

Identifying Performance Problems

- Many different factors contribute to poor performance
- There are many different solutions
- The keys are:
 - Knowing what "normal" looks like (i.e., profiling)
 - Knowing where to look
 - Knowing what to look for
 - Knowing what can be done
 - Knowing how to make a change and evaluate its effect
- The course will help teach you these things
- Experience will teach you how to use the knowledge effectively

Types of Performance Problem

- Problem Scope Understanding & Definition
 - Individual User
 - Only affects one user
 - Possibly caused by data being input into the program?
 - Client/PC hardware / software / network issue?
 - Application Components
 - Affects a group of users.
 - Problem localized to one application, screen, or frame.
 - Whole Application / System
 - Affects all users of an application or a system.
 - Problem usually at the machine, database, or installation level.

Types of Performance Problem

- **Problem Scope Understanding & Definition (Continued)**

- **Time Based**
 - Different times of the day
 - Different days of the week or month
 - Random
- **Area Based**
 - Regional
 - Individual remote sites
 - Departments
- **Load / Concurrency Based**
 - "Works great until everyone is using it"
 - "It worked fine in the test environment" (with limited concurrent activity)

Factors Affecting Performance

- Whatever the problem area there are many possible causes
- **Internal to Ingres**
 - Server configuration
 - Logging system configuration
 - Locking and concurrency
 - Key and index design
 - Table locations
 - Optimization statistics
 - And more

- **External to Ingres**
 - Database design
 - Application design
 - Query design
 - Hardware resources
 - CPU
 - Memory
 - I/O subsystem
 - Disk-level contention
 - RAID overhead
 - Controller-level contention

- Users give a *perceived* view of a problem
- Turn 'Perception' into 'Facts'
 - How slow is 'slow?'
 - What is the normal response time?
 - How long has the system been slow?
 - Is the system slow for everyone?
 - What has changed?
 - Application Information
 - Which screen are they using?
 - What data was entered?
 - Which button was clicked?
 - Benchmark and Metrics
 - Compare users times to "benchmark"
 - Know what "Normal" really is
 - Proactively gather performance metrics on the system

- **Slow Response from Ingres**

- **All DBMS Servers**
 - Locking problem
 - Logging problem
 - Cache performance
 - I/O bottleneck
- **Individual Server(s)**
 - CPU intensive queries (possibly disjoint with Cartesian Product)
 - I/O intensive queries
 - Disjoint queries where all tables in “where” clause not properly joined
 - Indexing and/or storage structure problem
 - Lacking or bad table statistics
 - Poor load balancing
 - Localized CPU starvation
 - Memory shortages (QSF, PSF, QEF)

- **Slow Response from Ingres (Continued)**

- **Specific Table(s)**
 - Key structures
 - Indexes
 - Optimization statistics
 - Rule of Thumb (ROT) #1: Optimize all key columns and columns typically referenced in the “where” clause of a query (optimizedb -zk)
 - ROT #2: Remove statistics having granularity of 20% or less (statdump -zdl)
 - ROT #3: Use as many histogram cells as needed (optimizedb -zu200 -zr200)
 - Physical placement
 - Page overflow
 - Compression?
- **Specific Applications or Queries**
 - Query Execution Plan (QEP)
 - Estimates versus actual (QE90 output, timing metrics in application)

- **Take Stock**
 - **Definite performance problem identified**
 - You can't fix what you can't find, so this is an important first step
 - **Snapshot of all relevant information taken**
 - This is the only way to quickly identify and resolve the root cause issue
 - **User perception and your perception**
 - Expectation setting is an important part of problem resolution
 - **Next step is to find the cause**
 - Examine each area of performance if appropriate

- **DMF – Data Manipulation Facility**
 - **Only part of the DBMS that does I/O**
 - **Maintains a memory cache of data pages**
 - Looked after by the 'Buffer Manager'
 - **Processes page requests (FIX CALLS) from other facilities (QEF, RDF etc)**
 - Page in Cache = HIT
 - Page not in Cache = I/O
 - **Objective of DMF cache tuning is a high Hit rate > 95%**
 - Minimize the I/O
 - Not always possible, especially in reporting and ad-hoc query systems
 - **It is important to understand the relationship between single-page and group cache buffers (more information on the next slide) to determine if the "mix" is appropriate for your given environment. If not, it could indicate performance problems such as excessive table scanning that would be masked by the DMF cache Hit rate.**

- DMF writes “dirty” (modified) pages to disk
 - Performed by write behind (WB) threads
 - Also performed during consistency points (CP) by CP threads and WB threads
- Types of Cache Pages
 - Single page cache
 - Most keyed accesses
 - Group cache
 - Multi page I/O's (read-ahead)
 - Table scans
 - Range searches

- Goals for DMF Tuning
 - Cache must be big enough to prevent forced writing or discarding of pages
 - You need to find the right balance for writing dirty pages to disk
 - Keeps the physical database consistent
 - Reduces 'spikes' during Consistency Points
 - Reduces recovery time

- **Many ways to tune a query**

- **Within the query**
 - Change/add restrictions
 - Change/add joins
 - Select fewer columns
 - Use functions
 - Don't use functions
 - Split the query (decompose into two or more queries using Global Temporary tables)
- **Outside the query**
 - Change/add physical keys
 - Change/add secondary indexes
 - Drop secondary indexes
 - Add statistics
 - Change statistics
- **And much more**

- **The Golden Rules**

- **Less I/O means faster queries**
- **Process the most restrictive restrictions first**
 - Get to the smallest possible dataset as soon as possible

- **Database designer / DBA create physical keys and indexes**

- **Using Indexes usually means less I/O**
- **Problem queries might not use keys & indexes or use the wrong ones**
- **Multi-column keys and indexes should have the most unique columns first to quickly identify the desired result set of data**
- **Conversely, too many indexes can increase I/O on updates, inserts, and deletes**
 - As a rule of thumb, more than 5-7 indexes are likely excessive (1-4 are ideal)
 - Of course, this can (and should) be tested and validated