



Managing DB2 Performance in a Multiplatform Environment

Jim Wankowski

November 16, 2005

Agenda

- Defining Performance
- Monitoring Methods
- Identifying/Resolving Performance Issues
 - ✓ Memory Management
 - ✓ Physical Design Considerations
 - ✓ Space Management
 - ✓ SQL Design
 - ✓ Coding Tips
 - ✓ Optimization
 - ✓ Tuning Examples

What is Performance?

Performance: *noun*

“The fulfillment of a claim, promise, or request”

- How does your company define performance?
 - System availability
 - Transaction throughput
 - Minimum response times (SLAs)

Subsystem/Instance Monitoring

All aspects of the DB2 subsystem or instance need to be monitored

- Take a look at the big picture
 - Think of DB2 as an ecosystem
- Do not tune for the sake of tuning!
 - Where are your bottlenecks?

Monitoring Methods

z/OS

- **Instrumentation Facility Component (IFC)**
 - **Statistics**
 - Global statistical data
 - **Accounting**
 - Start and stop times
 - Number of commits and aborts
 - The number of times certain SQL statements are issued
 - Number of buffer pool requests
 - Counts of certain locking events
 - Processor resources consumed
 - Thread wait times for various events
 - RID pool processing
 - Distributed processing
 - Resource limit facility statistics
 - **Performance**
 - Most detailed \$\$\$
 - Only use for short periods

L,U,W

- **Snapshot Monitor**
 - Show status of database for an instant in time
 - Monitor Switches need to be turned on at the instance level to collect data
 - Low overhead (~5%)
- **Event Monitor**
 - Historical collection of data
 - More overhead (~10-20%)
 - Main focus on application statistics

Memory Management

z/OS

- EDM Pool
- RID Pool
- Sort Pool
- Buffer Pool

L,U,W

- Catalog Cache
- Package Cache
- Sort Heap
- Buffer Pool

Minimizing the amount of disk access should be a key performance objective

Memory Usage

z/OS

EDM Pool

- “System Bufferpool”
 - Minimizes I/O against catalog and directory
- Contains
 - DBD (Database Descriptor)
 - CT (Cursor Table)
 - PT (Package Table)
 - SKCT (Skeleton Cursor Table)
 - SKPT (Skeleton Package Table)
 - Plan/Package authorization Cache (CACHESIZE > 0)
 - Dynamic SQL skeletons (Dynamic SQL caching active)

L,U,W

Catalog Cache

- Minimizes I/O against catalog
- Contains
 - SYSTABLES information
 - Authorization information
 - SYSDBAUTH
 - Execute privileges for routines

Package Cache

- Minimizes I/O against catalog
 - Loading packages
 - Having to prepare Dynamic SQL

Possible Performance Implications

z/OS

EDM Pool

- Increased I/O activity against DSNDB01
 - SCT02
 - SPT01
 - DBD01
- Increased response times due to loading the SKCTs, SKPTs and DBDs
- Re-preparation of Dynamic SQL
- Fewer threads used concurrently, due to a lack of storage
- Resource unavailable “-904”

L,U,W

Catalog Cache

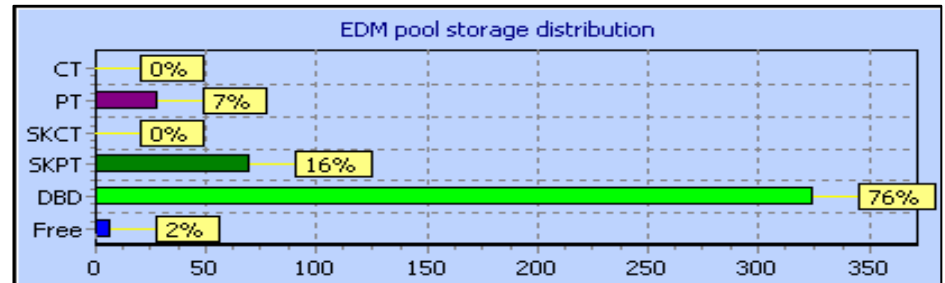
- Increased bind times
- Increased compile times
- Increase time to check DB and execution privileges

Package Cache

- Slower response time with Dynamic SQL

What to Monitor – z/OS

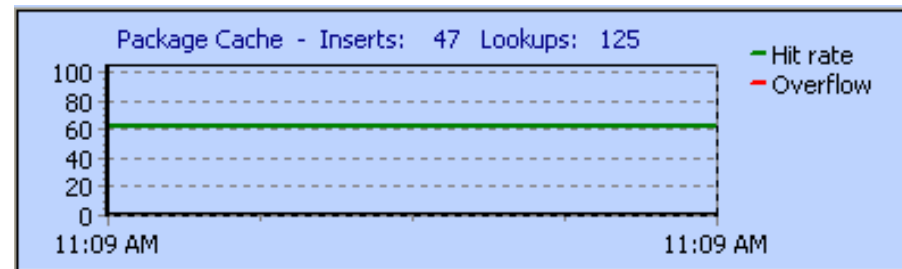
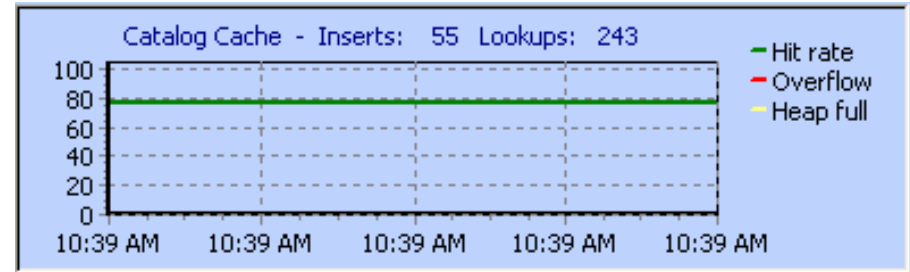
- EDM Pool hit ratio should be at least 80%
 - PT/CT's
 - 80-90%
 - DBD requests
 - 100%
 - Pages used for Package/Cursor tables <50% of pool



What to monitor - LUW

- Catalog Cache hit ratio 80-90%
 - Catalog Cache lookups
 - *Cat_cache_lookups*
 - Catalog Cache Inserts
 - *Cat_cache_inserts*
 - Catalog Cache Overflows
 - *Cat_cache_overflows*

- Package cache hit ratio
- Package cache overflows
 - *Pkg_cache_num_overflows*
- Package cache lookups
 - *Pkg_cache_lookups*
- Package cache inserts
 - *Pkg_cache_inserts*
- Package cache high water mark
 - *pkg_cache_size_top*



Managing EDM Pool Size

- Reduce size of DBD's
 - ✓ Try to minimize # of objects in database
 - ✓ Use 32K pieces for large databases
 - ✓ Run MODIFY utility regularly to remove old recovery info
 - ✓ Dropped objects
 - ✓ REORG tablespaces when tables dropped/recreated
- Dataspaces for Dynamic Caching
 - ✓ Specify a portion of EDM pool in a dataspace
 - ✓ EDMPOOL DATA SPACE SIZE > 0
- Avoid large plans
 - Use packages (avoid DBRM's)
 - Break applications into separate plans
- DEGREE(ANY) maintains 2 access paths
 - One each for parallelism YES/NO
- Bind w/AQUIRE(USE) vs. AQUIRE(ALLOCATE) when possible
- Use RELEASE(COMMIT) for infrequently used Plans/Pkg's
- Use RELEASE(DEALLOCATE) cautiously
 - Can cause EDM pool to grow enormously
 - Only change a few programs at a time

Sorting

z/OS

Sort Pool

- Memory area for all sort activity
- Sort pools should be made as large as possible
 - 240K – 64MB
 - The larger the sort pool, the more efficient the sort
 - Default to 1MB
 - Minimizes externalizing to physical sort files
 - DSNDB07

L,U,W

Sort Heap

- Number of pages available for private or shared sorts
 - Used by Optimizer for determining access paths
 - Sorting
 - Hash Joins
 - Index ANDing
 - Dynamic bitmaps
- Performance implications
 - Frequent large sorts
 - Non-piped vs. Piped

Optimizing Sort

z/OS

- **DSNDB07**
 - **Do not use BP0!**
 - 4 –5 equally sized datasets
 - Keep in primary space
 - Separate volumes
 - Separate BP's
 - Also used for:
 - View, temp table, and nested table expression materializations
 - Non-correlated in-lists

L,U,W

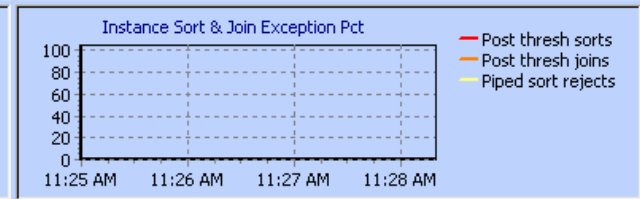
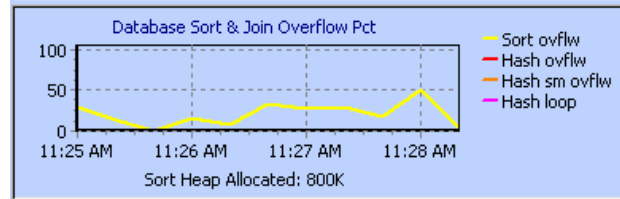
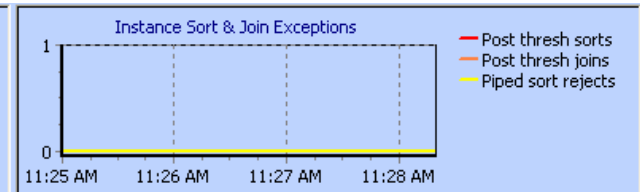
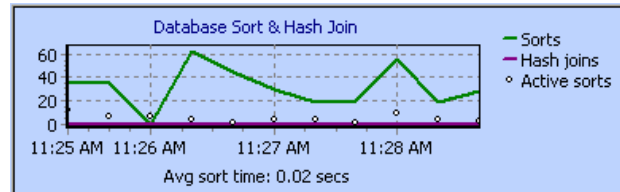
- **Avoid Sort Overflow**
 - If *sortheap* too small, sort will overflow into temp database tables
- **Avoid Non-Piped Sorts**
 - If sorted information must be stored in a temporary table vs. memory (*sortheap*)
- Determined at time of optimization

Optimizing Sort – All Platforms

- Proper indexing can minimize sorting
- Avoid ORDER by, Group By, Distinct
- Avoid sorting VARCHARs
- Only select required columns

What to Monitor - LUW

- Sort Heap Overflow
 - *Sortheap*
- Sort Heap Threshold
 - *Sheapthres*



RID Pool – z/OS

“Row Identifier” Pool

- Enforces unique keys during multi-row updates
- Used for storing and sorting RID’s for:
 - List Prefetch
 - Multiple index access
 - Hybrid Joins
- Performance Implication
 - If RID pool is too small above access paths revert to TS scans

What to Monitor – z/OS

- **Insufficient Pool Size**
 - RID pool too small
 - Recalculate size

- **RDS Limit**
 - RID list > 25% #rows in table
 - Prefetch turned off and TS scan results
 - Determined at Bind time
 - Have tables grown since BIND?
 - Make sure stats are accurate
 - RUNSTATS/REBIND

- **DM Limit**
 - RIDs req'd to satisfy query > 16 million
 - TS scan results
 - Is TS Scan best access?
 - Re-evaluate indexes
 - Add additional filtering

SQL statistics		
Event	Count	Per sec
Commits	18,836,894	49.71
Rollbacks	4,792	0.01
Incremental binds	70,285	0.19
Runtime reoptimizes	0	0.00
Direct row success	0	0.00
Direct row, index use	0	0.00
Direct row, tbs scan L	0	0.00
RID list success	520,010,971	1,372.21
RID failure, storage	0	0.00
RID failure, RDS limit	926	0.00
RID failure, DM limit	0	0.00
RID failure, size limit	1	0.00

Bufferpools

z/OS

- **Use DBM1 address space**
 - Virtual bufferpools
 - Can define up to 80 BP's
 - Limited to 1.6GB total
 - 1 TB in V8
- **DBM1 + Hiperpace**
 - Hiperpools
 - Additional 8GB of extended storage
 - “Holding tank” for infrequently updated data
- **MVS Dataspace**
 - Support up to 8M buffers
 - Allows for direct I/O from extended storage
- **Can be used by multiple objects**

L,U,W

- **IBMDEFAULTBP** automatically created with database
 - Additional pools created with DDL
- Uses memory from database shared memory (*database_memory*)
- 4 GB size limit
 - 32-bit machines can use extended storage cache
- Typically 1 BP for every page size

What to Monitor

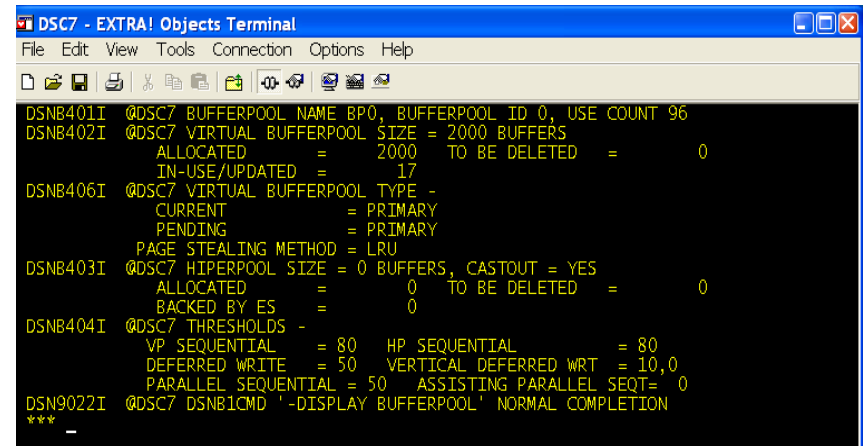
z/OS

- Bufferpool hit ratio
 - Accounting trace reports
- (GETPAGES – pages read)/GETPAGES
- Hiperpool hit ratio
 - Number of pages read from hiperpool/pages written
- Page externalization
 - Excessive writes to disk

L,U,W

- Overall hit ratio
 - Total # data/IX reads by BP
- Data hit rate
- Index hit rate
- Asynchronous page cleaners
 - *Num_iocleaners*

-Display Bufferpool(BP0)

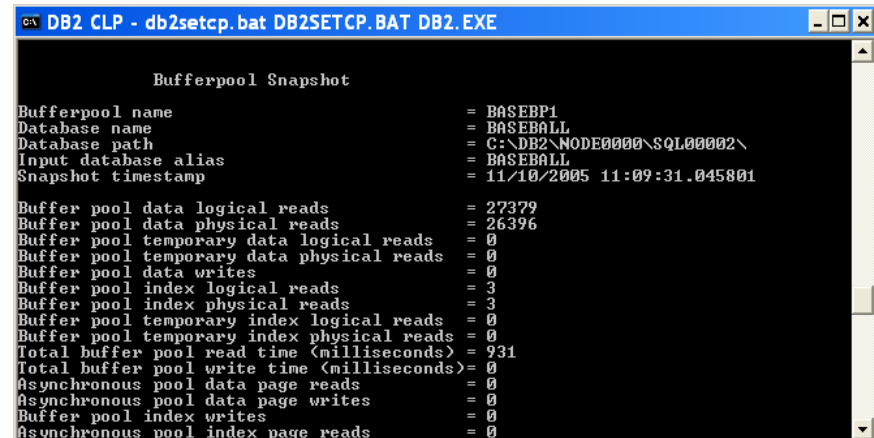


```

DSC7 - EXTRA! Objects Terminal
File Edit View Tools Connection Options Help

DSNB401I @DSC7 BUFFERPOOL NAME BP0, BUFFERPOOL ID 0, USE COUNT 96
DSNB402I @DSC7 VIRTUAL BUFFERPOOL SIZE = 2000 BUFFERS
          ALLOCATED      = 2000 TO BE DELETED    = 0
          IN-USE/UPDATED = 17
DSNB406I @DSC7 VIRTUAL BUFFERPOOL TYPE -
          CURRENT        = PRIMARY
          PENDING        = PRIMARY
          PAGE STEALING METHOD = LRU
DSNB403I @DSC7 HIPERPOOL SIZE = 0 BUFFERS, CASTOUT = YES
          ALLOCATED      = 0 TO BE DELETED    = 0
          BACKED BY ES   = 0
DSNB404I @DSC7 THRESHOLDS -
          VP SEQUENTIAL  = 80 HP SEQUENTIAL    = 80
          DEFERRED WRITE = 50 VERTICAL DEFERRED WRT = 10,0
          PARALLEL SEQUENTIAL = 50 ASSISTING PARALLEL SEQT= 0
DSN9022I @DSC7 DSNB1CMD '-DISPLAY BUFFERPOOL' NORMAL COMPLETION
***
    
```

Get Snapshot for Bufferpools on DB



```

DB2 CLP - db2setcp.bat DB2SETCP.BAT DB2.EXE

Bufferpool Snapshot
Bufferpool name           = BASEBP1
Database name             = BASEBALL
Database path              = C:\DB2\NODE0000\SQL00002\
Input database alias      = BASEBALL
Snapshot timestamp        = 11/10/2005 11:09:31.045801

Buffer pool data logical reads      = 27379
Buffer pool data physical reads     = 26396
Buffer pool temporary data logical reads = 0
Buffer pool temporary data physical reads = 0
Buffer pool data writes             = 0
Buffer pool index logical reads     = 3
Buffer pool index physical reads    = 3
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total buffer pool read time (milliseconds) = 931
Total buffer pool write time (milliseconds) = 0
Asynchronous pool data page reads   = 0
Asynchronous pool data page writes  = 0
Buffer pool index writes             = 0
Asynchronous pool index page reads  = 0
    
```

Effective Use of Bufferpools

“Rules of Thumb”

z/OS

The single biggest performance mistake is to stick everything in BP0!

- Separate bufferpools for:
 - ✓ Catalog and Directory (BP0)
 - ✓ DSNDB07
 - ✓ Tablespaces
 - ✓ Indexes
 - Large VPSIZE
 - More than one
 - ✓ Small, read-only tables
 - ✓ Large tablespaces w/random access
 - ✓ Small frequently updated tables and IX's
 - ✓ Test environment for isolating test cases

DB2 has 80 bufferpools available for a reason!

LUW

- Separate bufferpools for:
 - ✓ System Catalog
 - ✓ Sequentially scanned tables
 - ✓ Temporary table spaces
 - ✓ Small frequently updated tables
 - ✓ Small read-only tables
 - ✓ Large tables w/random access

Physical Design Considerations - Tablespace

z/OS

- Simple
 - Avoid
- Segmented
 - Best for most applications
 - Superior I/O performance over simple
- Partitioned
 - Best for large volumes of data
 - Manageable w/partition independence
- Large
 - LOB,CLOB,BLOB
 - Tedious to manage
 - Auxiliary tables

LUW

- Regular
 - Extent size
 - 2-256 pages
 - Space allocation
 - SMS vs. DMS
- Temporary
- Large
 - Much easier to manage than z/OS
 - No auxiliary tables

Efficient Space Management

z/OS

- Allocate DB2 datasets to use only primary space
 - Excessive extents can cause inefficient I/O
- Determine best page size for application
 - Data sharing
 - 8K or 16K page size can reduce overhead in coupling facility
- Use HSM to manage datasets
 - Move infrequently accessed datasets to slower devices

LUW

- SMS – “System Managed Space”
 - Advantages
 - Space not allocated until needed
 - Less work for creating TS
 - Disadvantages
 - Cannot add/modify containers
 - Cannot separate indexes
- DMS – “Database Managed Space”
 - Advantages
 - Easily extend size with ALTER
 - Tables can be split across multiple TS’s
 - Indexes can be separated from table data
 - Better performance overall vs. SMS
 - Disadvantages
 - More work to administer

Physical Design Considerations – Indexes All Platforms

- Define primary keys with unique indexes
- Use indexes on foreign keys
- Define unique indexes with include columns
 - For index only access
- Define Indexes on join columns
- Use indexes on highly sorted columns
 - Save clustering index for this purpose
- Minimize use of indexes on heavily updated or inserted tables
 - Avoid use of data tablespace (LUW)

Maintenance

- Proper maintenance is critical for optimal performance
 - Reorganization
 - Statistics Collection

Reorg

- What causes fragmentation?
 - Insert/Update
 - Check PCTFREE and FREEPAGE
 - VARCHAR fields being updated

Monitoring for TS and Table REORGs

z/OS

- Cluster Ratio < 95%
 - SYSINDEXES
 - CLUSTERATIO < 95%
 - SYSINDEXPART
 - FAROFFPOS > 10% of CARD
- Excessive row relocation
 - SYSTABLEPART
 - NEARINDREF+FARINDREF > 10% of CARD
- Excessive extents (>50)
- Excessive drop space
- Excessive drop space
 - Simple TS only
 - PERCDROP > 10%
- LOB tablespaces
 - SYSLOBSTATS
 - ORGRATIO > 2

L,U,W

- Cluster Ratio < 90%
 - SYSCAT.INDEXES
 - CLUSTERRATIO
- Overflow of Rows
 - SYSSTAT.TABLES
 - OVERFLOW
- Fetch Statistics
 - SYSCAT.INDEXES
 - Small # of
 - AVERAGE_SEQUENCE_FETCH_PAGES
 - Growth of AVERAGE_RANDOM_FETCH_PAGES
- Empty Pages
 - SYSCAT.TABLES
 - FPAGES-NPAGES

Monitoring for Index REORGs

z/OS

- Excessive distance between LEAF pages
 - SYSINDEXPART
 - LEAFDIST > 200
 - Can cause Pre-fetch to be disabled
 - Should be monitored for growth
 - LEAFFAR > 10% of NLEAF (SYSINDEXES)
 - Excessive extents (>50)

L,U,W

- Number of LEAF pages
 - SYSCAT.INDEXES
 - NLEAF
- Low Cluster Ratio
 - SYSCAT.INDEXES
 - CLUSTERRATIO

Statistics

Accurate statistics are a critical factor for performance monitoring and tuning

RUNSTATS provides statistical information for:

1. Optimization of SQL
2. Monitoring status of objects

RUNSTATS

- When to run RUNSTATS:
 - ✓ After LOAD, REORG and REBUILD IX
 - ✓ After creating new index
 - ✓ After heavy insert, update, delete activity
- Rebind After RUNSTATS?
 - ✓ New index created
 - ✓ Cluster ratio \neq 80%
 - ✓ Changes more than 20%
 - Cardinality
 - Index leaf distance

Application Design

- SQL Design considerations
- Optimization

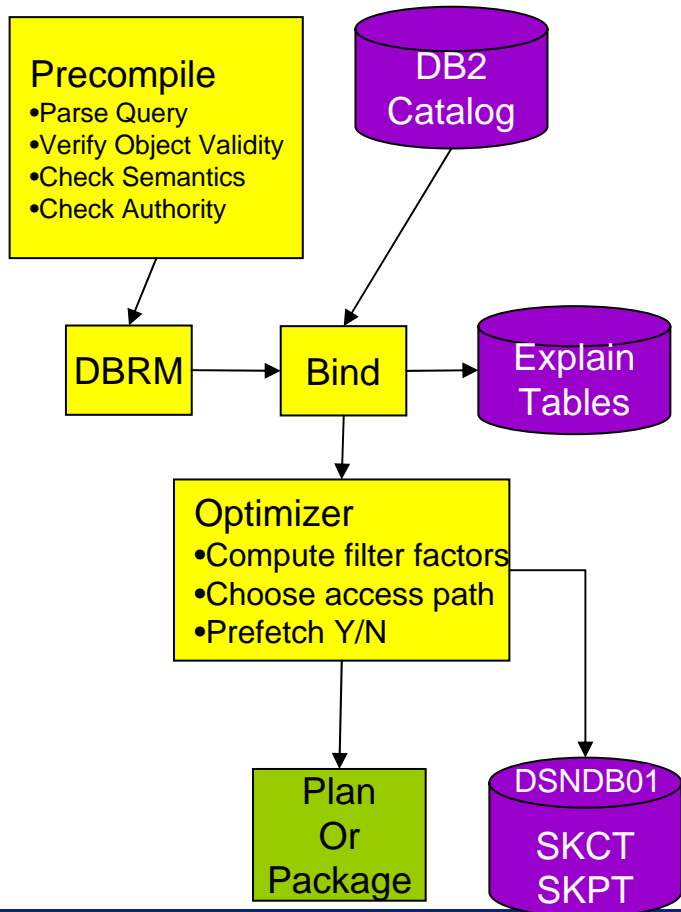
Efficient application design is the single most important aspect of an efficiently performing subsystem

SQL Coding Factors

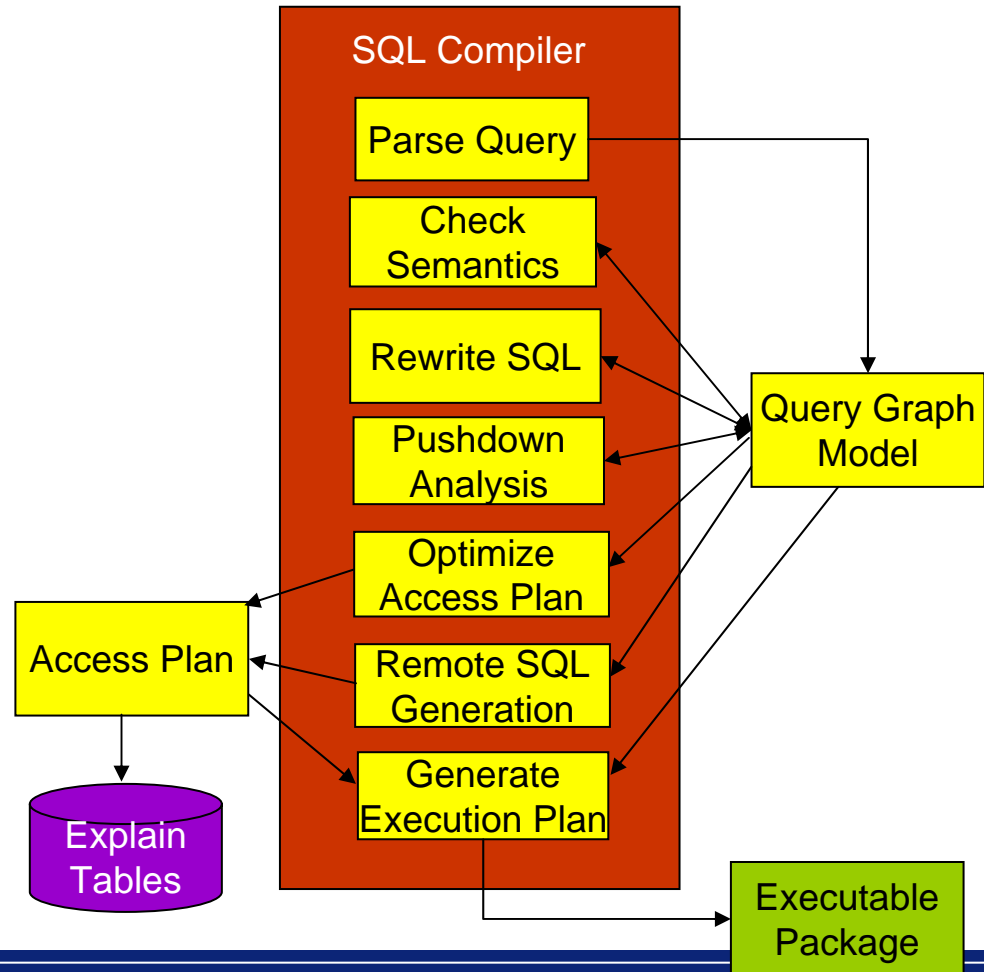
- Many ways to write a SQL to return the same data
- Small differences in coding SQL can have great performance implications
- Different SQL versions may produce different access plans

SQL Optimization

z/OS



L,U,W



Optimizer

z/OS

- Fixed optimization
- “HINTS” allow for some flexibility
 - Mainly used to maintain old access path
 - Across DB2 versions
 - After Rebinds
 - Must be turned on at install time
 - Need to modify PLAN_TABLE
 - Manual Process

L,U,W

- Much more flexible than z/OS
 - 7 levels of optimization
 - Adjusted based on query complexity

Optimizer Class

- DB2 Optimizer Class
 - Values are between 0 and 9, default is 5
 - Determines the intensity used by the DB2 SQL Compiler when rewriting SQL
 - Dynamic SQL can't spend time optimizing, use lower class
 - Static SQL optimizes once, use a higher class
 - "dft_queryopt" database setting
 - SET CURRENT QUERY OPTIMIZATION n

Level	Recommendation
0	Minimal amount of optimization. Only recommended for very simple SQL accessing well indexed tables. Only nested loop joins and IX scans enabled.
1	Similar to 0 except Merge Scan and TS scan enabled.
2	Recommended for very complex queries which are infrequently executed in a decision support or OLAP environment.
3	Closest to OS/390 optimizer. Recommended for queries with 4 or more joins.
5	DEFAULT – Most cost effective method for mix of simple and complex queries. Optimization will be automatically reduced for complex dynamic SQL if optimizer determines that the resources are not necessary.
7	Same as 5 except optimization not reduced for complex dynamic SQL
9	Used to determine whether more comprehensive optimization can generate better access plan for very complex long running queries using large tables

Optimization Tips – z/OS & LUW

- Make sure statistics are accurate
- Use stage 1 vs. stage 2 predicates
- Only select required columns
 - Avoid SELECT *
- Keep predicates as restrictive as possible
 - Minimize number rows returned
 - Minimize program filtering and let DB2 do the work
- Order predicates from most to least restrictive
- Avoid sorts
 - ASC/DESC indexes can help avoid excessive sorting
- Avoid UNION clause
 - IN, BETWEEN, or CASE more efficient
- Use Appropriate Optimizer Class (LUW)
 - 0 through 9 (5 default)
 - Use lower class for Dynamic SQL
 - Higher class for static
- Use “Optimize for n Rows”
 - Minimizes optimization cost
- Use “Fetch First n Rows”
- Avoid Data Type Conversions

EXPLAIN!!!

Tuning Examples

Optimizer Class Scenario - LUW

- **Scenario** – Complicated Join statement performing poorly (2 table join x 10 Unions)
- **Application** – Dynamic SQL
- **Optimizer Class** – 0
- **Possible Solution** – Up the Optimizer Class since preparation only occurs at bind time

Optimization Class	0	1	2	3	5	7	9
Total Cost (Timerons)	493416	148563	148563	148563	148563	148563	148563
I/O Cost	36933	23980	23980	23980	23980	23980	23980
Elapsed Time	0:0:1.234	0:0:0.891	0:0:0.828	0:0:0.703	0:0:0.750	0:0:0.796	0:0:0.797

35% improvement in elapsed time by bumping optimizer class up to 3

Union vs. IN vs. BETWEEN

- **Scenario** – Complicated Join statement performing poorly (2 table join x 10 Unions)
- **Application** – Dynamic SQL
- **Possible Solution** – Rewrite query with IN or BETWEEN clause

z/OS

	Processor Cost(ms)	Elapsed Time
UNION	25	0:0:0.266
BETWEEN	6	0:0:0.157
IN	12	0:0:0.141

Windows

	Total Cost (Timerons)	Elapsed Time
UNION	148563	0:0:0.797
BETWEEN	25875	0:0:0.328
IN	25832	0:0:0.297

**IN Clause performs best
on both platforms
+47% for z/OS
+62% for Windows**

Summary

- Build a strategy for how you will monitor/tune your DB2 environment
 - Try to identify most critical applications and begin focusing on those
 - Set realistic performance expectations
 - Tune, monitor, tune, monitor...

Quest Software: You Can Expect More



*For more information on Quest Software's
Solutions for DB2 and to download free trials, visit:*

www.quest.com/db2