor Demos

1. Script for creating a large table (5Mb log) & running a large query (The table should be larger than data cache for this demo to work)

```
dump tran pubtune with no_log
go
create table bigtable (a int, b char(255))
go
insert into bigtable values (1, "Once, when I walked up the stairs, I met
a man who wasn't there. He wasn't there again today, Oh how I wish he'd go
away!")
insert into bigtable values (2,"I was sad because I had no shoes, Until I
met a man who had no feet.  So I said, Dude!  Got any shoes you don't
need?")
go 5000
dump tran pubtune with no_log
go
insert into bigtable values (1, "Once, when I walked up the stairs, I met
a man who wasn't there. He wasn't there again today, Oh how I wish he'd go
away!")
insert into bigtable values (2,"I was sad because I had no shoes, Until I
met a man who had no feet.  So I said, Dude!  Got any shoes you don't
need?")
go 5000
dump tran pubtune with no_log
go
select count(*) from bigtable
go 500
EOF
```

2. The installpubs over-and-over script

```
isql -Usa -Pheydude -SNYDS1 <$SYBASE/scripts/installpubs2
isql -Usa -Pheydude -SNYDS1 <$SYBASE/scripts/installpubs2
isql -Usa -Pheydude -SNYDS1 <$SYBASE/scripts/installpubs2
isql -Usa -Pheydude -SNYDS1 <$SYBASE/scripts/installpubs2
isql -Usa -Pheydude -SNYDS1 <$SYBASE/scripts/installpubs2
isql -Usa -Pheydude -SNYDS1 <$SYBASE/scripts/installpubs2
isql -Usa -Pheydude -SNYDS1 <$SYBASE/scripts/installpubs2
isql -Usa -............
```

C

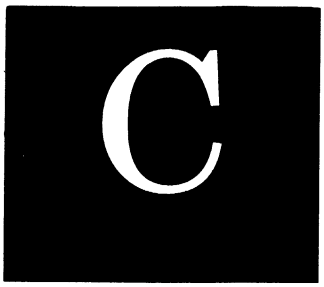
# Problem Analysis Walkthrough

System 10 Performance & Tuning

Version 2.2




*The Enterprise Client/Server Company™*



# Objectives

- Solve a performance problem using a systematic approach
- Obtain baseline metrics to measure the impact of modifications
- Use Sybase tools to evaluate and resolve performance problems

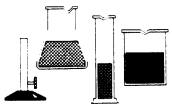
# Fundamental Tuning Guidelines

- Reduce contention for resources
- Reduce disk I/O
- Increase concurrency without increasing contention

Problem Analysis Walkthrough

## SQL Server Problem Analysis

1. Collect data
   - identify the performance problem
   - define performance requirements
   - measure the current performance
   - define SQL Server environment
   - analyze application design
2. Formulate hypothesis
3. Test hypothesis
4. If confirmed, implement;
   otherwise, collect more data or formulate another hypothesis



**Identify the problem**
- Determine the symptoms.
- Determine what components of the system model are affected.

**Define the performance requirements**
- Processing need.
- Frequency of execution.
- Response time required.

**Measure the current performance**
- Get baseline measurements using the appropriate tool given the component of the system that is affected.

**Define the SQL Server environment**
- Configuration at all layers.
- Limitations at all layers.
- Analyze application design.

**Analyze application design**
- Look at tables, indexes, transactions that comprise the application design.

**Formulate a hypothesis**
- Using the data, determine the problem and possible solution.

# Sample Problem

- Frequently-used query, DisplayTitle, runs poorly
- What should you do?

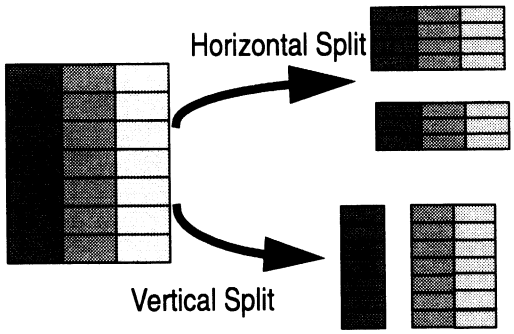
# Collect Data

- Ask more questions about its performance!
  - Constant? Dependent on number of users? All users experiencing same problem?
- Draw up a transaction-to-object-to-device/disk map so you can visualize where contention might be
- Display information (size, placement, indexes) on the objects involved
- Run the query yourself, displaying query plan, statistics, etc.

# Consider Your Options

- Determine
  - If database denormalization is needed
  - If the object placement supports the application
  - How best to index the tables involved
  - If the query could be rewritten to improve performance
  - Whether *tempdb* is being used heavily
  - If the memory allocation is sufficient
  - Whether or not the CPU is being used effectively
  - Whether or not there are any concurrency problems

# Is Denormalization Appropriate?




- What tools can we use to decide?
- What information do we need to create a baseline?
- How will we measure improvements?

**Tools**

- sp_estspace
- sp_helpindex
- showplan

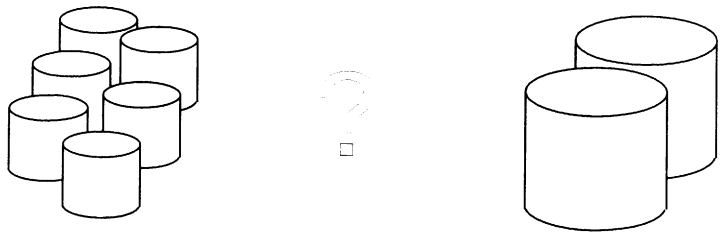
**Where Do I Begin?**

- Query plans will show the joins being performed and indexes that are being used.
- The size of the current tables and indexes will be used to measure the cost in space of any changes

**Did I Improve?**

- After the tables are denormalized, the new query plans will be compared to the old query plans to see if the number of amount of I/O has been reduced.

# How Should Objects Be Placed?



- What tools can we use to decide?
- What information do we need to create a baseline?
- How will we measure improvements?

**Tools**

- sp_helpdevice
- sp_helplog
- sp_helpsegment

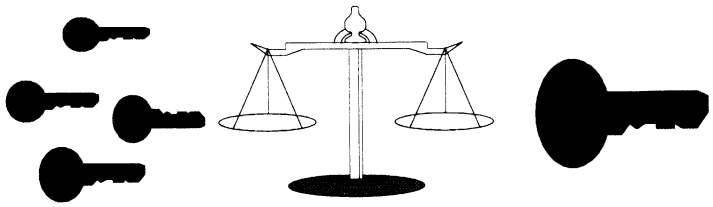
**Where Do I Begin?**

- What are the disk usage patterns?
- How are the current objects placed?

**Did I improve?**

- Have the disk usage patterns changed for the better?

# What Indexes Should Be In Place?



- What tools can we use to decide?
- What information do we need to create a baseline?
- How will we measure improvements?

**Tools**

- sp_helpindex
- sp_reportstats
- update statistics
- sp_configure extent io buffers
- showplan
- statistics io
- statistics time

**Where Do I Begin?**

- Do you have an accurate transaction model?
- How long do the present queries take?
- Are existing indexes being used?
- Can you cover queries?
- Are there any missing indexes?

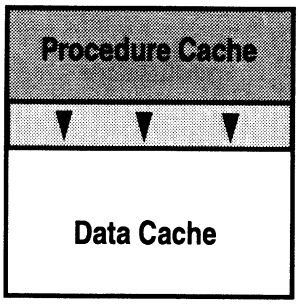
**Did I improve?**

- Are critical processes executing faster?
- Are the new indexes being used in the query plans?

Problem Analysis Walkthrough

# Is Memory Being Used Effectively?

- What tools can we use to decide?
- What information do we need to create a baseline?
- How will we measure improvements?




**Tools**

- dbcc indexalloc
- dbcc tablealloc
- sp_configure
- dbcc memusage
- statistics io

**Where Do I Begin?**

- What is the cache hit ratio?
- What are the sizes of the procedure and data caches?
- How much swapping is going on?

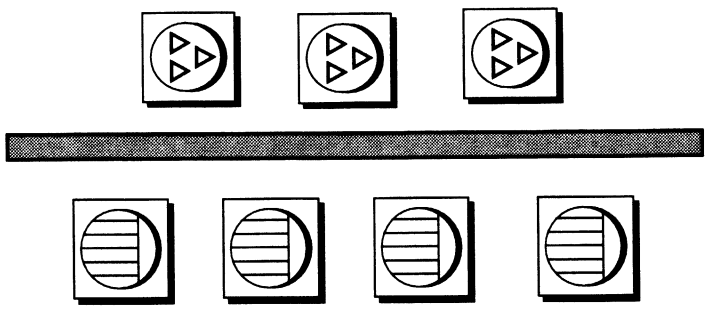
**Did I improve?**

- Did the cache hit ratio improve?
- Has swapping been reduced?

Problem Analysis Walkthrough

# Is The CPU Being Used Effectively?

- What tools can we used to decide?
- What information do we need to create a baseline?
- How will we measure improvements?



**Tools**

- sp_monitor
- O/S utilities
- SQL Monitor

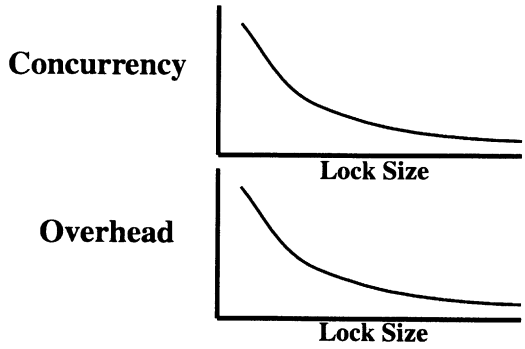
**Where Do I Begin?**

- How much is the CPU idle?
- How many on-line engines are there?
- What is the response time for critical processes?

**Did I improve?**

- Did the response time improve?
- Has CPU idle time been reduced?

# Are There Any Concurrency Issues?




- What tools can we use to decide?
- What information do we need to create a baseline?
- How will we measure improvements?

**Tools**

- sp_lock
- sp_who

**Where Do I Begin?**

- How much locking is going on?
- What are the most frequently executed transactions?
- What is the response time for critical processes?

**Did I improve?**

- Did lock contention go down?
- Has the response time improved?

Problem Analysis Walkthrough

# Lab C – Introduction to Problem Analysis

- Collect data
- Formulate hypothesis
- Test Hypothesis
- Implement if confirmed