Strategies for Optimizing the Data Access

This section explains SAP MaxDB optimization strategies.

**Moderators:** Christiane Hienger

**WIKI Space Editor:** Thiago Lütig.

The database administrator uses access plans, which are generated with the EXPLAIN statement, to understand the optimizer's data access strategy. The database system can generally use various different strategies to access data (known as search strategies). These search strategies (Terminology -> Strategy) differ in the access type and in the costs that they incur when executed. The Optimizer searches for search strategies for which the costs are as low as possible. Which search strategies are possible mainly depends on the search condition of the respective SQL statement.

---

### Single Table Optimizer Strategies

- Table Scan
- Equal Condition for Key
- Range Condition for Key
- In Condition for Key
- SUBQ Condition for Key
- Index Scan
- Index Only
- Equal Condition for Index
- Range Condition for Index
- In Condition for Index
- SUBQ Condition for Index
- Different Strategies for OR Terms
- No Strategy Now (Only at Execution Time)
- No Result Set Possible
Join Optimizer
Strategies

Introduction
- Join via Key Column
- Join via Key Range
- Join via Multiple Key Columns
- Join via Range of Multiple Key Columns
- Join via Indexed Column
- Join via Multiple Indexed Columns
- Join via Range of Multiple Indexed Columns

Relevant Documents
Recommended: MaxDB Documentation Optimizer Strategies.

Details

In Condition for Key

This strategy is selected, if the first k-1 columns of the primary key were qualified with equal conditions, and an IN condition was defined for the k-th key column (whereby 1 <= k <= n, where n is the number of key columns). This strategy is also selected especially if the first key column was represented by an IN condition. That is, the primary key columns before the column with the IN condition are specified with an equal condition. The individual values of an IN condition are processed as an equal condition in this strategy. You can access the corresponding table rows directly.

A temporary result set is created.

If the values in the IN condition are close together, or if many values are displayed, it can be more practical to use a range condition instead of an equal condition. This is displayed accordingly in the execution plan. As of MaxDB Version 7.5 several IN conditions can be taken into account in the primary key. Here you need to take into account that the second IN condition can only be processed with one range.

Example: The key of table zztele consists of the columns name, firstname and city. SELECT * FROM zztele WHERE name = 'Schmidt' AND firstname IN ('Anja', 'Elke')

back to top

SUBQ Condition for Key

This strategy is chosen if:
• the table has exactly one primary key and a subquery has been defined on this (example 1) or
• the table has a primary key across several columns and
• equal conditions are defined on all primary key columns that are located before the column with the subquery in the primary key and
• an IN or ANY condition is defined in the form of a subquery on one column of the primary key (example 2). For every results row of the subquery, a part of the primary key is searched sequentially.

Example 1: The primary key of table city consists of column zip. SELECT * FROM city WHERE zip IN (SELECT zip FROM customer WHERE name = 'Jenkins') Example 2: The primary key of table room consists of columns hno and type. SELECT * FROM room WHERE hno = 50 AND type IN (SELECT type FROM reservation WHERE hno = 50)

Index Scan
In this strategy the search area is not limited either. Here, the search area consists of the entire index (single or multiple index) whereby the data in the primary key table is accessed via the corresponding primary keys.

SUBQ Condition for Index
This strategy is chosen if:
• the table has a single index and a subquery has been defined on this index column (example 1) or
• the table has a an index across several columns and
• equal conditions are defined on all index columns that are located before the column with the subquery in the index and
• an IN or ANY condition is defined in the form of a subquery on one column of the index (example 2).

For every results row of the subquery, a part of the primary index key used (multiple index) is searched sequentially.

Example 1: The index consists of column state. SELECT * FROM city WHERE state IN (SELECT state FROM city WHERE zip > '90000') Example 2: The index consists of the columns name, firstname. SELECT * FROM customer WHERE name IN (SELECT name FROM employee) AND firstname > 'Frank'

No Strategy Now (Only at Execution Time)
In this subquery the column values are only known at the time when the subquery is executed. The best possible search strategy is determined at the time of execution, if the values of the subquery are available. This strategy appears for database queries that contain subqueries or correlated subqueries because the final strategy can only be specified after temporary results have been generated.

Join via Key Column
The first column (col1) is the only column of the primary key, the second column (col2) is a normal column. The second table (jointab) can be accessed directly via the key (col1).

Join via Key Range
The first column (col1) is the first column of the multiple primary key, the second column (col2) is a normal column. In the second table (jointab), the data is read sequentially within the primary key range.
Join via Multiple Key Columns

All join columns (col1 and col2) match the primary key columns of the second table (jointab). The data is read directly in the second table (jointab).

Join via Range of Multiple Key Columns

The join columns (col1 and col2) correspond to the first columns of the primary key. The data is read sequentially in the second table (jointab) within the primary key area.

Join via Indexed Column

There is a single index for the first join column (col1); the second column (col2) is a normal column. The data is read via the index of (col1).

Join via Multiple Indexed Columns

The join columns correspond to the columns of a multiple index. The first column (col1) is the first column of a multiple index and the second column (col2) is the second and last column of the multiple index. The data of the second table (jointab) is transferred via this index.

Join via Range of Multiple Indexed Columns

The join columns (col1 and col2) correspond to the first columns of a multiple index. The first column (col1) is the first column of the multiple index and the second column (col2) is the second column of the multiple index. In the second table (jointab), the data is read sequentially within the index range.

Details about Join

Introduction

A join can be executed across a maximum of 64 (MaxDB Version 7.5) or 254 (MaxDB Version 7.6) tables. In contrast, only about 180 tables can be used in a join view due to internal restrictions.

The order in which the tables are specified in the FROM clause of the SQL statement has no influence on the order of processing.

The Optimizer has to ensure that the processing order of the tables involved in the join is optimal. Only if you use the hint ORDERED, the join is processed in the order of the tables specified in the FROM clause.

For an outer join the outer join table is set to the end of the processing sequence (see also Optimizer Parameter OPTIMIZE _JOIN_ OUTER). To do so the Optimizer analyses the individual column statistics of the individual table columns that are involved in the join for join strategies. The strategies named below refer to join transitions from a table (scantab) to the following table (jointab) that is involved in the join. You can only optimize equal conditions. To illustrate the strategies, we have provided an example that can be used for each of the following strategies. Example: SELECT * FROM scantab, jointab WHERE scantab.A = jointab.col1 AND scantab.B = jointab.col2