

May 7-11, 2006
Tampa Convention Center
Tampa, Florida, USA

B13

DB2 for z/OS Design and Tuning Tips for Your Complex VLDB


IDUG® 2006
North America

Daniel L. Luksetich
Yevich, Lawson, and Associates

Thursday, May 11, 2006 • 10:00 a.m. – 11:10 a.m.

Platform: DB2 for z/OS

GoFurther



Dan Luksetich is a senior DBA with Yevich, Lawson, and Associates. Dan has been in Information Technology for the last 21 years, and has been working with DB2 for the last 16 years.

Dan's experience with DB2 includes programming, DBA, and System Programmer. Dan has extensive experience with DB2 for z/OS, and well as DB2 for AIX and Windows. Dan is a frequent presenter, and teaches the following subjects:

SQL Programming

SQL Tuning

Application Monitoring and Tuning

Stored Procedures, Triggers, and User Defined Functions

High Performance and VLDB

New Features

Disclaimer PLEASE READ THE FOLLOWING NOTICE

- The information contained in this presentation is based on techniques, algorithms, and documentation published by the several authors and companies, and in addition is the result of research. It is therefore subject to change at any time without notice or warning.
- The information contained in this presentation has not been submitted to any formal tests or review and is distributed on an “As is” basis without any warranty, either expressed or implied.
- The use of this information or the implementation of any of these techniques is a client responsibility and depends on the client’s ability to evaluate and integrate them into the client’s operational environment.
- While each item may have been reviewed for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere.
- Clients attempting to adapt these techniques to their own environments do so at their own risks.
- Foils, handouts, and additional materials distributed as part of this presentation or seminar should be reviewed in their entirety.

Agenda

- Present Our Concerns
 - Database are Growing to be Quite Large
 - Transaction Volumes are ever Increasing
 - More Complexity is Possible Within the Database
 - More Application Client Platform Options
 - We Now Live in a “Distributed World”
- Discuss Early Design Actions to Help Address Concerns
 - Physical Design
 - Complex Database Concerns
 - SQL Performance Options
 - Advanced Database Objects
- Discuss Monitoring and Tuning Techniques

Concerns

- **Database are Growing Quite Large**
 - Harder to Drop and Recreate a Really Big Table
 - Harder to Move the Table Data
 - Harder to Backup the Table
 - Need a Design to Minimize These Issues
 - Have to Get It Right the First Time
- **Transaction Volumes Increase**
 - Need to Determine Order of Most Incoming Transactions
 - Need to Minimize Change to Data
- **More Complexity in the Database**
 - More Advanced Database Objects
 - Need to Track and Maintain These Objects

Performance is a primary concern for most application implementations, but what's the cost of high performance? We use relational databases for a reason. They're supposed to make things easier by separating the user from the responsibility of having to know where and how their data is stored and managed. All the user needs to know is what they want, and the database is supposed to find it. The SQL language itself is a powerful programming language, enabling users to quickly answer complex questions. Advanced database objects such as constraints, triggers, stored procedures, and user-defined functions also play a significant role in application development and performance.

Any advanced database and application design can have a significant performance impact, so we spend time tuning, but at what cost? There have to be trade-offs in any database design. Do you want centralization, reusability, security, availability, ease of maintenance, quick time to delivery, or do you want the highest performance? Maybe you just want it all.

Let The Database Prove The Design

- To Much Time is Spent Debating VLDB SQL Performance During Design
 - “I Think it Should be Coded This Way Because...”
 - “I Didn’t Do That Because it Will Perform Badly”
 - “What’s the Best Way? We Want it as Fast as Possible”
- You Should Code to Solve the Business Rules First
 - Quick Wasting Time
 - Let The Database Choose the Access Path First
 - When There is a Problem, Fix It!
- If You Can’t Wait to Find Out
 - Test It!
 - Let the Database Drive the Design
 - Give the Proof to Management and Developers

Lots of Developer and DBA time can be spent in the physical design of a database, and database queries based upon certain assumptions about performance. A lot of this time is wasted because the assumptions being made are not accurate. A lot of time is spent in meetings where people are saying things such as “Well that’s not going to work”, “I think the database will choose this access path”, or “We want everything as fast as possible”. Then when the application is finally implemented, and performs horribly, people scratch their heads saying “...but we thought of everything!”.

Another approach for large systems design is to spend little time considering performance during the development, and set aside project time for performance testing and tuning. This frees the developers from having to consider performance in every aspect of their programming, and gives them incentive to code more logic in their queries. This makes for faster development time, and more logic in the queries means more opportunities for tuning (if all the logic were in the programs, then tuning may be a little harder to do). Let the logical designers do their thing, and let the database have a first shot at deciding the access path.

If you choose to make performance decisions during the design phase, then each performance decision should be backed by solid evidence, and not assumptions. There is no reason why a slightly talented DBA or developer can’t make a few test tables, generate some test data, and write a small program or two to simulate a performance situation and test their assumptions about how the database will behave. This gives the database the opportunity to tell you how to design for performance. Reports can then be generated, and given to managers. It’s much easier to make design decisions based upon actual test results.

Test Your Ideas!

- We've Got VLDB Design Issues
 - Blind Updates To the Database
 - Need High Speed Update Process
 - Balance Batch/Online Needs with Access Frequency
 - Table is Physically Too Large for a Single DB2 Table
 - Online Access to The Primary Index is Too Slow
 - We Need to Make an NPI to Access This Table
- We Need DB2 To Tell Us What To Do
 - Build Test Tables
 - Fabricate Data and Statements
 - Perform Data Analysis
 - Run Your Queries
 - Measure the Alternatives
 - Document and Report

The database itself can be your design guide. Every situation can be simulated relatively easily. With tools such as DB2 Personal Edition and REXX DBAs and developers can very quickly generate test data, and write small programs to simulate predicted program processes.

The following are real world examples of predictive performance analysis. In each situation a real database did not yet exist, and no programs had yet to be written.

Tools for Testing

- You Need a High Performance Design
 - However, No Application Programs Written Yet!!
- So, You Need to Simulate the Data and Access
- Tools – Data Generation (If you Don't Have a Tool)
 - REXX – Simple Scripting Tool
 - DB2 and Recursion to Generate Data
 - If V7 Then Use DB2 on Your PC
 - Then Import Data to Mainframe from IXF Format File
 - Type in Some Data and Repetitive SQL to Propagate
- Tools – Access Simulation (If you Don't Have a Tool)
 - COBOL
 - REXX
 - DB2 and Recursion to Generate Statements
 - If V7 Then Use DB2 on Your PC
 - Use Generated Table Data to Generate Statements
 - SQL Generating SQL

There are many ways to do predictive analysis and design testing. There are also a variety of tools available to do it. We use as many of them as are available to get the job done, usually dependent upon the people and machine resources that are available for testing.

Data Generation Example

- A look-up table was created to derive keys for a complex table design
 - The look-up table was to contain 300,000 rows of data
 - Not easy to type in!
 - The following is an example of the look-up table for twenty values and 20 partitions

LOOK_UP_TBL	
<u>STALE_IND</u>	CHAR(1) NOT NULL
<u>KEYVAL</u>	SMALLINT NOT NULL
<u>PART_NUM</u>	SMALLINT NOT NULL

Combination of two key columns determines the partition

© Copyright 2004, YL&A, All rights reserved.

8



In one design it was determined that data should be spread across multiple partitions of a partitioned tablespace. Data was to be split according to its age (stale or not stale) and based upon the value of last position of its key. So, if the stale indicator was 'S' (for stale data) then the data will go into partitions 1 through 10 depending upon the value of the last position of the key (each partition corresponds to the last value of the key, e.g. '0' goes into partition 1, and '9' goes into partition 10). If the stale indicator is 'F' (for fresh data) then the data will go into partitions 11 through 20 depending upon the value of the last position of the key (e.g. '0' goes into partition 11, and '9' goes into partition 20). We can build a look-up table that holds all of the combinations of stale indicator and last position value. In this particular case the resulting look-up table would have 300,000 rows of data. The test needed to be conducted, but typing in 300,000 rows of data was a bit difficult.

Data Generation Example

- Recursive Query to Generate the Gata
- Place This in an EXPORT Command (V7 or V8) or DSNTIAUL Unload (V8)

```

WITH LASTPOS (KEYVAL) AS
  (VALUES (0)
   UNION ALL
   SELECT KEYVAL + 1
   FROM   LASTPOS
   WHERE  KEYVAL < 9)
,STALETBL (STALE_IND) AS
  (VALUES 'S', 'F')
SELECT STALE_IND, KEYVAL
  ,CASE STALE_IND WHEN 'S' THEN
    CASE KEYVAL WHEN 0 THEN 1
    WHEN 1 THEN 2 WHEN 2 THEN 3
    WHEN 3 THEN 4 WHEN 4 THEN 4
    WHEN 5 THEN 6 WHEN 6 THEN 7
    WHEN 7 THEN 8 WHEN 8 THEN 9
    WHEN 9 THEN 10 END
    WHEN 'F' THEN
    CASE KEYVAL WHEN 0 THEN 11
    WHEN 1 THEN 12 WHEN 2 THEN 13
    WHEN 3 THEN 14 WHEN 4 THEN 15
    WHEN 5 THEN 16 WHEN 6 THEN 17
    WHEN 7 THEN 18 WHEN 8 THEN 19
    WHEN 9 THEN 20 END
  END AS PART_NUM
FROM   LASTPOS INNER JOIN
  STALETBL ON 1=1;

```

Recursive common table expression generates all key values

A second common table expression sets the values for stale or fresh data

A Cartesian join combines all the values, and the case expressions run the data transformation rules

© Copy



The result was that a recursive query was written to generate the data. This recursive query could be run on DB2 for z/OS, or it could be run on DB2 Personal Edition, and then the results imported into a DB2 for z/OS table. The following SQL statement is a smaller example of the actual statement that was run. This example uses common table expressions, recursion, a Cartesian join, and CASE expressions to generate the data for 20 partitions.

Statement Generation Example

- Some Design Situations Call for Small Scale Testing of SQL Statements to Prove a Concept
 - One Situation: Test an Insert Strategy for a High Volume Table
 - Table was Clustered by a Date Column and Account Identifier
 - Need to Test Sequential Versus Random Inserts on the Table
 - Statements are Generated on a PC then FTP'd to Mainframe

ACCT_HIST_TBL	
<u>HIST_EFF_DTE</u>	DATE NOT NULL
<u>ACCT_ID</u>	DEC(11,0) NOT NULL

Some design situations call for small scale testing of SQL statements to prove a concept. In one recent situation there was a need to test an insert strategy for a high volume table. The table was clustered by a date column and account identifier, and we needed to test sequential versus random inserts on the table.

Statement Generation Example

- Two Queries to Generate 50,000 Statements
 - One sequential

```
WITH GENDATA (ACCT_ID, HIST_EFF_DTE) AS
(VALUES (CAST(1 AS DEC(11,0)), CAST('2004-02-01' AS DATE))
UNION ALL
SELECT ACCT_ID + 5, HIST_EFF_DTE
FROM GENDATA
WHERE ACCT_ID < 249996
)
SELECT 'INSERT INTO YLA.ACCT_HIST (ACCT_ID, HIST_EFF_DTE)' CONCAT
' VALUES(' CONCAT CHAR(ACCT_ID) CONCAT ',' CONCAT ''''
CONCAT CHAR(HIST_EFF_DTE,ISO) CONCAT '''' CONCAT ');'
FROM GENDATA ORDER BY ACCT_ID;
```

50,000 inserts in key sequence, incrementing the ACCT_ID from 1 by 5 with each statement

- One random

```
WITH GENDATA (ACCT_ID, HIST_EFF_DTE, ORDERVAL) AS
(VALUES (CAST(2 AS DEC(11,0)), CAST('2003-02-01' AS DATE), CAST(1 AS FLOAT))
UNION ALL
SELECT ACCT_ID + 5, HIST_EFF_DTE, RAND()
FROM GENDATA
WHERE ACCT_ID < 249997
)
SELECT 'INSERT INTO YLA.ACCT_HIST (ACCT_ID, HIST_EFF_DTE)' CONCAT
' VALUES(' CONCAT CHAR(ACCT_ID) CONCAT ',' CONCAT ''''
CONCAT CHAR(HIST_EFF_DTE,ISO) CONCAT '''' CONCAT ');'
FROM GENDATA ORDER BY ORDERVAL;
```

50,000 inserts in random order (by using a RAND function)

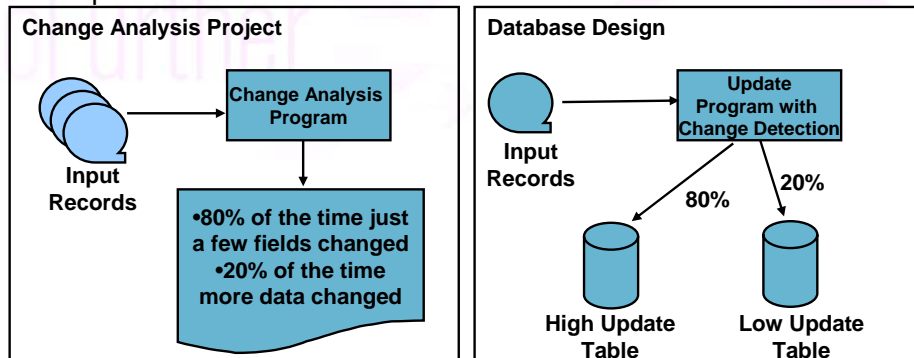
© Copy



No programs or programmers were available, and so recursive SQL was once again used to generate enough statements to test the theories.

Blind Update to the Database

- Update Records Contain Every Database Field
 - Do We Have to Update Every Field?
- Test the Input Data
 - Compare Successive Update Records
 - Determine Which Fields Change All The Time
- Apply The Test Results to Your Design
 - Change Detection Process
 - Separate Database Tables



© Copyright 2004, YL&A, All rights reserved.



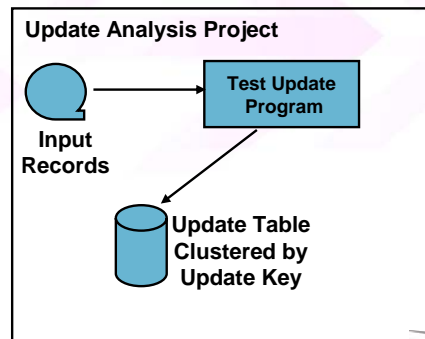
An application in design will have a large volume of “blind” updates. These are updates that arrive as full records, with no advance knowledge as to what fields in the update record actually need to be updated on the database. These blind updates have the potential of being very expensive if all of the database fields have to be updated all of the time. In addition, we have the potential for increased utility times for reorgs and copies if we are updating significant quantities of data.

During design we had to test the input data to determine what fields in the input change, and how often they change. Programs were written for the sole purpose of reading all of the input records over the course of several months, and then comparing records for the same input identifier to each other. The analysis proved that the vast majority of the time only a few fields actually changed. The infrequently changing fields can be utilized in a change detection process, and we can then build a database table that contains only the columns that are updated all of the time. This gives us more options for clustering, and reduces the demand for image copies and reorgs.

Without the change analysis project we would not have been able to determine the database design without guessing. With the change analysis project our design was practically dictated to us! By writing the programs to analyze the data there was no guesswork, no assumptions. The analysis enabled the physical design to happen, not guesswork!

High Speed Update Process versus Online Queries

- Batch Update Process Needs to be Very High Speed
 - As Many as 120 Million Updates per Day – Skip Sequential
 - Data Feed in CUST_ID, ACCT_NUM Order
- Online Process Not as High Volume
 - Purely Random Reads – 5 Million per Day
 - Requests Enter System via PRSN_ID, ACCT_NUM
- Test a Design for High Speed Updates
 - Split Test Table Data into High Speed Update Table
 - Cluster by Update Sequence
 - Write a Simple Test Program
 - Read Input Data
 - No Business Rules
 - Simply Locate Update Row
 - Verify Sequential Processing



13

© Copyright 2004, YL&A, All rights reserved.

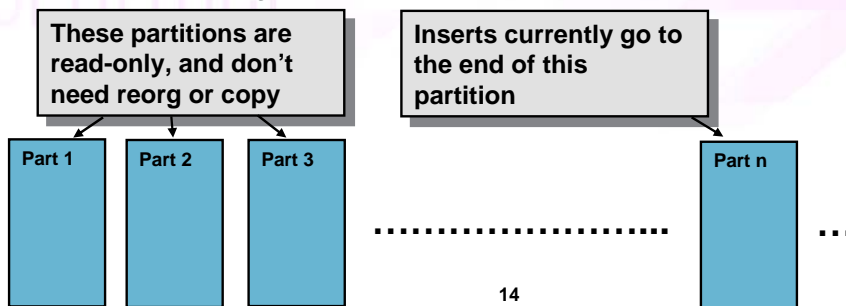


A batch update process is expected to have very high transaction volumes, while the online process will have relatively fewer random accesses to the same data. Since there is no chance for sequential access in the online process, we should cluster our update table by the key in which the batch updates come into the system. However, we do not know for sure that we'll get sequential detection on our batch update table because it's a highly skip sequential process. If not, we'll have to consider other design options. The online process reads the data by a different key than the batch process. Clustering the update table by the key of the batch process will hurt the online process. If there is no benefit to clustering by the update key, then why sacrifice online performance?

To answer this question we wrote a very simple test update program that simply reads in the real input records sorted in clustering sequence, and reads the update table. We monitor the test to make sure that sequential detection is activated. Testing did demonstrate that even though the batch process was skip sequential it did benefit significantly from clustering in the batch update order, and sequential detection. The table design was altered to support this clustering.

High Speed Update to Very Large Table

- Rule 1 – Don't Update
 - Insert Only to Ever Increasing Key Value
 - Take Advantage of "No Read" Insert
 - PCTFREE 0 and FREEPAGE 0 MEMBER CLUSTER YES
 - APARs PQ86037 and PQ87381
 - Fast Insert Access Path
 - Can Accommodate Current Data Plus Audit Requirements
 - Minimizes Database Maintenance
- If You Have to Update
 - Match Transaction Input to Clustering Sequence
- This Does Not Always Solve Index Insert Issues
 - Must Get Freepages to Avoid Index Split Searches



© Copyright 2004, YL&A, All rights reserved.

14

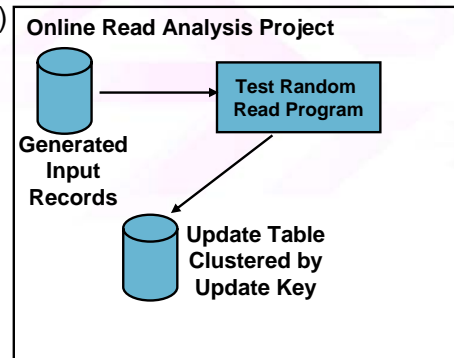


The DB2 insert algorithm is influenced by the member cluster setting for partitioned tablespaces. Read the diagnoses reference to understand this.

There is no member cluster type option for the index. Indexes with a true ascending key will simply add pages instead of split pages when they are full. However, if you have columns in the index that introduce any sort of randomness then DB2 will split pages when full and go into a page search algorithm. This algorithm will perform as exhaustive search before adding a page at the end. For this reason you need to somehow get free pages in you index space. This can be very difficult to do in a high performance environment when outages are minimal.

Online Access to Primary Index Too Slow / Need NPSI?

- Will Online Performance to Our Update Table be Acceptable?
- Do We Need an NPI?
 - I Hope Not!
- Test Simulated Online Random Reads
 - Write a Small Program to Simulate Online Reads
 - Monitor Random Access to Our Update Table
 - Clustering For Batch Results in Highly Random Index Reads
 - Index is Very Very Big (Billions of Rows)
- Verify Results
 - Online Index Access Unacceptable?
- Take Design Action
 - Reduce Size of Key
 - Isolate Index in Bufferpool
 - Increase Bufferpool Size
 - Last Report...NPSI
 - Must accommodate outages!



© Copyright 2004, YL&A, All rights reserved.

15



Since the update table is not clustered by the access key of the online application, we had to test to make sure that online read access to this table will be acceptable.

A small test program was written to simulate random access to the update table. It was determined that access to the index is much too slow, and actually takes longer than the entire projected online transaction time! Therefore, based upon the results of this test we have to take design action.

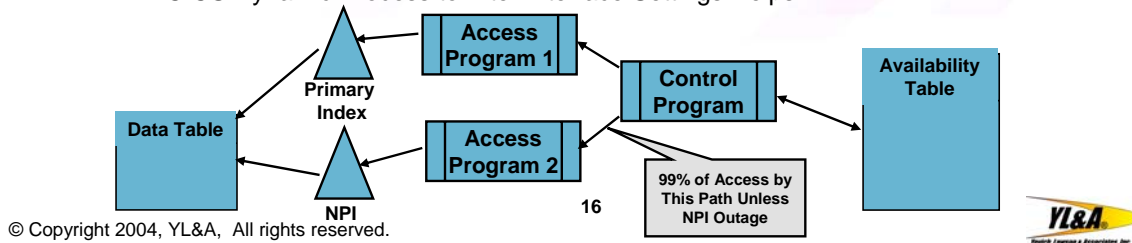
Several steps are taken, and tested in succession:

- Reduce the size of the index key
- Isolate the index its own bufferpool
- Increase the bufferpool size
- Add an NPI

Each variation above was tested using the simulated batch and online testing programs. The ultimate design decision will depend upon the testing results. Critical decisions, such as adding an NPI, will have to be fully considered and tested. This includes testing reorgs, rebuilds, and alternate queries to avoid the NPI during outages of that NPI.

Accommodating the NPSI Outage

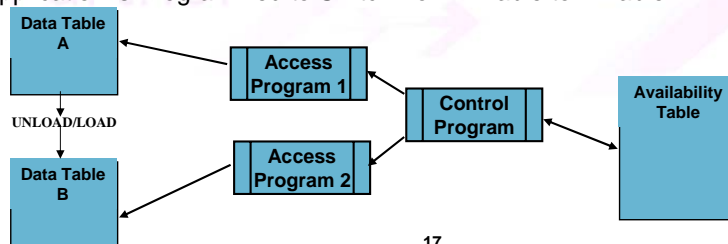
- We Are Evolving to 24X7 Availability
 - Not Up All Day, Just Good At Hiding Outages
- We Must “Hide” Our NPI Outage
 - Denormalize the Key Data
 - This Allows Multiple Access Paths to Table with NPI
 - Join to Table Can Take One of Two Paths
 - Build Two Access Programs
 - One Using NPI Access Path
 - One Using Primary Key Access Path
 - Build “Availability Table”
 - Table Dictates Which Path to Take
 - During NPI Outage Use This Table to Switch Paths
 - Instead of Outage We Have a “Slow Down”
 - CICS Dynamic Process to Alter Interface Settings Helps!



In one of our high performance account database designs we separated account table data that was actively changing on a regular basis from account database data that was more stable. The active data was clustered by the application update sequence in order to take advantage of high speed sequential processing. The more stable account data was clustered by the identifier of the person that held the account. Since a person can have more than one account in the database then clustering all of the accounts together for a person would improve the access time for retrieving a person. Access to the active table data, that was not clustered by the id of the person owning the accounts was much slower. For this reason an NPSI was built on the active data clustered by the update key. The NPSI was by the person id and the account id. Since the tablespace was in 205 partitions, and contained nearly 4 billion rows of data, the NPSI would be quite large and it would be very difficult to perform reorgs. The decision was made that during a scheduled reorg window the NPSI would be dropped. The batch application that updated the account data can be suspended during the reorg window, but the online application had to remain up in support of the 24X7 availability.

Accommodating a Table Outage

- We Have a Very Large Table
 - 1TB in Size
 - 1 PI and 2 NPSI
 - The NPSI's are for Access Paths So DPSI's Won't Work
- We Need 24X7 Access
 - Can't Get Online REORG Switch Due to Claims
 - NPSI's Take Too Long to REORG
- Tested Many Solutions – Best One is to Switch Tables
 - Old Unload/Load Technique
- Application is Programmed to Switch from A Table to B Table



© Copyright 2004, YL&A, All rights reserved.

17

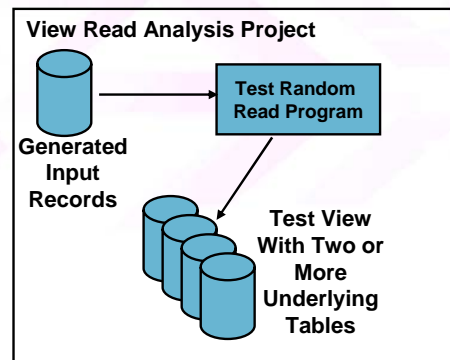


In a situation in which a very large table with NPSI's need a REORG we had to build the ability to switch between copies of the table in the application. We were able to get a period of time in which the table could be in read-only mode. During that time and unload/load process copied and organized the data into the second table. At the end of this process the application switched to the new table and out of read-only mode.

In situations in which read-only is not an option then changes can be captured in overflow tables until the switch is made.

Table is Physically Too Large

- Problem - Extremely Large Table
- Solution – UNION in View (Max Table Still 16TB for 4K Page in V8)
- Test This Solution!
 - Create Different Views Varying Number of Underlying Tables
 - Write a Test Read Access Program
 - Verify the Access Paths
 - Measure the Performance
- Take Design Action
 - Build a Lookup Table
 - Dynamic Inserts
 - Dynamic Updates
 - Dynamic Deletes
 - Eliminate Joins?
 - Take Advantage Of QB Pruning
 - Large Reduction in CPU
 - View Has Appropriate # of Tables
 - All Queries will Work Properly



© Copyright 2004, YL&A, All rights reserved.

18



There was a situation in which a table in our design would be extremely large in size, containing over 100 billion rows of data, and exceeding DB2 table size limits.

One solution to this problem is utilizing UNION everywhere to build a view containing a UNION of multiple underlying tables. However, there are several questions that will have to be answered about this type of implementation, such as:

- How many underlying table can I put in the view?
- Are query access paths affected?
- Can I update the view?

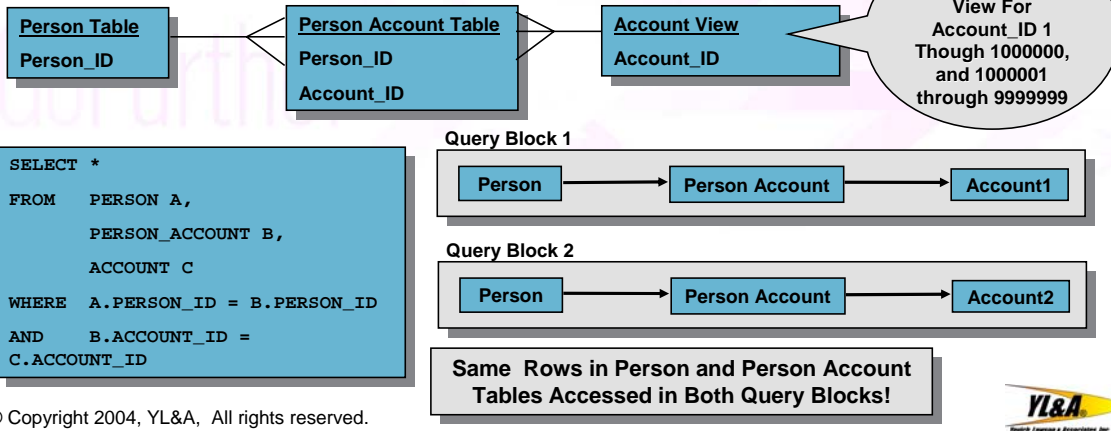
Several test views were created, and each contained a different number of underlying tables. A test program was written to read from the views via direct access and joins to other tables. Inserts, updates, and deletes are not possible, and so a method up dynamically performing these operations, without having to modify code when tables are added or removed, is invented.

This testing helps us create the best UNION in view design for our application, including:

- A good number of underlying tables
- A dynamic insert, update, and delete process using a finder table
- In-house standards for SQL access to the view in order to insure the application SQL takes advantage of UNION in view features such as query block pruning

UNION in View Tips

- Runtime Pruning for Host Variables
 - Only for Simplest of SQL Statements
 - Watch for APAR PQ92434 (Does not fix everything so you must test!)
- **May** Want to Avoid Joins
 - Repetitive Access to Outer Table for Each Query Block
 - May be Improved in Future DB2 Release (maybe Vnext+1...the one after Vnext)
 - No Query Block Pruning for Join Predicate
 - Host Variable or Literal Only



This redundant access will increase the number of getpages issued and CPU utilized for some queries that involve joins against views that employ UNION. In tests of a four table join, where the fourth table was a UNION in view large table design containing 8 underlying tables, and where query block pruning did not occur, there was a 100% increase in CPU utilization when compared to an equivalent query where the view was replaced by a single table (both queries returned the same results). In this situation, predicate transitive closure applied predicates from the union to the table joined to the view. However, since that table was the third table accessed in the join sequence the first two tables were accessed redundantly in each query block.

More Large Table Issues

- Advanced DASD Subsystem Really Making a Difference
 - Large Storage and Large Cache
 - Advanced Data Staging Algorithms
 - Fast Response (2.8ms for Random is Best Seen)
 - Parallel Access Volumes are a Must
 - Advanced Backup and Recovery Solutions
- Backup and Recovery
 - Can't Copy a Very Large NPSI
 - Only Single Dataset Allowed for Image Copy (59 Volume Limit)
 - Can't Always Rebuild an NPSI
 - Need DASD for Sorts
 - Need CPU
- Might be Better to Have Many Small Objects Instead of a Few Large Ones

SQL Performance Options

- DB2 Engines Moving Toward Support of ERPs and Warehouses
 - Query Rewrite; Blessing or Curse?
 - Merge Scan Versus Nested Loop Join
- Use Correlation for Transactions
 - Correlation Encourages Index Access
 - Correlation Encourages Nested Loop Join
- Want Subsecond Response?
 - Nested Loop Join
 - Lots of Choices for Correlated Table Expressions in V8
 - In V7 You Need APAR PQ66365 for Stage 1
- Processing Tons of Data?
 - Merge Scan Join

In my opinion, the LUW engine is primarily a warehouse engine. Large, complex queries are processed with absolutely astonishing speed. On an SMP machine, DB2 can use intra-partition parallelism quite effectively to improve query elapsed time. In this regard, the engine has a propensity towards a merge scan join. While merge scan is a significant performance advantage when the query is processing very large amounts of data, it can be a disadvantage for transaction processing, especially when fewer rows of data are expected to be processed. Correlating table expressions encourages the optimizer to use indexes to resolve the correlated references, which subsequently favors nested loop join over merge scan. This is demonstrated in the following real-world examples.

My concept of merge scan versus nested loop is cross-platform. Also, a lot of the optimization and query rewrite features of the DB2 UDB LUW engine are being incorporated into the mainframe DB2, so I've started to apply some of my LUW techniques up on the big iron!

Correlated Table Expression

- Returns Line Item and Summary Data in One Query
 - Nested Loop Join With Index Access
 - Good For Transactions or Processing Small Amount of Data
- Use Left Outer Join Instead for Large Data Volumes (Merge Scan)

```
SELECT TAB1.EMPNO, TAB1.SALARY,
       TAB2.AVGSAL, TAB2.HDCOUNT
FROM   DSN8710.EMP TAB1
       ,TABLE(SELECT AVG(SALARY) AS AVGSAL,
                COUNT(*) AS HDCOUNT
              FROM   DSN8710.EMP
              WHERE  WORKDEPT = TAB1.WORKDEPT) AS TAB2
WHERE  TAB1.JOB = 'SALESREP';
```

Index Access Comes “Built-In” for V8 (SARGSWRP default is YES)

DB2 for z/OS and OS/390 V7 APAR: PQ66365

And ZPARM SARGSWRP

This is a correlated nested table expression, sometimes referred to as a sideways reference.

If there is an index on the WORKDEPT column, the optimizer is likely to pick it based upon the predicate in the table expression. This can result in a dramatic improvement in query performance as apposed to an outer join. If you used an outer join to a nested table expression to solve this problem, you can expect a merge scan join, and a table scan on the EMP table to satisfy the second table expression. In contrast, this query should result in a nested loop join, using an index on the WORKDEPT column (assuming one exists). The TABLE keyword is required in order use the correlated reference. Correlated table expressions such as these have been used quite extensively in the billing application, resulting in fantastic transaction performance involve complex processes.

Before Performance Correlated Table Expression

- This Query Wants Account Information and the Most Recent Account History
 - Not Getting Good Access to Inner Most Table
 - Equality on ACCT_ID Not Pushed Down to Table “c”

```

select  acct_num, tab2.bil_date, tab2.dollar_amt
from    account a
left outer join
(select bil_date, dollar_amt
 from    acct_hist b
 where  b.bil_date =
        (select max(bil_date)
         from    acct_hist c
         where   c.acct_id = b.acct_id)) as tab2
on      a.acct_id = tab2.acct_id

```

In this situation, we desire the information for an account. We also want the most recent historical dollar amount, if that information exists.

This seems like a reasonable request, but there is a chance that the optimizer may not choose to use the provided index (first column is the acct_id of the acct_hist) for the “select max” subquery.

This is a cross-platform query.

After Performance Correlated Table Expression

- Correlated Table Expression
 - Correlated Reference Encourages Index Access

```
select acct_num, tab2.bil_date, tab2.dollar_amt
from account a
left outer join
Table(select bil_date, dollar_amt
from acct_hist b
where b.bil_date =
(select max(bil_date)
from acct_hist c
where c.acct_id = a.acct_id)) as tab2
on a.acct_id = tab2.acct_id
```

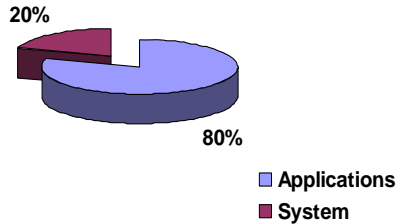
By changing the correlated reference from the acct_hist table (the table with the correlation name “b”) to the account table (correlation name “a”) we may be able to influence the optimizer to an index that its was less likely to choose with the previous query.

This is a cross-platform query.

Steps To Quickly Measuring Application Performance

- Gather Package Level Performance Information
 - The Proper Accounting Traces Must be Set
- Collect Performance Information Over 1 or More Days
 - Summarize the Costs by Package
 - Number of Statements
 - Elapsed Time per Statement
 - CPU Consumption per Statement
- Identify the “Heavy Hitters” (AKA Low Hanging Fruit)
 - Attack those “Heavy Hitters”

Percentage of Tuning Opportunities



25

© Copyright 2004, YL&A, All rights reserved.

As much as we hate to hear it, there's no silver bullet for improving overall large system performance. We can tune the DB2 subsystem parameters, logging and storage, but often we're only accommodating a bad situation. Here the "80/20" rule applies; you'll eventually have to address application performance.

There's a way to quickly assess application performance and identify the significant offenders and SQL statements causing problems. You can quickly identify the "low-hanging fruit," report on it to your boss, and change the application or database to support a more efficient path to the data. Management support is a must, and an effective manner of communicating performance tuning opportunities and results is crucial.

DB2 Accounting Performance Impact

- Addresses System Wide Application Performance
 - Utilize a Batch Reporting System
 - We Need DB2 Package Level Reporting
 - DB2 Accounting Traces 1,2,3,7,8
 - Reports Are Inputs to Spreadsheets and Charts
- The Impact is Minimal, the Benefits Substantial
 - DB2 Accounting Traces 1,2,3,7,8 to SMF Increases CPU by 4.3%
 - If Your NOT Already Capturing That Information to an Online Monitor!

Monitor Product	Percent Overhead
SMF	4.30%
Product 1	-0.42%
Product 2	-1.36%
Product 3	10.51%

Note: Product 3 anomaly due to performance class 30 trace that can be turned off

26

© Copyright 2004, YL&A, All rights reserved.



DB2 accounting traces will play a valuable role in reporting on application performance tuning opportunities. DB2 accounting traces 1, 2, 3, 7, and 8 must be set to monitor performance at the package level. Once you do that, you can further examine the most expensive programs (identified by package) to look for tuning changes. This reporting process can serve as a quick solution to identifying an application performance problem, but can be incorporated into a long-term solution that identifies problems and tracks changes.

There's been some concern about the performance impact of this level of DB2 accounting. The IBM DB2 Administration Guide states that the performance impact of setting these traces is minimal and the benefits can be substantial. Tests performed at a customer site demonstrated an overall system impact of 4.3 percent for all DB2 activity when accounting classes 1, 2, 3, 7, and 8 are started. In addition, adding accounting classes 7 and 8 when 1, 2, and 3 are already started has nominal impact, as does the addition of most other performance monitor equivalent traces (i.e. your online monitor software).

Summarized Accounting Information

- Summarized DB2 Accounting Information
 - Package Level Summary Reports are Produced
 - Reports are Formatted Such That They Can be Fed into Spreadsheet Software
 - REXX is a Good Tool For This
 - Some Reporting Software is also Very Flexible
 - Or Sometimes Even "Cut and Paste"
- Produce Daily Spreadsheet Data
 - Produce a Spreadsheet by Package
 - Package Name, Number of Executions, Total Class 2 Elapsed,
 - Total Class 2 CPU, Elapsed Per Execution, CPU Per Execution
 - Elapsed Per SQL Statement, CPU Per SQL Statement
 - Use Naming Standards to Summarize by Application
 - Packages Belonging to the Accounting Application:
 - ACCT001, ACCT003, ACCT003, ACCT004
 - Package Name Prefix Used to Summarize by Application
 - ACCT

© Copyright 2004, YL&A, All rights reserved.

27



To effectively communicate application performance information to management, the accounting data will must be organized and summarized up to the application level. You need a reporting tool that formats DB2 accounting traces from System Management Facilities (SMF) files to produce the type of report you're interested in. Most reporting tools can produce DB2 accounting reports at a package summary level. Some can even produce customized reports that can filter only the information you want out of the wealth of information in trace records.

If you lack access to a reporting tool that can filter out just the pieces of information desired, you can write a simple program in any language to read the standard accounting reports and pull out the information you need. REXX is an excellent programming language well-suited to this type of "report scraping," and you can write a REXX program to do such work in a few hours. You could write a slightly more sophisticated program to read the SMF data directly to produce similar summary information if you wish to avoid dependency on the reporting software.

Spreadsheet Listing of Package Level Information Now We can Search for Our “Heavy Hitters”

- This Spreadsheet is Sorted by CPU Descending

Package	Executions	Total Elapsed	Total CPU	Total SQL	Elaps/Execution	CPU/Execution	Elapsed/SQL	CPU/SQL
ACCT001	246745	75694.2992	5187.4262	1881908	0.3067	0.021	0.0402	0.0027
ACCT002	613316	26277.2022	4381.7926	1310374	0.0428	0.0071	0.02	0.0033
ACCTB01	8833	4654.4292	2723.1485	531455	0.5269	0.3082	0.0087	0.0051
RPTS001	93	6998.7605	2491.9989	5762	75.2554	26.7956	1.2146	0.4324
ACCT003	169236	33439.2804	2198.0959	1124463	0.1975	0.0129	0.0297	0.0019
RPTS002	2686	2648.3583	2130.2409	2686	0.9859	0.793	0.9859	0.793
HRPK001	281	4603.1262	2017.7179	59048	16.3812	7.1804	0.0779	0.0341
HRPKB01	21846	3633.5143	2006.6083	316746	0.1663	0.0918	0.0114	0.0063
HRBKB01	505	2079.5351	1653.5773	5776	4.1178	3.2744	0.36	0.2862
CUSTB01	1	4653.9935	1416.6254	7591111	4653.9935	1416.6254	0.0006	0.0001
CUSTB02	1	3862.1498	1399.9468	7971317	3862.1498	1399.9468	0.0004	0.0001
CUST001	246670	12636.0232	1249.7678	635911	0.0512	0.005	0.0198	0.0019
CUSTB03	280	24171.1267	1191.0164	765906	86.3254	4.2536	0.0315	0.0015
RPTS003	1	5163.3568	884.0148	1456541	5163.3568	884.0148	0.0035	0.0006
CUST002	47923	10796.5509	875.252	489288	0.2252	0.0182	0.022	0.0017
CUST003	68628	3428.4162	739.4523	558436	0.0499	0.0107	0.0061	0.0013
CUSTB04	2	1183.2068	716.2694	3916502	591.6034	358.1347	0.0003	0.0001
CUSTB05	563	1232.2111	713.9306	1001	2.1886	1.268	1.2309	0.7132

Things That are of Interest

- High Overall CPU
- High Number of SQL Statements
- High CPU per SQL Statement

28

© Copyright 2004, YL&A, All rights reserved.



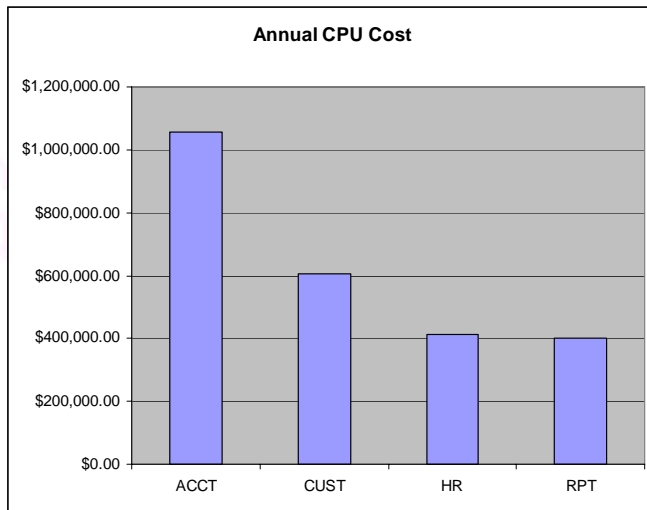
You can process whatever types of reports you produce so that a concentrated amount of information about DB2 application performance can be extracted. This information is reduced to the amount of elapsed time and CPU time the application consumes daily and the number of SQL statements each package issues daily. This highly specific information will be your first clue as to which packages provide the best DB2 tuning opportunity.

Once the standard reports are processed and summarized, all the information for a specific interval (say one day) can appear in a simple spreadsheet. You can sort the spreadsheet by CPU descending. With high consumers at the top of the report, the low hanging fruit is easy to spot.

ACCT001, ACCT002, RPTS001, and RPTS002 represent the best opportunities for saving CPU, so examine those first. Without this type of summarized reporting, it's difficult to do any sort of truly productive tuning. Most DBAs and systems programmers who lack these reports and look only at online monitors or plan table information are really just shooting in the dark.

Do You Need To Convince The Boss?

- Use Naming Standards to Categorize Packages Into Applications
 - Turn CPU Into Dollars
 - Your Capacity Planner Should Have a Formula



29



© Copyright 2004, YL&A, All rights reserved.

To do this type of tuning, you need buy in from management and application developers. This can sometimes be the most difficult part because, unfortunately, most application tuning involves costly changes to programs. One way to demonstrate the potential ROI for programming time is to report the cost of application performance problems in terms of dollars. This is easy and amazingly effective!

The summarized reports can report on information on the application level. An in-house naming standard can be used to combine all the performance information from various packages into application-level summaries. This lets you classify applications and address the ones that use the most resources.

For example, if the in-house accounting application has a program naming standard where all program names begin with "ACCT," then the corresponding DB2 package accounting information can be grouped by this header. Thus, the DB2 accounting report data for programs ACCT001, ACCT002, and ACCT003 can be grouped together, and their accounting information summarized to represent the "ACCT" application.

Most capacity planners have formulas for converting CPU time into dollars. If you get this formula from the capacity planner, and categorize the package information by application, you can easily turn your daily package summary report into an annual CPU cost per application.

Isolate Analysis to The Packages of Interest

- We Need to Drill Down to the Program
 - Which DB2 Statements are Executing?
 - How Often Do They Execute?
 - How Much CPU Does Each Statement Consume?
- There Are Tools To Help US
 - Program Source Code Listings
 - DB2 Performance Trace
 - Online Monitor Tools
 - Batch Analysis Tools
 - DB2 Explain
- Bottom Line - We need Statement Level Information at This Point

Once you've found the highest consuming packages and obtained management approval to tune them, you need additional analysis of the programs that present the best opportunity. Ask questions such as:

- Which DB2 statements are executing and how often?
- How much CPU time is required?
- Is there logic in the program that's resulting in an excessive number of unnecessary database calls?
- Are there SQL statements with relatively poor access paths?

Involve managers and developers in your investigation. It's much easier to tune with a team approach where different team members can be responsible for different analysis.

Useful PLAN_TABLE Query

```

SELECT SUBSTR(DIGITS(QUERYNO),5) CONCAT '-' CONCAT
SUBSTR(DIGITS(QBLOCKNO),4) CONCAT '-' CONCAT SUBSTR(DIGITS(PLANNO),4) AS
Q_QB_PL ,PROGNAME AS PNAME ,SUBSTR(CHAR(METHOD),1,1) AS MT
,SUBSTR(TNAME,1,18) AS TNAME ,CHAR(TABNO) AS T_NO ,ACCESSTYPE AS AT
,CHAR(MATCHCOLS) AS MC ,SUBSTR(ACCESSNAME,1,8) AS ACC_NM ,INDEXONLY AS IX
,SORTN_JOIN CONCAT SORTC_UNIQ CONCAT SORTC_JOIN CONCAT SORTC_ORDERBY
CONCAT SORTC_GROUPBY AS NJ_CUJOG ,PREFETCH AS PF ,COLUMN_FN_EVAL AS
CFE ,CHAR(MIXOPSEQ) AS MIX ,ACCESS_DEGREE AS A_DEG ,JOIN_DEGREE AS J_DEG
,PARALLELISM_MODE AS P_MODE ,MERGE_JOIN_COLS AS MJC ,CORRELATION_NAME
AS COR_NM ,PAGE_RANGE AS PG_RG ,JOIN_TYPE AS JT ,WHEN_OPTIMIZE AS WH_OP
,QBLOCK_TYPE AS QB_TYP ,BIND_TIME AS B_TM ,HINT_USED AS HINT
,PRIMARY_ACCESSTYPE AS PR_ACC

FROM ACCT.PLAN_TABLE A

WHERE PROGNAME IN ('ACCT001' , 'RPTS001')

WHERE BIND_TIME = (SELECT MAX(BIND_TIME) FROM ACCT.PLAN_TABLE B WHERE
A.PROGNAME = B.PROGNAME AND A.COLLID = B.COLLID)

ORDER BY PROGNAME, BIND_TIME, Q_QB_PL, MIX;

```

If plan table information isn't available for the targeted package, then you can rebind that package with EXPLAIN(YES). If it's hard to get the outage to rebind EXPLAIN(YES) or a plan table is available for a different owner id, you could also copy the package with EXPLAIN(YES) rather than rebinding it.

PLAN_TABLE Query Results

Q_QB_PL	PNAME	MT	TNAME	T_NO	AT	MC	ACC_NM	IX	NJ_CUJOG	PF
000640-01-01	ACCT001	0	PERSON_ACCT	1	I	1	AAIXPACO	N	NNNNN	S
000640-01-02	ACCT001	3		0		0		N	NNNYN	
001029-01-01	RPTS001	0	PERSON	1	I	1	AAIXPRCO	N	NNNNN	S
001029-01-02	RPTS001	4	ACCOUNT	2	I	1	CCIXACCO	N	NNNNN	L
001029-01-03	RPTS001	1	ACCT_ORDER	3	I	2	CCIXAOCO	N	NNNNN	

Our Most Expensive Program is Sorting!

Our Expensive Reporting Program is Performing a Hybrid Join!

These simple access path issues may not mean much, but when you combine them the run times and system info, they can mean the world!

32

© Copyright 2004, YL&A, All rights reserved.



Here, our most expensive program issues a simple SQL statement with matching index access to the PERSON_ACCT table, and it orders the result, which results in a sort. Programmers advise that the query rarely returns more than a single row of data. In this case, a bubble sort in the application program replaced the DB2 sort. The bubble sort algorithm was almost never used because the query rarely returned more than one row, and the CPU associated with DB2 sort initialization was avoided. Since this query was executing many thousands of times per day, the CPU savings were substantial.

Our high-consuming reporting program is performing a hybrid join. While we don't frown on hybrid joins, our system statistics were showing us that there were Remote Data Services Relational Data System (RDS) subcomponent failures in the subsystem. We wondered whether this query was causing an RDS failure, and reverted to a tablespace scan. This turned out to be true. We tuned the statement to use nested loop over hybrid join, the RDS failures and subsequent tablespace scan were avoided, and CPU and elapsed time improved dramatically.

While these statements may have not caught someone's eye just by looking at EXPLAIN results, when combined with the accounting data, they screamed for further investigation.

Useful Index Query

```
SELECT I.TBNAME CONCAT '!' CONCAT I.NAME , K.COLNAME , K.COLSEQ
, DECIMAL(I.CLUSTERRATIOF * 100,5,2) AS CRATIO
, INTEGER(I.FIRSTKEYCARDF) AS FI_KEY
, INTEGER(I.FULLKEYCARDF) AS FU_KEY
, INTEGER(C.COLCARDF) AS CCARD , C.COLTYPE , C.LENGTH
, C.SCALE , C.NULLS , K.ORDERING , I.UNIQUERULE
FROM SYSIBM.SYSTABLES T ,SYSIBM.SYSINDEXES I ,SYSIBM.SYSKEYS K
, SYSIBM.SYSCOLUMNS C
WHERE T.NAME IN ('ACCOUNT', 'ACCT_ORDR')
AND T.CREATOR = 'ACCT'
AND I.TBNAME = T.NAME
AND I.TBCREATOR = T.CREATOR
AND K.IXNAME = I.NAME
AND K.IXCREATOR = I.CREATOR
AND C.TBNAME = T.NAME
AND C.TBCREATOR = T.CREATOR
AND C.NAME = K.COLNAME
ORDER BY I.TBNAME, I.NAME, K.COLSEQ
```

Here is a useful index query.

Useful Index Query Results

	COLNAME	COLSEQ	CRATIO	FI_KEY	FU_KEY	CCARD	COLTYPE	LENGTH	SCALE	NULLS	ORDERING	UNIQUERULE
ACCOUNT.CCIXACC0	PRIM_RECORD_ID	1	99.99	33805359	33830794	33805359	DECIMAL	10	0	N	A	U
ACCOUNT.CCIXACC0	ACCT_NBR	2	99.99	33805359	33830794	33830794	DECIMAL	10	0	N	A	U
ACCOUNT.CCIXACU0	ACCT_NBR	1	86.59	33830794	33830794	33830794	DECIMAL	10	0	N	A	P
ACCOUNT.CCIXACU1	SEC_RECORD_ID	1	86.19	12556250	33830794	12556250	DECIMAL	10	0	Y	A	U
ACCOUNT.CCIXACU1	ACCT_NBR	2	86.19	12556250	33830794	33830794	DECIMAL	10	0	N	A	U
ACCT_ORDER.CCIXAOC0	PRIM_RECORD_ID	1	99.95	33918694	33944195	33918694	DECIMAL	10	0	N	A	D
ACCT_ORDER.CCIXAOC0	ACCT_NBR	2	99.95	33918694	33944195	33944070	DECIMAL	10	0	N	A	D
ACCT_ORDER.CCIXAON0	BATCH_DT	1	21.14	6042	1391945	6042	DATE	4	0	N	A	D
ACCT_ORDER.CCIXAON0	BATCH_NBR	2	21.14	6042	1391945	8064	DECIMAL	5	0	N	A	D
ACCT_ORDER.CCIXAOU0	ACCT_NBR	1	94.89	33944070	96046838	33944070	DECIMAL	10	0	N	A	P
ACCT_ORDER.CCIXAOU0	ORDER_NBR	2	94.89	33944070	96046838	22	DECIMAL	3	0	N	A	P

Table Name
and Index
Name

Index Columns
and Sequence

Clustering

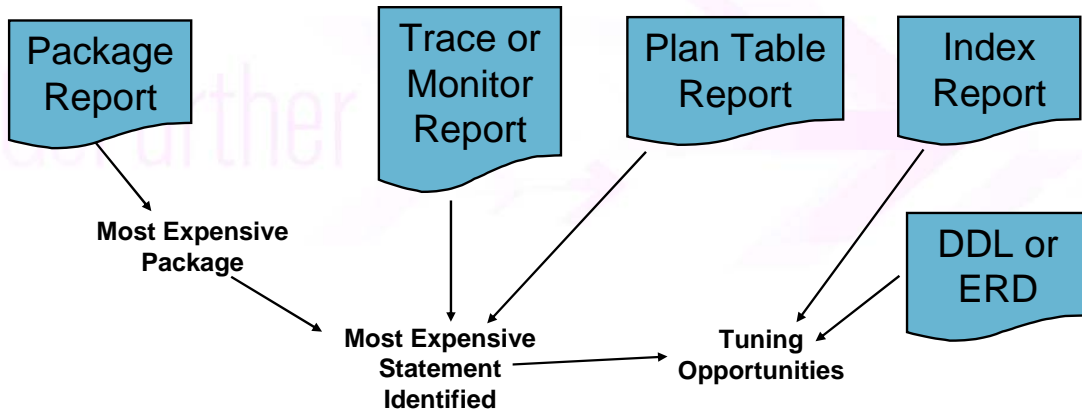
Cardinality

Really Useful Report When Examining
And Tuning SQL!

- This report should include the index name, table name, columns names, columns sequence, cluster ratio, clustering, first key cardinality, and full key cardinality. Use this report when tuning SQL statements identified in the plan table or trace/monitor report. There may be indexes you can take advantage of, add or change.

Result – Tune the Most Expensive Statements First

- All the Information Gathered Comes Together
 - Package Level Summaries
 - Plan Table Information
 - Supporting Index Information
 - Trace or Other Monitor Information



© Copyright 2004, YL&A, All rights reserved.

35



The application performance summary report identifies applications of interest and provides the initial tool to report to management. You can also use these other tools, or pieces of documentation, to identify and tune SQL statements in your most expensive packages:

- Trace or monitor report: You can run a performance trace or watch your online monitor for the packages identified as high consumers in your package report. This type of monitoring will help to drill down to the high-consuming SQL statements within these packages.
- Plan table report: Run extractions of plan table information for the high-consuming programs identified in your package report. You may quickly find some bad access paths that can be tuned quickly. Don't forget to consider the frequency of execution as indicated in the package report. Even a simple thing such as a small sort may be really expensive if executed often.
- DDL or ERD: You're going to need to know about the database. This includes relationships between tables, column data types, and knowing where data is. An Entity Relationship Diagram (ERD) is the best tool for this, but if none is available, you can print out the Data Definition Language (DDL) SQL statements used to create the tables and indexes. If the DDL isn't available, you can use a tool such as DB2LOOK (yes, you can use this against a mainframe database) to generate the DDL.

Capturing Dynamic Statement Information

- Applications Moving Away From Traditional Mainframe Programming
 - The Mainframe DB2 Database is Now the “Ultimate” Database Server
 - The Mainframe is Just Another Machine on the Wire!
- Lots of Dynamic SQL Coming From Distributed Applications
 - How Are These Applications Identified?
 - CORR ID **appl.exe** or javaw.exe or java.exe
 - Authorization ID
 - OR Applications Set Special Accounting Fields
 - ACCTSTR, APPLNAME, USERID, WKIRSTNAME
 - Correlation Id
 - How Do We Capture These SQL Statements?
 - Performance Trace on the Server
 - CLI and/or JDBC Trace on the Client
 - DB2 CLI/ODBC/JDBC Statement Capture - Static Profiling
 - NOT available with V8 universal java driver (IBM says solution is SQLJ)
 - Online Performance Monitor
 - Performance Trace
 - Other Performance Monitor Tool
 - Either by default or add-on feature

© Copyright 2004, YL&A, All rights reserved.

36



Many Web-based or client/server applications being developed today use DB2 on the mainframe as the database server. These types of applications, though capable of using static SQL, are using dynamic SQL to access DB2. This dynamic SQL presents a challenge for DBAs and systems programmers responsible for capturing and reporting on inefficient SQL. This challenge lies in the fact that most of the SQL is accounted for under just a few packages, and so our package-level accounting methodology lumps all these applications and queries together.

However, there are several ways you can capture and isolate accounting information for these dynamic applications. You can use the correlation id to identify the application, although this isn't effective for Java, since the executable is actually the Java virtual machine. You can set specific authorization ids for each application, and then further itemize your package-level information by application using those ids. You can also use special DB2 accounting fields (e.g., ACCTSTR) that the applications can set when they establish their connections (your reporting program must be able to read these newer fields).

Issues With Dynamic SQL

- How Can We Make Our Distributed Dynamic SQL Faster?
 - Use Parameter Markers Instead of Literals
 - “Almost” Always Better
 - Turn on Dynamic Statement Cache
 - Parameter Markers Help
 - Use a Three Tier Architecture
 - Fixed Authorization ID, and Application Security
 - Reuse Database Connections – Connection Pooling
 - Consolidate Code Into Stored Procedures
 - Multiple SQL Statements per Procedure
 - Includes Some Application Logic
 - Return Only Results
 - DB2 CLI/ODBC/JDBC Statement Capture – Static Profiling
 - Allows Dynamic SQL to be Static Bound SQL
 - Parameter Markers Help

Techniques for capturing dynamic SQL statements include running a performance trace or a Call Level Interface (CLI) or Java DataBase Connectivity (JDBC) trace. Each technique will get you the statement text and response time; the performance trace will also report on CPU consumption, getpages and index access, etc. Online performance monitors are also useful, but more dependent upon timing to actually view the SQL statement executing. This can be difficult to do if there are many statements that run rather quickly.

Another lesser-known option is to use the DB2 CLI/ODBC/JDBC static profiling feature. Although this feature's primary purpose is to capture SQL statements for binding into the database as static, it will capture all SQL statements issued during the time it's set in capture mode. A DB2 command is used to bind the statements into a package, and the EXPLAIN(YES) bind option can be used to generate access path information.

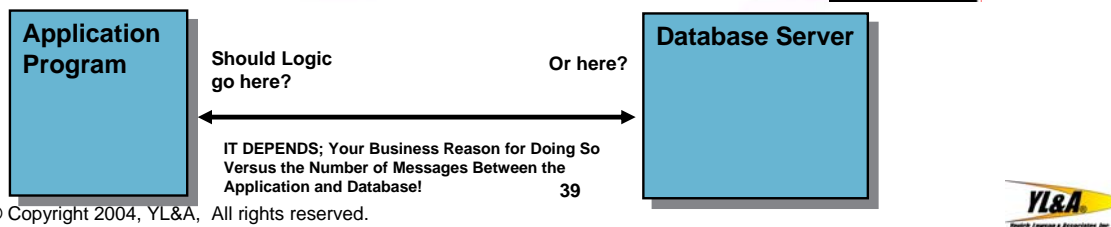
More Issues with Dynamic SQL - Linux

- Evolving from Traditional OS to Linux
 - More Transportable
 - Cheaper?
 - We Now Have zLinux for Mainframe
- Moving z/OS Application to Linux?
 - Now a Distributed Dynamic SQL Application
 - Even from zLinux
 - Certain Trade-Offs for Remote Connection
 - DDF CPU Overhead
 - No RELEASE(DEALLOCATE) Packages
 - Lose Benefits of This Setting Across Commits
 - IPROCS, SPROCS, UPROCS
 - Sequential Detection
 - Index Look-Aside
 - Not Good for Batch Processing
 - High Performance DASD Subsystems Can Help!
 - Detect Sequential Access and Pre-Stage Data
 - Large Cache Sizes (Current 64GB) Help Avoid Actual I/O's

Be careful if you're trying to move your high performance application off of the mainframe z/OS operating system. That is, there are costs to pay if you move from a local application to remote.

Complex Database Objects

- You Add Complexity to the Database for a Reason
 - Centralized Logic (Reusable Code)
 - Faster Time to Delivery (Easier to Program)
 - Performance
 - Sometimes!
- Complex Objects Include
 - Database Enforced RI and Check Constraints
 - Complex SQL
 - User-Defined Functions
 - Triggers
 - Stored Procedures



© Copyright 2004, YL&A, All rights reserved.

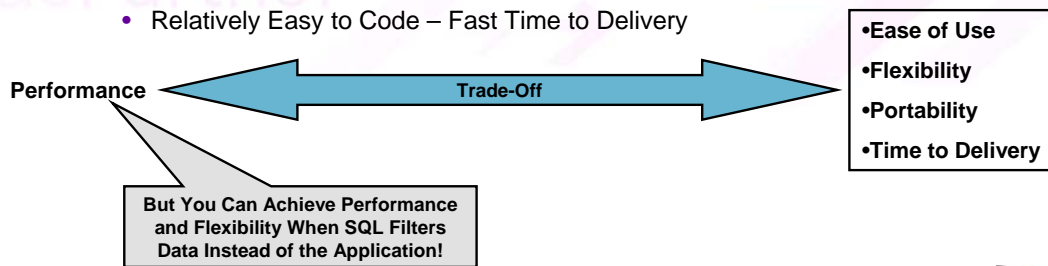
There are many advanced features of DB2 for z/OS that let you take advantage of the power of the mainframe server:

These advanced features facilitate rapid application development by pushing some of the logic of the application into the database server. Usually, advanced functionality can be incorporated into the database using these features at a much lower development cost than coding the feature into the application itself. A feature such as database-enforced Referential Integrity (RI) is a perfect example of something that's easy to implement in the database, but would take significantly longer time to code in a program.

These advanced database features also let you place application logic as part of the database engine itself, making this logic more easily reusable. Reusing existing logic will mean faster time to market for new applications that need that logic; having the logic centrally located makes it easier to manage than client code. Often, having data-intensive logic located on the database server will result in improved performance as that logic can process the data at the server, and only return a result to the client.

Complex Objects Trade-Offs

- Complex SQL
 - Performance Improvement
 - Data is Filtered or Aggregated
 - Transactions Process Little Data
 - One Large Query Instead of Many Small Queries
 - Business Rules Pushed Towards Data
 - Set Processing or Filtering
 - Little or No Logic in SELECT List
 - Avoid UDFs, CASE Expressions, Data Conversions
 - Flexibility
 - SQL is Highly Portable
 - Relatively Easy to Code – Fast Time to Delivery



© Copyright 2004, YL&A, All rights reserved.

40



The reusability, flexibility, and portability of complex SQL doesn't come without a performance price. Complex SQL can be an extreme performance advantage if the program processes written into the SQL statement aggregate or filter data. If the complex SQL statement logic is data-intensive, then it will be a performance gain over the equivalent COBOL or Java logic. If the application program can issue one large statement that returns a result rather than many smaller statements that return data, it will gain all the advantages that complex SQL has to offer and a performance advantage. If a complex SQL statement processes data, rather than filters it, then it can be a performance detriment. SQL statements that use lots of UDFs, Computer-Aided Software Engineering (CASE) expressions, and data conversions in the SELECT clause may impact performance. Nested table expressions that contain expressions in a SELECT clause and then have references to those nested expressions reused in outer expressions may also significantly impact performance.

So, complex SQL statements can be a performance advantage or disadvantage. Data-intensive logic is almost always a performance advantage; data-processing logic is almost always a performance disadvantage. The trade-off is being able to balance the performance with the reusability, flexibility, and portability of complex SQL,

Complex Objects Trade-Offs

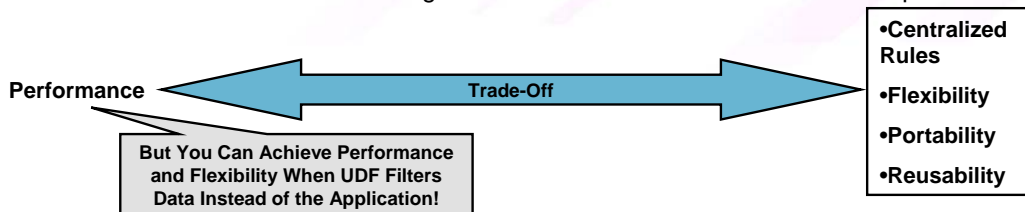
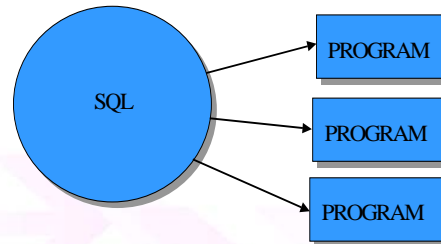
- User-Defined Functions

- Performance Improvement

- Used to Access Legacy Data Stores
 - Perform Complex Business Rules
 - Set Processing or Filtering

- Flexibility

- SQL is Highly Portable
 - Centralize Various Business Rules
 - SQL Scalar UDFs Have Casting Overhead
 - External UDFs Have Task Switch Overhead
 - Deterministic Setting Affects Performance Of Nested Table Expressions



© Copyright 2004, YL&A, All rights reserved.

41



UDFs provide a major breakthrough in database programming technology; they actually let developers and DBAs extend the capabilities of the database. This allows for more processing to be pushed into the database engine, which allows these types of processes to become more centralized and controllable. Virtually any type of processing can be placed in a UDF, including legacy application programs. This can be used to create some amazing results, as well as push legacy processing into SQL statements.

Once your processing is inside SQL statements, you can put those SQL statements anywhere. So that anywhere you can run your SQL statements (say, from a Web browser), you can run your programs! So, just like complex SQL statements, UDFs place more logic into the highly portable SQL statements.

Also just like complex SQL, UDFs can be a performance advantage or disadvantage. If the UDFs process large amounts of data and return a result to the SQL statement, they may be a performance advantage over the equivalent client application code. However, if a UDF is used to process data only, then it can be a performance disadvantage, especially if the UDF is involved invoked many times or embedded in a table expression, as data type casting and task switch overhead are expensive (DB2 V8 relieves some of this overhead). Converting a legacy program into a UDF in about a day's time, invoking that program from an SQL statement, and then placing that SQL statement where it can be accessed via a client process may just be worth that expense.

UDFs for Legacy Data Access

- Use a Query to Read a Table
 - Use Table Data to Invoke a Legacy Process
 - OR Just the Table Function
 - Return Everything in a Query!
- Put That Query Anywhere!
 - WEB Enable Your Legacy Processing in Days

```
SELECT *
FROM TABLE (CALCUDF()) AS LEGACY_DATA;
```

```
SELECT      TAB2.COLA, TAB2.COLB, TAB1.COL1
FROM        (SELECT      COL1, COL2, COL3, COL4
              FROM FINANCETLB
              WHERE      ACCT_ID = 123) AS TAB1
            ,TABLE (CALCUDF(TAB1.COL2,
                            TAB1.COL3, TAB1.COL4))
              AS TAB2;
```

42

© Copyright 2004, YL&A, All rights reserved.



You can make a correlated reference from within the invocation of a table function. This allows you to send data into a table UDF (the same could be done for a scalar UDF in a correlated nested table expression) from another table.

This allows for some wonderful and amazingly sophisticated processing. You can basically take some of your current or legacy programs, and use this technique to embed them into SQL statements.

I know...I've done it!

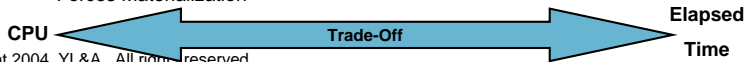
UDF and Expression Impact on NTE's and Performance

- User Defined Functions Can Change Your Access Path
 - We Can Use This to Increase Performance, or Kill It!
 - Your Results May Vary
- This Has to do With "DETERMINISTIC" or "NOT DETERMINISTIC" Functions

```
SELECT    total_months1 - total_months2,
          total_months1, total_months2
FROM
  (SELECT  TOTMON(rdate1, rdate2) as total_months1,
          TOTMON(rdate3, rdate4) as total_months2
   FROM    YLA.DATE_TABLE) AS TAB1
```

**Both total_months1
and total_months2
Referenced Twice
Here**

- If TOTMON is Deterministic, Then It is Executed 4 Times
 - NTE is Merged with Outer SELECT
- If TOTMON is Not Deterministic, It is Executed 2 Times
 - BUT, TAB1 is Materialized First!
- This May Not be a Big Deal in this Example, but is HUGE for Complex Queries
- This Applies to all UDFs, SQL Based or External!
 - Also to ANY Expression (e.g. CASE, CAST, ETC.)
 - RAND() is Not Deterministic
 - You Can Use the RAND() Function as a Performance Tuning Tool!
 - Forces Materialization



© Copyright 2004, YL&A, All rights reserved.



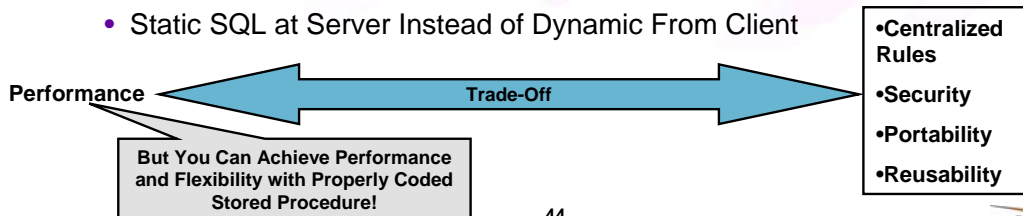
SQL sourced UDFs can give you some performance surprises. In particular, defining a UDF as deterministic can affect performance.

A deterministic function is a function that returns the same value if the same values are passed to it, like a SUBSTR function. A non deterministic function is one that can return different results even if passed the same values, like a RAND function.

DB2 will merge the expression represented by a deterministic function if it is nested in a table expression, as long as the nested expression does not have to be materialized by DB2. A non deterministic function will force a nested table expression to be materialized. This can lead to a variety of SQL performance issues.

Complex Objects Trade-Offs

- Stored Procedures
 - Performance Improvement (for Remote Clients)
 - Used to Access Legacy Data Stores
 - Business Logic and Multiple SQL Statements Encapsulated in Stored Procedure
 - Filtering and Set Processing
 - Only Result Returned to Client
 - Flexibility
 - Centralized Business Rules
 - Security
 - Availability
 - Static SQL at Server Instead of Dynamic From Client



© Copyright 2004, YL&A, All rights reserved.

44

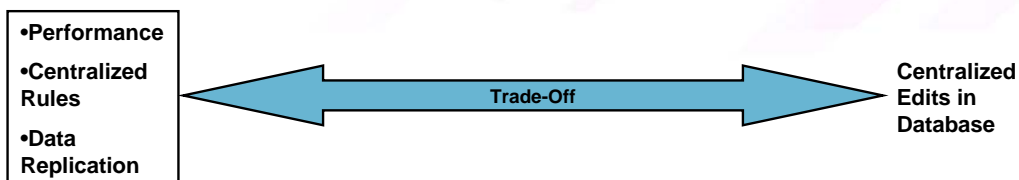


The major advantage to stored procedures occurs when they're implemented in a client/server application that must issue several remote SQL statements. The network overhead involved in sending multiple SQL commands and receiving result sets is significant, so proper use of stored procedures to accept a request, process that request with encapsulated SQL statements and business logic, and return a result will lessen the traffic across the network and reduce the application overhead.

If a stored procedure is coded in this manner, then it can be a significant performance improvement. Conversely, if the stored procedures contain only a few or one SQL statement, the advantages of security, availability, and reusability can be realized, but performance will be worse than the equivalent single statement executions from the client due to task switch overhead.

Complex Objects Trade-Offs

- Triggers and Constraints
 - Performance When Used Properly
 - Push Data Intense Application Logic to Database
 - Proper Indexes a Must
 - NOT a Replacement for Edits
 - Edits Best Done in Application Code
 - Flexibility
 - Centralized Business Rules
 - Replication of Data
 - Audit and Historical (Transaction History)



Triggers and constraints ease the programming burden because the logic, in the form of SQL, is much easier to code than the equivalent application programming logic. This helps make the application programs smaller and easier to manage. In addition, since the triggers and constraints are connected to DB2 tables, they're centrally located rules and universally enforced. This helps to ensure data integrity across many application processes. Triggers can also be used to automatically invoke UDFs and stored procedures, which can introduce some automatic and centrally controlled application logic.

There are advantages to using triggers and constraints. They certainly provide for better data integrity, faster application delivery time, and centrally located reusable code. Since the logic in triggers and constraints is usually data-intensive, they typically outperform the equivalent application logic simply because no data has to be returned to the application when these automated processes fire. There's one trade-off for performance, however. When triggers, RI or check constraints are used in place of application edits, they can be a serious performance disadvantage. This is especially true if several edits on a data-entry screen are verified at the server. It could be as bad as one trip to the server and back per edit. This would seriously increase message traffic between the client and the server. For this reason, data edits are best performed at the client when possible.

Final Note on Complex Objects

- We Promote the Creation of a Procedural DBA and/or Programming Group
 - Central Control of all Database Program Objects
 - Can Monitor and Control Stored Procedures, UDFs, and Triggers
 - Advanced SQL
 - Referential Integrity and Constraints
 - Can Maintain a “Story Board” of Relationships of Objects
 - Objects Can be Nested
- Without This Group
 - Stronger Service Level Agreements
 - Especially Between Database Administrators and Application Developers
- Database Management Tools Become Critical
 - Performance Monitors
 - Administration Tools

Advanced SQL, UDFs, stored procedures, triggers, and constraints are all powerful database features that can be used to add more control, flexibility, reusability, and security into your database engine. They allow business logic that's data-intensive to be placed close to the data, which can provide a significant performance boost. These advanced database features also introduce more implementation choices for security, location of application code, availability, and flexibility. Centralized reusable objects improve time to delivery of new applications, and UDFs and stored procedures allow for legacy programs to be placed into SQL and rapidly made Web-available. Properly implement these features to ensure the best performance. In situations where these choices result in less than optimal performance, the business case for implementing these advanced features needs to be weighed against that cost.

Session B13

DB2 for z/OS Design and Tuning Tips for Your Complex VLDB

Daniel L. Luksetich

Yevich, Lawson, and Associates

Dan_Luksetich@YLIASSOC.COM

Dan_Luksetich@DB2EXPERT.COM



If anyone tells you they are a DB2 expert they are lying. The breath of its features are too wide for one to master all of it!