SAP® MaxDB™ Expert Sessions

SAP MaxDB SQL Query Optimization - Part 2

Christiane Hienger SAP AG
Holger Becker SAP AG

August 29, 2012
1. Introduction

2. Update Statistics & System Table  FILES

3. JOIN Optimizer Strategies

4. Hash Join

5. ORDERED HINT
After this presentation, you will be able to:

- Understand the join processing
- Understand the meaning of update statistics and file directory counters
- Understand the usage of SQL strategies of the join optimizer
- Analyze SQL join statements
In this session, we use the table ZZTELE with approx. 115,000 records for the examples. The primary key is defined on the columns NAME, VORNAME, STR.

The uniqueness of the primary key ensures that we only have one entry with the same name, first name and street. The records of the table are sorted in key sequence – name, vorname, str.

You can get the table and the primary key definition with the following SQL statement:

```sql
Select * from domain.columns where tablename = 'ZZTELE'
```
To explain strategies which can be used for joins, the examples also refer to the table ZZSTADTTEIL with approx. 20000 records and table ZZMASTER.

The primary key of table ZZSTADTTEIL is defined on column PLZ (zip code). For each zip code there is one entry.

The table is sorted via zip code.

Table ZZMASTER has a multiple key defined on columns YEAR, NAME and VORNAME. The table is sorted by the year of the Master graduation, Name and Vorname.
### Table Examples: ZZCODE

#### Create Table ZZCODE

```sql
( STR CHAR(40),
  NR INT,
  PLZ CHAR(5),
  ORT CHAR(25),
  CODE CHAR(31)
)
```

# of records: around 115,000
Indexes enable faster access to the rows of a table. The indexes of a table can be determined using the system table INDEXCOLUMNS.

```
SELECT owner, tablename, indexname, type, columnname, 
    sort, columnno, datatype, len, createdate 
FROM domain.indexcolumns 
WHERE owner = <owner> 
AND schemaname = <schema> 
AND tablename = <table_name> 
ORDER BY owner, tablename, indexname, columnno
```

You can create an index (also known as secondary key) to speed up the search for database records in a table. In technical terms, indexes are data structures (consisting of one or more inverting lists), which store parts of the data of a table in a separate B* tree structure. This storage sorts the data according to the inverting key fields that were used. Due to this type of storage, the table data can be accessed faster using the indexed columns than without the relevant index.

For more information about indexes use SAP note 928037 FAQ SAP MaxDB Indexes
A join is an SQL statement that links multiple tables with each other. A result table is created.

An **inner join** is the most common join operation. Inner join creates a result by combining column values of two tables (A and B) based upon the **join predicate**.

The join predicate is defined in an ON clause and specifies a comparison between two values or lists of values of both tables.

MaxDB handles four types of JOIN: INNER, OUTER (Full, LEFT and RIGHT), UNION

An **outer join** does not require each record in the joined tables to have a matching record. The result contains each record—even if no other matching record exists. We distinguish between left and right outer join.

A **left outer join** returns all the values from an inner join plus all values in the left table that do not match to the right table added by NULL values for the left table.

A **right outer join** returns all the values from the right table and matched values from the left table added by NULL values for the right table.
When a join is optimized, first the optimal access strategy for each single table is calculated.

Then the optimizer decides which order of the tables will be processed in the join execution. The calculation of the costs are based on the optimizer statistics.

Outdated optimizer statistics may have an extreme influence on the chosen access strategy and therefore on the runtime of the SQL command. E.g. After a dataload the statistics are outdated. But only if the relationship of the data (Distinct values) was changed new optimizer statistics are necessary to find the best strategy.
For a JOIN, the optimizer looks for the most suitable access path for each table.

Then the join optimizer decides in which order the tables will be processed and connected with each other. For the join columns, the values are unknown before the execution. Therefore, the join optimizer works with statistical values for columns.
Update Statistics (1)

**UPDATE STATISTICS**  [<owner>.]<table_name>
[ESTIMATE [SAMPLE <unsigned_integer> <PERCENT,ROWS>]]

To determine the best possible access path, in particular for joins, the Optimizer requires statistical information. If such information is not up-to-date, the system may make erroneous strategic decisions.

**UPDATE STATISTICS** determines values about the size of a table as well as the size and value distribution of indexes.

**UPDATE STATISTICS** should be executed following large-scale change transactions (INSERT/LOAD, UPDATE, DELETE).

Start using the DBM command sql_updatestat and sql_updatestat_per_systemtable or via the CCMS (transactions DB13, DB21, DBACOCKPIT).

For the table itself, Update Statistics only determines data if the current size information is not already in the file directory. This does not apply to tables created with databases of versions < 7.6 and for which no size information could yet be determined in the file directory.

Update Statistics determines statistics data for all columns that are primary key or index columns. It also determines the statistics data for all columns outside of the primary key and the index, if statistics are available. Additionally it determines the statistics data of all entries in system table SYSUPDSTATWANTED.

If the Optimizer discovers tables with outdated statistics data, they are inserted into the table SYSUPDSTATWANTED. The DBM command sql_updatestat_per_systemtable executes Update Statistics for all tables listed in SYSUPDSTATWANTED.

The DBM command sql_updatestat executes Update Statistics for all tables in the database.

Update Statistics imports the data for a table from all data volumes in parallel for update statistics computed (not estimate). This makes it very speedy.

As of version 7.6, the sampling procedure in the standard uses a new algorithm for calculating the statistics data. You can determine the algorithm to be used with the parameter UPDATESTAT_SAMPLE_ALGO. The new algorithm generates more accurate statistics with fewer records read.

The programs "xpu" and "updcol" are no longer available as of version 7.6.

Additional information about Update Statistics: FAQ note 927882
For tables that grow and shrink very quickly, such as spool tables, for example, it is a good idea to set the sampling rate to 0. This prevents Update Statistics from being requested and executed for these tables.

With the following command dbmcli starts an Update Statistics with sampling for all tables of one schema:

```
sql_updatestat SAP<SID>.* estimate
```
Requested Updates shows if an Update Statistics is requested for this table. It shows the content of system table SYSUPDSTATWANTED.

Update Standard executes an Update Statistics table.

You can use Update (Column Statistics) to create column statistics for specified columns.

In the Optimizer Statistics view the column and table statistics are listed.
The one table Optimizer only uses the statistics data for tables if the counters for size data are not in the file directory.

The join optimizer uses the column statistics created with Update Statistics in the system table `OPTIMIZERSTATISTICS`. 

```
SELECT * FROM OPTIMIZERSTATISTICS
WHERE tablename = '...' 
```

- Shows the current statistic values that will be used by the optimizer to determine the strategy.

<table>
<thead>
<tr>
<th>TABLENAME</th>
<th>INDEXNAME</th>
<th>COLUMNNAME</th>
<th>DISTINCTVALUES</th>
<th>PAGECOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZZTELE</td>
<td>ADDINFO</td>
<td></td>
<td>1969</td>
<td>?</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>CODE</td>
<td></td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>NAME</td>
<td></td>
<td>13363</td>
<td>?</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>NR</td>
<td></td>
<td>255</td>
<td>?</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>ORT</td>
<td></td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>PLZ</td>
<td></td>
<td>20001</td>
<td>?</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>STR</td>
<td></td>
<td>8</td>
<td>?</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>YORNNAME</td>
<td></td>
<td>5156</td>
<td>?</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>CODE</td>
<td></td>
<td>?</td>
<td>1155</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>ZZTELE~1</td>
<td></td>
<td>?</td>
<td>1165</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>ZZTELE~3</td>
<td></td>
<td>?</td>
<td>1112</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>ZZTELE~4</td>
<td></td>
<td>?</td>
<td>1334</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>ZZTELE~2</td>
<td></td>
<td>?</td>
<td>1548</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>TABLE STATISTICS</td>
<td></td>
<td>114199</td>
<td>1800</td>
</tr>
</tbody>
</table>
For tables that were created with versions < 7.6, the counters for size data in the file directory after upgrade to version 7.5 are not yet available. You can determine the counters with a CHECK DATA in the ADMIN state or with CHECK TABLE WITH SHARE LOCK. CHECK TABLE sets a share lock for the duration of the check.

After the upgrade from versions < 7.6 to versions >= 7.6, all table names are transferred to the table SYSUPDATECOUNTERWANTED. With every restart and in periodic intervals, the database attempts to determine the counters for all remaining tables in SYSUPDATECOUNTERWANTED for the file directory. A share lock is set on a table during processing. Determination of the counters is immediately terminated for a table if the share lock causes a lock collision.

The values for TREENINDEXSIZE, TREELEAVESIZE and LOBSIZE are shown in KB.

For tables, ENTRYCOUNT shows the number of records per table. For indexes, ENTRYCOUNT shows the number of different values for the secondary key.
An execution plan or access path shows how MaxDB accesses the requested data (index access, table scan, key range, key equal, index equal, and so on). An EXPLAIN plan (execution plan) displays the strategy the Optimizer selects to run a special SQL statement. These EXPLAINs are used to analyze long running SQL statements. An EXPLAIN plan can only be displayed for SELECT statements.

In the ABAP-based SAP application server, EXPLAIN is available in transactions ST05, DB50 and DBACockpit (in the command monitor). The SQL editor of the Database Studio can send an EXPLAIN via context menu (right mouse click) to the database. The output is shown in a separate window.

There are additional EXPLAIN statements which are useful for join analysis.

EXPLAIN JOIN and EXPLAIN SEQUENCE are used by the development to find optimizer problems.

Interested people can find additional information can be found in the SCN using the following links:

- EXPLAIN SEQUENCE - > https://wiki.sdn.sap.com/wiki/display/MaxDB/MaxDB+Explain+SEQUENCE
EXPLAIN shows:

- one block for each table from the SELECT-FROM list
- the order of the strategies reflects the order of execution
- COPIED / NOT COPIED --> Result set is generated/not generated
- "Estimated costs" provides an estimation about the number of read/write accesses
- Applied Query Rewrite rules

<table>
<thead>
<tr>
<th>SCHEMANAME</th>
<th>TABLENAME</th>
<th>COLUMN_OR_INDEX</th>
<th>STRATEGY</th>
<th>PAGECOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>Table 1</td>
<td>Names of key or index columns</td>
<td>Name of chosen strategy for this table</td>
<td>Number of pages in system table Optimizerstatistics</td>
</tr>
<tr>
<td>Schema</td>
<td>Table 2</td>
<td>Names of key or index columns</td>
<td>Name of chosen strategy for this table</td>
<td>Number of pages in system table Optimizerstatistics</td>
</tr>
<tr>
<td>Result name</td>
<td></td>
<td></td>
<td>RESULT IS (NOT) COPIED, COSTVALUE IS</td>
<td>Estimated costs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Applied Query Rewrite rules</td>
<td>1</td>
</tr>
</tbody>
</table>
Search conditions used by the optimizer to determine the optimal search strategy are:

- Equality conditions
- Range conditions
- IN conditions

The best strategy is chosen by the Optimizer. The basis of decision making is the cost for each evaluated strategy.

The SQL Optimizer also converts conditions under certain circumstances. If a single value is specified in an IN condition multiple times, the condition is converted into an equality condition.
Joins are executed with the Nested Loop method. In doing so for the single join transitions no result sets are built. The nested loop join uses one join input as the outer input table and one as the inner input table. The outer loop consumes the outer input table row by row. The inner loop, executed for each outer row, searches for matching rows in the inner input table. Only the final result is fully created before the first row is delivered. -> this is a advantage for SQL commands with restriction of ROWNO

As of version 7.7 there is no more possibility to choose between Sorted Merge or Nested Loop by a parameter setting (JOIN_OPERATOR_IMPLEMENTATION). There are only marginal disadvantages concerning CPU usage for Nested Loop with the current algorithms. Therewith the Nested Loop can deliver the result faster and with the use of less resources.

The Optimizer starts with that table which related to the total execution plan results in the lowest total costs. You should take care that convenient indexes exist.

In the example the Optimizer starts with a large table customer.

For each hit in customer (outer table) the inner table reservation is read. Each hit in reservation is inserted immediately into the final result.

As soon as the number of requested rows ( rowno = 6 ) has been reached the join process stops and the result can be delivered to the application.
Here is an example for nested loop join processed via index strategies.
The analysis and optimization of complex joins is one of the most difficult tasks in the SQL statement analysis.

For the access to the first table have a closer look to the local predicates. Can the primary key be used to access the table or can the access be optimized with an additional index.

For each join with MaxDB it is very important to have good join transition. The number of records read can be reduced by creating convenient indexes for the join transition. During join performance analysis a focus should always be if the best join transition is used.
zzstadtteil.plz is the sole primary key column
zzstadtteil.ort is a standard column

```
SELECT * FROM zztele JOIN zzstadtteil ON zztele.Plz = zzstadtteil.Plz,
AND zztele.Ort = zzstadtteil.Ort
WHERE zztele.name = 'Mueller'
```

The join transition from table zztele to table zzstadtteil is specified via column PLZ. Table zzstadtteil has a single key on column PLZ.
The key of table zzstadtteil is qualified in the join predicate. So a JOIN VIA KEY strategy can be used. Because table zzstadtteil only has a single key column on plz the join transition can be done with the strategy JOIN VIA KEY COLUMN.
**Join via multiple key columns**

zztele.name is the first primary key column  
zztele.vorname is the second primary key column  
zztele.str is the last/third primary key column

```
SELECT * FROM zztele JOIN zzmaster ON zztele.name = zzmaster.name 
   AND  zztele.vorname = zzmaster.vorname
WHERE zztele.str = 'Alt Moabit'
   AND zzmaster.Year = '2000'
```

<table>
<thead>
<tr>
<th>TABLENAME</th>
<th>COLUMN_OR_INDEX</th>
<th>STRATEGY</th>
<th>PAGECOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZZMASTER</td>
<td>YEAR</td>
<td>RANGE CONDITION FOR KEY (USED KEY COLUMN)</td>
<td>1</td>
</tr>
<tr>
<td>ZZTELE</td>
<td>NAME</td>
<td>JOIN VIA MULTIPLE KEY COLUMNS</td>
<td>3200</td>
</tr>
<tr>
<td></td>
<td>VORNAME</td>
<td>(USED KEY COLUMN)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STR</td>
<td>(USED KEY COLUMN)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO TEMPORARY RESULTS CREATED</td>
<td></td>
</tr>
<tr>
<td>JDBC_CURSOR_71</td>
<td></td>
<td>RESULT IS COPIED, COSTVALUE IS</td>
<td>2</td>
</tr>
</tbody>
</table>

Remember: zztele key: Name, Vorname, Str

If the key of a joined table exists of more than one column and the complete key is qualified the join strategy is the same as **JOIN VIA KEY COLUMN**. Only the name (**JOIN VIA KEY COLUMN / JOIN VIA MULTIPLE KEY COLUMNS**) differs if the joined table has one or several key columns. This is because of historical reasons.

If the complete multiple key is qualified in the join predicates the strategy is called **JOIN VIA MULTIPLE KEY COLUMNS**.
zztele.name is the first primary key column
zztele.ort is a standard column

SELECT * FROM zztele JOIN zzmaster ON zztele.name = zzmaster.name
AND   zztele.ort = zzmaster.ort
WHERE zzmaster.Year = '2000'

<table>
<thead>
<tr>
<th>TABLENAME</th>
<th>COLUMN_OR_INDEX</th>
<th>STRATEGY</th>
<th>PAGECOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZZMASTER</td>
<td>YEAR</td>
<td>RANGE CONDITION FOR KEY</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(USED KEY COLUMN)</td>
<td></td>
</tr>
<tr>
<td>ZZTELE</td>
<td>NAME</td>
<td>JOIN VIA KEY RANGE</td>
<td>3200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO TEMPORARY RESULTS CREATED</td>
<td></td>
</tr>
<tr>
<td>JDBC_CURSOR_25</td>
<td></td>
<td>RESULT IS COPIED , COSTVALUE IS</td>
<td>13</td>
</tr>
</tbody>
</table>

If the key of a joined table exists of more than one column and only the first column of the multiple key is qualified the join transition is done via a **KEY RANGE**.

If only the first column of the primary key is qualified via a join predicate the join strategy is called **JOIN VIA KEY RANGE**.
If the key of a joined table exists of more than one column and only a part of the multiple key is qualified the join transition is done via a key range.

If there is more than one key column part of the join predicates but not all primary key columns are qualified then we are talking about the join strategy **JOIN VIA RANGE OF MULTIPLE KEY COLUMNS**.

The strategy **JOIN VIA RANGE OF MULTIPLE KEY COLUMNS** is nearly the same as the strategy **JOIN VIA KEY RANGE**. The difference is the number of key columns of the joined table and has historical reasons too.
For the next examples about *JOIN VIA INDEX* accesses the tables *ZZTELE, ZZCODE, ZZMASTER* and *ZZSTADTTEIL* are used.

The slide lists the indexes which exist on these tables.

*ZZTELE~3, ZZTELE~4* and *CODE* are single indexes (secondary keys).

*ZZTELE~2* and *ZZTELE~1* are multiple indexes (secondary keys).

*ZZMASTER* and *ZZCODE* do not have any indexes.
During join performance analysis an additional focus should be to check if the best join transition is used and if we can optimize the join transition by creating a new index.

The following slides explain the join strategies via index access.
In this SQL statement a local predicate is specified (stadtteil) on table zzstadtteil.

The join transition between ZZSTADTTEIL and ZZTELE is specified via column PLZ and column ORT.

Table ZZTELE has a single index on column PLZ. Column ORT is neither part of an index nor part of the primary key.

The index ZZTELE~3 of table ZZTELE is qualified in the join predicate. So a JOIN VIA INDEX strategy can be used. Because index zztele~3 is a single index on column PLZ the join transition can be done with the strategy JOIN VIA INDEXED COLUMN.
On table ZZTELE there exists a multiple index zztele~2 on columns STR,NR. The join transition qualifies the complete index ZZTELE~2.

For the join transition a strategy called *JOIN VIA MULTIPLE INDEXED COLUMNS* can be used.

This is same strategy as *JOIN VIA INDEX COLUMN*. The only difference is that we have a multiple index instead of a single index.

```sql
SELECT *
FROM zztele JOIN zzcode ON zztele.str = zzcode.str
AND  zztele.ort = zzcode.ort
WHERE zzcode.code = 'Cable TV'
```
If the index of a joined table exists of more than one column and only a part of the multiple secondary key is qualified the join transition is done via an index range.

If there is more than one index column part of the join predicates but not all secondary key columns are qualified then we are talking about the Join strategy

**JOIN VIA RANGE OF MULTIPLE INDEXED COLUMNS.**
The hash join strategy is employed when a join transition to a small table is done and it is probable that a large number of records needs to be read from the small table several times. In this case it would be faster to import the small table once and generate a temporary hash table. Searching for the keys in a hash table is faster than searching via the B* tree of the table. The accesses on the hash table need not to be synchronized.

The strategy "TABLE HASHED" identifies the join via a hash table.

**JoinHashMinimalRatio** – default 1
The minimal ratio between size of tables joined so far to the size of the next table to be joined which has to be equal or exceeded to use hashing for this next table.

**HashJoinSingleTableMemorySize** (MAX_SINGLE_HASHTABLE_SIZE)
The maximum table size in KB for which hash joins will be executed. If HashJoinSingleTableMemorySize = 0 then no hash tables will be created during join execution.

**HashJoinTotalMemorySize** (MAX_HASHTABLE_MEMORY)
As there can be multiple hash joins running at the same time, the amount of memory used for all hashes might become excessive if it is unlimited. This parameter sets the upper limit for the memory provided for all hash joins that are running in parallel. If during join execution a join transition qualifies for a hash join but the overall memory used for all hash joins would be more than HashJoinTotalMemorySize a regular join will be executed instead.

If HashJoinTotalMemorySize = 0 then no hash joins will be executed.
Hash Join (1)

SELECT zzstadtteil.* FROM zztele JOIN zzstadtteil
ON zztele.plz = zzstadtteil.plz
Hints provide the Optimizer with rules that it can use if necessary.

Example:

```
SELECT /*+ORDERED*/ zztele.plz, zzstadtteil.stadtteil
FROM zzstadtteil, zztele
    WHERE zztele.plz = zzstadtteil.plz
    AND zzstadtteil.stadtteil = 'Moabit'
```

Hints are supported as of:
- MaxDB Version 7.5
- WebAS ABAP Version 6.20

MaxDB supports the several hints, see SAP note 832544 FAQ SAP MaxDB Hints for detailed information.

During join performance analysis the ORDERED Hint can be used to force a special order of table processing.
Feedback and further information:
http://www.sdn.sap.com/irj/sdn/maxdb

SAP® MaxDB™ 7.8 Shadow Page Algorithm
Disclaimer

This presentation reflects current planning.

Contents may be changed without prior notice, and are in no way binding upon SAP.