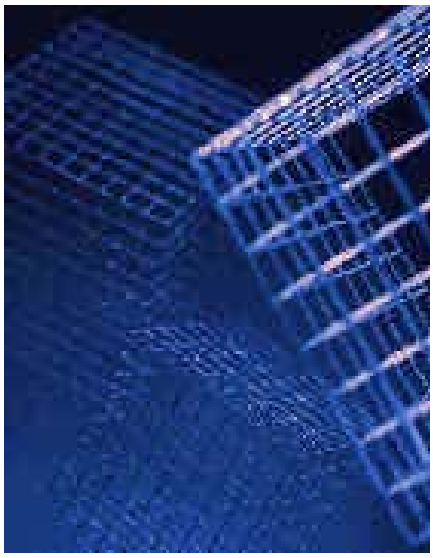
Optimizer: SAP DB



Version 7.4

Copyright

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Typographic Conventions

Type Style	Description
Example text	Words or characters that appear on the screen. These include field names, screen titles, pushbuttons as well as menu names, paths and options. Cross-references to other documentation.
Example text	Emphasized words or phrases in body text, titles of graphics and tables.
EXAMPLE TEXT	Names of elements in the system. These include report names, program names, transaction codes, table names, and individual key words of a programming language, when surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Screen output. This includes file and directory names and their paths, messages, source code, names of variables and parameters as well as names of installation, upgrade and database tools.
`EXAMPLE TEXT`	Keys on the keyboard, for example, function keys (such as `F2`) or the `ENTER` key.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.

Optimizer: SAP DB 7.4 6
Factors that Influence the Processing of an SQL Statement 6
Runtime Improvement for SQL Applications 7
Search Condition 8
Equality Condition 8
Area Condition 8
IN Condition 9
LIKE Condition 9
Search Strategy 10
Sequential Search 10
Search Conditions for Key Columns 10
Search Conditions for Inverted Columns 11
Examples: Search Conditions for Inverted Columns 11
Search Conditions for UPDATE Statements 12
Cost Determination 12
Search Conditions Linked with OR 13
Postponement of the Search to the FETCH Time 13
Join 14
List of All Search Strategies 15
CATALOG KEY ACCESS 16
CATALOG SCAN 16
CATALOG SCAN USING USER EQUAL CONDITION 16
DIFFERENT STRATEGIES FOR OR-TERMS 16
EQUAL CONDITION FOR INDEX 17
EQUAL CONDITION FOR INDEX (SUBQ) 17
EQUAL CONDITION FOR KEY COLUMN 17
EQUAL CONDITION FOR KEY COLUMN (SUBQ) 17
IN CONDITION FOR INDEX 17
IN CONDITION FOR KEY COLUMN 18
JOIN VIA INDEXED COLUMN 18
JOIN VIA KEY COLUMN 18
JOIN VIA KEY RANGE 18
JOIN VIA MULTIPLE INDEXED COLUMNS 18
JOIN VIA MULTIPLE KEY COLUMNS 19
JOIN VIA RANGE OF MULTIPLE INDEXED COL. 19
JOIN VIA RANGE OF MULTIPLE KEY COLUMNS 19
NO RESULT SET POSSIBLE 19
NO STRATEGY NOW (ONLY AT EXECUTION TIME) 19
RANGE CONDITION FOR KEY COLUMN 20

RANGE CONDITION FOR KEY COLUMN (SUBQ) ... 20
RANGE CONDITION FOR INDEX ... 20
RANGE CONDITION FOR INDEX (SUBQ) ... 20
INDEX SCAN ... 20
TABLE SCAN ... 21
EXPLAIN Statement ... 21
EXPLAIN Statement for Joins ... 21
EXPLAIN Statement for Complicated SELECT Statements ... 22
EXPLAIN Statement for SELECT Statements with Subqueries ... 22
EXPLAIN Statement: Columns O,D,T,M ... 22
Terms and Examples ... 23
Inverted Column ... 23
Inversion List ... 24
Examples ... 24
example ... 24
ind ... 24
indf ... 24
invcolumn1 ... 24
invcolumn2 ... 25
one ... 25
ten1 ... 25
ten2 ... 25

Optimizer: SAP DB 7.4

This documentation describes how the SAP DB Optimizer processes SQL statements. It provides information about the meaning and possibilities of creating optimal SQL statements for your application.

For general information about the SAP DB database system, see the documentation The SAP DB Database System at the following Internet address: **www.sapdb.org** → *Documentation.*

For information about the syntax of the SQL statements that can be entered in SAP DB, see the Reference Manual: SAP DB 7.4.

Optimizing SQL Statements

Determining the optimal SQL statements for a SAP DB application is an important task. Optimal SQL statements significantly influence the performance of the entire SAP DB application. An SQL statement is optimal if it is written in such a way that it can be processed in as short a time as possible with a minimal storage area requirement.

To process SQL statements optimally, the factors that influence the processing of an SQL statement [Page 6] must be considered. In particular, you should consider the influence of the search conditions [Page 8] on the processing of an SQL statement. There are a series of additional possibilities for runtime improvement for SQL applications [Page 7].

The SQL statements should be written in such a way that search strategies [Page 10] can be used. The following goals should be achieved in doing so:

Search only those lines that it is necessary to search

Restrict results tables and temporary results tables to the smallest possible size

Postponement of the Search to the FETCH Time [Page 13]

If a large number of changes has been made in a table, you should run the UPDATE STATISTICS statement. This statement updates the details about the tables that are necessary for the work of the Optimizer (such as value distribution, size of tables, and inversions).

If you consider all of these hints, you can write optimal SQL statements.

Checking SQL Statements

Check your SQL statements for correctness, for example, with the help of the SAP DB tool SQL Studio.

Analyze your SQL statements with regard to the performance that can be achieved through them. To do this, use the appropriate EXPLAIN statements [Page 21].

Only after you have completed these checks of your SQL statements should you insert them into your application program.

Factors that Influence the Processing of an SQL Statement

The following factors play a role during the processing of an SQL statement:

- Type of physical storage of a table

- Size of the tables (Number of lines and B* tree pages)
- Definition of the table (especially the key columns)
- Definition of indexes
- Type of SQL statement (SELECT, INSERT, UPDATE, DELETE)
- Elements of a SELECT statement (ORDER clause, UPDATE clause, DISTINCT specification, and specification of FOR REUSE)
- Changes that are specified in the UPDATE statement
- Search conditions [Page 8]

These factors especially influence the processing of an SQL statement if the SQL statement triggers a search of a large quantity of data. This generally happens with the following SQL statements:

```
SELECT ...
INSERT ... SELECT ...
UPDATE ... WHERE <search_condition>
DELETE ... WHERE <search_condition>
```

Runtime Improvement for SQL Applications

The following are general hints that can contribute to an improved runtime for SQL applications.

- When you are building a database, you should derive the definition of the tables from a previously performed examination of the structures. When defining the key columns, you should ensure that the columns that are especially selective, and for which search conditions are specified especially often, are placed at the beginning of the key. This creates the possibility of only having to consider a very small part of the table when processing a SELECT statement.
- Only columns with as high a selectivity as possible should be inverted [Page 23]. Do not choose columns such as gender or marital status as index keys, due to their low number of different values. These columns can only be used very rarely for a non-sequential search, as these would usually be more costly than a sequential search [Page 10].
- In the case of relatively static datasets, a large number of columns can be inverted. In doing so, you must ensure, as with the definition of the key columns, that you specify columns that are as selective as possible, and that are often used in equality conditions [Page 8] at the beginning of the index.
- You should never invert all of the columns that are used in search conditions. The space for the indexes and the cost for their maintenance is considerable.
- If a large number of changes has been made in a table, you should run the UPDATE STATISTICS statement.
- You should only formulate search conditions [Page 8] that cannot be fulfilled by all lines. Applications are often written so that the user determines the values of a search condition. If the user does not enter any values, standard values are inserted into the search condition, so that it always returns “true”. The database system must now evaluate this ineffective search condition for every line that is to be searched. It is better to execute different SELECT statements depending on a user entry.
- You should place the most selective search conditions at the beginning of the condition, as it may be possible to terminate the check earlier.

- The specification
  `columnx IN (1,13,24,...)`
  is better than
  `columnx=1 OR columnx=13 OR columnx=24 OR ...`

Search Condition

Search conditions (<search_condition>) are often specified in SQL statements. However, not all of the search conditions that can be entered are used by the Optimizer to determine an optimal search strategy [Page 10], but rather only a few of these possible search conditions. To be able to write optimal SQL statements, you should have a detailed knowledge of the search conditions that can be evaluated by the Optimizer.

The search conditions that the Optimizer can use to determine the optimal search strategy are the following:

- Equality Condition [Page 8]
- Area Condition [Page 8]
- IN Condition [Page 9]
- LIKE Condition [Page 9]

If the search conditions in an SQL statement cannot be converted into one of the forms listed above, they cannot be used when selecting a search strategy.

The order of search conditions that are linked by equal Boolean operators has no influence on the selection of a search strategy.

If you use UPDATE statements, you can only use the search conditions with restrictions (Search Conditions for UPDATE Statements [Page 12]).

Equality Condition

Equality conditions are search conditions [Page 8] with a comparison predicate, to which the following rules apply:

- The only operator is the **relational operator =**.
- Equality conditions have one of the following forms:
  `<column_spec> = <extended_value_spec>`
  `<column_spec> = <subquery>`

Only equality conditions that satisfy these prerequisites can use the Optimizer to determine an optimal search strategy [Page 10].

Conditions in the format `...NOT (<column_spec> <> <extended_value_spec>)` or `...NOT (<column_spec> <> <subquery>)` are, where possible, converted into an expression without NOT, with a correspondingly negated operator =. The system then continues processing the condition in this converted format.

Area Condition

Area conditions are search conditions [Page 8] with a comparison predicate or a BETWEEN predicate, to which the following prerequisites apply:

- The only valid operators are the **relational operators < | <= | => | >** and the operator **BETWEEN**.
- Area conditions have one of the following formats:
  `<column_spec> < < | <= | => | > > <extended_value_spec>`
  `<column_spec> BETWEEN <extended_value_spec> AND <extended_value_spec>`

Only area conditions that satisfy these prerequisites can use the Optimizer to determine an optimal search strategy [Page 10].

Whether the area is defined using a BETWEEN operator or using a combination of the relational operators <= and => is irrelevant to the search strategy.

Conditions in the format `...NOT (<column_spec> < < | <= | => | > <extended_value_spec>)` or. `...NOT (<column_spec> NOT BETWEEN <extended_value_spec> AND <extended_value_spec>)` are, where possible, converted to an expression without NOT, with a correspondingly negated operator. The system then continues processing the condition in this converted format.

IN Condition

IN conditions are search conditions [Page 8] with an IN predicate, to which the following rules apply:

- The operator is the **operator IN**.
- IN conditions have one of the following formats:
  `<column_spec> IN (<extended_value_spec>,...)`
  `<column_spec> IN <subquery>`

Only IN conditions that satisfy these prerequisites can use the Optimizer to determine an optimal search strategy [Page 10].

Conditions in the format `...NOT (<column_spec> NOT IN (<extended_value_spec>,...))` or. `...NOT (<column_spec> NOT IN <subquery>)` are, where possible, converted to an expression without NOT, and with a correspondingly negated operator. The system then continues processing the condition in this converted format.

LIKE Condition

LIKE conditions are search conditions [Page 8] with a LIKE predicate, to which the following rules apply:

- The operator is the **operator LIKE**.
- LIKE conditions have the following format:
  `<column_spec> LIKE <extended_value_spec>`
- The value specification used (`extended_value_spec`) may not begin with the SQL syntax elements match_string or match_set.

Only LIKE conditions that satisfy these prerequisites can use the Optimizer to determine an optimal search strategy [Page 10].

Conditions with the format `...NOT (<column_spec> NOT LIKE <extended_value_spec>)` are, where possible, converted into an expression

`<column_spec> LIKE <extended_value_spec>`. The system then continues processing the condition in this converted format.

Search Strategy

SQL statements should be written in such a way that the following aims for the processing of an SQL statement are achieved when a search strategy is selected:

- Search only those lines that it is necessary to search
- Restrict results tables and temporary results tables to the smallest possible size

The search strategy that is selected is largely dependent upon the form of the search condition [Page 8] specified in the SQL statement. The search strategy is also selected so that the costs for the processing of the SQL statement are minimized.

For more detailed information about the selection of the search strategy, see the following sections:

- Sequential Search [Page 10]
- Search Conditions for Key Columns [Page 10]
- Search Conditions for Inverted Columns [Page 11]
- Search Conditions for UPDATE Statements [Page 12]
- Cost Determination [Page 12]
- Search Conditions Linked with OR [Page 13]
- Postponement of the Search to the FETCH Time [Page 13]
- Join [Page 14]
- List of All Search Strategies [Page 15]

You can display the search strategy used for an SQL statement with the help of the EXPLAIN statement [Page 21].

Sequential Search

In principle, every search can be performed sequentially through the entire table. This is necessary in the following cases:

- No search conditions [Page 8] have been specified
- No search condition has been specified either for key columns or for inverted columns.

If the possible non-sequential search strategies [Page 10] would be more costly than the sequential search, the table is processed sequentially.

Search Conditions for Key Columns

If search conditions [Page 8] are specified for key columns, the search is limited to the corresponding part of the table, that is, the search strategy [Page 10] is determined as specified in the table.

Search Condition	Search Strategy

Equality condition [Page 8] for every key column	The system accesses the appropriate table line(s) directly.
IN condition [Page 9] for one key column and an equality condition for all other key columns	The system accesses the appropriate table line(s) directly.
Any other search condition for the key columns	Upper and lower limits are created for the key columns for the valid search area. All search strategies, including strategies through inversions, use the knowledge of this valid search area.

Search Conditions for Inverted Columns

If you specify search conditions [Page 8] for inverted columns [Page 23], the search strategy [Page 10] is determined by the system as specified in the table.

Search Condition	Search Strategy
Equality condition [Page 8] for all inverted columns	Only those lines whose keys are contained in the associated inversion list [Page 24] are accessed.
IN condition [Page 9] for inverted columns	The system accesses the lines whose keys are contained in the inversion lists.
Area Conditions [Page 8] There is only one condition (<, <=, >, >=) for one of the two area limits (upper or lower limit).	The system accesses the lines whose keys are contained in the inversion lists that were determined by the area.
Area conditions Both area limits are specified. It is irrelevant to the selection of the search strategy whether this specification is made using a BETWEEN operator or using two conditions (<= or >=) for the same column linked by AND.	The system accesses the lines whose keys are contained in the inversion lists that were determined by the area.

In the case of the equality/IN conditions and the area conditions, there are also queries for which it is not necessary to access the lines, as all required values are contained in the inversion list(s).

See also:

Examples: Search Conditions for Inverted Columns [Page 11]

Examples: Search Conditions for Inverted Columns

Search conditions for inverted columns [Page 11] of the example [Page 24] table.

Excerpt from SQL Statement	Search Strategy
`... where firstkey >= 123`	The table is searched sequentially from the line with the key ‘123’.

`... where invcolumn1 = 'Miller' and firstkey >= 123`	The inversion list with the value ‘Miller’ is searched from the key ‘123’ to the end of the list.
`... where invcolumn1 = 'Miller' and invcolumn2 < 'C'`	The system creates a logical inversion list that contains all inversion lists of invcolumn 2 that begin with a value smaller than 'C' (' ', 'A', 'B'). The intersection of the logical inversion list and the inversion list with the value ‘Miller’ is determined and the entirety of this intersection is processed.
`... where invcolumn1 = 'Miller' and invcolumn2 = 'Don'`	The intersection of both inversion lists is determined and processed completely.
`... where invcolumn1 IN ('Miller', 'Smith', 'Hawk')`	The entirety of three inversion lists are processed.
`... where invcolumn2 > 8965 and firstkey = 34 and secondkey between 12 and 18`	All inversion lists of invcolumn2 whose values are greater than 8965 are processed. However, they are only considered within the key limits ’34, 12’ and ’34, 18’.
`... where multinvcolumn1 = 'Düsseldorf' and multinvcolumn2 = '40223' and multinvcolumn3 = 10000`	The entirety of the named index ind [Page 24] with the values ‘Düsseldorf’, ‘40223’ and 10000 is processed.
`...` where multinvcolumn1 = 'Düsseldorf' and multinvcolumn2 between '40221' and '40238'	The inversion list within and including the values ‘Düsseldorf’, ‘40221’ (binary zeros), and ‘Düsseldorf’, ‘40238’ (binary ones) is processed.

Search Conditions for UPDATE Statements

If you use UPDATE statements, the following restrictions apply to the use of search conditions [Page 8].

If the new value of a column is calculated in an arithmetical expression, an inversion of this column, that is, the corresponding index, cannot be used during the search.

```
UPDATE <table_name> SET columnx = columnx + 3 WHERE columnx
IN (100, 103, 106, 109, 112)
```

This SQL statement can lead to erroneous results, if the inversion lists were gradually processed with the values 100, 103, 106, 109, and 112.

You should also consider this factor when using FOR UPDATE in the SELECT statement.

Cost Determination

The system determines the cost of every possible search strategy [Page 10].

This is necessary for the following reasons:

- There can be several search strategies, of which the best must be determined

- There are cases, in which searching with the help of an index would be more costly than a sequential search [Page 10].

The costs relate to the expected number of I/O processes that would have to be executed due to the chosen search strategy.

These costs are also output as a result of the EXPLAIN statement [Page 21].

Search Conditions Linked with OR

There are search conditions [Page 8] linked with OR. The system analyzes the individual search conditions.

If the system finds an equality condition [Page 8] for key columns, it ignores the other search conditions linked with OR.

If the system does not find any equality conditions for key columns, the other search conditions linked with OR, that the system has not considered until then, are analyzed.

Procedure

1. Conversion of the search condition into the disjunctive normal form
   Example:
   ```
   b1 and b2 and (b3 or b4 and b5)
   ```
   results in the disjunctive normal form
   ```
   (b1 and b2 and b3) or (b1 and b2 and b4 and b5)
   ```
2. Analysis of the new expression
   Every bracketed expression is analyzed separately. If the analysis of the first expression results in the best search strategy, this is used. However, if the analysis of each bracketed expression provides a better search strategy, there are usually as many search strategies as there are bracketed expressions to be analyzed.
3. Cost determination
   The costs of the various search strategies are added together. If the total is lower than the determined cost for the search strategy without consideration of the search conditions linked with OR, the various search strategies are used.

If applicable, the EXPLAIN statement [Page 21] displays the search strategy DIFFERENT STRATEGIES FOR OR-TERMS [Page 16].

Postponement of the Search to the FETCH Time

A goal of optimization is to save storage area; that is, the creation of results tables should be avoided. Except for the cases in which the syntax element FOR REUSE or the existence of a join forces the creation of results tables, the creation of a results table is avoided where possible.

A possible search strategy [Page 10] is therefore the postponement of the search to the FETCH time (FETCH statement), without the creation of a results table. In this way, no memory is used for results, quick access to the first results is possible, and a quicker comparison can be made of the received and the desired results.

The postponement of the search to the FETCH time is not possible for all SQL statements. The following is an overview of a number of SQL statements for which a postponement is **not** permissible:

SELECT for multiple tables (Join)
SELECT ... FOR REUSE

SELECT DISTINCT ... (in most cases)
SELECT ... ORDER BY ... (in most cases)

SELECT ... ORDER BY

If the ORDER clause was specified, it is only possible to avoid the creation of a results table, if all of the following conditions are met:

- Neither a DISTINCT specification (<distinct spec>) (with a few exceptions), nor the syntax element FOR REUSE, are specified.
- The columns by which the system is to sort form an index in the specified order and with the specified sorting (ascending or descending).

You can use the EXPLAIN statement [Page 21] to display whether a results table is created (RESULT IS COPIED) or not (RESULT IS NOT COPIED).

Join

Some background information about the processing of joins is required to understand how search strategies [Page 10] are found for joins.

Creating a Results Table for All Tables Linked by a Join

You can use a join to link up to 64 tables with each other. The system links the tables in steps; that is, one join is created from two tables. An additional table is then linked with the join result to form a new join.

1. The first table is searched.
2. The result is stored in a temporary results table.
3. The temporary results table is sorted by the join columns of the following join step.
4. The next table that still exists is searched.
5. The existing temporary results table is linked with the new table.
6. The result is stored in a new temporary results table.
7. This temporary results table is sorted by the join columns of the following join step.
8. The old temporary results table is deleted.
9. If there are more tables, the procedure is repeated from step 4.

The last temporary results table is the results table that the user wants.

Optimization of the Tables Linked by a Join

You can only save time or space during the processing of tables linked by a join if the temporary results tables are as small as possible and that the system can access the lines of each of the new tables to be linked as directly as possible.

The Optimizer therefore attempts to place small tables with restrictive search conditions at the beginning of the series of tables to be processed, in order to maintain small temporary results tables.

The order in which the tables are specified in the FROM clause of the SELECT statement has no influence on the order of processing. The Optimizer determines the order of processing itself.

Search Strategies

You can display the search strategies using the EXPLAIN statement (EXPLAIN Statement for Joins [Page 21]).

The following search strategies can be used to access lines in the new table, starting from the join column values of the old temporary results table.

JOIN VIA INDEXED COLUMN [Page 18]

JOIN VIA KEY COLUMN [Page 18]

JOIN VIA KEY RANGE [Page 18]

JOIN VIA MULTIPLE INDEXED COLUMNS [Page 18]

JOIN VIA MULTIPLE KEY COLUMNS [Page 19]

JOIN VIA RANGE OF MULTIPLE INDEXED COL. [Page 19]

JOIN VIA RANGE OF MULTIPLE KEY COLUMNS [Page 19]

Multi-Column Key and Index Strategies

If two columns to be compared in a join step are not of the same length, not all of the search strategies can be used. We recommend that you define columns that are to be linked using a join with the same domain to avoid this restriction.

List of All Search Strategies

You can use the EXPLAIN statement [Page 21] to display the selected search strategy [Page 10].

List of All Possible Search Strategies

CATALOG KEY ACCESS [Page 16]

CATALOG SCAN [Page 16]

CATALOG SCAN USING USER EQUAL CONDITION [Page 16]

DIFFERENT STRATEGIES FOR OR-TERMS [Page 16]

EQUAL CONDITION FOR INDEX [Page 17]

EQUAL CONDITION FOR INDEX (SUBQ) [Page 17]

EQUAL CONDITION FOR KEY COLUMN [Page 17]

EQUAL CONDITION FOR KEY COLUMN (SUBQ) [Page 17]

IN CONDITION FOR INDEX [Page 17]

IN CONDITION FOR KEY COLUMN [Page 18]

JOIN VIA INDEXED COLUMN [Page 18]

JOIN VIA KEY COLUMN [Page 18]

JOIN VIA KEY RANGE [Page 18]

JOIN VIA MULTIPLE INDEXED COLUMNS [Page 18]

JOIN VIA MULTIPLE KEY COLUMNS [Page 19]

JOIN VIA RANGE OF MULTIPLE INDEXED COL. [Page 19]

JOIN VIA RANGE OF MULTIPLE KEY COLUMNS [Page 19]

NO RESULT SET POSSIBLE [Page 19]

NO STRATEGY NOW (ONLY AT EXECUTION TIME) [Page 19]

RANGE CONDITION FOR KEY COLUMN [Page 20]

RANGE CONDITION FOR KEY COLUMN (SUBQ) [Page 20]

RANGE CONDITION FOR INDEX [Page 20]

RANGE CONDITION FOR INDEX (SUBQ) [Page 20]

INDEX SCAN [Page 20]

TABLE SCAN [Page 21]

CATALOG KEY ACCESS

CATALOG KEY ACCESS is a search strategy [Page 10]. The SQL statement contains equality conditions [Page 8], so that the query can be processed using an access to key columns in the database catalog (for example, equality conditions for OWNER and TABLENAME for queries for DOMAIN.TABLES).

See also:

List of All Search Strategies [Page 15]

CATALOG SCAN

CATALOG SCAN is a search strategy [Page 10]. The system performs a sequential search of the database catalog.

See also:

List of All Search Strategies [Page 15]

CATALOG SCAN USING USER EQUAL CONDITION

CATALOG SCAN USING USER EQUAL CONDITION is a search strategy [Page 10]. The system performs a sequential search [Page 10] of the database catalog entries that describe objects of the identified user.

See also:

List of All Search Strategies [Page 15]

DIFFERENT STRATEGIES FOR OR-TERMS

DIFFERENT STRATEGIES FOR OR-TERMS is a search strategy [Page 10]. The system converted and analyzed search conditions linked with OR [Page 13]. If the system finds an equality condition [Page 8] for key fields, it ignores the other search conditions linked with OR. If the system does not find an equality condition, the system determines different search strategies for the different parts of the search conditions linked with OR.

See also:

List of All Search Strategies [Page 15]

EQUAL CONDITION FOR INDEX

EQUAL CONDITION FOR INDEX is a search strategy [Page 10]. An equality condition [Page 8] was specified for all columns of an index. The system accesses the corresponding table lines directly, with the help of the appropriate inversion list [Page 24].

See also:

List of All Search Strategies [Page 15]

EQUAL CONDITION FOR INDEX (SUBQ)

EQUAL CONDITION FOR INDEX (SUBQ) is a search strategy [Page 10]. An equality condition [Page 8] was specified for all columns of an index. The system accesses the corresponding table lines for each hit line of the subquery directly, with the help of the appropriate inversion list [Page 24].

See also:

List of All Search Strategies [Page 15]

EQUAL CONDITION FOR KEY COLUMN

EQUAL CONDITION FOR KEY COLUMN is a search strategy [Page 10]. The table has only one key column, for which an equality condition [Page 8] was specified. The system accesses the appropriate table lines directly.

See also:

List of All Search Strategies [Page 15]

EQUAL CONDITION FOR KEY COLUMN (SUBQ)

EQUAL CONDITION FOR KEY COLUMN (SUBQ) is a search strategy [Page 10]. The table has only one key column, for which an equality condition [Page 8] was specified. The system accesses the appropriate table line directly for each hit line of the subquery.

See also:

List of All Search Strategies [Page 15]

IN CONDITION FOR INDEX

IN CONDITION FOR INDEX is a search strategy [Page 10]. An IN condition [Page 9] was specified for the first column of an index that consists of multiple columns, or equality conditions [Page 8] were specified for the first k-1 columns of an index of this type and an IN condition was specified for the k-th column. The system accesses the corresponding table lines directly, with the help of the appropriate inversion lists [Page 24].

See also:

List of All Search Strategies [Page 15]

IN CONDITION FOR KEY COLUMN

IN CONDITION FOR KEY COLUMN is a search strategy [Page 10]. The table has only one key column, for which an IN condition [Page 9] was specified. The system directly accesses the table lines concerned.

See also:

List of All Search Strategies [Page 15]

JOIN VIA INDEXED COLUMN

JOIN VIA INDEXED COLUMN is a search strategy [Page 10] for joins [Page 14]. An index was created for the join column that contains only this column. The system accesses the data through the index of the column whose name is displayed.

See also:

List of All Search Strategies [Page 15]

JOIN VIA KEY COLUMN

JOIN VIA KEYCOLUMN is a search strategy [Page 10] for joins [Page 14]. The join column is the only key column. The system accesses the table lines in the new table directly.

See also:

List of All Search Strategies [Page 15]

JOIN VIA KEY RANGE

JOIN VIA KEY RANGE is a search strategy [Page 10] for joins [Page 14]. The join column whose name is displayed is the first key column. Within the key area, the system accesses the table lines in the new table sequentially.

See also:

List of All Search Strategies [Page 15]

JOIN VIA MULTIPLE INDEXED COLUMNS

JOIN VIA MULTIPLE INDEXED COLUMNS is a search strategy [Page 10] for joins [Page 14]. The specified join columns can be combined into a complete index that incorporates multiple columns. The system accesses the data through this index.

See also:

List of All Search Strategies [Page 15]

JOIN VIA MULTIPLE KEY COLUMNS

JOIN VIA MULTIPLE KEY COLUMNS is a search strategy [Page 10] for joins [Page 14]. The specified join columns can be combined into a multi-column key for the new table. The system accesses the table lines in the new table directly.

See also:

List of All Search Strategies [Page 15]

JOIN VIA RANGE OF MULTIPLE INDEXED COL.

JOIN VIA RANGE MULTIPLE INDEXED COL. is a search strategy [Page 10] for joins [Page 14]. The specified join columns can be combined to form the start of a complete index that incorporates multiple columns. Within the index area, the system accesses the lines of the table.

See also:

List of All Search Strategies [Page 15]

JOIN VIA RANGE OF MULTIPLE KEY COLUMNS

JOIN VIA RANGE MULTIPLE KEY COLUMNS is a search strategy [Page 10] for joins [Page 14]. The specified join columns can be combined to form the start of a multi-column key for the new table. Within the key area, the system accesses the table lines in the new table sequentially.

See also:

List of All Search Strategies [Page 15]

NO RESULT SET POSSIBLE

The Optimizer identifies search conditions [Page 8] that logically cannot have a result (NO RESULT SET POSSIBLE). In this case, no search is performed.

See also:

List of All Search Strategies [Page 15]

NO STRATEGY NOW (ONLY AT EXECUTION TIME)

NO STRATEGY NOW (ONLY AT EXECUTION TIME) is a search strategy [Page 10]. In a subquery, the corresponding column values are only known at the execution time for the subquery. The search strategy for the most effective access to the corresponding table for the subquery is only determined when these values are available.

See also:

List of All Search Strategies [Page 15]

RANGE CONDITION FOR KEY COLUMN

RANGE CONDITION FOR KEY COLUMN is a search strategy [Page 10], with which a sequential search [Page 10] is performed in part of the table.

See also:

List of All Search Strategies [Page 15]

RANGE CONDITION FOR KEY COLUMN (SUBQ)

RANGE CONDITION FOR KEY COLUMN (SUBQ) is a search strategy [Page 10], with which a sequential search [Page 10] is performed in part of the table with every hit line of the subquery.

See also:

List of All Search Strategies [Page 15]

RANGE CONDITION FOR INDEX

RANGE CONDITION FOR INDEX is a search strategy [Page 10]. Equality [Page 8] or area conditions [Page 8] have been specified for the first k columns of an index. The system accesses the corresponding table lines directly, with the help of the inversion lists [Page 24] in the area.

See also:

List of All Search Strategies [Page 15]

RANGE CONDITION FOR INDEX (SUBQ)

RANGE CONDITION FOR INDEX (SUBQ) is a search strategy [Page 10]. Equality [Page 8] or area conditions [Page 8] have been specified for the first k columns of an index. The system accesses the corresponding table lines for each hit line of the subquery directly, with the help of the appropriate inversion lists [Page 24] in the area.

See also:

List of All Search Strategies [Page 15]

INDEX SCAN

INDEX SCAN is a search strategy [Page 10], with which a sequential search [Page 10] is performed in the entire specified index.

See also:

List of All Search Strategies [Page 15]

TABLE SCAN

TABLE SCAN is a search strategy [Page 10], with which a sequential search [Page 10] is performed in the entire table.

See also:

List of All Search Strategies [Page 15]

EXPLAIN Statement

The user can use EXPLAIN statements to find out which search strategy [Page 10] the system is using to perform the specified SELECT statement. For an explanation of all search strategies, see the List of All Search Strategies [Page 15].

Information about the EXPLAIN Statement

- For a complete syntax description for the EXPLAIN statement, see the *Reference Manual: SAP DB 7.4*, under EXPLAIN Statement.
- Result of the cost determination for a search strategy: You can find this in the PAGECOUNT column of the EXPLAIN statement output
- Special features for an EXPLAIN statement for joins [Page 21]
- Special features for an EXPLAIN statement for complicated SELECT statements [Page 22]
- Special features for an EXPLAIN statement for SELECT statements with subqueries [Page 22]
- Explanation of the EXPLAIN Statement: Columns O,D,T,M [Page 22]

EXPLAIN Statement for Joins

You can also use the EXPLAIN statement [Page 21] for joins [Page 14].

Result of the EXPLAIN Statement

- Display of the order in which the tables are processed when the SELECT statement is executed
- Display of whether the join column values of the old temporary results table can be accessed directly or through an inversion to the lines of a new table
- Display of the strategy used to search in the new table, if the lines of this table cannot be accessed directly or using an inversion

```
EXEC SQL EXPLAIN SELECT one [Page 25].key ten1 [Page 25].keyft1,
ten2 [Page 25].keyft2
FROM one, ten1, ten2
WHERE ten1.keyft1 < 100
  AND ten1.ft1    = one.keyf
  AND one.indf [Page 24]    = ten2.keyft2
  AND ten2.keyft2 < 100;
```

These EXPLAIN statement produces the following output:

TABLE NAME	COLUMN_ OR_INDEX	STRATEGY	PAGE COUNT
TEN1		RANGE CONDITION FOR KEY COLUMN [Page 20]	1250
ONE	KEF	JOIN VIA KEY COLUMN [Page 18]	125
TEN2	KEYFT2	JOIN VIA KEY COLUMN	1463
		RESULT IS COPIED, COSTVALUE IS	97

EXPLAIN Statement for Complicated SELECT Statements

Certain SELECT statements are so complicated that they are separated into several internal SELECT steps.

For SELECT statements of this type, the EXPLAIN statement [Page 21] outputs several search strategies [Page 10]. In this output, INTERNAL TEMPORARY RESULT is given as the table name. Searches through internal temporary results of this type can only ever be performed sequentially.

EXPLAIN Statement for SELECT Statements with Subqueries

To determine the cost of search strategies [Page 10] with which, the value of a column is compared with the hit lines of a subquery, you must know the number of hit lines. You can therefore only determine the cost of the search strategy after the subquery has been processed.

For a SELECT statement with subqueries, the EXPLAIN statement [Page 21] determines the possible search strategies, but does not execute the contained subqueries. Therefore, the search strategy NO STRATEGY NOW (ONLY AT EXECUTION TIME) [Page 19] is usually displayed for the outer SQL statement.

SQL statements that can only be processed on an index are an exception to this rule.

EXPLAIN Statement: Columns O,D,T,M

Column O (Only Index)

'*'

This strategy uses only the specified index to process the command. The system does not access the basis table data. To do this, it is necessary that only columns that are contained in the index structure are addressed as a result of selected columns (<select_column>) or in a WHERE clause, if one exists.

Column D (Distinct Optimization)

Column D only has an entry if column O contains the character '*'.

- 'C' (Complete Secondary Key)
  All columns of an index (and only these) are specified as a result of selected columns (<select_column>), after the keyword DISTINCT (DISTINCT specification (distinct_spec)), in any order. The system accesses the values for each index column only once. The system does not create a results table (such as `SELECT DISTINCT <all_columns_of_the_index> FROM ...`).
- 'P' (Partial Secondary Key)
  The first k (k < Total number of index columns) columns of an index are specified are specified in any order as a result of selected columns (<select_column>), after the key word DISTINCT. The system accesses the values for each index column only once. The system does not create a results table (such as `SELECT DISTINCT <first_k_columns_of_the_index> FROM ...`).
- 'K' (Primary Key)
  All columns of an index, and the first k (k <= total number of columns in the key) columns of the key are specified in any order as a result of selected columns (<select_column>), after the keyword DISTINCT. The system accesses the values for the corresponding index and key columns only once. The system does not create a results table (such as `SELECT DISTINCT <all_columns_of_the_index_+_first_k_columns_of_the_key> FROM ...`).

Column T (Temporary Index)

'*'

A temporary index is created internally, in which the keys of the hit lines determined through the corresponding index are stored in ascending order. The system accesses the basis table using this temporary index.

Column M (More Qualifications)

'*'

There are search conditions for index or key columns that cannot be used for the direct containment of the area for an index access (for example, in the case of an equality [Page 8] /IN condition [Page 9], for the first and third columns of a multi-column index, only the first search condition of the search strategy [Page 10] is used for access). These search conditions flow into the corresponding index strategy. They are used to restrict accesses to the basis table.

Terms and Examples

Inverted Column [Page 23]

Inversion List [Page 24]

Examples [Page 24]

Inverted Column

An inverted column is a column that is part of at least one index.

Inversion List

An inversion list is a list of keys that belong to an index. In each column of the index, all table lines whose keys are specified in the list contain the same value for this column.

Examples

- Definition of tables (example [Page 24], one [Page 25], ten1 [Page 25], ten2 [Page 25]).
- Definition of indexes (invcolumn1 [Page 24], invcolumn2 [Page 25], indf [Page 24], ind [Page 24]).

example

Definition of the table `example`

```
EXEC SQL CREATE TABLE example
  (firstkey         FIXED (3) KEY,
   secondkey        FIXED (4) KEY,
   normalcolumn     FIXED (5),
   invcolumn1       CHAR (15),
   invcolumn2       CHAR (9),
   multinvcolumn1   CHAR (22),
   multinvcolumn2   CHAR (5,
   multinvcolumn3   CHAR (8,
   PRIMARY KEY (firstkey, secondkey));
```

Two single-column indexes (invcolumn1 [Page 24] and invcolumn2 [Page 25]), and one multi-column index ind [Page 24] are defined for the example table.

ind

Definition of the index `ind`

```
EXEC SQL CREATE INDEX ind ON example [Page 24] (multinvcolumn1,
multinvcolumn2, multinvcolumn3);
```

indf

Definition of the index `indf`

```
EXEC SQL CREATE INDEX indf on one [Page 25] (indf);
```

invcolumn1

Definition of the index `invcolumn1`

```
EXEC SQL CREATE INDEX invcolumn1 ON example [Page 24] (invcolumn1);
```

invcolumn2

Definition of the index `invcolumn2`

```
EXEC SQL CREATE INDEX invcolumn2 ON example [Page 24] (invcolumn2);
```

one

Definition of the table `one`

```
EXEC SQL CREATE TABLE one
  (keyf          FIXED (6),
   indf          FIXED (3),
   ...,
   PRIMARY KEY (keyf));   /*1000 rows*/
```

The index indf [Page 24] is created for the table `one`.

ten1

Definition of the table `ten1`

```
EXEC SQL CREATE TABLE ten1
  (keyft1          FIXED (6),
   ft1             FIXED (3),
   ...,
   PRIMARY KEY (keyft1));   /*10000 rows*/
```

ten2

Definition of the table `ten2`

```
EXEC SQL CREATE TABLE ten2
  (keyft2          FIXED (6),
   ...,
   PRIMARY KEY (keyft2));   /*10000 rows*/
```