PROCEEDINGS OF SCIENCE

Query Optimization Ways of DB2 to Improve Database Performance

Jianfeng Tang[1]
School of Software Engineering, Tongji University
Shanghai, 201804, P.R.China
E-mail:billtangjf@tongji.edu.cn

Zewen Hua[a], **Jinsong Feng**[b]
School of Software Engineering, Tongji University
Shanghai, 201804, P.R.China
E-mail:[a]*1452758@tongji.edu.cn;*[b]*fenjinsong@tongji.edu.cn*

How to write the query that meets the requirements of DB2 optimizer is the most concerns of programmers and form partial ability of the programmer to directly control the query performance. In this paper, we analyse how to improve the performance of DB2 from the point of view of Query. The research of this paper includes: (1) how to optimize the query, (2) run RUNSTATS tool to collect statistical object information, (3) the influence of FF (Filter Factor) and statistical information on query efficiency, (4) how to use EXPLAIN tool to view the access path, and (5) discuss the effect of join mode on performance. In this paper, we draw the following conclusions:(1) One of the most important factors affecting the performance of SQL is the choice of access path. The basic generation access path comes from the statistical information of query table, the filter factor of predicate, etc. (2) Before we do the query tuning, the first operation is to run the RUNSTATS program on related tables and indexes. (3) Different join methods under different situations can have different impacts on performance.

CENet2017
22-23 July 2017
Shanghai, China

[1]Speaker

PoS(CENet2017)092

1.Introduction

From the principle of SQL compiler implementation, we can see that one of the most important factors affecting the performance of SQL is the choice of the access path [1]. A lot of access paths towards one statement. For example, there are six kinds of join orders towards three tables, as well as a lot of choices of access paths in each table: index only, index+data scan, RSCAN, etc. However, DB2 optimizer will choose only one path. We can see the access that the path DB2 optimizer will finally adopt according to the column values in the explain table includes the information of the order of MINI-PLAN, how does each MINI-PLAN access data and so on.

And the basis of generating access path comes from statistical information of query table and Filter Factor of predicate, etc.[2]. We should gather as much statistical information as possible to know more accurately a certain moment Filter Factor. As a result, before we do the query tuning, the first operation is to run RUNSTATS program on related tables and indexes.

In addition, different join methods under different situations can have different impacts on the performance. The paper will discuss the four different join methods.

2.Query Optimization Ways of DB2 to Improve the Database Performance

2.1 RUNSTATS Statistics Object Information

In general, there are two kinds of statistical information collected by DB2's RUNSTATS program. One is frequency, indicating that the percentage of rows that satisfy the condition accounts for the number of rows. The other is cardinality, indicating how many different values a column (set) can take. For the table space, statistical information such as CARDF, NACTIVE, NPAGESF, etc. is generally required. For index, statistical information such as NLEAF, NLEVELS, CLUSTERRATIOF, etc. is generally required.

Next, the value of FF (Filter Factor) can be calculated by statistical information. FF exerts great influences on the access path and affects the response time of DB2. Therefore, timely access to the statistical information can enable DB2 to take the latest access path. Otherwise, if there is a change in the DB2 object and the optimizer is still using the old access path, it will inevitably bring about reduced performance [3]. In order to show how the level of details of the statistical information affects the response time of the DB2, the next section uses a simple QUERY to analyze the response time of the DB2 from less to more according to the statistical information.

For equal predicate, the FF formula is FF=1/CARDF. When CARDF=-1 (without RUNSTATS), the optimizer will calculate FF according to the relationship as shown below.

Predicate Type	Filter Factor
Col = literal	1/25
Col <> literal	1 – (1/25)
Col IS NULL	1/25
Col IS NOT DISTINCT FROM	1/25
Col IS DISTINCT FROM	1 – (1/25)
Col IN (literal list)	(number of literals)/25
Col *Op* literal	1/3
Col LIKE literal	1/10
Col BETWEEN literal1 and literal2	1/10

Note:
Op is one of these operators: <, <=, >, >=.
Literal is any constant value that is known at bind time.

Table1: Relationship between Filter Factor and Predicate Type

For the range predicate, if it is greater than or equal to VAR1, FF=(HIGH2KEY-VAR1) / (HIGH2KEY-LOW2KEY). If it is less than or equal to VAR1, FF=(VAR1-LOW2KEY)/ (HIGH2KEY-LOW2KEY). If HIGH2KEY or LOW2KEY all equal to -1. The optimizer will calculate FF according to the chart shown below.

COLCARDF	Factor for Op	Factor for LIKE or BETWEEN
>=100000000	1/10,000	3/100000
>=10000000	1/3,000	1/10000
>=1000000	1/1,000	3/10000
>=100000	1/300	1/1000
>=10000	1/100	3/1000
>=1000	1/30	1/100
>=100	1/10	3/100
>=2	1/3	1/10
=1	1/1	1/1
<=0	1/3	1/10

Note: Op is one of these operators: <, <=, >, >=.

Table2: Calculation Basis of Filter Factor to the Optimizer

2.2 FF (Filter Factor) and Statistical Information

With FF and statistical information, we can initially count how many index pages a query needs to access, how many rids a query needs to access and how many rows to return. Here are some simple SQL statements and statistics to illustrate. For example, there is such an SQL:

Select c2 from t1 where c1=? In this SQL, index l1(c1, c2, c3),t1 CARDF=100000,c1 COLCARDF=5,l1 NLEAF=10000 ， l1 NLEVELS=3. For equal predicate, FF=1/COLCARDF. So, it needs to access 3+ (1/5) *10000=2003 index pages, deal with (1/5) *100000=20000 RIDs, and returns (1/5) *100000=20000 records.

With a matching predicate, the situation will become much better. For example, select c3 from t1 where c1=? and c2 =?. In this SQL, c2 COLCARDF=10, FULLKEYCARDF=65000. Other statistics are the same as above. In this case, it needs to access 3+ (1/5) * (1/10) *10000=203 index pages, needs to process (1/5) * (1/10) *100000=2000 RIDs, and returns (1/5) * (1/10) *100000=2000 records.

From this point of view, we can see that using matching predicateas much as possible can reduce the number of processing DB2 objects so as to achieve the purpose of improving the efficiency of DB2.

For SQL: select c4 from t1 where c1=? and c3>50, t1 CARDF=1 Million, NPAGES=100000, NLEAF=10000,c1 COLCARDF=10,c3 COLCARDF=10120. Since c3>? is a range predicate, so the default FF for DB2 is 1/100. It needs to access the 3+ (1/10)

*10000=1003 index pages, access the 1/10*1000000=100000 index rows, and access (1/10) * (1/100) *1000000=1000 records.

Also for the above SQL, if the statistics contain low2key and high2key, then FF will be much more accurate, and the screening cost will be much more accurate, too. If low2key=0, high2key=100, then the FF of C3 is (100-50)/(100-0)=0.5. For the matching cost, the above information is the same. But for the screening cost, it needs to access (1/10)*0.5*1000000=50000 records. Thus, if RUNSTATS does not count low2key and high2key at this time, the optimizer will still take the previous access path, which filters too many records. At this point, DB2 will take RSCAN way to access data, which will have a great negative impact on the performance.

If RUNSTATS contains statistics for frequency, a more accurate access path will be generated [4]. For example, select c4 from t1 where c1='A', index l1(c1,c2,c3), the following frequency will be added to the statistical information of c1: (A.0.75), (B,0.15), (C,0.05), (D,0.02) and (E,0.03); therefore, for the matching cost, it needs to access 3+0.75*10000=7503 index pages, needs to access 0.75*100000=75000 RIDs, and returns 0.75*100000=75000 records, which is closer to the true value than the previous 20000 records. At this point, if we use the "where c1= 'Z'", then the FF=0, which needs to access 3 index pages, 0 RID, and returns 0 record, thus greatly improves the efficiency of SQL. In the DB2 V9 version, RUNSTATS can also count Histogram. Using RUNSTATS INDEX (l1 NUMCOLS(1)) HISTOGRAM, the following C1 statistics are obtained:

QUAN.NO	LOWVALUE	HIGHVALUE	CARD	FREQUENCY
1	0	5	4	18%
2	7	30	20	22%
3	31	36	3	18%
4	37	49	12	19%
5	55	100	10	23%

Table3: C1 Statistics Represented in the Form of Histogram

T1 CARDF=100000,c1 COLCARDF=49, l1 NLEAF=10000,l1 LEVELS=3

For example, "select c4 from t1 where c1=2", and for matching cost, the statement needs to access 3+(0.18/4)*10000=453 index pages, access (0.18/4)*100000=4500 RIDs, and return (0.18/4)*100000=4500 records.

With the DB2 version getting higher and higher, DB2 will do more and more detailed work in the field of statistical information. For example, in the example above, FF is 0.045 instead of 1/49=0.02, so that the generated access path is more consistent with the actual data distribution andthereby improves the efficiency of data query.

2.3 Using EXPLAIN Tool to View the Access Path

Upon analysis of the first two sections, we can guide the DB2 optimizer to generate the best access path. But DB2 will not be "smart" enough to go to work in accord with our "guidelines", thus we have to look at the specific access path that DB2 generates. We can use EXPLAIN tool to view the access path. Of course, if we use the database tuning tools (such as OSC, OE), we can view the access path in a graphical way.

A SQL plan is usually composed of a number of mini plans. In simple terms, mini plans are the implementation steps of each query block, which are finally linked and generates the plan through the way of Left-Deep-Tree, as shown in the following figure:

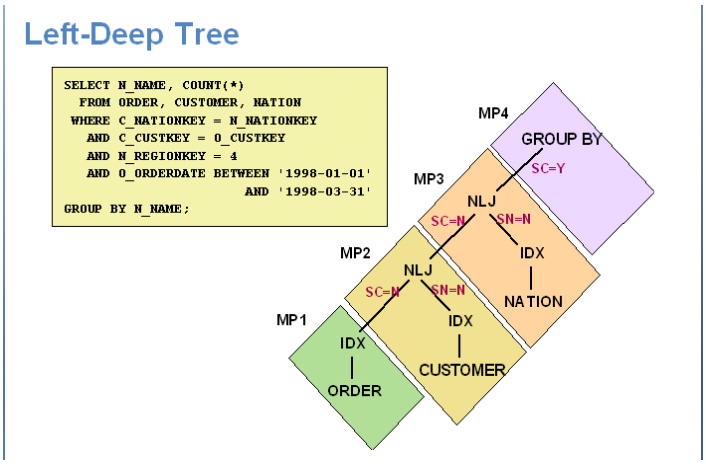

Figure 1: Specific SQL and Details of SQL Plan

The steps using EXPLAIN statement to view access path are as follows:

1.Create PLAN TABLE, and the syntax is as follow:

CREATE TABLE userid.PLAN_TABLE

(QUERYNO INTEGER NOT NULL,

…)

In the database-name.table-space-name CCSID EBCDIC;

2. We can use the EXPLAIN tool to tune the SQL statement, and the specific syntax is shown as follow:

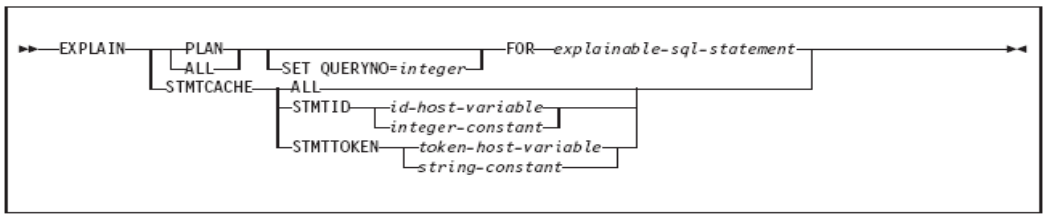

Table 4: EXPLAIN Syntax

Use SELECT to fetch needed records from the PLAN_TABLE and the following figure is a simple example of PLAN_TABLE.

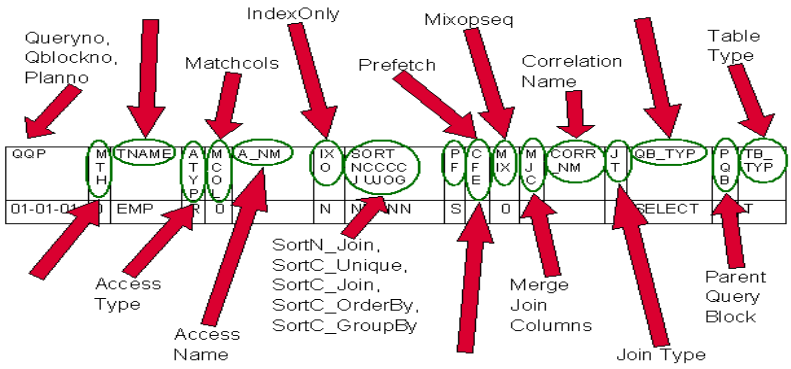

Figure 2: A Simple Example of PLAN_TABLE

For example, we can first run the EXPLAIN command and then the SQL statement: SELECT ACCESSTYPE FROM userid.PLAN_TABLE. If the value of ACCESSTYPE equals to R, then it means DB2 uses TableScan method to fetch data. If we add index on the appropriate column and run the EXPLAIN command again, the value of ACCESSTYPE may equals to I, which means DB2 uses IndexScan method to fetch data, which can improve the efficiency of the query.

2.4 Effect of Join Mode on Performance

In most cases, the SQL statements are related to join methods. DB2 for z/OS has four kinds of join methods(Due to limited space of the paper, the fourth join method: Star Join(JOIN_TYPE=S) is ommited). You can view every join method through the METHOD column in PLAN_TABLE. Different join methods under different situations can have different effects on performance. In the following part, we will discuss different join methods.

2.4.1 Nested Loop Join (NLJ in short, METHOD=1)

The following figure is the example of PLAN_TABLE which reflects the nested loop join. NLJ takes the "first inside and then outside mode", that is, it accesses the external table once. And the internal table access time lies on the size of the filtered outside table. DB2 optimizer uses the NLJ under the following conditions:

1) The outside table is very small (the table which METHOD=0).

2) FF of the outside table is very small so that the filtered outside table is very small, too.

3) The inside table has an efficient and high aggregation index.

4) The inside table has very few data pages accessed.

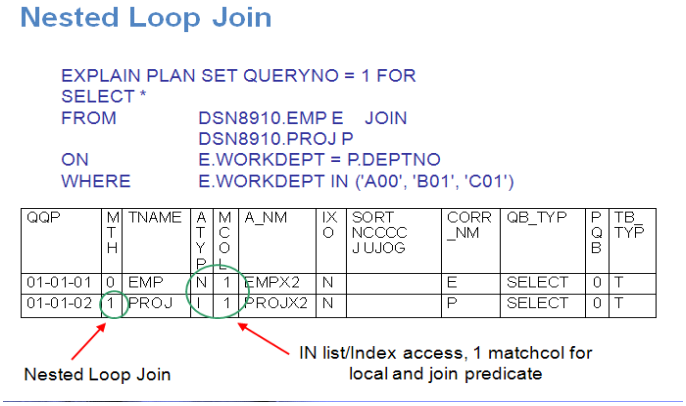

Figure 3: The PLAN_TABLE of Nested Loop Join

Note: the value "1" in the MTH column signifies DB2 takes the Nested Loop Join mode.

2.4.2 Sort Merge Join (SMJ in short, METHOD=2)

As shown in the following fig., there is an example of PLAN_TABLE towards SMJ. DB2 scans both outside and inside table according to the column of join. If there is no index in the outside or inside table, DB2 will do sort on unordered table in accordance with the join column before join operation. The sort results will be placed into the work file. When the value of the join column in the outside table is equal to that of the join column in the inside table, then the join records are returned. From the join process of SMJ, we can see that the key to improve the SMJ efficiency is to avoid sorting, that is, to create an index in the join column. As seen from the following figure, the efficiency of the SMJ is not high. Because SORT_NEW_JOIN=Y, that is, there is a sorting process in the inside table and the result of the sort is written into the work file. One of the solutions is to build an index on DEPNO column. In general, the DB2 optimizer uses SMJ in the following circumstances:

1) The outside and the inside tables that satisfy the condition are all very large and there is no index on the matching predicate.

2) There are very few columns from the inside table in SELECT statement.

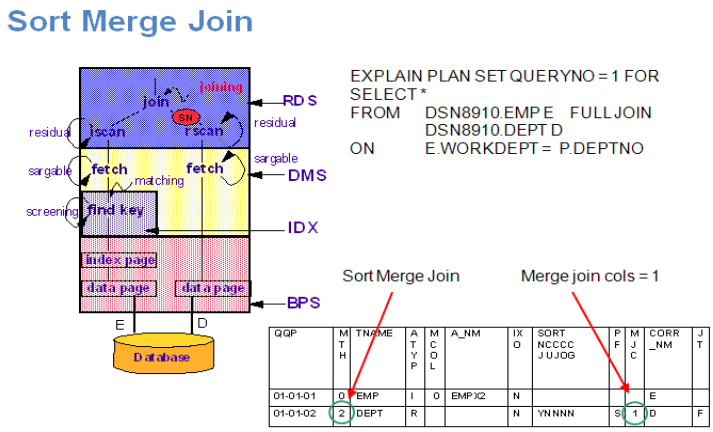

Figure 4: The PLAN_TABLE of Sort Merge Join

2.4.3 Hybrid Join(METHOD=4)

As shown in the following figure, there is an example of PLAN_TABLE towards Hybrid Join. The prerequisite for Hybrid Join is that the index must be built on the join column of the inside table.

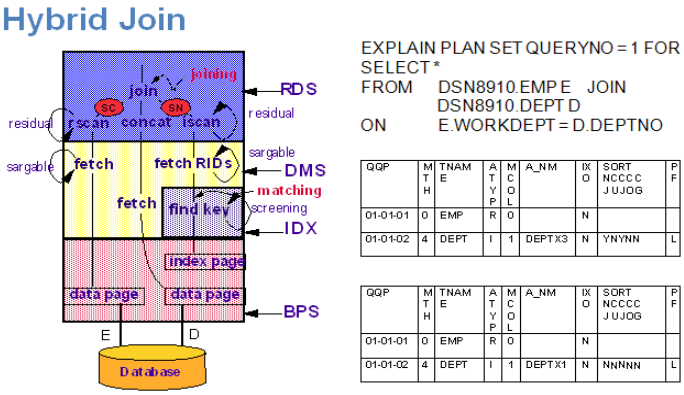

Figure 5: The PLAN_TABLE of Hybrid Join

The process of Hybrid Join is shown as follows:

- Scan the outside table in an efficient way.
- Join the RID of join column of the outside and inside table to form the intermediate table in Phase 1.
- Do the sort of the intermediate table in phase1 according to the order of RIDs to generate the intermediate table in Phase 2.
- Use RID in phase 2 to extract the corresponding records from the inside table with list prefetch.
- Merge records in the intermediate table in Phase 2 with records taken from Step 4 to generate the result table.

3.Conclusion

This paper analyzes how to improve the performance of DB2 from the point of view of Query. In this paper, we can draw the following conclusions.

One of the most important factors affecting the performance of SQL is the choice of access path. The access path will be finally adopted by DB2 optimizer will finally adopt according to

the column values in the explain table. And the basic generating access path comes from statistical information of the query table, Filter Factor of predicate, etc. We can gather as much statistical information as possible to know more accurately a certain moment Filter Factor. Before we do the query tuning, the first operation is to run RUNSTATS program on related tables and indexes. In addition, different join methods under different situations may have different impacts on performance.

References

[1] Angela Yang, *Use DB2 optimization techniques to get and maintain a good query execution plan*[C]. Proceedings of the 2010 Conference of the Centre for Advanced Studies on collaborative research, Nov, 2010:385-386.

[2] Andrews Tony, *DB2 SQL tuning tips for z/OS developers*[M]. IBM Press, Oct,2012.

[3] Bedoya Hernando, *Preparing for and Tuning the SQL Query Engine on DB2 for i5/OS*[M]. IBM Redbooks, Sep,2006.

[4] Bruni Paolo, *IBM DB2 9 for z/OS: New Tools for Query Optimization*[M], IBM Redbooks, Dec,2007.