SQL Performance for a Big Data 22 Billion row data warehouse

Dave Beulke – dave @ d a v e b e u l k e.com
Dave Beulke & Associates

Session: F19
Friday May 8, 2015 8:00 – 9:00 | Platform: z/OS

D a v e @ d a v e b e u l k e . c o m

- Member of the inaugural IBM DB2 Information Champions
- One of 45 IBM DB2 Gold Consultant Worldwide
- President of DAMA-NCR, Past President of International DB2 Users Group - IDUG
- Best speaker at CMG conference & former TDWI instructor

- Former Co-Author of certification tests
  - DB2 DBA Certification tests
  - IBM Business Intelligence certification test
- Former Columnist for IBM Data Management Magazine

Weekly Performance Tips:
www.DaveBeulke.com

- Consulting
  - CPU Demand Reduction Guaranteed!
  - DB2 Performance Review
  - DW & Database Design Review
  - Security Audit & Compliance
  - DB2 11 Migration Assistance
  - DB2 10 Performance IBM White Paper & Redbook

- Teaching Educational Seminars
  - DB2 Version 11 Transition
  - DB2 Performance for Java Developers
  - Data Warehousing Designs for Performance
  - How to Do a Performance Review
  - Data Studio and pureQuery

- Extensive experience in Big Data systems, DW design and performance
  - Working with DB2 on z/OS since V1.2
  - Working with DB2 on LUW since OS/2 Extended Edition
  - Designed/implemented first data warehouse in 1988 for E.F. Hutton
  - Working with Java for Syspedia since 2001 – Syspedia - Find, understand and integrate your data faster!

Big Data Era bends the SQL Rules

- **AGENDA**
  - The bigger the database the more attention needed to:
    - Standard SQL Rules (30+)
    - Database Design
    - Database Conditions
    - Database Indexes
    - DB2 SQL being used

Standard SQL Tips

1. **Only SELECT the columns** needed because it optimizes the I/O between DB2 and your program.
2. **Only SELECT rows needed** for the process or transaction and use
   FETCH FIRST x ROWS ONLY whenever possible.
3. **Code WHERE clauses with indexed columns** that have unique or good indexes.
4. **Reference only tables that are absolutely necessary** for getting the data.
5. **Code as many WHERE column clauses as possible** because the DB2 engine filters data faster than any application code.

Standard SQL Tips

6. **Prioritize the WHERE column clauses** to maximize their effectiveness. First code the WHERE column clauses that reference indexed keys, than the WHERE column clauses that limit the most data, than WHERE clauses on all columns that can filter the data further.

7. **Code SQL JOINs instead of singular SQL access whenever possible**. A single SQL JOIN is always faster than two SQL statements within an application program comparing and filtering the result set data.

8. **Code SQL tables JOINs when there is a common indexed column or columns shared** by both or all of the JOINed tables. If a common column is not indexed talk to a DBA and make a new index if possible.

9. **When coding JOINs make sure to code the most restrictive JOIN table first** and provide as many indexed and restrictive WHERE clauses as possible to limit the amount of data that needs to be JOINed to the second or subsequent tables.

10. **Use DB2 OLAP DB2 functions** to optimize your DB2 SQL answer result sets. Using these SQL DB2 OLAP functions is usually much faster than application code summing, totaling and calculations over your data.

Standard SQL Tips

11. **Leverage DB2 functions appropriately.** Analyze the SQL to determine the need for the various DB2 functions. Report writers generate SQL with multiple, duplicate or unnecessary DB2 functions. AVG, SUM etc..
12. **Examine the data columns and tables SELECTed in the SQL.** Many of these new big data report writers have nice GUI interfaces that allow the users to point and click on anything. Dup of #1?
13. **Verify the DB2 table joins for improving your DB2 SQL performance.** evaluate table indexes' uniqueness, cardinality, relationship to the table partitioning or column frequencies.
14. **Eliminate data translations.** Sometime the report writer generated SQL uses many CAST, SUBSTR, CASE statements, constants, or formulas in the SQL. The generated SQL may also have math formulas compared to DB2 table columns within the SQL. Check the ones that are not index-able and clean them for performance.
15. **Understand the ORDER of JOIN and Access.** By making sure the JOIN of the smaller table drives the data SELECTed from the big data DB2 table, your DB2 SQL should eliminate/minimize as much data as possible in the first JOIN. Also minimize SQL such as ORDER BY, GROUP BY, DISTINCT, UNION, or JOIN predicates that do not leverage the clustering order of the table, because they will require a DB2 sort.

Standard SQL Tips

16. **Understand all SQL phrases that results in a SORT.** DB2 SQL workloads Sorts are one of the most expensive operations especially with Big Data. i.e. - Distinct, Order by, UNION etc.

17. **Verify Scalar function is index-able.** In DB2 11 many scalar function became index-able not all of them like DATE(*timestamp-column*).

18. **Remove all math from the WHERE Clause** if possible. Math in WHERE clauses sometime disrupts the data type comparisons and precision. Pre-calculate math before SQL statement and always verify index-ability of any that are questionable.

19. **Understand the number of trips to the database**. Use multi-row SELECT and INSERT operations whenever possible to access or to the add the database.

20. **Minimize and optimize any WHERE ‘OR’ clauses**. Make sure that the ‘OR’ does not make the entire WHERE clause become a table scan. Optimize all ‘OR’ phases and turn them into a correlated sub-query if possible.

Standard SQL Tips

21. **SELECT from INSERT, UPDATE whenever possible**. Avoids multiple trips to the database and is great for identity, sequence and any other database column keys
22. **Use MERGE to reference the database** and make sure the data is there and updated. Avoid complex program logic to perform the data checking and then update.
23. **Create indexes on all WHERE clause expressions** that are not already index-able. Companies have unique situations and data types. Create Indexes on Expressions that are used in your daily SQL processing.
24. **Use Common Table Expression (CTEs) whenever possible** because they can get the data so that the result set can be used repeatedly and efficiently.
25. **Use DB2 TEMP tables for all data that is referenced more than three times** in the flow of daily business. It is more efficient to get the TEMP result set working instead of repeated redundant data access. Remember they can also be set up as NOT LOGGED.

Standard SQL Tips

26. **Use DB2 Virtual Indexes to model your potential Indexes** Model INCLUDE Column, Index on Expression, Uniqueness and other new index definition possibilities.

27. **Use Optimizer Filter Factor Hints to give full information.** JOIN Skew/Correlation, accurate statistics, complex predicates and host variable and special register information.

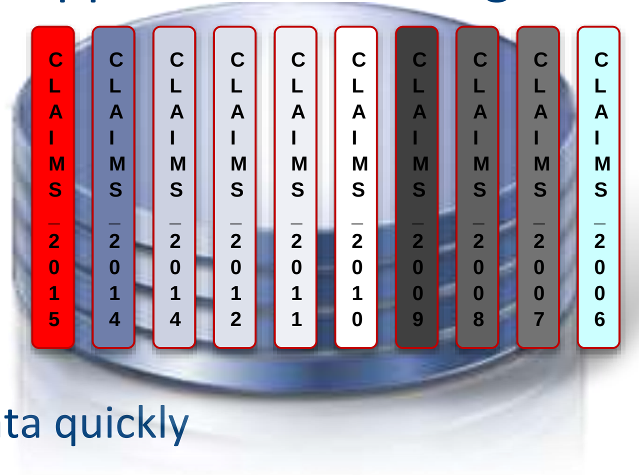
28. **Analyze the SYSSTATFEEDBACK table contents** to understand your SQL Access Path results better. Can be used to generate better RUNSTATS input.

29. **Leverage EXCLUDE NULL when creating indexes** to reduce index size. Verify that the NULL-ability is not within the application WHERE clauses.

30. **Overriding predicate selectivity with the DSN_PREDICAT_% tables.** Populate these tables with the details of the selectivity of the tables' uniqueness to improve optimizer knowledge for better access paths and improved performance.

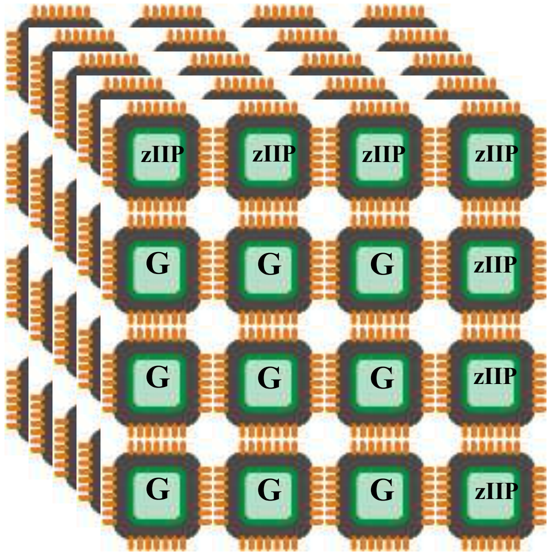
Big Data Database Design

- Big data physical objects I/O bound operations
  - Partition the database tables to minimize the data required for daily SQL
  - Use the thousands of partitions available for a design
    - How many parallel processes are your applications running today?



- Separate old data from new data
  - Current Year, Quarter, Week, Day
  - Temporal tables with the HISTORY tables
    - Complicates the SQL also can make a lot of data quickly
  - Materialized Query Table – YTD sales figures
    - Or composite tables to separate via TIME axis Year, Quarter, Week, Day
    - Or composite tables to separate via Sales territory axis Country, Region, State, City, Zip code

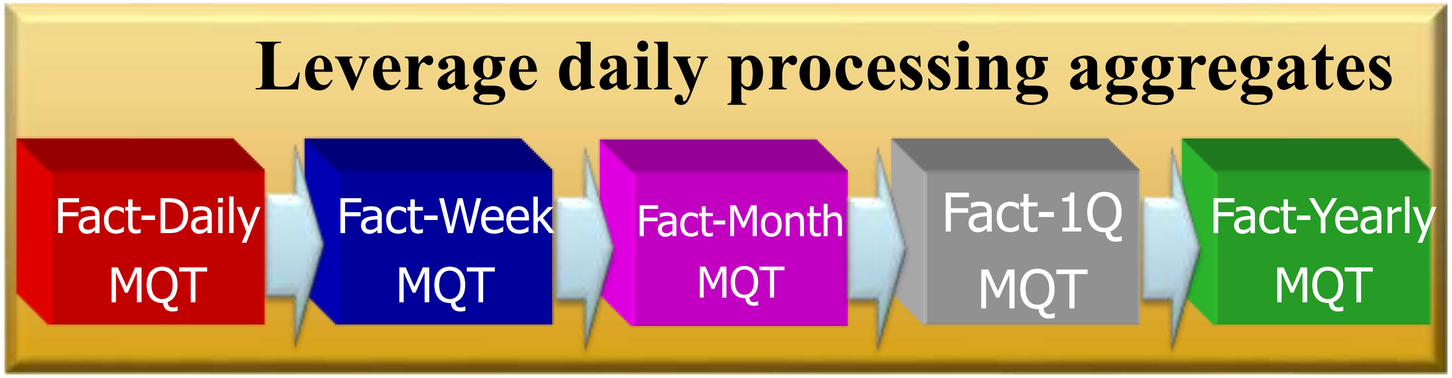
Database Design for parallelism



- Partitioning to leverage ??? more concurrent processes
  - Same partitioning limit keys across multiple table spaces
    - Via Customer number across those related tables
    - Via Product SKU number across all the product related data
- Partitioning design leverages time properly
  - The active partitions are only a segment of the entire table
  - Concentrates the I/O into the right sized portion of the database
  - Current history available - Ancient history is not in database/archived easily
- Indexes (NPIs or DPSIs) are appropriately designed
  - Partitioned for parallelism and recovery
  - Table clustered for SQL efficiencies

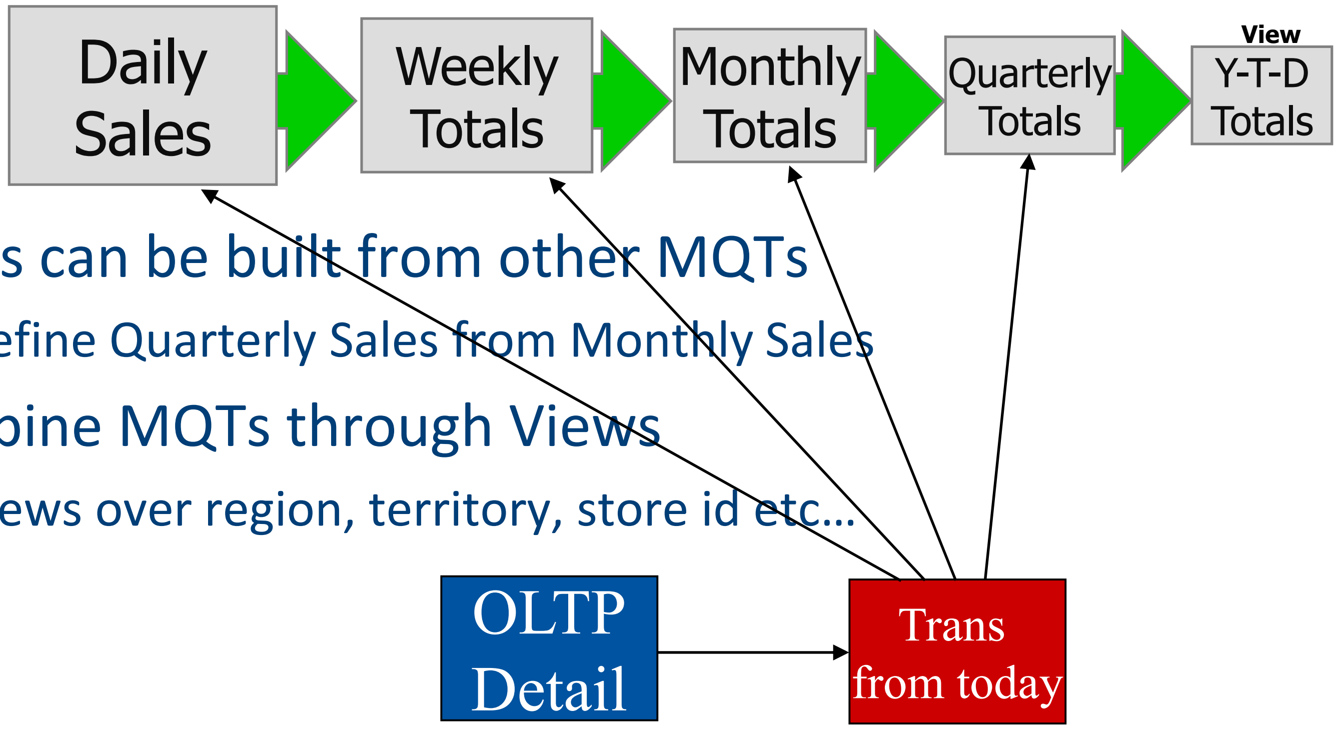
Big Data Flow

- Design encourages concurrent loads and queries
  - Either through bulk, trickle or custom loads
  - Automatically aggregates data within hierarchy
  - Massively parallel processing of all workload aspects



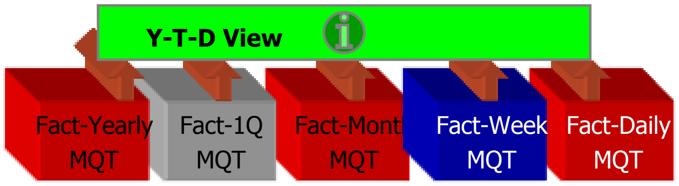
Materialized Query Table-MQT Example

- Daily sales figures feed MQTs
  - Create additive MQTs
  - MQTs created for all analysis comparison points
- MQTs can be built from other MQTs
  - Define Quarterly Sales from Monthly Sales
- Combine MQTs through Views
  - Views over region, territory, store id etc…



MQT – 10 to 1000 times improvement!

- 5B rows per year–10 per 4k page= ½B pages
- MQT aggregates save large amounts of everything
    - Create aggregates for every possibility
        - "On Demand" information
        - Sales by department
        - Sales by zip code
        - Sales by time period – day/week/month/quarter/AP
    - All reporting and analysis areas
    - Trace usage to create/eliminate aggregates
- Total by month ½B I/Os versus 12 I/Os



Database Conditions

- RUNSTATs – Sample your database 5%
  - Table
    - Page size Optimization – Do you have the correct DB2 Page Size?
  - Column Cardinality
    - Important Columns
      - Index columns
      - JOIN columns
      - All columns used within any WHERE clauses
  - Index
    - Cardinality – Make sure to get detailed statistics
    - INCLUDE Columns

Database Conditions

- How long does it take until your Index goes bad?
- You can collect distribution statistics when you collect inline statistics when you run the following utilities:
  - LOAD
  - REORG TABLESPACE
- You can collect histogram statistics when you collect inline statistics when you run the following utilities:
  - LOAD
  - REBUILD INDEX
  - REORG INDEX
  - REORG TABLESPACE

Database Conditions

- Identify your VOLATILE TABLEs
  - Targeted for SAP Customers and the SAP Cluster Tables
  - Favors using Index access whenever possible
    - **Avoids list prefetch**
    - Can be a problem for OR predicates or UPDATEs at risk of loop

- Optimize Page level options
  - PAGE ROW Ratio needs to be optimized for storage and Update frequency
  - PCTFREE FOR UPDATE in DB2 11
    - Help keep clustered longer, reduces the use of overflow records and indirect references

Statement Level Hints Steps

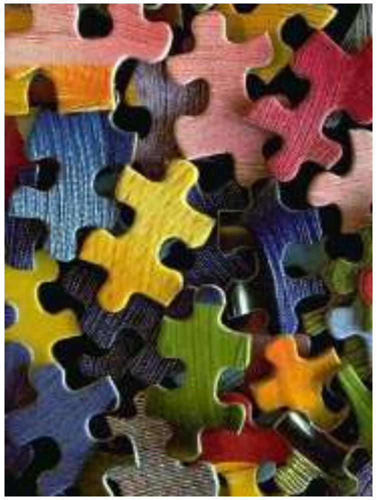
1. Zparm required - Enable OPTHINTS
2. Populate DSN_USERQUERY_TABLE with SQL text
   - From SYSPACKSTMT or Statement Cache
3. Populate PLAN_TABLE with corresponding hints
   - Make sure to match QUERYNO in both tables - DSN_USERQUERY_TABLE & PLAN_TABLE
4. BIND QUERY to put Hint into the repository
   - Next dynamic prepare should use the Hint
   - FREE QUERY to remove it

Database Design

- Use correct database column definitions
  - So all the DB2 Online Analytic Processing-OLAP functions can be used

- Understand the new Stage 1 predicates processed
  - value BETWEEN col1 AND col2
  - value BETWEEN column-expression AND column-expression
  - SUBSTR(COLX, 1, n) = value ****NOTE only from first position****
  - DATE(timestamp-column) = value
  - YEAR(date-column) = value
  - CASE WHEN THEN ELSE END = value

Steps to resolve SQL problems



1. Start with all EXPLAIN tables information
2. Research the database objects, Tables, Indexes
3. Research and understand the statistics
   a. Understand where the problems are:

      CPU, I/O, #-Sorts, #-tables, wrong indexes
4. Using all WHERE clauses possible – most restrictive first
5. How many times accessing the same table
6. Change the SQL table order? Sorts
7. Filter Factors, uneven data distribution, range predicates
   a. Are Stats current, RTS Usage & Can you change the objects statistics to improve?
8. Can you use Hints to improve the optimizer access path choice
9. Buy a IBM DB2 Analytic Accelerator - IDAA

Thank You!

Dave@ d a v e b e u l k e . c o m

Session: F19

Title: **SQL Performance for a Big Data 22 Billion row data warehouse**

Please fill out your session evaluation before leaving!