Performance Tuning Batch Cycles and Table Partitioning

Robert Williams
One-point Solutions Inc.
rwilliams@one-point.com

March 11 2010
Platform: LUW
Agenda

- Introduction – A little about me
- About the DWP/EDW environment I work on
- Thoughts on compression
- The real overhead of monitoring
- Performance tuning the batch cycle
- Maintenance – When is it necessary and what options to use
- Statistical views
- Working with event monitors to help identify tuning opportunities
- db2advis a few thoughts
- A quick overview of MDC and MQT
- Table Partitioning
- Open Questions
A little about me

• Responsible for over 50 TB of online data
• idug.org webmaster
  • Use the site – thousands of presentations and over 450 hours of videos
  • Live broadcasting of a track this year from the conference for free
• Advanced Certified DB2 DBA / Certified DB2 Developer / Certified Stored Proc 9.5 Developer/ Certified Infosphere / Certified Websphere ND/ Certified Data Stage 8.0
• Working towards a certificate in mainframe computing from Ryerson University and PMP designation
Data Warehouse Platform

- Currently holds 42 TB of data over 6 physical nodes + 1 Backup
- 64 Processors and 192 GB of memory
- Currently holds:
  - EDW – Enterprise Data Warehouse
  - ELM – Expected Loss Modeling
  - ESI – Enterprise Systems Warehouse
  - HR Warehouse
  - ECD – Expected Credit Default Modeling
- Hundreds of disks
- Currently growing by 2-4 TB of data per quarter
- SAS for analytics
- AIX 5.3 – GPFS, Hitachi HDLM, Shared SAN, TSM
- Have had a lot of platform issues
- 2010 we have approx 19 projects on the go with an average cost of $920,000
Thoughts on compression

- Internal charges are $3.00 per GB per month for SAN storage plus 45 cents per GB to backup
- Reduced the size of our database by just under 10 TB
- Savings of over 360,000 a year
- Cause our system to reach saturation level
- Added an additional 8 CPUs (Still worth the investment)
- Our derivatives back end has seen a reduction of about 11% elapsed time and big savings on storage (generate a 100 GB a week of XML data stored in DB2)
  - Down to 52 GB a week
Where Do I Spend Time Performance Tuning

- Disable monitoring that is not needed
  - No use of any “tool” at TD for monitoring performance yet
  - We have several simple bash scripts executed by crontab
- Batch cycle performance tuning and training
  - At least at TD we have close to 0 performance tuning of our batch cycle
- Maintenance and statistics granularity
- Physical design
  - Indexing
    - Lots on idug.org in content and video section
  - DDL Design
    - Hardest to change but typically the best results
THE COST OF MONITORING
Review: Snapshots and Monitors

- **Database Snapshots**: a point-in-time view of monitor elements.
  - Examples:
    - Database snapshot
    - All Applications, (or specific application) snapshot
    - All Bufferpools (or specific bufferpool) snapshot
    - Locks snapshot, etc...

- **Database Monitor Switches**: determine amount of detail collected by snapshots.
  - BUFFERPOOL monitor
  - LOCK monitor
  - SORT monitor
  - STATEMENT monitor
  - TABLE monitor
  - UOW monitor

- When database monitor switches are OFF, select monitor elements in snapshots will contain no data (no collected)
Benchmark Environment Summary

- **Server**
  - WebSphere Application Server
    - Hardware
      - 2 CPU, 3.00 GHz
      - Memory: 3G
    - Software
      - OS: Windows 2003 R1 SP2
      - WAS 6.1
  - Database Server
    - Hardware
      - 2 CPU, 3.00 GHz
      - Memory: 4G
    - Software
      - OS: RedHat Enterprise Linux 4 (x86)
      - DB2 9.5.4

- **Clients**
  - 50 concurrent clients
  - Workload generated by WSWS
  - Benchmark Application - DayTrader
    - Trade 6
    - J2EE 1.4
    - Servlet 2.4, JSP 2.0
    - EJB 2.1
    - JMS 1.1
    - App Client 1.4
DB2 Snapshot Switches – CPU usage analysis

- Workload used for Analysis:
  - Run 50 Daytrader clients and compare CPU and memory increment across scenarios

- Baseline: No Switches enabled
  - Result establishes basis of comparison for Scenario 1 and Scenario 2

- Scenario 1: All switches enabled
  - CPU increased 16.27%
  - Memory increased 0%

- Scenario 2: All switches except statement enabled
  - CPU increased 7.5%
  - Memory increased 0%
DB2 Snapshot Switches – Individual CPU Usage

- Test scenario:
  - Run 50 Daytrader clients, turning only one switch one by one
  - Compare CPU and memory increase due to single switch enabled

- Result:
  - “Statement” switch has largest performance impact on DB2 server. (CPU increase is 11.0%)  
  - “UOW” switch has 2nd largest performance impact on DB2 server (CPU increase is 3.75%)
  - Total CPU increase for other five switches cumulative: 3.75%
Performance Test with ITM DB2 agent

- Test scenarios:
  - running 50 Daytrader clients, start the agent, comparing the CPU and memory increment
  - No situation and historical collection is enabled.
  - No query is issued.
- Test result:
  - Running ITM DB2 agent in minimum steady state had little performance impact on DB2 server
    - 0.08% CPU increment
    - 0% memory increment
  - Working on perf benchmarks to measure situation, historical collection and sample queries for the upcoming release.
Additional questions we still have

• From the IBM doc
  • Recommendation: Any switch (except dft_mon_timestamp) that is turned ON instructs the database manager to collect monitor data related to that switch. Collecting additional monitor data increases database manager overhead which can impact system performance. Turning the dft_mon_timestamp switch OFF becomes important as CPU utilization approaches 100%. When this occurs, the CPU time required for issuing timestamps increases dramatically. Furthermore, if the timestamp switch is turned OFF, the overall cost of other data under monitor switch control is greatly reduced.
  • Extra CPU does not equal elapsed time. Home much longer can we expect our queries to run as a rule of the monitor switches
BATCH CYCLE
PERFORMANCE TUNNING
Load vs Online Load vs Import

**Offline Load**
- Prevents read/write access to table
- Fastest way to load data with the lowest resources used
- Several orders of magnitude faster than import

**Online Load**
- Same performance as offline load
- Puts the table in read only mode
- If merging indexes in same tablespace only (1-3)% slower than offline load

**Import**
- Basically bulk SQL insert
- Slow
- Holds locks on new data
- Can cause issues with table scans
- Watch commitcount
### IXF vs ASC vs DEL vs CURSOR

- A lot of the time people just choose DEL (CSV) for export and import operations

<table>
<thead>
<tr>
<th>Load from Cursor</th>
<th>PC/IXF Format</th>
<th>ASC</th>
<th>DEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Fastest way to move/transform data between tables&lt;br&gt;• Quicker than INSERT select and avoids the lock overflow issue&lt;br&gt;• No export needed&lt;br&gt;• Can pull directly from other DBMS both DB2 and non DB2 with federation (check license on box)</td>
<td>• Easily 40% quicker to load if not I/O bound compared to ASC and DEL&lt;br&gt;• Much lower CPU requirements&lt;br&gt;• Binary data format (hard to read)&lt;br&gt;• Works for exchange&lt;br&gt;• Can only import file on z/OS db2</td>
<td>• Non delimited ASCII file – IE column based separation&lt;br&gt;• Great for compatibility between different operating systems and database systems&lt;br&gt;• Very similar performance to DEL</td>
<td>• Delimited ASCII file&lt;br&gt;• 1, 2, 3, 4, 5&lt;br&gt;• Great for compatibility between different operating systems and database systems&lt;br&gt;• Very similar performance to ASC</td>
</tr>
</tbody>
</table>
INPUT

1. `emp.idx`
   - 10 ~ 1
   - 20 ~
   - 30 ~ 3
   - 30 ~ 4
   - 40 ~ x
   - 50 ~ 6
   - 50 ~ 7
   - 80 ~ 8

OUTPUT

2. `employee`
   - 10 ~ 1
   - 30 ~ 3
   - 50 ~ 6
   - 80 ~ 8

3. `empexp`
   - 30 ~ 4 `timestamp` `msg`
   - 50 ~ 7 `timestamp` `msg`

Exception Table

Rows not loaded last column does not comply with `NOT NULL` and `NUMERIC` requirements
Optimal load parameters and throughput

<table>
<thead>
<tr>
<th>File</th>
<th>Load Options</th>
<th>GB / h / CPU</th>
<th>Seconds</th>
<th>Rows/Sec</th>
<th>Improv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXF</td>
<td>ANYORDER, DATA BUFFER 40000, CPU_PARALLELISM 5, DISK_PARALLELISM 8</td>
<td>50.8±0.2</td>
<td>391±1</td>
<td>0</td>
<td>127900±30</td>
</tr>
<tr>
<td>ASC</td>
<td>ANYORDER, FASTPARSE, DATA BUFFER 40000, CPU_PARALLELISM 6, DISK_PARALLELISM 8</td>
<td>29.2±0.1</td>
<td>872±3</td>
<td>57300±200</td>
<td>36%</td>
</tr>
<tr>
<td>DEL</td>
<td>ANYORDER, FASTPARSE, DATA BUFFER 40000, CPU_PARALLELISM 6, DISK_PARALLELISM 8</td>
<td>28.0±0.1</td>
<td>912±1</td>
<td>54800±60</td>
<td>29%</td>
</tr>
<tr>
<td>CURSO R</td>
<td>ANYORDER, DATA BUFFER 40000, CPU_PARALLELISM 2, DISK_PARALLELISM 8, INTRA_PARALLEL ON, DFT DEGREE 18</td>
<td>14.0±0.1</td>
<td>996±7</td>
<td>50200±300</td>
<td>14%</td>
</tr>
</tbody>
</table>
Fastparse and Anyorder

Fastparse
• This parameter reduces syntax checking on the input data
  • DB2 guarantees the table will be architecturally correct but if you send in garbage data the table will be filled with garbage

Anyorder
• Load utility will not maintain the input source record order when writing data pages. Is your data cluster sensitive?

Performance degradation observed by not using FASTPARSE and ANYORDER

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IXF</th>
<th>DEL</th>
<th>ASC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fastparse</td>
<td>N/A</td>
<td>18%</td>
<td>16%</td>
</tr>
<tr>
<td>Anyorder</td>
<td>22%</td>
<td>13%</td>
<td>14%</td>
</tr>
</tbody>
</table>
Summary of DPF Load Options and Performance

<table>
<thead>
<tr>
<th>Partitioned Load Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Mode</td>
</tr>
<tr>
<td>PARTITION_ONLY + LOAD_ONLY (ASC)</td>
</tr>
<tr>
<td>ANYORDER, FASTPARSE</td>
</tr>
<tr>
<td>PARTITION_AND_LOAD (ASC)</td>
</tr>
<tr>
<td>ANYORDER, FASTPARSE</td>
</tr>
<tr>
<td>PARTITION_AND_LOAD (CURSOR)</td>
</tr>
<tr>
<td>ANYORDER, INTRA_PARALLEL ON, DFT_DEGREE 6</td>
</tr>
<tr>
<td>LOAD_ONLY_VERIFY_PART (IXF)</td>
</tr>
<tr>
<td>ANYORDER</td>
</tr>
<tr>
<td>LOAD_ONLY_VERIFY_PART (IXF + INDEX)</td>
</tr>
</tbody>
</table>
Collecting Statistics With Load

- Don’t collect statistics after every load unless there is a significant change in cardinality (# of rows)
  - Setup weekly/month jobs – Disable auto runstats
- Collecting statistics with load is more efficient as it doesn’t require an additional scan of data

<table>
<thead>
<tr>
<th>Statistics option</th>
<th>IXF</th>
<th>ASC</th>
<th>DEL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (s)</td>
<td>Degr</td>
<td>Time (s)</td>
</tr>
<tr>
<td>Yes</td>
<td>1259 + -5</td>
<td>-222%</td>
<td>1269 + -4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes with distribution</td>
<td>1654 + 28</td>
<td>-323%</td>
<td>1667</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How To Run Load With Runstats

- Developers find it confusing as the argument is STATISTICS
  - USE PROFILE
  - LOAD FROM test.ixf of IXF INSERT INTO test.test STATISTICS
    USE PROFILE
  - If no profile exists a warning will be issued

- How to setup a profile
  - RUNSTATS ON TABLE test.test SET PROFILE ONLY

- How to check the current profile
  - SELECT STATISTICS_PROFILE FROM SYSCAT.TABLES
    WHERE TABNAME = tablename
THINK Before You Write Your Runstats Statement

• If a table is used for OLTP, 1-1 relationships, has primary and foreign keys defined, and proper indexing, you probably do not need it with distribution. NO SKEW
  • Several people use this for every runstats statement. We always want our batch cycles to be as short as possible.

• 1-N or N-N relationships need with distribution.

• Really consider TABLESAMPLE of a few percentage unless there is serious skew to be worried about on large tables.
Why You Probably Want Table Sample on Weekly Maintenance

- As projects grow we see batch cycles getting tighter and tighter. Over time, many things will be overlooked.
- `SELECT SUM(sales) / 0.02 FROM transactions TABLESAMPLE SYSTEM(2);`
Automatic Statistics Profiling

- Great feature if you are unsure about the profile you should be using - DB2 will generate it for you!
  - Doesn’t work in DPF/MPP, Federation and Intra-partition parallelism
  - Less suitable for environments that are typically transactional

- How to use it
  - Set AUTO_STATS_PROF and AUTO_PROF_UPD to on in test systems during the PAT release.
    - Run a production workload or realistic tests
    - Transfer profile to production on deployment
Generated Always and the Load Overhead (NP)

- Identity column maintenance is expensive, as the processing agent needs to go to the system catalog to retrieve the next value
- Please, PLEASE do not add a surrogate key on child relationships and dimensions

<table>
<thead>
<tr>
<th>Generated column definition</th>
<th>Degradation</th>
<th>Degradation IXF</th>
<th>Projected Degradation DEL &amp; ASC</th>
</tr>
</thead>
<tbody>
<tr>
<td>C24 INTEGER GENERATED ALWAYS AS (C1+C22)</td>
<td>38 sec</td>
<td>(9.7 ± 0.4)%</td>
<td>4%</td>
</tr>
<tr>
<td>C48 BIGINT GENERATED ALWAYS AS (C22 + CAST(C5 AS BIGINT) + C9 + ASCII(SUBSTR(C8, 9, 1)) + LENGTH(C42) + DAYS(C35))</td>
<td>182 sec</td>
<td>(47 ± 1)%</td>
<td>18%</td>
</tr>
<tr>
<td>C1 INTEGER GENERATED ALWAYS AS IDENTITY</td>
<td>1024 sec</td>
<td>(262 ± 9)%</td>
<td>102%</td>
</tr>
</tbody>
</table>
Generated Always and the Load Overhead (P)

- As identity columns cannot be calculated on the partition loading, additional requests must flow between the partition and the catalog. Hence the detrimental effect on performance.

Cost of evaluating generated column expressions Partitioned Load

<table>
<thead>
<tr>
<th>Generated column definition</th>
<th>Degradation</th>
</tr>
</thead>
<tbody>
<tr>
<td>C24 INTEGER GENERATED ALWAYS AS (C1+C22) in partitioning key</td>
<td>120 sec</td>
</tr>
<tr>
<td>C24 INTEGER GENERATED ALWAYS AS (C1+C22) outside of partitioning key</td>
<td>60 sec</td>
</tr>
<tr>
<td>C1 INTEGER GENERATED ALWAYS AS IDENTITY in partitioning key</td>
<td>1909 sec</td>
</tr>
</tbody>
</table>
| C1 INTEGER GENERATED ALWAYS AS IDENTITY outside of partitioning key | 990 sec   | (12.5 ± 0.3)%
|                                                                  | (6.2 ± 0.2)% |
|                                                                  | (198 ± 2)%  |
|                                                                  | (103 ± 1)%  |
MDC Load Performance

• Make sure your data is presorted
  • If you have over 512 cells you will notice a significant slowdown when randomized.

• If data is not random but over a small set of cells you will still notice a slowdown of roughly 70%. (ASC DEL)
COPY YES Is it really worth it to break point in time recovery?

• The additional processing overhead of COPY YES has negligible costs.
  • This has been validated by running large loads with the copy to parameter set to /dev/null.
• Simply delete the load copy when you do not need point in time recovery.
• DRT will be fighting any new deployments without COPY YES.
ALTER TABLE NOT INITALLY LOGGED

• Do not do this for performance reasons
  • Logging does not add much of a performance overhead if you are not on a shared disk configuration
• Insert 5,000,000 logged vs not logged
  • 2.8% ± .4% elapsed time increase over 7 runs
Why Are We Putting Our Tables Offline For Loads?

- **Bulk loads are mainly used for reporting/data warehouse/DSS.**
- The tables involved are typically loaded during an overnight batch cycle.
- We typically have access to previous data (day or month) earlier.
  - Why do we wait until night time for batch cycles?
  - Online load has close to 0 performance overhead.
  - LOAD can be throttled to ensure no user impact.
- Performance overhead:
  - IXF (-0.5±0.6)%
  - DEL (0.3±1.2)%
  - ASC (-0.2±2.3)%
- Performance of the online rebuild index mode is also negligible provided the index is being rebuilt in the same tablespace where the indexes for the table currently reside.
Dual Logging on a Separate SAN/Disk and File System

- A new requirement as a result of an audit is that we make use of dual logging.
- Make sure they are on different systems.
- As long as both hard drives are fast there should be close to 0 overhead.
- We had serious whole sale back end issues that could have been prevented if this was in place.
Type 2 Slowly Changing Dimensions

• Right now we are running multiple queries to maintain type SCD. Using the merge statement permits you to reduce time between staging and SCD by 60-70%.

• Look at the merge statement so you can execute it in a single pass.

MERGE INTO archive ar
USING (SELECT activity, description FROM activities) ac
ON (ar.activity = ac.activity)
WHEN MATCHED THEN
UPDATE SET
description = ac.description
WHEN NOT MATCHED THEN
INSERT
(activity, description)
VALUES (ac.activity, ac.description)
AIX Parameters

- tcp_nodelayack – disable delay of acknowledgements
  - Default: 0
  - Recommend: 1
    - TCP deliberately delays sending acknowledgements by up to 200ms to try a piggyback
    - Packet acknowledgements with response packet are for large POST requests (multimegabyte)
    - This can cause low data transfer rates due to 150-200ms delays between packets.
MAINTENANCE
To Much Maintenance in Batch Jobs

• General problem I see is too much maintenance in batch cycles
• I can’t count how often we have load, reorg, and runstats on every job; this is pointless.
• Too much maintenance doesn’t make the database go faster it just consumes more batch time, CPU cycles, and power.
• Consider doing weekly maintenance outside of your batch cycle.
  • Cut several hours out of batch cycle by doing this
Online vs Offline reorg

Offline
- Much faster
- CAN PUT IT IN READ ONLY
- If you have a failure might need to restore and rollforward
  - Put tables into their own tablespace
- Indexes get rebuilt
- Use of temporary tablespace
- Supports all types of tables
- Compression
- Support for MDC tables
- Long and LOB tables

Online
- Slower -> Permits full r/w although it can take up to 10-20 times longer
- Entire table is not reorganized at once nor copied to temp space
- Just shuffled around
- Big logging issue if data is out of order of clustering index (up to 2.5 times table)
- No index cleanup
- Reclaims free space in phase 4
- Online table compression (9.7)
- Recoverable
- Imperfect clustering
- Can pause and restart
### When to do maintenance

<table>
<thead>
<tr>
<th>Action</th>
<th>runstats</th>
<th>Runstat</th>
<th>Runstats with distribution</th>
<th>Online Reorg</th>
<th>Offline Reorg</th>
<th>Index Reorg Rebind</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load a table</td>
<td>No</td>
<td>&gt;10%</td>
<td>No</td>
<td>No -pre sort</td>
<td>No -pre sort</td>
<td>No</td>
</tr>
<tr>
<td>Import</td>
<td>&gt;10%</td>
<td>N/A</td>
<td>large joins or skew</td>
<td>No -pre sort</td>
<td>No -pre sort</td>
<td>No</td>
</tr>
<tr>
<td>Insert row</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Delete row</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Update Row</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Delete &gt; 10%</td>
<td>Yes</td>
<td>N/A</td>
<td>large joins or skew</td>
<td>Won't clean up index -avoid if possible</td>
<td>No -pre sort</td>
<td>Yes</td>
</tr>
<tr>
<td>Insert &gt; 10%</td>
<td>Yes</td>
<td>N/A</td>
<td>large joins or skew</td>
<td>Only if not in cluster order</td>
<td>No -pre sort</td>
<td>No</td>
</tr>
<tr>
<td>Update &gt; 10%</td>
<td>Yes</td>
<td>N/A</td>
<td>large joins or skew</td>
<td>Only if change in cluster column</td>
<td>No -pre sort</td>
<td>No</td>
</tr>
<tr>
<td>OLTP Data warehouse</td>
<td>Yes</td>
<td>N/A</td>
<td>No (joins or skews)</td>
<td>If reorogchk says to</td>
<td>Never</td>
<td>Never</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Yes on weekend</td>
<td>Weekends</td>
<td></td>
<td>Weekends</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>N/A</td>
<td>No</td>
<td>No</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Backup Compression

- It’s now included in the license
- Reduces size by 20%-50%
- Even though we have a very fast etherchannel back end it still reduced total elapsed time by about 11%
STATISTICAL VIEWS
What statistics are captured by RUNSTATS

- I have had to learn to get good with this – Otherwise I have to justify the cost of several thousands of dollars a year

- Which statistics are captured during runstats?
  - How many rows exist/pages/active blocks in table
    - For each column in a table, optionally records
      - Number of data values, avg. length of data values, data range information
  - WITH distributions
    - Top 10 quantities
    - Consider a population of 10 data values \{3, 6, 7, 8, 10, 13, 15, 16, 20\}.
    - Top 10 highest repeated values
  - Multi-column statistics
  - Index clustering (DETAILED index statistics)
    - Column group statistics
      - Better estimate data page fetches \rightarrow\text{determinates}\ curve of I/O vs buffer size
      - Accounts for benefit of large buffers

- Ok but how can DB2 estimate the result set size for...?
  - Complex Predicates
    - WHERE NAME LIKE ‘%ROB%’
  - Relationships among complex predicates
    - WHERE PRODUCT = ‘DVD’ and PRICE < ‘15.00’
    - With distributions on all columns
  - Relationships across tables
    - SELECT ..., FROM CUSTOMER C, FACT F
      WHERE C.NAME = ‘ROB’ AND C.CUST_ID = F.CUST_ID
SELECT r.consumer_id, r.reward_amount, r.reward_uuid,
s.chain_location_id, s.store_uuid, r.create_date,
r.volume_limit, r.expiration_date
FROM enterprise.reward r, enterprise.store s
WHERE r.reward_type = 1 AND r.reward_availability = 1 AND s.store_uuid =
r.store_uuid AND com_reward_uuid IS NULL
Estimated Cost = 45785.304688
Estimated Cardinality = 639.102234
Optimizer Plan:

RETURN
    (1)
     |
HSJOIN
    (2)
      /_________
      \_______/
     TBSSCAN    NLJOIN
    (3)         (4)
       |         /_________
       |         \_______/
Table: FETCH     IXSCAN
ENTREPRISE    (---)
STORE          (4)
RIDSCN Table: Index: Table:
    (6) ENTERPRISE         ENTERPRISE
| P.REWARD            SQL080221174306020
| P.SALE_TRANSACTION
| SORT
    (7)
    |
IXSCAN
    (8)
    /\                     /\
Index: Table: ENTERPRISE ENTERPRISE
P.RE_OEUUUD_IDC P.REWARD

SELECT r.create_date, r.consumer_id, r.reward_amount, r.reward_uuid,
s.chain_location_id, s.expiration_date
FROM enterprise.reward r, enterprise.store s
WHERE r.reward_type = 1 AND r.reward_availability = 1 AND s.store_uuid =
r.store_uuid AND com_reward_uuid IS NULL
Estimated Cost = 7397.978516
Estimated Cardinality = 38.130344
Optimizer Plan:

RETURN
    (1)
     |
HSJOIN
    (2)
      /_________
      \__________
     TBSSCAN    NLJOIN
    (3)         (4)
       |         /_________
       |         \__________
Table: FETCH     IXSCAN
ENTREPRISE    (---)
STORE          (4)
RIDSCN Table: Index: Table:
    (6) ENTERPRISE         ENTERPRISE
| P_RE_OEUUUD_IDC P_REWARD
| SQL080221174306020 P_SALE_TRANSACTION
| SORT
    (7)
    |
IXSCAN
    (8)
    /\                     /\
Index: Table: Index: Table:
ENTREPRISE ENTERPRISE ENTERPRISE ENTERPRISE
P_RE_OEUUUD_IDC P_REWARD REWARD_AVIL P_REWARD
Statistical Views

- Statistics are associated with the view and stored in the catalog.
- Query does not need to reference the view to make use of the statistics.
- This matching is done via the code base that was used for MQT rewriting.
- The view is not materialized.
- The views help improve the selectivity estimates for predicates in the query.
Enable a View for Optimization

- **ALTER VIEW STATEMENT**
  - Enable a view for optimization:
    - `ALTER VIEW AML_CORE.THIS_VIEW ENABLE QUERY OPTIMIZATION`
  - Disable a few for optimization
    - `ALTER VIEW AML_CORE.THIS_VIEW DISABLE QUERY OPTIMIZATION`
  - **RUNSTATS** command
    - `RUNSTATS ON TABLE AML_CORE.THIS_VIEW WITH DISTRIBUTION`
The view created to inform the optimizer

CREATE VIEW TEST AS

SELECT * FROM ENTERPRISE.P_REWARD,
 ENTERPRISE.P_SALE_TRANSACTION

WHERE _P_REWARD.SALE_TRANSACTION_IDC =
 P_SALE_TRANSACTION.SALE_TRANSACTION_IDC

• Which statements will use the new meta information
  • SELECT * FROM ENTERPRISE.P_REWARD,
    ENTERPRISE.P_SALE_TRANSACTION WHERE
    _P_REWARD.SALE_TRANSACTION_IDC =
    P_SALE_TRANSACTION.SALE_TRANSACTION_IDC AND ID = 1

  • SELECT * FROM ENTERPRISE.P_REWARD,
    ENTERPRISE.P_SALE_TRANSACTION, ENTERPRISE.P_SALE_ITEM
    WHERE _P_REWARD.SALE_TRANSACTION_IDC =
    P_SALE_TRANSACTION.SALE_TRANSACTION_IDC AND ....

  • SELECT * FROM ENTERPRISE.P_REWARD,
    ENTERPRISE.P_SALE_TRANSACTION WHERE ID = 1
DB2 Design Advisor

db2advis -d sample -i sql.in -t 5 -m MICP -k HIGH -l 50

- Input can be one of:
  -s "sql stmt" | -i <infile> | -w <workload> | -g | -qp

- Example of input file called sql.in:

```
--#SET FREQUENCY 100
SELECT COUNT(*) FROM EMPLOYEE;
SELECT * FROM EMPLOYEE WHERE LASTNAME='HAAS';
--#SET FREQUENCY 1
SELECT AVG(BONUS), AVG(SALARY) FROM EMPLOYEE
     GROUP BY WORKDEPT ORDER BY WORKDEPT;
```

- MDC requires data in table
- Partitioning requires DPF
db2advis -d CRSLE -t 3 -s " SELECT lmeadm.audit_log.user_id ,
timestamp(lmeadm.audit_log.change_date_time)
as change_date_time FROM lmeadm.audit_log
WHERE lmeadm.audit_log.table_name = 'movement' AND
lmeadm.audit_log.primary_key_values = ? AND
lmeadm.audit_log.company_id = ? AND
lmeadm.audit_log.changed_values LIKE
 '%status|A|P%' ORDER BY
lmeadm.audit_log.change_date_time FETCH FIRST 1 ROWS ONLY"

execution started at timestamp 2006-12-15-07.36.31.246857
 Recommending indexes...
 total disk space needed for initial set [2535.966] MB
 total disk space constrained to [18569.313] MB
 Trying variations of the solution set.
 Optimization finished.
 1 indexes in current solution
 [46561.2773] timerons (without recommendations)
 [ 30.6148] timerons (with current solution)
 [99.93%] improvement
-- LIST OF RECOMMENDED INDEXES
-- =============================
-- index[1], 2535.966MB
CREATE INDEX "DB2INST1"."IDX612151436400000" ON "LMEADM"."AUDIT_LOG"
   ("PRIMARY_KEY_VALUES" ASC, "TABLE_NAME" ASC, "COMPANY_ID" ASC)
   ALLOW REVERSE SCANS ;
COMMIT WORK ;
RUNSTATS ON TABLE "LMEADM"."AUDIT_LOG" FOR INDEX
   "DB2INST1"."IDX612151436400000" ;
COMMIT WORK ;
--
-- RECOMMENDED EXISTING INDEXES
-- =============================
-- RUNSTATS ON TABLE "LMEADM"."AUDIT_LOG" FOR INDEX "LMEADM"
 -- "X_AUD_DSP_OPT" ;
-- UNUSED EXISTING INDEXES
-- ============================
-- DROP INDEX "LMEADM"."X_AUD_AUDIT.Sort" ;
7 solutions were evaluated by the advisor
DB2 Workload Performance Advisor tool is finished.
2010 IDUG North America

- .Net/Visual Basic
- Access Path Mgmt
- App Architecture
- App Dev Informix
- App Dev LUW
- App Dev z/OS
- App Dev
- Application Design for LUW
- Application Design for z/OS
- Application Developer
- Architect
- Archiving
- Auditing
- Business Intelligence
- Business Intelligence/Data Warehousing
- Connectivity
- DBA Informix
- DBA LUW
- DBA z/OS
- Data Architecture
- Data Sharing
- Data Warehousing
- Database Administration for LUW
- Database Administration for z/OS
- Disaster Recovery
- ERP
- For IT Managers

- For New Users
- Heterogeneous Env.
- High Availability
- IMS
- Information Integration
- Informix
- Java
- Java & Web
- LINUX
- Manager
- New Release of DB2
- New Release of DB2 z/OS
- New Release of Informix
- New Release of LUW
- New to DB2
- New to Informix
- Performance - DB
- Performance - SQL
- Performance - System
- Platform Transition
- Replication
- SOA
- SQL Features
- Security
- Sysprog
- Systems Programmer
- Systems Programming
- Tools and Utilities
- Triggers/SPs/UDFs
- WebSphere
USING EVENT MONITORS AND DB2PD TO FIND POOR RUNNING SQL
If you have a good PAT environment or can afford a 20% slow down

- CREATE EVENT MONITOR EVM_STMT FOR STATEMENTS WRITE TO TABLE
- This records the a large number of metrics for every statement that gets executed
- You can then query the table evm_stmt_stmt to find high resources statements
- Make use of GROUP BY and strip out predicates with perl regular expression
- SELECT * FROM EVM_STMT_STMT ORDER BY ROWS READ
- Look at index on those statements
What information does stmt evmt give?

Consistency Token : AAAAAANBX
Package Version ID :
Cursor : SQLCUR201
Cursor was blocking: FALSE
Text : select nominationid from nomination where nomdate < current

Start Time: 04/14/2008 22:14:12.416802
Stop Time: 04/14/2008 22:14:12.872888
Elapsed Execution Time: 0.052746 seconds
Number of Agents created: 1
User CPU: 0.015625 seconds
System CPU: 0.031250 seconds
Statistic fabrication time (milliseconds): 0
Synchronous runstats time (milliseconds): 0
Fetch Count: 1210
Sorts: 0
Total sort time: 0
Sort overflows: 0
Rows read: 1210
Rows written: 0

Internal rows deleted: 0
Internal rows updated: 0
Internal rows inserted: 0
Bufferpool data logical reads: 0
Bufferpool data physical reads: 0
Bufferpool temporary data logical reads: 0
Bufferpool temporary data physical reads: 0
Bufferpool index logical reads: 0
Bufferpool index physical reads: 0
Bufferpool temporary index logical reads: 0
Bufferpool temporary index physical reads: 0
Bufferpool xda logical page reads: 0
Bufferpool xda physical page reads: 0
Bufferpool temporary xda logical page reads: 0
Bufferpool temporary xda physical page reads: 0
What happens if you can’t take the performance hit

- You can make use of the SYSIBMADM.SNAPSTMT view in a cronjob
- `SELECT SUBSTR(STMT_TEXT,1,30) AS STMT_TEXT, ROWS_READ, ROWS_WRITTEN, STMT_OPERATION FROM SYSIBMADM.SNAPSTMT`
- Still might take 10%+ overhead + latching
- If you can’t take that use `db2pd --apinfo` and `db2explain` or explain the dynamic statement cache
- `db2pd -latches`
A QUICK OVERVIEW OF MDC AND MQT
A quick note on the licensing

• If you have V8 it was included in the license and you can still use it as long as you remained in support
• In V9 it was the performance optimization feature that you could get on workgroup and enterprise
• Now you can only use it with enterprise
• It was initially included across all the Infosphere line
• But now they got rid of the cheap versions of Infosphere
MQT – Materialized Query Tables

CREATE TABLE dba.PG_SALES_SUM AS (
    SELECT ti.id AS prodline, pg.id AS pgroup,
           loc.country, loc.state, loc.city,
           l.name AS linename, pg.name AS pgroup,
           YEAR(pdate) AS year, MONTH(pdate) AS month,
           t.status,
           SUM(ti.amount) AS amount,
           COUNT(*) AS count
    FROM cube.transitem AS ti, cube.trans AS t,
         cube.loc AS loc, cube.pgroup AS pg,
         cube.prodline AS l
    WHERE ti.transid = t.id
    AND ti.pgid = pg.id
    AND pg.lineid = l.id
    AND t.locid = loc.id
    AND YEAR(pdate) > 1990
    GROUP BY ti.id, pg.id, loc.country, loc.state, loc.city,
            year(pdate), month(pdate), t.status, l.name, pg.name
) DATA INITIALLY DEFERRED REFRESH DEFERRED;

SET CURRENT REFRESH AGE=ANY

SELECT YEAR(pdate) AS year,
       SUM(ti.amount) AS amount
FROM cube.transitem AS ti,
     cube.trans AS t,
     cube.loc AS loc, cube.pgroup AS pg,
     cube.prodline AS l
WHERE ti.transid = t.id
    AND ti.pgid = pg.id
    AND pg.lineid = l.id
    AND t.locid = loc.id
GROUP BY year(pdate);

REFRESH TABLE dba.SALES_CUBE;
We make very limited use in DWP of MQTs

- In the areas that we do use them we have had very good success with MQTs
- One major issue for us is the catalog locking during a refresh
  - Holds IX on syscat.tables preventing DDL, grants, and backups of SMS table spaces
  - We use SMS quite a bit
## Settings DEV vs PAT vs PROD

### DEV
- PAT is not DEV
- Tighter lock timeout
- Smaller bufferpools
- Smaller prefetchers in tablespace
- Smaller sort heap
- Make sure `capture_lock_timeout` is on
- STMM disabled
- Disable automatic maintenance

### PAT
- Before testing new release request refresh
- Settings must mirror production except
  - Much lower lock timeout
- Copy over latest statistics from production if subsample
- Make sure `capture_lock_timeout` is on
- Disable automatic maintenance

### PROD
- Make use of STMM unless you have a reason not
- Disable automatic maintenance
- Kept at 1-2 fix packs behind latest unless there exists a specific reason to go to the latest version
TABLE PARTITIONING
What is Table Partitioning?

- Allows a single logical table to be broken up into multiple separate physical storage objects
  - Partition boundaries are specified value ranges for partition key
  - Across One or Multiple Table Spaces

- Completely Transparent to Applications
  - Access by Column and Table Names
  - Not concerned where actual data resides

Without Partitioning

Annual Sales

With Partitioning

Applications see single table

SALES Jan Part
SALES Feb Part
SALES Mar Part
Table Partitioning Benefits

- Improve Query Performance with Partition Elimination
  - Optimizer is Data Partition Aware
    - Limits SQL access to relevant Partitions in “Where Clause”

- Allows for optimized roll-in / roll-out processing of Ranges
  - Partitions easily added/removed
    - Offline but quick (seconds)
  - No Data Movement / Near Instantaneous

- Optimize the management of huge tables
  - Unlimited in size
    - Limits are per partition (not table)
  - Place Ranges across multiple Table Spaces
  - Backup/Restore Ranges in separate Table Spaces

- Greater Index Placement Flexibility
  - Indexes can be stored in separate SMS Table Spaces
    - Table Partition Tables only
  - Can exploit large table spaces
DB2 LUW V9 Partitioning Terminology

Database Partitioning

- Distributing data by key hashing across logical nodes of the database (DPF)
- V9 “Partition Key” clause renamed “Distribute By”
- Create table Test
  (Account_Number integer
   ,Trade_date date)
  **Distribute by** (Account Number) **using hashing**

Table (Range) Partitioning

- Splitting data by key range over multiple physical objects within a logical database partition
- Create table Test
  (Account_Number integer
   ,Trade_date date)
in ts1,ts2,ts3
  **Partition by range**(Trade_date)
  (STARTING '1/1/2000' ENDING '3/31/2000',
  STARTING '4/1/2000' ENDING '6/30/2000',
  STARTING '7/1/2000' ENDING '9/30/2000')
DB2 LUW V9 Partitioning Terminology (Continued)

**Multi-Dimensional Cluster**

- Organizing data in table (or range of a table) by multiple key values (Cells)
- V9 “Partition Key” clause renamed “Distribute By”
- Create table order

(Account_Number integer
,Trade_Date date
,Region char(10) {such as north,south,east,west}
,order_month integer generated always as
month(order_dt))
in ts1

Organized by dimensions (region, order_month)
Table Partitioning with DPF and MDC

CREATE TABLE TestTable
(A INT, B INT, C INT, D INT ...)
IN Tablespace A, Tablespace B, Tablespace C
INDEX IN Tablespace B
DISTRIBUTE BY HASH (A)
PARTITION BY RANGE (B) (STARTING FROM (100) ENDING (300) EVERY (100))
ORGANIZE BY DIMENSIONS (C,D)
Partition Elimination: Table Scans

```sql
SELECT * FROM t1
WHERE
  year = 2001 AND month > 7
- Will only access data in table space tbsp3 and tbsp4

SELECT * FROM t1
WHERE
  A > 50 AND A < 150
- Will only access data in tbsp1 and tbsp2
```
Storage Mapping: Indexes are Global in V9/V9.5

- Indexes are *global* (in V9)
  - Index RIDs contain 2-byte partition ID
  - V9.7 – local indexes
- Each index is in a separate storage object than the table
  - By default, in the same table space as the first data partition
  - Can be created in different table space, via
    - `INDEX IN` clause on `CREATE TABLE` (default is table space of first partition)
- Recommendation
  - Place indexes in LARGE table spaces

```sql
CREATE TABLE t1(c1 INT, c2 INT, ...)
  IN tbsp1, tbsp2, tbsp3
INDEX IN tbsp4
PARTITION BY RANGE(a)
  (STARTING FROM (1) ENDING (100)
   EVERY (33));
CREATE INDEX i1(c1);
CREATE INDEX i2(c2) IN tbsp5
```
V9.7 Partitioned (Local) Indexes

- Will support the ability to create local indexes
- This will relieve current issues associated with Roll-In processing, mainly the global index maintenance and associated logging
- Allows for reorg table at the range partition level (not V9.7 GA)

- Improved ease of use with respect to range level compression
Roll-In Overview (with Table Partitioning)

- Build a new Table "NewMonthSales"
- Populate the new Table
- Drain Queries
- ALTER TABLE Big Table ...
  ATTACH PARTITION ...
  STARTING '03/01/2005'
  ENDING   '03/31/2005'
  FROM TABLE NewMonthSales

- Very fast operation
- No data movement required
- Index maintenance done later
- COMMIT
  - New data still not visible
- SET INTEGRITY FOR Big_Table.p3
  - Potentially long running operation
    - Validates data
    - Maintains global indexes, MQTs
    - Increase log space usage
    - V9.7 Partition Indexes
      - Reduce logging
      - Avoid Global Index Maintenance
  - Existing data available for Read/Write access while executing
  - COMMIT
    - New data visible
Roll-Out Overview (with Table Partitioning)

- Use DETACH to roll-out a range of data
  - Queries are drained and table Z locked by DETACH (sub-second)
  - Data disappears from view immediately upon DETACH commit
  - Index maintenance is done in the background asynchronously
    - Avoided with V9.7 Partition Index
  - Delete triggers do not fire for DETACH
  - Dependent MQTs go offline and can be refreshed incrementally via SET INTEGRITY
  - Rolled-out data is available in a new, separate table
  - Rolled-out data can be dropped, archived, moved to HSM

- DETACH syntax example:
  
  ```sql
  ALTER TABLE sales DETACH
  PARTITION part2 INTO jan00
  ```

Range partition becomes a stand-alone table
Asynchronous Index Cleanup after DETACH (Global Indexes)

- AIC is a feature to cleanup indexes
  - Low priority, throttled, background process
  - Reclaims space in index (keys corresponding to data rolled-out)
  - Automatically started when DETACH is committed (or after refresh of dependent MQTs)
  - Pauses if lock conflict with user activity
  - Hardens progress periodically; picks up where it left off after shutdowns
- SYSCAT.DATAPARTITIONS shows ‘I’ in STATUS while executing
- Not required for V9.7 Partition Index

```sql
$ DB2 LIST UTILITIES SHOW DETAIL

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Database Name</th>
<th>Partition Number</th>
<th>Description</th>
<th>Start Time</th>
<th>State</th>
<th>Invocation Type</th>
<th>Throttling</th>
<th>Priority</th>
<th>Progress Monitoring:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ASYNCH INDEX CLEANUP</td>
<td>WRNR</td>
<td>0</td>
<td>Table: T1, Index: I1</td>
<td>12/15/2005 11:15:01.978513</td>
<td>Executing</td>
<td>Automatic</td>
<td></td>
<td>50</td>
<td>Total Work: 5 pages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Completed Work: 0 pages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Start Time: 12/15/2005 11:15:01.980518</td>
</tr>
</tbody>
</table>

$ DB2 UTIL_IMPACT_PRIORITY FOR 3 TO 90
```
Design Considerations

- Which tables benefit from being range partitioned
  - Very Large tables
  - Roll-out requirements
  - Considerations for Roll-In
    - Consider MDC ranges with fine granularity (ex. Daily)
      - Less logging than SI
      - More predictable elapsed time
    - Queries accessing range subsets of a table
      - Partition Elimination

- Which column(s) to partition on
  - Dates (roll-out)
  - Facilitate Partition elimination

- Granularity of ranges should match roll-in/roll-out
  - If Ranges match your roll-out criteria
    - Detach avoids need for Reorg to reclaim space

- Consider placing different ranges into separate table space
  - Facilitate Back Up/Recovery
    - Can backup/restore an entire Range Partition in a separate tablespace
    - Separate Active from Historical data
Table Partition Design Considerations (continued)

- Set Integrity
  - Consider Logging Impact and Single UOW Impact of attaching large ranges
  - Elapsed Time and log requirements during SI depend:
    - Size of Partition
    - Number of Indexes
  - Avoid Index maintenance and logging with V9.7 Partition Index during SI

- DPF can help reduce the granularity of global indexes

- Specify Global Index placement in separate large tablespaces
  - Supported for DMS and SMS (with range partitioning only)
  - Default placement puts every index in tablespace with first range
  - Example
    - Create Index indexname on tablename in tabspc1

- Exploit V9.5 Automatic Compression Dictionary Creation
  - Occurs for Loads, Redistribute, Import or Inserts
  - Threshold to create dictionary varies – At least 700 KB
  - With V9, entire table Reorg needed to build a compression dictionary for a range (if not built prior to attach)
    - Inspect is another technique to build a dictionary if the dictionary does not already exist
Benefits of MDC for Roll-In/Roll-Out

- **MDC Roll-In**
  - Less Index I/O
    - Block Index updated only when block full
  - Faster Insert
    - No page splitting
    - Reuse existing empty blocks
  - Less Locking
    - Specify Create/Alter Table XXXX locksize blockinsert
    - Block Level (rather than row) Locking for Insert operations

- **MDC Rollout**
  - Less Logging – few bytes per page
    - Except when Refresh Immediate MQT exists – Row deletes with logging
  - Delete by page not row
    - Except when Refresh Immediate MQT exists – Row deletes with logging
  - May have faster elapsed time than Attach rollout with MQT (requires SI)
Typical Approach

- Application Requirements
  - Queries are frequently accessing a given day
  - Data is frequently added daily
  - Table Partitioning on Day would result in too many partitions
  - Rollout usually occurs weekly or monthly – typical reporting boundary
  - Continuous Updating

- Consider this Design Approach
  - Single Column MDC on Day
    - To facilitate Roll-In
  - Table Partition by Week or Month
    - Same time dimension but higher granularity
    - To facilitate Roll-Out

- Benefit:
  - Less Range Partitions
  - Faster Roll-In and Roll-Out
Index Clustering with Table Partitions

Consider prefixing clustered index with Table Partition Key or Correlated Key (partition by Quarter, index by Date)
- PARTITION BY RANGE (Month, Region)
  - CREATE INDEX ... (Month, Region, Department) CLUSTER

Not Clustered

Clustering Doesn't Match Partitioning

Clustering with Partition Key as Prefix

Range scans may require a working set of K pages

Range scans may require a working set of 3 pages

Range scans may require a working set of 1 page
Table Partition Elimination

- To benefit from partition elimination
  - Prefix Cluster key with Partition Key (prior slide)
  - Range Partitioning column frequently used in Where Clause
  - Leading columns of Composite Partition Key in where clause to get partition elimination
- Table Partition Keys can be a generated column
  - Generated Columns participate in Partition Elimination
- Partition Elimination works for Multiple Ranges
  - Ex. Where Jan or Dec
- Join predicates pushed down at run time
  - Used for Inner access of nested loop join only
- Parameter markers pushed down at run time when values are bound at Execution Time
  - Not reflected in Explain
Operational Considerations

- Failed load
  - Can’t drop partition table
  - Need to terminate load first

- Load Replace of a partition will replace the entire table (all ranges)
  - Specify Load Insert when appending data

- Explicitly Name Range Partitions
  - Easier to manage than system generated names

- Partition names cannot be reused after detach until Async Index cleanup completes

- Asynchronous Index Cleanup (not for V9.7 Partition Index)
  - Data is available during process
  - Detach data is not viewed

- Always issue Commit after Attach, Detach, SI, etc. to release locks
Table Partitioning Attachment Alternatives

- **Two Table Partitioning techniques to Roll-in Data:**
  - **Alter Attach** Partition and execute Set Integrity
    - Populate table offline, attach table and execute SI
    - **Advantage:**
      - Concurrent access – all previous partitions available for update
      - No partial data view (all or nothing)
    - **Disadvantage:**
      - Requires additional log space *(minimized with partition index)*
      - Long elapsed time *(minimized with partition index)*
      - Requires Drain of Queries
  - **Alter Add** Partition and Populate (faster but offline)
    - Attach empty table and then populate empty range partition (load or insert)
    - **Advantage:**
      - Faster Elapsed Time – Set Integrity not required
      - No log space for index maintenance
    - **Disadvantage:**
      - Partial Data View with Inserts (not Load)
      - Load allows Read Only access to older partitions
      - Compressed Data
      - V9 Need to Reorg entire table to build compression dictionary in new range tables (load or inserts will automatically build the compression dictionary in V9.5)
Partition Indexes Benefits (V9.7)

- Support for Partition Index for Range Partitioned Tables
  - Relieve issues associated with Roll-In processing
    - Reduce SET INTEGRITY time
    - Reduce Logging costs with Attach and Detach
    - Avoid Global Index Maintenance – Less I/O – Reduced Elapse Times
- Set Integrity performs following tasks:
  - Range validation
  - Global index
  - Partition index creation (if not already built prior to Attach)
  - MQT Maintenance
- Less locking for:
  - Index Scans and Concurrent IUD operations across partitions
- Partition Reorg much faster (V9.7)
  - Improve performance Drop Index and Create Index
- Avoid Asynchronous Index Cleanup for Detach
- Improve Query Performance for most queries
  - Benefit from reduce Index Levels – NON-DPF
Design Implications to exploit V9.7 Partition (Local) Index

- Non-Unique Indexes can be Partitioned
  - No restrictions
- Unique Indexes
  - Can be Partitioned only if a superset of Table Partitioning Key
    - (For DPF – superset of Data Base Partitioning Key)
    - Example:
      - Data Base Partition Key – Account_Num
      - Table Partition Key – Sales_Month,
      - Potential Unique Index - Account_Num, Sales_Month, Store_Num
  - Verify that uniqueness is required
    - Downstream data source may enforce Uniqueness
    - Uniqueness may not be required
    - May increase sorting
      - Distinct, Order By, Group By
- Unique partitioned indexes have to be created before attach
V9.7 Offline Partition Reorg Best Practice

- Set Integrity Rapid with Partition Indexes
  - Assuming No MQTs or Non-Partition (Global) Indexes

- Reorg of a table partition viable with partition indexes
  - Detach Table Partition
  - Reorg the table and indexes
  - Attach Table Partition
  - Run Set Integrity (fast) with partition indexes

- V9.7 Partition Reorg
  - Syntax:
    - Reorg Table YearData on Data Partition JuneMonth
  - Concurrent Read/Write Access only if no global indexes
Best Practice for Roll-In of V9 Compressed Range Partition Tables

- Technique to rapidly Attach Range with **V9 Compressed data only** (Non-Compression use Alter Add)
  - Load a subset (true random sample) of data into a separate DB2 table
  - Alter standalone table to enable compression
  - Reorg subset of data to build compression dictionary
  - Empty table (Import replace with /dev/nul) or Minimal data – (dictionary retained)
  - Alter Attach the table as a new range (dictionary retained)
  - Set integrity – Rapid with no/minimal data
  - Populate Data:
    - Batch Window
      - Load Data or Insert Data (compression will occur)
    - Continuous Update
      - Insert Subselect from Staging Tables or ETL (compression will occur)

V9.5 Easier Range Attachment with Compressed Data:
- Alter Attach the empty table with Set Integrity – Rapid with no data
  - Or Alter Add Partition
  - V9.5 Populate Data in Partition:
    - Load or Insert subselect provides Automatic Compression Dictionary Creation

V9.7:
- Load all data into source table and Compress Data
- Alter Attach the source table into target table
  - Rapid if Partition Indexes are implemented and Non-Partition Indexes Avoided
Best Practices for Roll-In of Table Partition

- Issue COMMITs after ATTACH, SET INTEGRITY
  - ATTACH Z locks table until committed
  - New data is not visible after SET INTEGRITY until committed

- SET CURRENT LOCK TIMEOUT statement
  - Prevent SET INTEGRITY from failing on lock conflict
  - Executable statement / Add to script before Set Integrity

- Plan for query draining by ATTACH
  - ATTACH won’t complete until it drains existing queries for the table
  - Meanwhile, no new queries can start

- Specify ALLOW WRITE ACCESS with SET INTEGRITY
  - The default is the old, offline behavior
  - Also available: ALLOW READ ACCESS

- Specify Exception Tables during Set Integrity

- Consider doing Roll-in and Roll-out together
  - ATTACH and DETACH in the same transaction minimizes the time that the table is unavailable
  - Use different partition name for attached partition
Best Practice for Roll-Out of Table Partition

- After Detaching a partition
  - Identify Dependent MQTs
    - MQTs are offline until Set Integrity executed
  - Check DB2 Catalog
    - SYSCAT.DATAPARTITIONS shows 'D' in STATUS field
      - For detached Data Partition
    - SYSCAT.TABLES shows 'L' for TYPE field
      - For Detached Table

- Consider execution of Set Integrity in one pass
  - Faster - One table scan
  - Longer UOW
  - Example
    - Set Integrity for MQT1, MQT2, MQT3 immediate checked
References

- References
- IDUG – Table Partitioning Best Practices NA 2009
- A few more will update it on the website
2010 IDUG North America

Rob Williams
rwilliams@one-point.com