www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 6 Issue 8 August 2017, Page No. 22259-22275
Index Copernicus value (2015): 58.10 DOI: 10.18535/ijecs/v6i8.07

Explain the 'Explain'

Dr SP Rahmesh[1], Uthresh Kumar[2], Thamilselvi[2], Abila Jacqulin[2], Aarthi Rekha[2], Buvaneshwari[2]

Senior Project Manager, HCL Technologies [sulur.rahmesh@ thryvedigital.com],
Uthresh.Neelakandan@thryvedigital.com
Sureshkumar.Thamilselvi@thryvedigital.com
Abila.JacqulinSp@thryvedigital.com
Aarthi.Rekha@thryvedigital.com
Buvaneswari.Rajagopal@thryvedigital.com
Thryve Digital LLP, Chennai

Abstract

The ultimate objective and focus of IT Shops today is to bring down the cost of operations and to run the business efficiently. High cost element to maintain the application and database is measured using the measure called MIPS [millions of instructions per second] and is calculated based on MIPS usage by CPU. In this paper we investigate in detail the causes behind high usage of CPU and the ways in which the MIPS Usage can be curtailed and the cost for it. This paper deals with the universal database DB2 which runs on mainframe MVS operating system. As all the older legacy systems were developed on mainframe using COBOL & DB2 sub-systems, re-engineering the applications is needed to bring the cost down. This paper discusses various sub topics on why and how the MIPS can be brought down with no impact to the application and business. Programs written with poor readability in an unstructured manner and without cost consciousness are the key factors causing performance issues now-a-days.

Optimizer, a component which decides the optimal access path to fetch data from database is based on statistical data captured in system tables. By adopting various approaches and disciplines outlined in this paper surely would guide application programmers to script their queries appropriately and as a result, organization's objectives and goals can be met. In computing, optimization is the process of modifying a system to improve its efficiency where the system can be a single entity or a collection of entities.

Keywords: Predicates, MIPS, Indexes, CPU, Optimizer, Explain, SQL, Programming logic

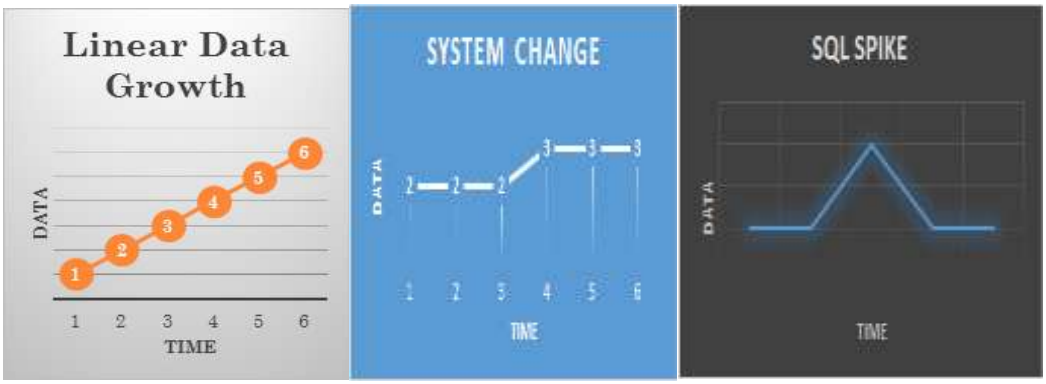
What causes DB2 performance issues?

Usual suspects are

- Organic growth of data [linear in nature] - Size and volume of data in database grows gradually and over a period time [2 or 3 decades], the application programs and query statements in the programs perform at low level as it has to handle huge volume of data.
- Workload – [addition of new applications / functionality or new installations / instances] – As the business scenarios and complex requirements keep growing, the application programs have become

like Spaghetti. Application portfolio studies have become hard due to such multifarious scenarios. Also the emergent needs of customers, instances of the same application is being installed at various locations to cater different geographies. This adds more complexities beyond control.

- Changes to the system classifications are
  - Systematic changes [changes to operating system] – Changing technologies and innovations introduced operating system changes on a regular basis. Modifications in Architecture is happening endlessly for security purposes
  - Application modifications – Due to business, functional & legal obligations, applications growing in size incessantly.
- Spikes due to one off execution of SQL statements causing usage of higher MIPS and unplanned outage of system resources. Due to some untoward incidents, SQL statements might take excess CPU Time causing delay



- Asleep at the wheel, which means irregular execution of utilities like RUNSTATS, REORG. Basically DB2 loves to statistics and performs all its activities and makes decisions only based on the statistical information that gets recorded in system tables for any and every change that happens in the DB2 sub system. Optimizer the core component of DB2 decides the access path to data for every SQL based on the statistical information available in the tables. To ensure the statistical data available in the system tables are unswerving, DB2 utility RUNSTATS which serves this purpose. Regular execution of this utility will ensure the Statistical data are updated in tables. Similarly the REORG utility organizes the data in the table, indexes in all table spaces. As the I/O operations are heavy in all business applications, this utility arranges data in a disciplined manner which advances the performance of the DB2 sub system.

How to analyze performance issues with DB2

Activities on exceptions to look into, are listed below

Look for number of times the issue has happened – Examine the problem whether is persistent or a one-off incident. If it's a rare occurrence look for system or operating environmental parameters to understand the issue. Many times, such scenarios may not warrant a solution. But for regular issues,

it's essential to analyze the complete cycle of events, availability of system resources, conflicts, deadlock or issues with access paths for the SQL or anything to do with DB2 objects in particular. The analysis of output from Explain command would serve plenty of information for deep-dive analysis and investigation

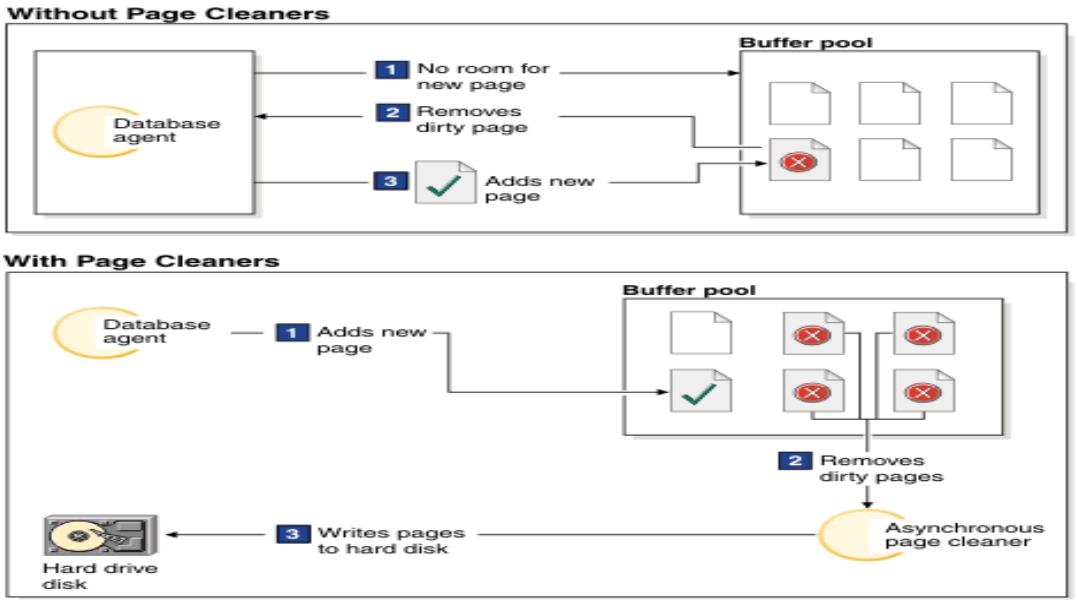
Connection with DB2 issues – Validate the connection with DB2 sub system to avoid errors around connectivity

Page cleaning – [Ref: IBM knowledge center]

In a well-tuned system, usually the page-cleaner agents that write changed or dirty pages to disk. Page-cleaner agents perform I/O as background processes and allow applications to run faster because their agents can perform actual transaction work. Page-cleaner agents are sometimes referred to as asynchronous page cleaners or asynchronous buffer writers, because they are not coordinated with the work of other agents and work only when required independently.

To improve performance for update-intensive workloads, one might prefer to enable proactive page cleaning, whereby page cleaners behave more proactively in choosing which dirty pages get written out at any given point in time. This is particularly true if snapshots reveal that there are a significant number of synchronous data-page or index-page writes in relation to the number of asynchronous data-page or index-page writes.

Figure below explains page cleaning process. Asynchronous page cleaning. Dirty pages are written out to disk.



Sort Pool – [Is sort happening more or less?] – Sorting is an internal to DB2 activity happens on various scenarios based on the way the SQLs are written. Based on the business needs, the query can be written but with all cautions. Because the internal sort operations, take too much of time to complete the query. For example, UNION ALL is a key parameter which combines the results of two different queries without doing any sorting & duplicate checking but UNION removes the duplicates in the

output. It's programmer's responsibility to decide appropriate parameters in such a way to bring the CPU time down.

Have a holistic view of the system to inch close to the root cause and to look for optimum solution. Expectations of the IT Shops is to have a highly stable system to support their business. DBAs expect no surprises and wants up front communications regarding the changes that happens around the application and the environment. It's DBAs responsibility to ensure stability of DB2 sub system on an overall assessment

1 - Analyzing Programming Logic

A Programmer's analysis should naturally focus on identifying the problem prone code and to look for permanently arresting it from happening again. An important step in analyzing the DB2 performance related issues is to find the query which consumes higher CPU and locks resources. Firstly, isolate the problem to the smallest number of factors as possible. Before starting the analysis of the SQL or database request, one should understand that other variables that are involved in the total response time are

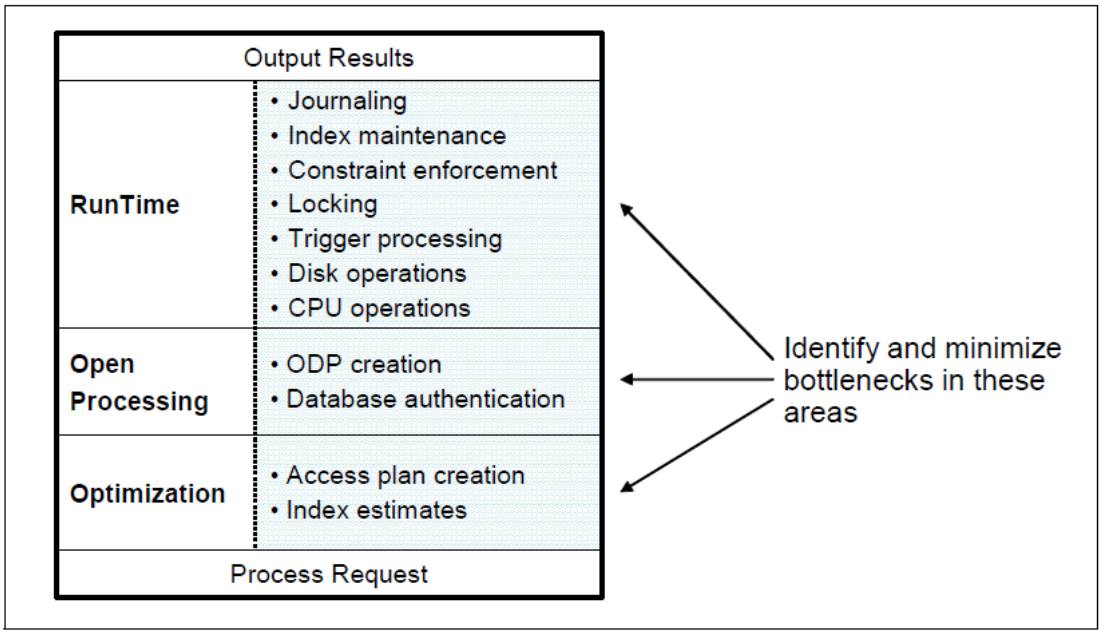
- User display I/O
- Authentication
- Data processing
- Communications

Once the query causing the issue is identified, understand what it takes to execute a DB2 query. Upon the execution of a DB2 query, three main operations around are

- Optimization time
- Open processing time
- Run time

It can be found from the diagram below that which operations affect optimization time, open processing time, and run time. It is paramount important to identify and minimize the bottlenecks in the above three main areas of the processing of any DB2 query to Improve the Performance. Programmers should be empowered to rewrite the query, re-design the programming logic and flow to ensure the improvements in efficacy.

DOI: 10.18535/ijecs/v6i8.07



2 - Pruning Select List

SQL statements have a major impact on database performance, hence it's important to understand how SQL statement works, optimizer chooses shortest access path, indexes used and etc. Select list is one among the vital statement influences the consumption of CPU time and performance.

Some important basic best practices that can be followed while writing a select statement in DB2 are listed below:

- *Disallow SELECT * in application programs*

  SELECT only the rows or columns needed in the program and no extra column should be selected as it avoids space, load

- *Do not SELECT columns with known static values*

  Disallow Condition from SELECT list if Condition = Value

- *Make WHERE clause data types match*

  Costs more to navigate indexes when data types are not matching

- *Avoiding Joins, Distinct and complex functions to improve query optimization*

  The DB2 optimizer might not be able to efficiently run SQL statements that contain non-equality join predicates, data type mismatches on join columns, unnecessary left or right outer joins and other complex search conditions

- *Optimizing by fetching what is needed*

  Consider defining unique, check and referential integrity constraints. These constraints provide semantic information that allows the DB2 optimizer to rewrite queries to eliminate joins, push FETCH FIRST n ROWS down through joins. Use OPTIMIZE FOR Clause requests special of the select statement

- *Usage of literals in right place*

Use constants and literals if the values will not change and this increases optimizer's accuracy using statistics

- *Avoid using functions in predicates.*

  The index is not used by the database if there is a function on the column.

  For example:

  *SELECT * FROM TABLE1 WHERE UPPER (COL1) ='ABC'*

  As a result of the function UPPER (), the index on COL1 is not used by the optimizer. If the function cannot be avoided in the SQL, create a function-based index in Oracle or generated columns in DB2 to improve performance

- *Avoid Scalar functions*

  Never put SQL scalar functions on columns in the WHERE clause predicates.

  For example: WHERE YEAR (HIREDATE) = 2009 should be recoded as WHERE HIREDATE BETWEEN '2009-01-01' and '2009-12-31'. When placing SQL scalar functions on columns in the SELECT portion of the SQL statement does incur some minimal overhead, but applying it to a column in the WHERE clause causes the predicate to become stage 2 non indexable

["MIPS & Salaries are the highest unit cost per month" by Nancy White – Former CEO – Certegy Corp.]

- *Usage of host variables for mathematics*

  Performing mathematics on the left predicates should be avoided.

  For example: WHERE HIREDATE – 7 DAYS > :HV-DATE should be recoded as WHERE HIREDATE > :HV_DATE + 7 days, As Having the mathematics on the host variable won't cause a problem.

- *Existence checking queries*

  To check existence of rows in tables select count (*) is being used which is inefficient. Instead Use select 1 into host variable along with Fetch First 1 Row only Condition, which reduces the runtime ultimately

3 - Minimize Trips to DB2

Number of times database is queried to fetch or manipulate the data should be minimized to the extent possible, as every trip to DB2 adds cost to operations. Few tips are listed below to achieve this objective.

- Identify the loops / trips / threads to database to make it as minimal as possible
- Ensure right programming methodologies are applied with all in-built functions utilized
- Use the Bind parameters / options appropriately for row processing
- Use in-built data storage options [use of multi-dimensional arrays] to bring down the I/O operations
- Use multi-row fetch options to avoid singleton fetch operations & look at options to move SQLs written outside perform loops, wherever possible.
- Avoid cursors inside another cursor which is using the same set of tables. Need to ensure DB2 latches or uncommitted reads are supporting this scenario
- Use default options while insert time for unknown fields [avoid using host variables for defaults]
- Use NO UPDATE options while updating fields
- Move – 7 to Null Indicator field [when no change, skip column]
- Move – 5 to Null indicator field [when defaults to be used]
- For mass inserts or updates into Database, use the load utilities. While doing mass operations, drop the indexes and re-create the index objects.
- Use options like a) for Fetch Only in Cursors, b) Optimize for n Rows, c) Fetch First n Rows Only d) Use 'Stored Procedures' f) Reduce network traffic
- Use the NULL indicators to systemic treatment of nulls to satisfy CODD Rules
- Use commits in application all programs performing heavy I/O operations should in-built restart logic / strategy in ABEND situations. Recommend to have DB2 restart table containing a package name, last key rules, commit frequency and etc

4 - Predicates stage 1, 2 & 3

A *predicate* specifies a condition that is true, false, or unknown about a given value, row, or group. The type of a predicate depends on its operator or syntax. And it can be organized into the following types:

- Sub query predicate
- Equal predicate
- Range Predicate
- IN predicate
- Not Predicate

Predicates can be grouped into four categories that are determined by how and when the predicate is used in the evaluation process. Applying the predicates for the SQL statements are in stages as in the following list, ordered in terms of performance starting with the most favorable: .

- Indexable - predicate can match index entries

- Non- Indexable - predicates that cannot match index entries
- Stage 1 - Predicates that can be applied during the first stage of processing are called *Stage 1 predicates*. These predicates are also sometimes said to be *sargable [an IBM term meaning Searchable Argument]*
- Stage 2 - predicates that cannot be applied until the second stage of processing are called *stage 2 predicates*, and sometimes described as *non sargable* or *residual* predicates.

The STAGE column of DSN_FILTER_TABLE indicates the stage at which a predicate was applied. The stage 2 predicates are applied on the returned data rows. The predicate might be indexable and stage 1, if both sides contain the same data type. Otherwise, the predicate is stage 2. The following sets of conditions make the predicate as stage 2:

- The syntax of the predicate
- Data type and length of constants or columns in the predicate. A simple predicate whose syntax classifies it as indexable and stage 1 might not be indexable or stage 1 because of data types that are associated with the predicate. For example, a predicate that is associated with either columns or constants of the DECFLOAT data type is never treated as stage 1. Similarly, a predicate that contains constants or columns whose lengths are too long also might not be stage 1 or indexable.
- Whether DB2 evaluates the predicate before or after a join operation. A predicate that is evaluated after a join operation is always a stage 2 predicate.
- Join sequence -The same predicate might be stage 1 or stage 2, depending on the join sequence. Join sequence is the order in which DB2 joins tables when it evaluates a query. The join sequence is not necessarily the same as the order in which the tables appear in the predicate. The join sequence can be determined by executing EXPLAIN on the query and examining the resulting plan table.

There are tremendous performance benefits to be gained when the query can be processed as Stage 1 or indexable rather than Stage 2. Here, the stage 2 and stage 1 predicates are listed.

Stage 1 Predicates	Stage 2 Predicates
COL = value	value BETWEEN COL1 AND COL2
COL = noncol expr	COL BETWEEN COL1 AND COL2
COL IS NULL	value NOT BETWEEN COL1 AND COL2
COL op value	value BETWEEN col expr and col expr
COL op noncol expr	T1.COL <> T2.COL
COL BETWEEN value1 AND value2	T1.COL1 = T1.COL2
COL BETWEEN noncol expr 1 AND noncol expr 2	T1.COL1 op T1.COL2

COL BETWEEN expr-1 AND expr-2	T1.COL1 <> T1.COL2
COL LIKE 'pattern'	COL = ALL (noncor subq)
COL IN (list)	COL <> (noncor subq)
COL IS NOT NULL	COL <> ALL (noncor subq)
COL LIKE host variable	COL NOT IN (noncor subq)
COL LIKE UPPER ('pattern')	COL = (cor subq)
COL LIKE UPPER (host-variable)	COL = ALL (cor subq)
COL LIKE UPPER (SQL-variable)	COL op (cor subq)
COL LIKE UPPER (CAST ('pattern' AS data-type))	COL op ANY (cor subq)
COL LIKE UPPER (CAST (host-variable AS data-type))	COL op ALL (cor subq)
COL LIKE UPPER (CAST (SQL-variable AS data-type))	COL <> (cor subq)
T1.COL = T2.COL	COL <> ANY (cor subq)
T1.COL op T2.COL	(COL1,...COLn) IN (cor subq)
T1.COL = T2 col expr	COL NOT IN (cor subq)
T1.COL op T2 col expr	(COL1,...COLn) NOT IN (cor subq)
COL = (noncor subq)	T1.COL1 IS DISTINCT FROM T2.COL2
COL op (noncor subq)	T1.COL1 IS DISTINCT FROM T2 col expr
COL = ANY (noncor subq)	COL IS NOT DISTINCT FROM (cor subq)
(COL1,...COLn) IN (noncor subq)	EXISTS (subq
COL = ANY (cor subq)	expression = value
COL IS NOT DISTINCT FROM value	expression <> value
COL IS NOT DISTINCT FROM noncol expr	expression op value
T1.COL1 IS NOT DISTINCT FROM T2.COL2	expression op (subq)
T1.COL1 IS NOT DISTINCT FROM T2 col expr	NOT XMLEXIS
COL IS NOT DISTINCT FROM (noncor subq)	

Some example queries below to see the effect of rewriting the stage 2 SQL to stage 1 SQL.

Original predicate or query	**Optimized predicates**	**Comments**
c1 between 5 and 10	c1 ≥ 5 and c1 ≤ 10	The BETWEEN predicates are rewritten into the equivalent range delimiting predicates so that they can be used internally as though the user specified the range delimiting predicates.
c1 not between 5 and 10	c1 < 5 or c1 > 10	The presence of the OR predicate does not allow the use of a start-stop key unless the DB2 optimizer chooses an index-ORing plan.

DOI: 10.18535/ijecs/v6i8.07

Original predicate or query	Optimized predicates	Comments
SELECT * **FROM** t1 **WHERE EXISTS** (**SELECT** c1 **FROM** t2 **WHERE** t1.c1 = t2.c1)	**SELECT** t1.* **FROM** t1 EOJOIN t2 **WHERE** t1.c1= t2.c1	The sub query might be transformed into a join.
SELECT * **FROM** t1 **WHERE** t1.c1 **IN** (**SELECT** c1 **FROM** t2)	**SELECT** t1* **FROM** t1 EOJOIN t2 **WHERE** t1.c1= t2.c1	This is similar to the transformation for the EXISTS predicate example in the previous row.
c1 like 'abc%'	c1 ≥ 'abc X X X ' and c1 ≤ 'abc Y Y Y'	If we have c1 as the leading column of an index, DB2 generates these predicates so that they can be applied as range-delimiting start-stop predicates. Here the characters X and Y are symbolic of the lowest and highest collating character.
c1 like 'abc%def'	c1 ≥ 'abc X X X ' and c1 ≤ 'abc Y Y Y' and c1 like 'abc%def'	This is similar to the previous case, except that we have to also apply the original predicate as an index Sargable predicate. This ensures that the characters definite match correctly.

Stage 3: Application filtering

These are the predicates that are returned to an application and the application determines whether to keep the row or not. These stage 3 predicates are not an actual category described under DB2 but from application program's filtering logic. Program logics and filtering techniques used are very much part of stage 3 predicates.

5 - Verifying local filtering sequence

When writing an SQL statement with multiple predicates, determine the predicate that will filter out the most data from the result set to place that predicate at the start of the list. By sequencing the predicates in this manner, the subsequent predicates will have less data to filter. The DB2 optimizer by default will categorize the predicate and process as detailed below.

- DB2 always applies index predicates first, following the order in which the index is created. It's a good practice to keep the order of predicates as in index.
- Stage 1 non-index predicates are applied in the following order:

  1. Equal predicates 2. Range predicates 3. In-list and like predicates
- Stage 2 predicates are applied

However, if SQL query presents multiple predicates that fall into the same category, these predicates will be executed in the order that they are written. So, it is important to sequence the predicates placing the predicate with the most filtering at the top of the sequence.

1. The most-filtering predicates should be coded before the least-filtering predicates. The most restrictive condition should be listed first, so that extra processing of the second condition can be eliminated.

2. Code SQL with AND predicates coded with most-filtering predicates before least-filtering predicates and OR predicates coded the opposite way, with least-filtering predicates coded before most-filtering predicates.
3. Code join predicates first, followed by local predicates (predicates on a single table) in the same as the named tables appear in the FROM clause.
4. The order of predicate filtering is mainly dependent on the join sequence, join method, and index selection. The order the predicates physically appear in the statement only come into play when there is a tie with one of the above listed categories.

6 – Table Joins

A join statement in SQL relates, associates, or combines multiple table structures together in several different ways, producing a resultant single tabular structure. Join using ON options are supposedly more efficient that joins with conditions as WHERE parameters. It is possible to replicate ON functionality via WHERE implementation, but it is much slower.

- For LEFT OUTER JOIN, pushing predicates of the right table from the WHERE clause into the ON condition helps the database optimizer to generate a more efficient access path. Predicates of the left table can stay in the WHERE clause.

- Similarly, for the SQL queries with the RIGHT OUTER JOIN, predicates for the right table should be moved from the WHERE clause into the ON condition.

Imagine a DB server trying to join two tables, the join condition is specified as a WHERE clause, then the DB Cross joins (union) all results into memory and then removes the combinations that don't pass the filters. An ON condition can be processed during that first CROSS JOIN itself and results in smaller memory use and fewer passes over the dataset.

There will be a difference in **LEFT OUTER JOIN ON** condition and a **WHERE** condition. The ON condition stipulates which rows will be returned in the join, while the WHERE condition acts as a filter on the rows that were returned. While performing a LEFT OUTER JOIN, make sure that any filter conditions on the table are in the **ON** clause, not the WHERE clause. Right and left outer joins are functionally equivalent. Neither provides any functionality that the other does not, so right and left outer joins may replace each other if the table order is switched.

Join performs better than sub query because it does not have to do a correlated evaluation, on a row-by-row basis. In most cases JOINs are faster than sub-queries and it is very rare for a sub-query to be faster. In JOINs RDBMS can create an execution plan that is better for query and can predict what data should be loaded to be processed which saves time, unlike the sub-query where it will run all the queries and load all their data to do the processing.

7 Access Path

DOI: 10.18535/ijecs/v6i8.07

An access path through which the data from DB2 travels back to the application program to satisfy the SQL statements. It specifies the indexes, tables, access methods, order in which objects are accessed and etc. DB2 selects the access paths static SQL statements when application program is bound or rebound into a package while DB2 selects the access paths for dynamic SQL statements when the statements are issued. The DB2 optimizer can choose from a variety of different techniques as it creates optimal access paths for each SQL statement. These techniques range from a simple series of sequential reads to much more complicated strategies such as using multiple indexes to access data based on the statistical data recorded in the tables.

To select efficient access paths, DB2 relies on the following elements:

- Queries that use effective predicates
- Indexes that support efficient data access
- Statistics that describe the data sufficiently and accurately

Analyzing access path

Performance of queries can be assessed using appropriate current statistics that are available in the database objects referenced by an SQL statement.

To investigate access path complete the following investigations:

1. Check the **accuracy and completeness of statistics** for the objects in the SQL statement. Inaccurate statistics often result in inaccurate access paths chosen
   - Invoke the REORG utility to reorganize the DB2 objects and to refresh the content. By Invoking the DSNACCOX stored procedure to determine when to execute REORG is needed
   - Invoke the RUNSTATS utility to capture statistics
2. Execute **EXPLAIN** statement to capture access path information
3. Check whether **indexes** are used and **how many matching columns** are used. The fastest way to access DB2 data is to use possible indexes. Indexes are structured in such a way as to increase the efficiency of finding a particular piece of data. It is essential to have limited but effective indexes as having more indexes lessens the performance and adds cost. When indexes are not defined properly, the selectivity of indexes would be low and it's paramount important to ensure higher selectivity of the indexes. Avoid redundant indexes to have better run time for batch or online applications. At the same time, consider over loading existing indexes by adding columns to encourage index-only access searches to happen.

   There are different types of indexed access. Table space scan / Index scan / Index only *are the options the Optimizer chooses* to access the data.

   - Direct index lookup:

     For DB2 to perform a direct index lookup, values must be provided for each column in the index.

   - Index scan:

i. Matching index scans (MATCHCOLS>0)

In a matching index scan, predicates are specified on either the leading or all of the index key columns. These predicates provide filtering, only specific index pages and data pages need to be accessed. If the degree of filtering is high, the matching index scan is efficient.

ii. Non-matching index scans(ACCESSTYPE='I' and MATCHCOLS=0)

In a non-matching index scan no matching columns are in the index. Consequently, all of the index keys must be examined.

iii. Index-only access(INDEXONLY='Y')

An index scan occurs when the database manager accesses an index to narrow the set of qualifying rows (by scanning the rows in a specified range of the index) before accessing the base table, to order the output or to retrieve the requested column data directly

❖ Table space scan

When index access is not possible, DB2 uses a table space scan. DB2 typically uses the sequential prefetch method to scan table spaces. A table space scan on a partitioned table space can be more efficient than a scan on a non-partitioned table space. DB2 can take advantage of the partitions by limiting the scan of data in a partitioned table space to one or more partitions.

- Table space scan access (ACCESSTYPE='R' and PREFETCH='S')
- Sequential prefetch (PREFETCH='S')

8 EXPLAIN: To understand the access path

The EXPLAIN statement obtains information about access path selection for an explainable statement. A statement is explainable if it is a SELECT, MERGE, or INSERT statement, or the searched form of an UPDATE or DELETE statement. The PLAN_TABLE contains information about access paths that is collected from the results of EXPLAIN statements.

The information in the plan table helps to perform the following tasks:

- Determine the access path that DB2 chooses for a query
- Design databases, indexes, and application programs
- Determine when to rebind an application

For each access to a single table, EXPLAIN indicates whether DB2 uses index access or a table space scan. For indexes, EXPLAIN indicates how many indexes and index columns are used and what I/O methods are used to read the pages. For joins of tables, EXPLAIN indicates the join method and type, the order in which DB2 joins the tables, and the occasions when and reasons why it sorts any rows. For programs that do static bind process, either hard code the value for predicates in the code or rebind the program with REOPT parameter, which triggers optimizer to look for access path at run time. This would improve the performance in many scenarios, where DBA adds many indexes or any changes to table space.

Questions to analysis Access Path using EXPLAIN

Investigation of access paths that DB2 uses to process SQL statements by using the PLAN_TABLE data to answer certain questions.

The following questions can be used to guide initial analysis of the access paths.

How are indexes used to access the data?

The ACCESSTYPE and MATCHOLS values contain information about the use of indexes in an access path.

- Is an index used? For information about interpreting index access, see Index access (ACCESSTYPE is 'I', 'IN', 'I1', 'N', 'MX', or 'DX')

1. How many indexes are used? For information about interpreting access through multiple indexes, see Multiple index access (ACCESSTYPE='M', 'MX', 'MI', 'MU', 'DX', 'DI', or 'DU')
2. How many index columns are used in matching? For more information about finding the number of matching index columns, see Matching index scan (MATCHCOLS>0) and the number of index columns used for matching (MATCHCOLS=n)
3. Is the query satisfied by the index alone? For more information about analyzing index-only access, see Index-only access (INDEXONLY='Y')
4. How many index screening columns are used? See Index screening to analyze further

Is direct row access used?

5. Direct row access can only be used only when the table contains a column of the ROWID data type. For information about direct row access, see Direct row access (PRIMARY_ACCESSTYPE='D') and ROWID data type.

What possibly costly operations are used?

6. Is a view or nested table expression materialized? Analyzing materialization, see View and nested table expression access.
7. Was a scan limited to certain partitions? See Prefetch access paths (PREFETCH='D', 'S', 'L', or 'U') to analyze use of page-range screening
8. What prefetch type is expected? See Prefetch access paths (PREFETCH='D', 'S', 'L', or 'U').
9. Is data accessed or processed in parallel? see Parallel processing access (PARALLELISM_MODE='C')
10. Is data sorted? To analyze the use of sort operations, check Sort access
11. Is a sub-query transformed to a join? Check sub-query access
12. When are aggregate functions evaluated? Check Aggregate function access (COLUMN_FN_EVAL)
13. Is a complex trigger WHEN clause used? Check Complex trigger WHEN clause access (QBLOCKTYPE='TRIGGR').

DOI: 10.18535/ijecs/v6i8.07

By using the values captured in EXPLAIN tables and instances of PLAN_TABLE in particular, it's possible to understand how Explain works & what changes can be taken up to improve the performance and bring the cost down.

Usually, the optimizer does not consider the order in which tables appear in the FROM clause when choosing an execution plan. The optimizer makes this choice by applying the following rules listed below.

14. The optimizer chooses the execution plan with the fewest nested-loops operations in which the inner table is accessed with a full table scan.
15. If there is a tie, the optimizer chooses the execution plan with the fewest sort-merge operations.
16. If there is still a tie, the optimizer chooses the execution plan for which the first table in the join order has the most highly ranked access path:
17. If there is a tie among multiple plans whose first tables are accessed by the single-column indexes access path, the optimizer chooses the plan whose first table is accessed with the most merged indexes.
18. If there is a tie among multiple plans whose first tables are accessed by bounded range scans, the optimizer chooses the plan whose first table is accessed with the greatest number of leading columns of the composite index.

If there is still a tie, the optimizer chooses the execution plan for which the first table appears later in the query's FROM clause.

Examples

Example to analysis Access Path using EXPALIN

The job was running long time, due to Query performance in the COBOL module.

Query used in the COBOL module

```
SELECT  A.CTHLD_MBR_ID
      ,C.PLC_ID
      ,C.SUB_ID
      ,C.ASSGN_MBR_ID
      ,A.TIMESTAMP
  FROM  WWWW A
      ,XXXX B
      ,YYYY C
      ,ZZZZ D
WHERE A.CI_ID = :WS-CI-ID
AND A.CNTR_ID = :WS-INT-CNTR-ID
AND A.CTHLD_MBR_ID = B.CTHLD_MBR_ID
AND B.CTHLD_MBR_ID = C.CTHLD_MBR_ID
AND C.CTHLD_MBR_ID = D.CTHLD_MBR_ID
AND A.KY_ASGN_MBR_ID = D.KY_ASGN_MBR_ID
AND B.EFF_DT <= :WS-CANCEL-DATE-F
AND ( C.MAINT_TYPE_CD = '021'
OR C.MAINT_TYPE_CD = '025' )
ORDER BY A.TIMESTAMP DESC,
       A.CTHLD_MBR_ID DESC
FOR FETCH ONLY
WITH UR
```

DOI: 10.18535/ijecs/v6i8.07

Result of Explain for this Query

PLANNO	METHOD	CREATOR	TNAME	TABNO	ACCESSTYPE	MATCHCOLS
1	0	DB2TEST	WWWW	1	I	2
2	1	DB2TEST	XXXX	2	I	1
3	1	DB2TEST	YYYY	3	I	1
4	1	DB2TEST	ZZZZ	4	I	1
5	3		0		0	

ACCESSCREATOR	ACCESSNAME
DB2TEST	XMBD5012
DB2TEST	XMBD3391
DB2TEST	XMBD311C
DB2TEST	XMBD503C

On the basis of Explain information, able to see all the four tables [WWWW, XXXX, YYYY, ZZZZ], all the predicates used are using the Indexes as given above. But the matchcols for the table WWWW is 2 column but there is a possibility to improve by adding an index. Similarly the ZZZZ table uses the index with matchcols 1, which can be improved by creating another suitable index

Recommendations for additional indexes are

Table: WWWW:

Another index with the following 4 columns:

CI_ID
CNTR_ID
CTHLD_MBR_ID
KY_ASGN_MBR_ID

Table: ZZZZ:

Another index with the following 2 columns:

CTHLD_MBR_ID
KY_ASGN_MBR_ID

Conclusion

CPU Consumption and elapsed time of various applications are the major contributors of cost for all IT shops. Growing complex business requirements keeps adding the application complexity and cost in a linear fashion.

Optimization techniques and tools [Operations Research tools] are being applied across all the industries to achieve optimum usage of resources to gain cost benefits. IT industries also now-a-days started adopting lean principles, statistical tools and techniques to achieve cost benefits. IT consultants

have come-up with plenty of analysis tools and functions to optimize the cost incurred for various applications across all the business domains and environments. In particular, the database cost is too heavy and so this paper has been written to share the experiences about how to deal with DB2 operations. Explain being the command used to learn the access path, which triggers overall research and analysis of SQLs written to meet the business needs, this paper focused more on EXPLAIN. Various sub topics discussed in this paper by the programmers would guide and help many programmers and beginners to understand the concepts and walk towards enhancing the efficiency to meet operational critical success factors.

Beyond the usage of tools, utilities and optimization techniques, skillful programs can still beat these tools with the help of all the experience, use of best practices, tricks & tips in handling the SQL commands and scripting structured application programs.

["A little knowledge that acts is worth indefinitely more that much knowledge that's idle"]

References

1. IBM Manual on 'Database Performance and query optimization' Version 7.3, pp 11 – 42, 155 – 171
2. Jeff Sullivan, 'What can cause performance issues in DB2', Lecture recorded on 17th Apr, 2010, recorded by YouTube
3. Tony Andrews, 'Advanced Query Tuning with IBM Data Studio', Lecture produced by Curl Films, Themis Education, 27th Mar 2011
4. Sheryl Larsen, 'DB2 for z/OS Best Practice: Advanced SQL Performance Insights', Lecture recorded on 7th March 2014