# SAP® MaxDB™
# Introduction to Query Optimization
# Version 7.8

Heike Gursch
Christiane Hienger

THE BEST-RUN BUSINESSES RUN SAP™

# Optimization Overview

Introdcution

Explain

Query Rewrite

Single table optimizer

Join Optimizer

Update Statistics

File Directory counter

## Goals of Optimization

Goal: Minimizing resource-consumption like

- CPU-time
- **I/O-load**
- Memory
- Disk space

SQL Commands affected by optimization

- SELECT
- UPDATE
- DELETE
- INSERT/SELECT



SQL Optimizer is a part of the kernel of the database system. It analyzes SQL queries and selects the best search strategy for accessing the data. You can specify the search condition in the SQL statement in the WHERE clause or via the join condition.

Goals of Optimization:
An SQL performance analysis involves the identification, analysis, and optimization of SQL statements that are responsible for the highest load as regards I/O at the database level. These statements are also called "processing-intensive SQL statements".

Regular analysis and optimization of expensive SQL statements provides THE most important basis for high-performance system operation. Resource-intensive SQL statements are directly responsible for increased I/O and CPU activities, and therefore result in a poor data cache hit ratio.
More information can be found in SAP note 819324: FAQ SAP MaxDB SQL Optimization

## Cost-based Optimizer

Cost-based optimizer

- Search strategy is determined via
  - current column content (values)
  - available indexes
  - estimated count of (page) accesses
- ‚***Lowest cost' strategy will be used.***



Cost-based optimizers determine the best search strategy with the help of information about the size of the table and values within the table columns.

A cost-benefit plan is created for the various access options. The best strategy is chosen to execute the command depending on the values sent in the WHERE condition. Therefore, the eventual search strategy can only be determined at the time of execution.

**MaxDB supports cost-based optimizers.**

Before the optimization **Query Rewrite** checks if the statement can be rewritten in a reasonable way. This check and conversion is done rule-based.

## Which condition will be evaluated ?

Single table SELECT

- Column = value
- Column <, <=, >=, > value
- Column BETWEEN value AND value
- Column IN ( value, value, ... )
- Column LIKE string value (including %,?,...)
- Column = (ANY) <subquery>
- Column IN <subquery>



What is a selection or also called search condition?

Selection conditions ( search conditions) are specified in the WHERE part of the SQL statement. Within the framework of a selection condition, a column is compared with one or several actual values (for example, "MANDT = :A0", "BDAT greater than '20050821' ").

Search conditions used by the optimizer to determine the optimal search strategy are:

- Equality conditions
- Range conditions
- IN conditions
- LIKE conditions

The best strategy is chosen by the optimizer. The basis of decision making are 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.

## Introduction into SQL Optimization

SELECT * FROM zztele WHERE STR = ‚Wexstr‘

| NAME | VORNAME | STR | NR | PLZ | ORT | CODE | ADDINFO |
|---|---|---|---|---|---|---|---|
| A-J de Groot | Hugo | Dummy | 1 | 10000 | Berlin | | Growth Mkt-Sapier |
| A-J de Groot | Hugo | Dummy1 | 2 | 10001 | Berlin | | Growth Mkt-Sapier |
| A-J de Groot | Hugo | Dummy2 | 3 | 10002 | | | Growth Mkt-Sapier |
| A-J de Groot | Hugo | Dummy3 | 4 | 10003 | | | Growth Mkt-Sapier |
| A-J de Groot | Hugo | Dummy4 | 5 | 10004 | | | Growth Mkt-Sapier |
| A-J de Groot | Hugo | Wexstr | 6 | 10005 | Berlin | | Growth Mkt-Sapier |
| A.S. | Raghavendra | Dummy | 7 | 10006 | | | Development India |
| A.S. | Raghavendra | Dummy1 | 8 | 10007 | | | Development India |
| A.S. | Raghavendra | Dummy2 | 9 | 10008 | Berlin | | Development India |
| A.S. | Raghavendra | Dummy3 | 10 | 10009 | | | Development India |
| A.S. | Raghavendra | Dummy4 | 11 | 10010 | | | Development India |
| A.S. | Raghavendra | Wexstr | 12 | 10011 | | | Development India |
| ABABSA | Monia | Dummy | 13 | 10012 | | | GESTION INDUSTR |

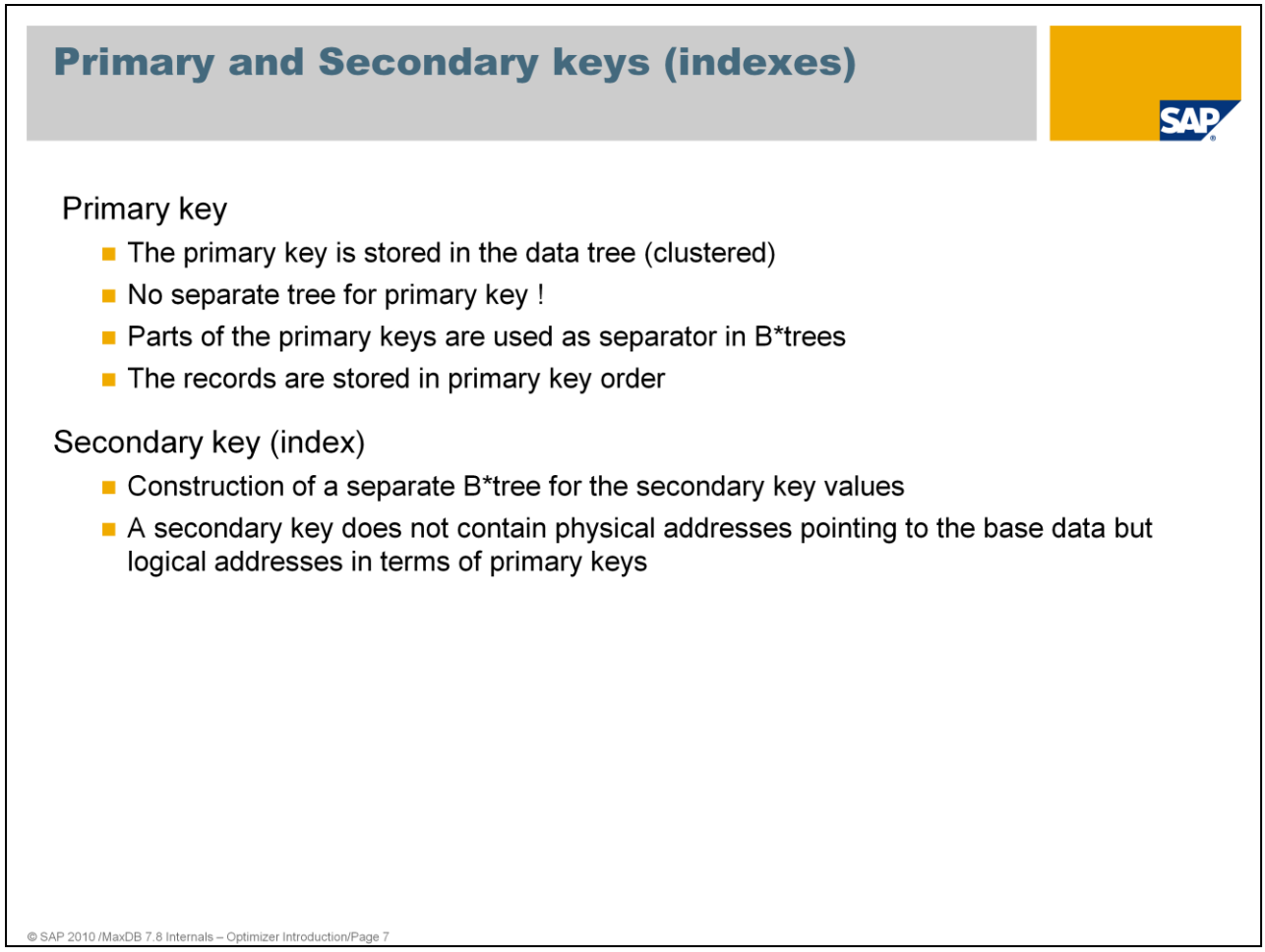


To fullfill the qualification (WHERE STR = ‚Wexstr‘) the total table can be read and each record can be compared with the qualification.

To minimize the costs the optimizer tries to reduce the area on the table which has to be read to check the qualification and deliver the result.

If the table is sorted by the columns of the qualification, binary search is possible. An area can be found, which includes all the requested result (START and STOP key). You can sort the table by a single column or several columns.

These kind of sorts are called Index. A special kind of index is the primary key.




Each database table has a primary key (primary index). The primary key is either defined by the user or generated by the system. A user-defined primary key can consist of one or more columns. The primary key must have a unique value for each table row.

The MaxDB primary key is a UNIQUE index that is implemented directly on the data tree. The data is sorted by the primary key.

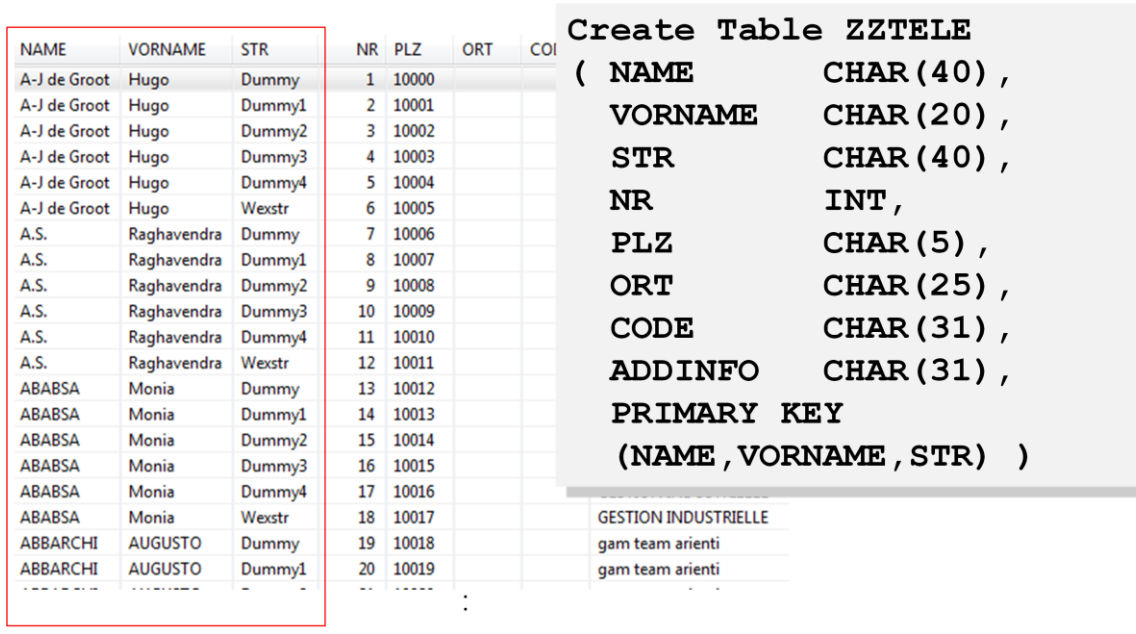
A separate B* tree is created for a secondary key (or index). The secondary key (index) contains no physical addresses on the data tree, instead it contains logical addresses in the form of primary keys. An index is a database object that can be created for an individual column or a series of columns in a database table.

The data of the secondary index is sorted by the index column(s).

You can create an index to speed up the search for database records in a table. In technical terms, indexes are data structures (consisting of one or more primary key 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 (index columns). Due to this type of storage, the table data can be accessed faster using the indexed columns.

For more information about indexes use SAP note 928037 FAQ SAP MaxDB Indexes

## Table Examples: ZZTELE



| NAME | VORNAME | STR | NR | PLZ | ORT | COI | |
|---|---|---|---|---|---|---|---|
| A-J de Groot | Hugo | Dummy | 1 | 10000 | | | |
| A-J de Groot | Hugo | Dummy1 | 2 | 10001 | | | |
| A-J de Groot | Hugo | Dummy2 | 3 | 10002 | | | |
| A-J de Groot | Hugo | Dummy3 | 4 | 10003 | | | |
| A-J de Groot | Hugo | Dummy4 | 5 | 10004 | | | |
| A-J de Groot | Hugo | Wexstr | 6 | 10005 | | | |
| A.S. | Raghavendra | Dummy | 7 | 10006 | | | |
| A.S. | Raghavendra | Dummy1 | 8 | 10007 | | | |
| A.S. | Raghavendra | Dummy2 | 9 | 10008 | | | |
| A.S. | Raghavendra | Dummy3 | 10 | 10009 | | | |
| A.S. | Raghavendra | Dummy4 | 11 | 10010 | | | |
| A.S. | Raghavendra | Wexstr | 12 | 10011 | | | |
| ABABSA | Monia | Dummy | 13 | 10012 | | | |
| ABABSA | Monia | Dummy1 | 14 | 10013 | | | |
| ABABSA | Monia | Dummy2 | 15 | 10014 | | | |
| ABABSA | Monia | Dummy3 | 16 | 10015 | | | |
| ABABSA | Monia | Dummy4 | 17 | 10016 | | | |
| ABABSA | Monia | Wexstr | 18 | 10017 | | | GESTION INDUSTRIELLE |
| ABBARCHI | AUGUSTO | Dummy | 19 | 10018 | | | gam team arienti |
| ABBARCHI | AUGUSTO | Dummy1 | 20 | 10019 | | | gam team arienti |

```
Create Table ZZTELE
( NAME        CHAR(40),
  VORNAME     CHAR(20),
  STR         CHAR(40),
  NR          INT,
  PLZ         CHAR(5),
  ORT         CHAR(25),
  CODE        CHAR(31),
  ADDINFO     CHAR(31),
  PRIMARY KEY
  (NAME,VORNAME,STR) )
```

**# of records: around 115,000**



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, first name, street

You can get the table and the primary key definition with the following SQL statement:
*Select * from domain.columns where tablename = 'ZZTELE'*

## Table Examples: ZZSTADTTEIL , ZZMASTER

| PLZ | ORT | STADTTEIL |
|---|---|---|
| 10950 | | dummy |
| 10951 | | dummy |
| 10952 | | dummy |
| 10953 | | dummy |
| 10954 | | dummy |
| 10955 | | dummy |
| 10956 | | dummy |
| 10957 | | dummy |
| 10958 | | dummy |
| 10959 | | dummy |
| 10960 | | dummy |
| 10961 | Berlin | Kreuzberg |
| 10962 | | dummy |
| 10963 | | dummy |
| 10964 | | dummy |
| 10965 | | dummy |
| 10966 | | dummy |
| 10967 | Berlin | Kreuzberg |

**around 20,000 records**

```
Create Table ZZSTADTTEIL
( PLZ          CHAR(5),
  ORT          CHAR(25),
STADTTEIL      CHAR(40),
  PRIMARY KEY
  (PLZ) )
```

```
Create Table ZZMASTER
( YEAR         INT,
  NAME         CHAR(40),
  VORNAME      CHAR(20),
  UNI          CHAR (40),
  ORT          CHAR (25),
PRIMARY KEY
  (YEAR,NAME,VORNAME) )
```

| YEAR | NAME | VORNAME | UNI |
|---|---|---|---|
| 1988 | Teo | Hoe Sing | LMU München |
| 1999 | Doin | Xenia | FU Berlin |
| 1999 | Toscani | Marybeth | LMU München |
| 2000 | Aimonsri | Kanokporn | FH Ludwigshafen |
| 2000 | Hofmann | Martin | HU Berlin |
| 2000 | Lueck | Christina | LMU München |
| 2000 | MORONI | STEFANO | TU Berlin |
| 2000 | Reijer | Lars | TU Berlin |
| 2002 | Diekmann | Burkhard | FU Berlin |
| 2008 | Tilborg | Machiel | HPI Potsdam |

**10 records**



To explain strategies which can be used for subqueries, 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.

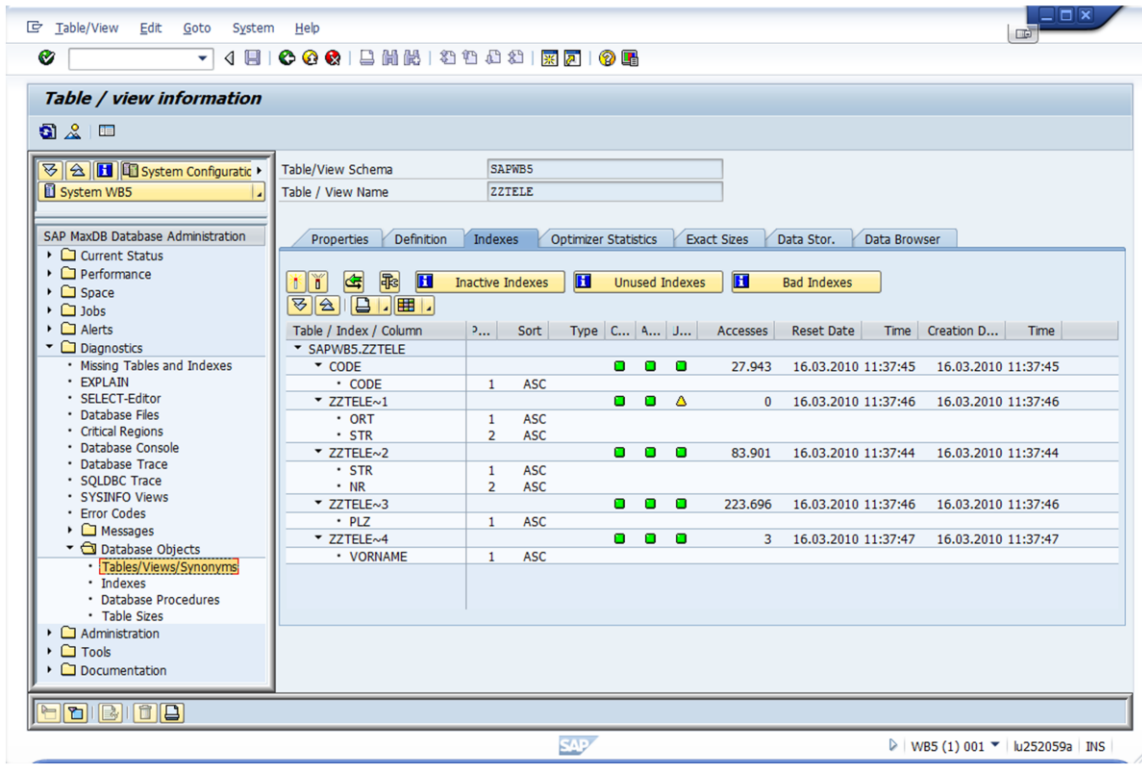
# Table Examples: ZZCODE

| STR | NR | PLZ | ORT | CODE |
|---|---|---|---|---|
| Dummy | 91 | 10090 | Berlin | |
| Dummy1 | 92 | 10091 | Berlin | Television |
| Dummy2 | 93 | 10092 | Berlin | Cell phone tower |
| Dummy3 | 94 | 10093 | Berlin | |
| Dummy4 | 95 | 10094 | Berlin | Cable TV |
| Wexstr | 96 | 10095 | Berlin | Cable TV |
| Dummy | 97 | 10096 | Berlin | |
| Dummy1 | 98 | 10097 | Berlin | Television |
| Dummy2 | 99 | 10098 | Berlin | Cell phone tower |
| Dummy3 | 100 | 10099 | Berlin | |
| Dummy4 | 101 | 10100 | Berlin | Television |
| Wexstr | 102 | 10101 | Berlin | Cable TV |
| Dummy | 103 | 10102 | Berlin | |
| Dummy1 | 104 | 10103 | Berlin | Television |
| Dummy2 | 105 | 10104 | Berlin | |
| Dummy3 | 106 | 10105 | Berlin | |
| Dummy4 | 107 | 10106 | Berlin | Television |
| Wexstr | 108 | 10107 | Berlin | Cable TV |
| Dummy | 109 | 10108 | Berlin | |
| Dummy1 | 110 | 10109 | Berlin | Television |
| Dummy2 | 111 | 10110 | Berlin | |
| Dummy3 | 112 | 10111 | Berlin | |
| Dummy4 | 113 | 10112 | Berlin | Television |
| Wexstr | 114 | 10113 | Berlin | Cable TV |

```
Create Table ZZCODE
( STR        CHAR(40),
  NR         INT,
  PLZ        CHAR(5),
  ORT        CHAR(25),
  CODE       CHAR(31)
)
```

**# of records: around 115,000**

# Indexes created on table ZZTELE






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*

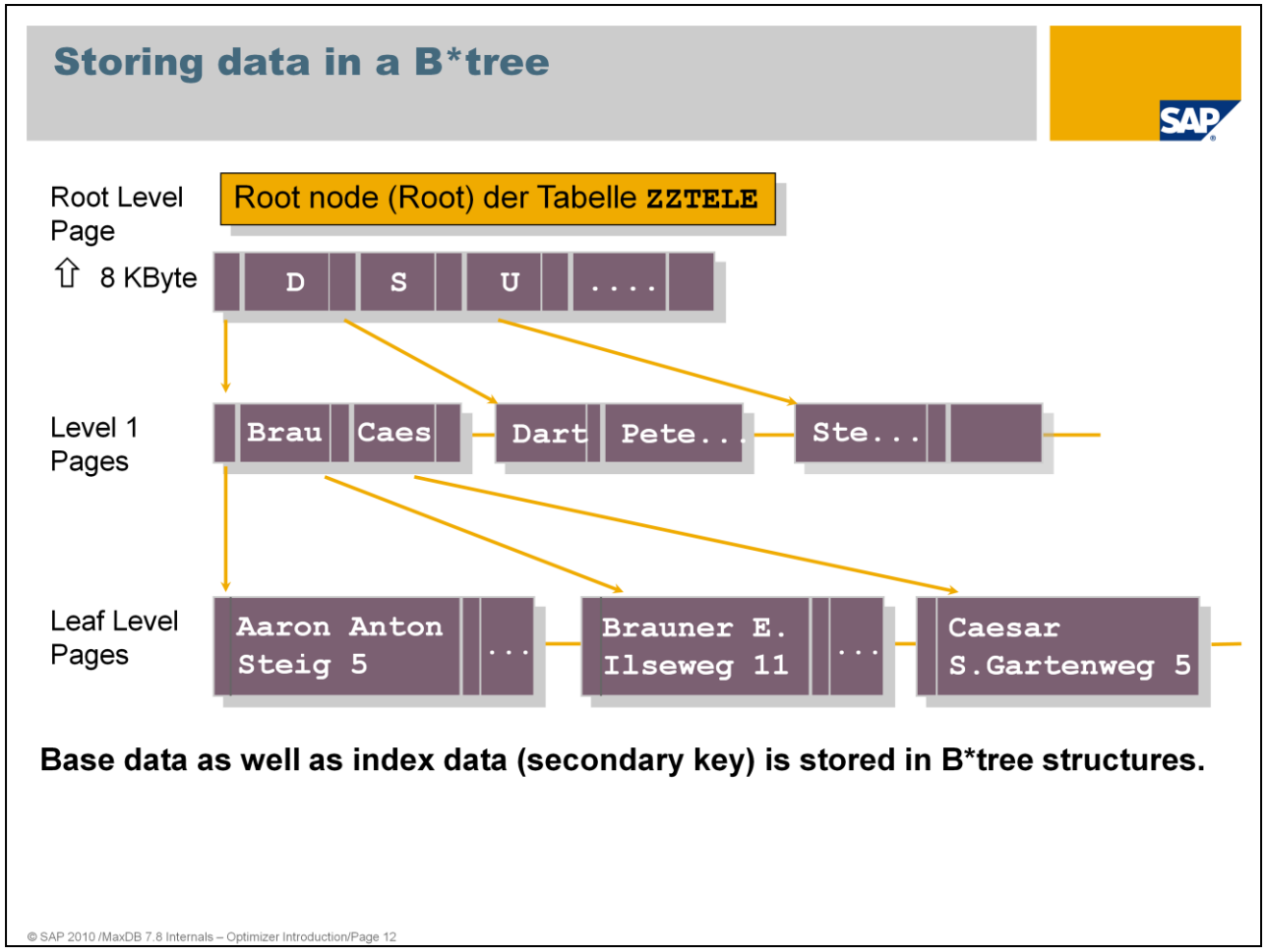
*FROM domain.indexcolumns*

*WHERE owner = <owner>*

*AND schemaname = <schema>*

*AND tablename = <table_name>*

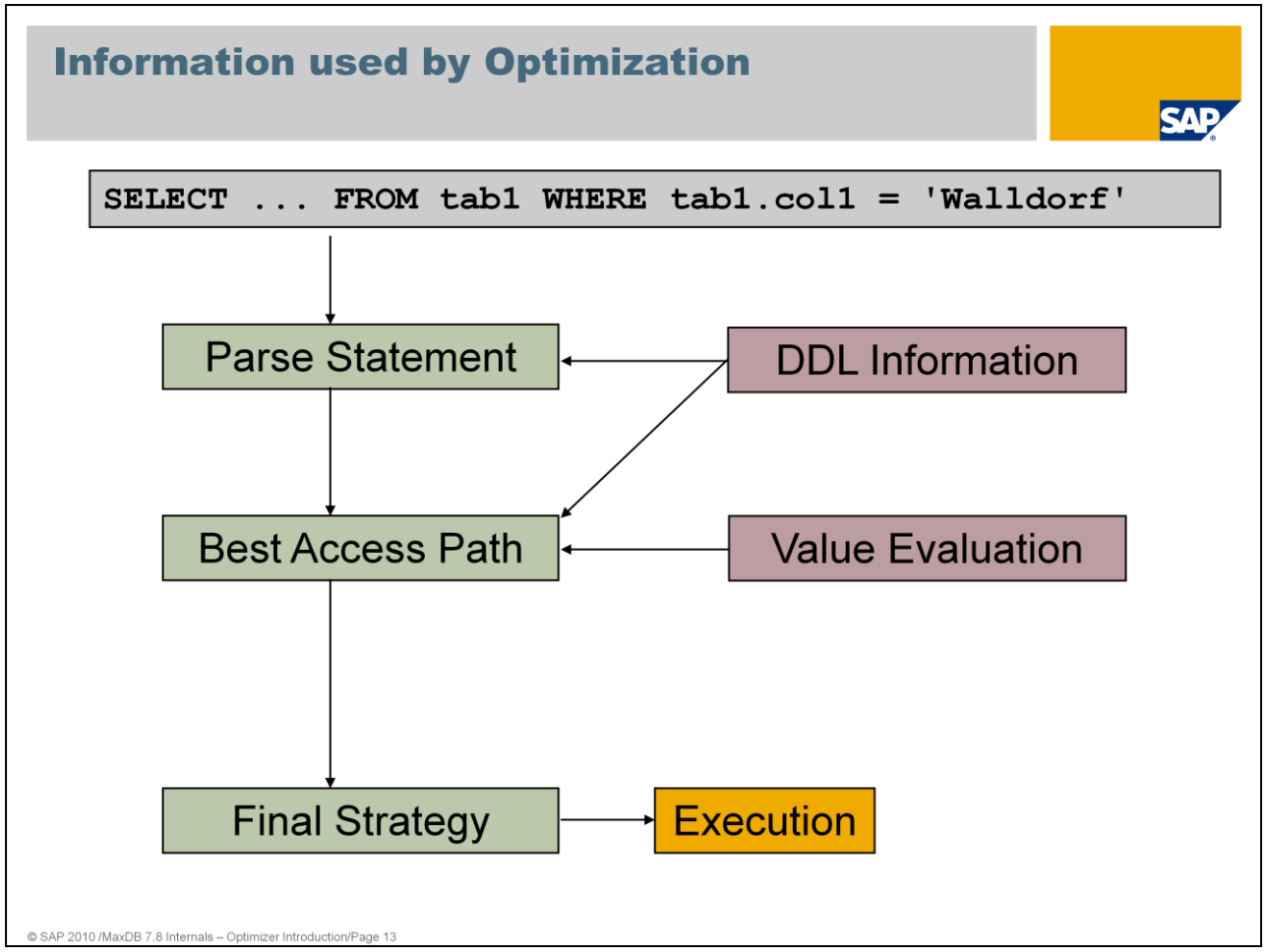
*ORDER BY owner, tablename, indexname, columnno*




The data of the base tables and the indexes are stored in B*Tree format.

Every B*Tree for a table has one root page with a size of 8 KB.

For more information about B* Tree see Expert Session no. 15 SAP MaxDB No Reorganization Principle




First, the SQL parser processes an SQL statement. It performs a syntactic and semantic analysis. In the semantic analysis, tables and their column data are checked.

The optimizer determines which primary and secondary keys are available for the table and checks whether a corresponding key can be used to search for values.

The number of pages that have to be read in the primary or secondary index is determined by generating a start and a stop key. Depending on the number of pages of the table or index, it is decided whether it is worthwhile to search using the index. The number of pages of the entire table is located in the so called filedirectory

At the end, the optimizer builds a strategy with which the SQL statement will be executed.

In the R/3 System environment, SQL statements with bind variables are parsed (:A0, :A1, and so on). These bind variables may contain other specific values. All SQL statements that only vary in values are also regarded as different statements. The same SQL statement executed with different values can therefore also have different run schedules. These commands are structurally the same and are listed individually in the Command Monitor with different values.

## Explain (1)

Input : EXPLAIN <SELECT Command>

Output : Description of search strategy

- EXPLAIN is used with SELECT commands that access tables and views
- EXPLAIN does not execute the specified SELECT command.
- The explain command cannot be used with UPDATE, DELETE or INSERT commands



An explain 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 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. Other SQL statements must be rewritten to display an explain plan. For example, an UPDATE statement can be converted into a SELECT FOR REUSE. Example:

```
UPDATE ZZTELE                    SELECT * FROM ZZTELE
SET ADDINFO = 'ledig'            WHERE NAME = 'Mueller'
WHERE NAME = 'Mueller'             AND     VORNAME = ' Egon'
AND     VORNAME = ' Egon‘          AND     STR = ' Wexstraße'
AND     STR = ' Wexstraße‘         FOR REUSE
```

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.

## Explain (2)

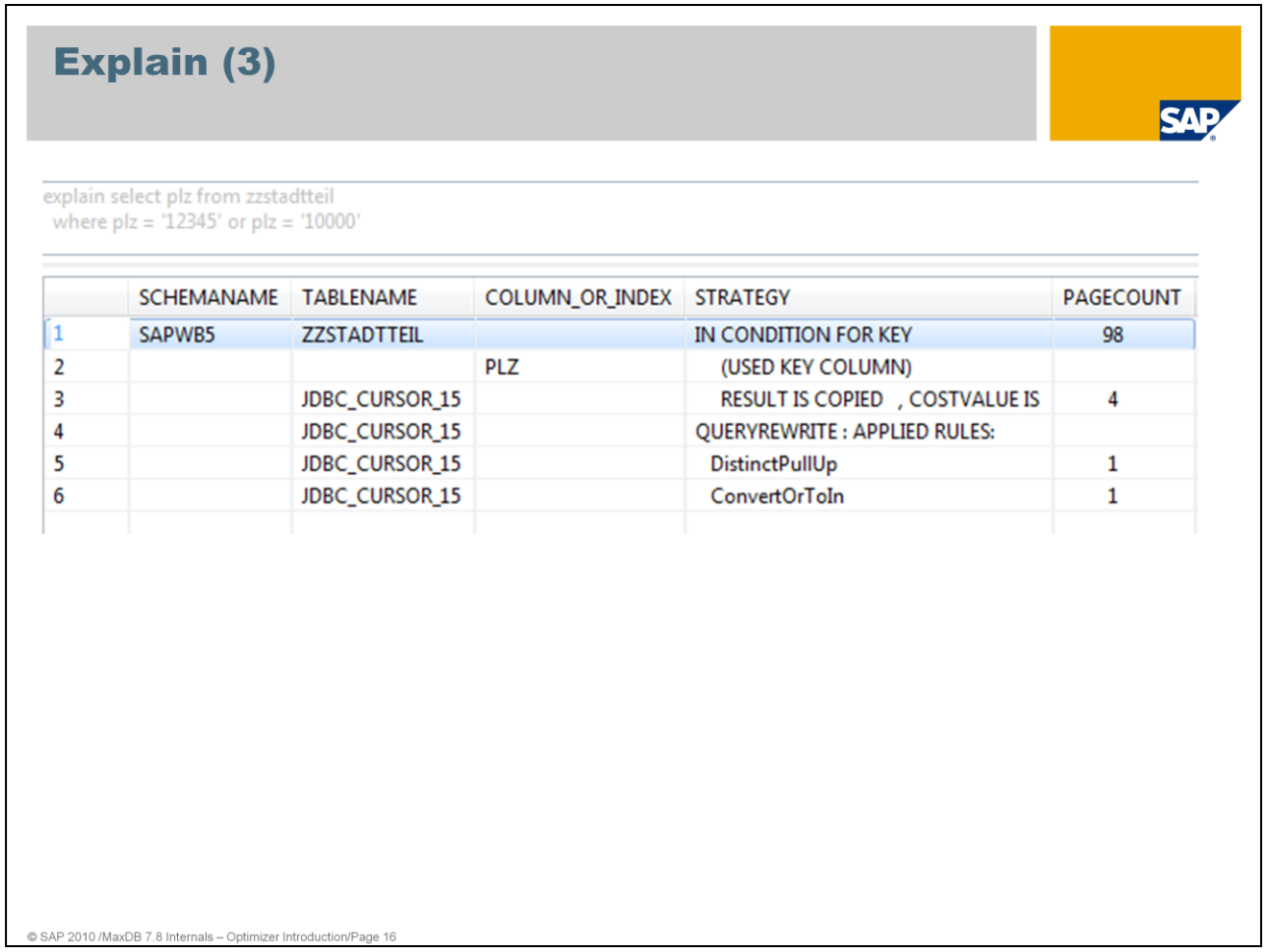
| SCHEMANAME | TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|---|
| Schema | Table 1 | Names of key or index columns | Name of chosen strategy for this table | Number of pages In system table Optimizerstatistics |
| Schema | Table 2 | Names of key or index columns | Name of chosen strategy for this table | Number of pages in system table Optimizerstatistics |
| | Result name | | RESULT IS (NOT) COPIED, COSTVALUE IS | Estimated costs |
| | | | Applied Query Rewrite rules | 1 |



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

## Explain (3)



```
explain select plz from zzstadtteil
 where plz = '12345' or plz = '10000'
```

| | SCHEMANAME | TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|---|---|
| 1 | SAPWB5 | ZZSTADTTEIL | | IN CONDITION FOR KEY | 98 |
| 2 | | | PLZ | (USED KEY COLUMN) | |
| 3 | | JDBC_CURSOR_15 | | RESULT IS COPIED , COSTVALUE IS | 4 |
| 4 | | JDBC_CURSOR_15 | | QUERYREWRITE : APPLIED RULES: | |
| 5 | | JDBC_CURSOR_15 | | DistinctPullUp | 1 |
| 6 | | JDBC_CURSOR_15 | | ConvertOrToIn | 1 |



Here is an example for an explain plan on a single table access. The Optimizer is using the strategy *In Condition for key* on column *plz.* A temporary result is created – *Result is copied.* The estimated costs have a value of 4.

*Select plz from zzstadtteil where plz = ‘12345’ or plz = ‘10000’*

Furthermore the explain tells us the usage of Queryrewrite rules.

**DistinctpullUp** does not have any effects on the execution, but is required for other rules to work.

**ConvertOrToIn** rewrites OR predicates as IN statements.

## Query Rewrite

Query Rewrite rebuilds SQL statements by the use of rules to enable the optimizer to find the best strategy.

Example: ConvertOrToIn

*select * from zztele*
*where PLZ = '10967' or PLZ = '15099' or PLZ = '12047'*

➔
```
SELECT "NAME", "VORNAME", "STR", "NR",
       "PLZ", "ORT", "CODE", "ADDINFO"
FROM "SAPR3"."ZZTELE" AS "_T1"
WHERE PLZ in ('10967', '15099', '12047')
```

Parameter: EnableQueryRewrite=YES, QueryAnalysisMode=EXTENDED



Query Rewrite investigates the statement after the syntactical analysis.

Query Rewrite does a semantical analysis and rebuilds the statement if rules can be applied. Several rules can be applied to one query.

The rearranged statement with the possible execution plans is stored in internal format within Shared SQL or the catalog cache, respectively. The optimizer determines the best execution plan for the rearranged statement.

Query Rewrite works rule-based. Statistical data is not taken into account. There is no evaluation of data.

For more information check FAQ note 1368477 about QueryRewrite.

## Query Rewrite Rules

Changing the column ACTIVE in the view QUERYREWRITERULES effects if a rule is switched on or off.

SQL SQL | Result (1)

```
select * from queryrewriterules
```

| | RULENAME | ACTIVE | COMMENT |
|---|---|---|---|
| 1 | AddLocalPredicates | YES | Add local predicates |
| 2 | CombineExternalSelects | YES | Combine external selects |
| 3 | CombineToAnyOrAll | YES | Combine ORed/ANDed predicates to ANY/ALL predicate |
| 4 | ConvertExceptToAntiSemiJoin | YES | Convert Except to anti-semi-join |
| 5 | ConvertIntersectToSemiJoin | YES | Convert Intersect to semi-join |
| 6 | ConvertUnionToORQuery | YES | Convert Union to OR-query |
| 7 | DistinctPullUp | YES | Pull up distinct information |
| 8 | DistinctPushDown | YES | Push down distinct information |
| 9 | FlattenSubqueries | YES | Flatten subqueries |
| 10 | MergeQueries | YES | Merge queries |
| 11 | NormalizePredicates | YES | Normalize predicates |
| 12 | OptimizeAggregates | YES | Optimize aggregates |
| 13 | OptimizeExpressions | YES | Optimize expressions |
| 14 | OptimizeJoins | YES | Optimize joins |
| 15 | OptimizePredicates | YES | Optimize predicates |
| 16 | OptimizeSubqueries | YES | Optimize subqueries |
| 17 | PushDownJoins | YES | Push down joins |
| 18 | PushDownPredicates | YES | Push down predicates |
| 19 | PushDownProjection | YES | Push down projection |
| 20 | RemoveDispensableConstants | YES | Remove dispensable constants |
| 21 | RemoveDispensableGroupBy | YES | Remove dispensable GroupBy-columns |
| 22 | RemoveDispensableOrderBy | YES | Remove dispensable OrderBy-columns |
| 23 | ReorderJoins | YES | Reorder joins |
| 24 | ReorderPredicates | YES | Reorder predicates |
| 25 | ReorderUnions | YES | Reorder unions |
| 26 | SubstituteBushyJoins | YES | Substitute right side of a bushy join by a from-select |
| 27 | SubstituteViews | YES | Substitute complex views and join views by the corresponding from-select |



You can influence the use of Query Rewrite by setting the parameter EnableQueryRewrite.

Furthermore you have the possibility to switch single rules on or off. Use an UPDATE statement on table QUERYREWRITERULES to set the attribute ACTIVE for the corresponding rule to YES or NO.

```
UPDATE queryrewriterules
   SET ACTIVE = 'YES' WHERE RULENAME = 'AddLocalPredicates'
```

To activate the rule changes to all applications execute: diagnose share parse reject

# Monitoring Query Rewrite

EXPLAIN QUERYREWRITE shows the result of Query Rewrite as SQL statement.

```
explain queryrewrite
select distinct *
from zztele
```

| | STATEMENT |
|---|---|
| 1 | SELECT "NAME", "VORNAME", "STR", "NR", "PLZ", "ORT", "CODE", "ADDINFO" FROM "SAPA15"."ZZTELE" AS "_T1" |

The view MONITOR indicates how often rules were applied.

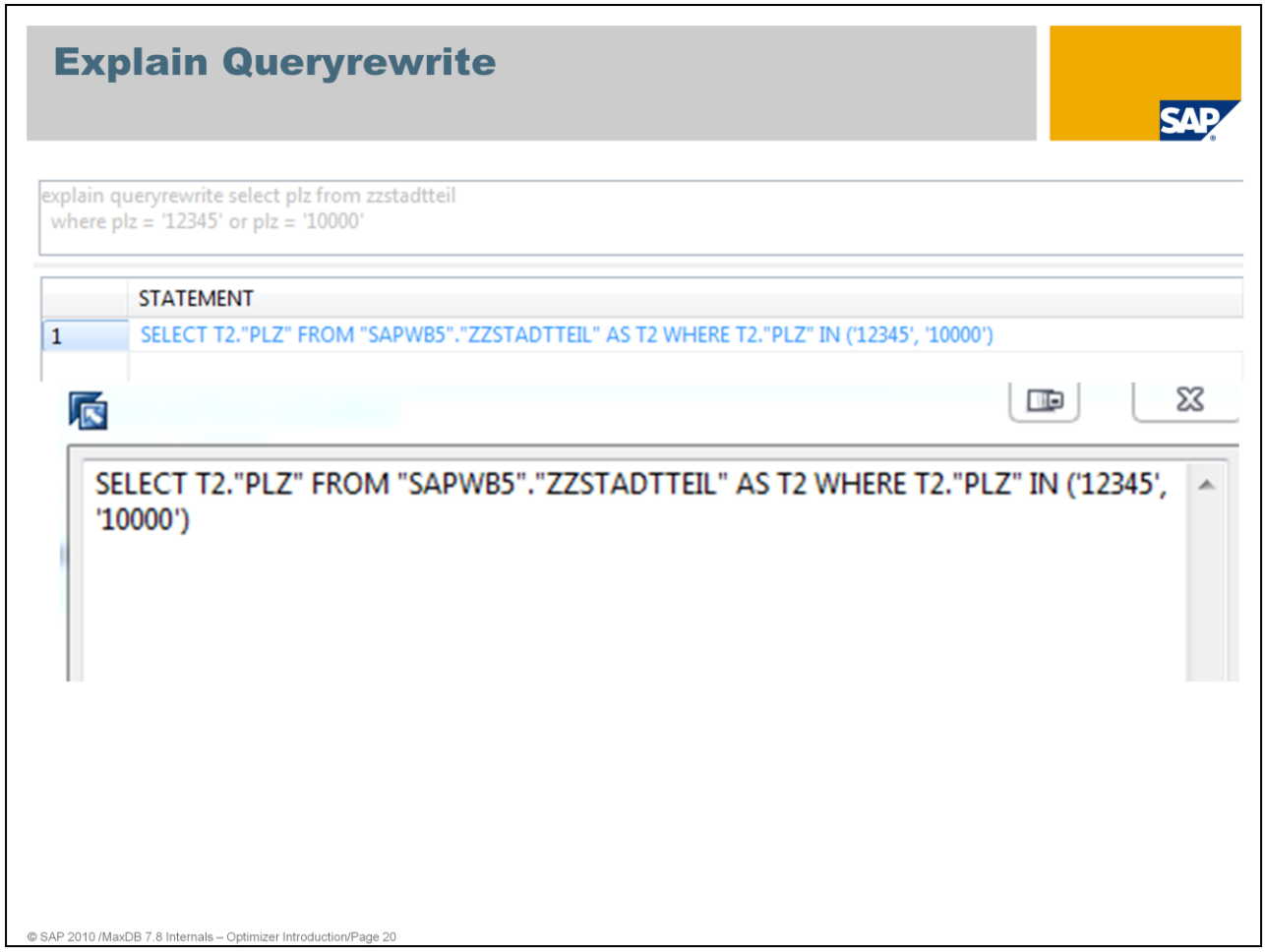
```
select * from monitor
where type = 'REWRITE'
```

| | TYPE | DESCRIPTION | VALUE |
|---|---|---|---|
| 1 | REWRITE | SubstituteViews | 1960631 |
| 2 | REWRITE | SubstituteBushyJoins | 0 |
| 3 | REWRITE | DistinctPullUp | 1767894 |
| 4 | REWRITE | DistinctPushDown | 114373 |
| 5 | REWRITE | OptimizeSubqueries | 0 |
| 6 | REWRITE | OptimizePredicates | 8543 |
| 7 | REWRITE | OptimizeExpressions | 0 |
| 8 | REWRITE | OptimizeJoins | 8 |
| 9 | REWRITE | OptimizeAggregates | 0 |
| 10 | REWRITE | ReorderPredicates | 0 |
| 11 | REWRITE | ReorderJoins | 43 |
| 12 | REWRITE | ReorderUnions | 0 |
| 13 | REWRITE | PushDownPredicates | 1828128 |
| 14 | REWRITE | PushDownProjection | 1813928 |
| 15 | REWRITE | PushDownJoins | 0 |
| 16 | REWRITE | FlattenSubqueries | 1360 |
| 17 | REWRITE | MergeQueries | 1959877 |
| 18 | REWRITE | RemoveDispensableConstants | 0 |



EXPLAIN QUERYREWRITE displays the SQL statement after it has been converted by QueryRewrite.

SELECT * FROM MONITOR
WHERE Type = ‚REWRITE‘




Use the statement EXPLAIN QUERYREWRITE <select> to display the result of the rewrite. In SAP MaxDB versions below 7.8, this output is restricted to 2500 characters. This means that truncated rewrites may occur.

Note that the result of EXPLAIN QUERYREWRITE is not an SQL statement that can necessarily be executed using the Database Studio. You can use the right mouse to expand the complete SQL statement.

Further information about QueryRewrite: SAP note: 1368477 FAQ: SAP MaxDB QueryRewrite

## Search Condition and Search Strategies (1)

```
SELECT VORNAME,NAME,PLZ,ORT,STR,ADDINFO
  FROM ZZTELE
 WHERE ADDINFO LIKE 'Training%'
```

| VORNAME | NAME | PLZ | ORT | STR | ADDINFO |
|---|---|---|---|---|---|
| MICHELA | ANDREONI | 10108 | Berlin | Dummy | Training Administration |
| MICHELA | ANDREONI | 10109 | Berlin | Dummy1 | Training Administration |
| MICHELA | ANDREONI | 10110 | Berlin | Dummy2 | Training Administration |
| MICHELA | ANDREONI | 10111 | Berlin | Dummy3 | Training Administration |
| MICHELA | ANDREONI | 10112 | Berlin | Dummy4 | Training Administration |
| MICHELA | ANDREONI | 10113 | Berlin | Wexstr | Training Administration |
| Svein | Aasen | 10186 | Berlin | Dummy | Training - NEW EMP |
| Svein | Aasen | 10187 | Berlin | Dummy1 | Training - NEW EMP |
| Svein | Aasen | 10188 | Berlin | Dummy2 | Training - NEW EMP |
| Svein | Aasen | 10189 | Berlin | Dummy3 | Training - NEW EMP |
| Svein | Aasen | 10190 | Berlin | Dummy4 | Training - NEW EMP |
| Svein | Aasen | 10191 | Berlin | Wexstr | Training - NEW EMP |
| Keiichiro | Abe | 10258 | Berlin | Dummy | Training Mgmt & Admin |
| Keiichiro | Abe | 10259 | Berlin | Dummy1 | Training Mgmt & Admin |
| Keiichiro | Abe | 10260 | Berlin | Dummy2 | Training Mgmt & Admin |

Search Strategy

| STRATEGY | PAGECOUNT |
|---|---|
| TABLE SCAN | 3212 |
| RESULT IS NOT COPIED , COSTVALUE IS | 3212 |
| QUERYREWRITE : APPLIED RULES: | |
| DistinctPullUp | 1 |

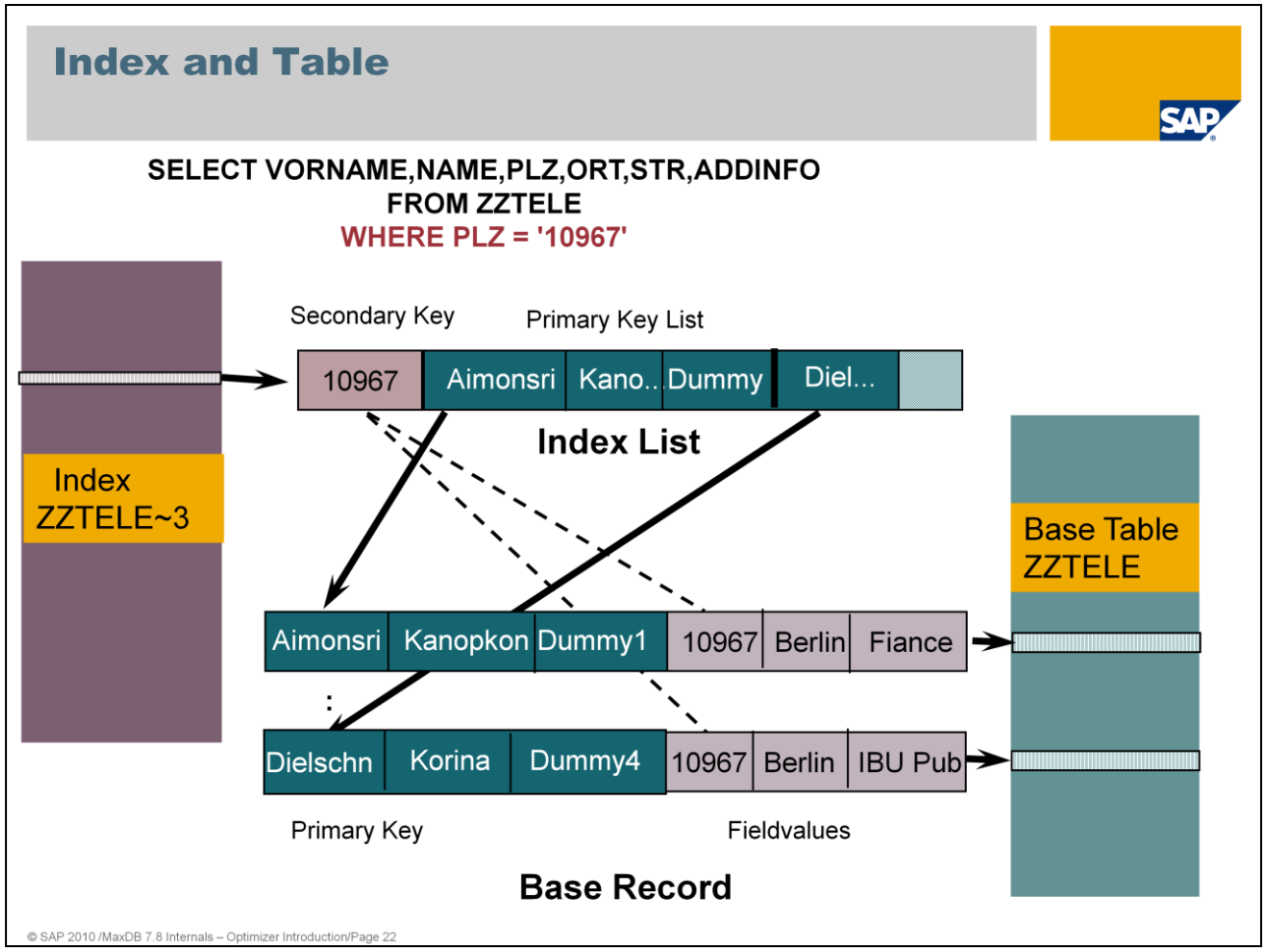


**Search conditions** are specified in the WHERE part of an SQL statement. The WHERE part is used by the optimizer to find the best **search strategy** to deliver the result.

Note: The **order** of the column specification in the SELECT list only influences the optimizer search strategy for DISTINCT statements.

Without an explicit order by option in the WHERE condition the result is sorted by the primary key *Name,Vorname,Str*.

Exception: When an Index only strategy is used the result is sorted by the index order. Always ORDER BY should be used if a special sort of the result is requested.




An index contains the data of the secondary key as well as the respective primary key. Using the primary key, the data can be found in the base table. For each index, a B* tree is created, which is sorted according to the values of the secondary key.

There is no record ID or anything similar. The unique ID of a record is the primary key.

If no primary key was specified with the table creation, the database generates the internal field SYSKEY of the type CHAR(8) BYTE. This field is filled with unique values.

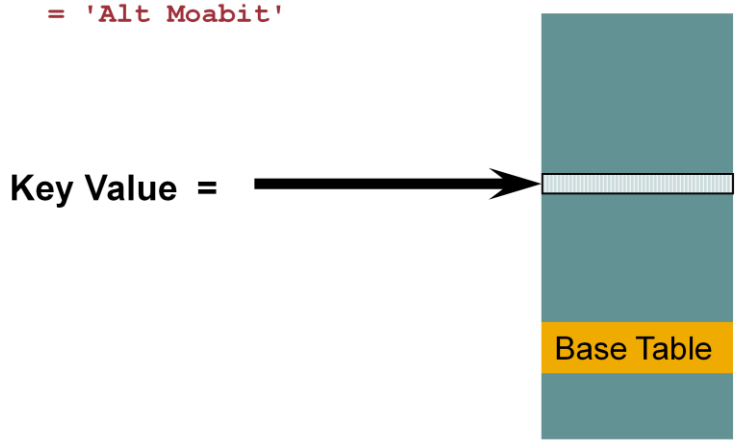
Searching via an index is relatively costly.

On the following slides you will find examples of search strategies. The list of strategies is not complete. A complete list of search strategies can be found in the documentation.

Basic Information -> Background Knowledge -> SQL Optimizer -> Search Strategy -> List of all search strategies

# EQUAL CONDITION FOR KEY

```
SELECT * FROM zztele
  WHERE Name     = 'Aaron'
  AND   Vorname  = 'Anton'
  AND   Str      = 'Alt Moabit'
```






Remember the key definition of table ZZTELE is *Name, Vorname, Str*

EQUAL CONDITION FOR KEY provides an efficient access path through "direct access" to the base table.

The optimizer takes decision for this strategy already at the time of the parsing because, independent of the data in the search conditions, no better search strategy is possible.

# EQUAL CONDITION FOR KEY - Example

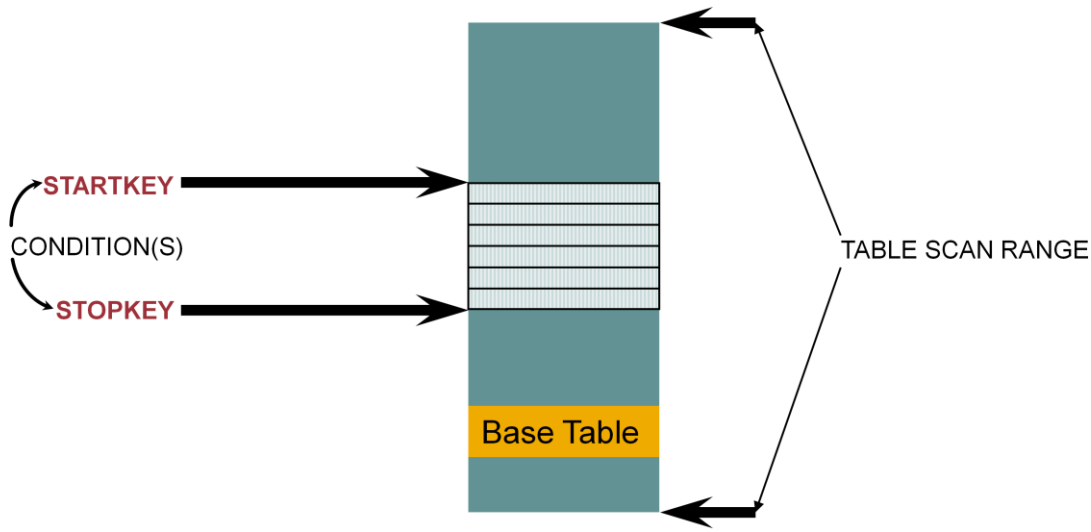
```
SELECT * FROM zztele
  WHERE Name     = 'Aaron'
  AND   Vorname  = 'Anton'
  AND   Str      = 'Alt Moabit'
```

| SCHEMANAME | TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|---|
| SAPWB5 | ZZTELE | | EQUAL CONDITION FOR KEY | 3200 |
| | | NAME | (USED KEY COLUMN) | |
| | | VORNAME | (USED KEY COLUMN) | |
| | | STR | (USED KEY COLUMN) | |
| | JDBC_CURSOR_44 | | RESULT IS NOT COPIED , COSTVALUE IS | 1 |
| | JDBC_CURSOR_44 | | QUERYREWRITE : APPLIED RULES: | |
| | JDBC_CURSOR_44 | | DistinctPullUp | 1 |

| NAME | VORNAME | STR | NR | PLZ | ORT | CODE | ADDINFO |
|---|---|---|---|---|---|---|---|
| Aaron | Anton | Alt Moabit | 96 | 10559 | Berlin | X | Testperson |

# RANGE CONDITION FOR KEY

```
SELECT * FROM zztele WHERE Name = 'Schmidt'
                     AND   Vorname like 'A%'
SELECT * FROM zztele
```






If a portion of the start of the primary key is specified in the WHERE condition, the strategy RANGE CONDITION FOR KEY will be executed.

A special case of key range is the table scan. The start key is located at the beginning of the table and the stop key at the end of the table.

The base table will be searched completely (TABLE SCAN).

An intermediate result set is not generated.

# RANGE CONDITION FOR KEY - Example

```
SELECT * FROM zztele WHERE Name = 'Schmidt'
                     AND   Vorname like 'A%'
```

| SCHEMANAME | TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|---|
| SAPWB5 | ZZTELE | | RANGE CONDITION FOR KEY | 3212 |
| | | NAME | (USED KEY COLUMN) | |
| | | VORNAME | (USED KEY COLUMN) | |
| | JDBC_CURSOR_28 | | RESULT IS NOT COPIED , COSTVALUE IS | 1 |
| | JDBC_CURSOR_28 | | QUERYREWRITE : APPLIED RULES: | |
| | JDBC_CURSOR_28 | | DistinctPullUp | 1 |

```
SELECT * FROM zztele
```

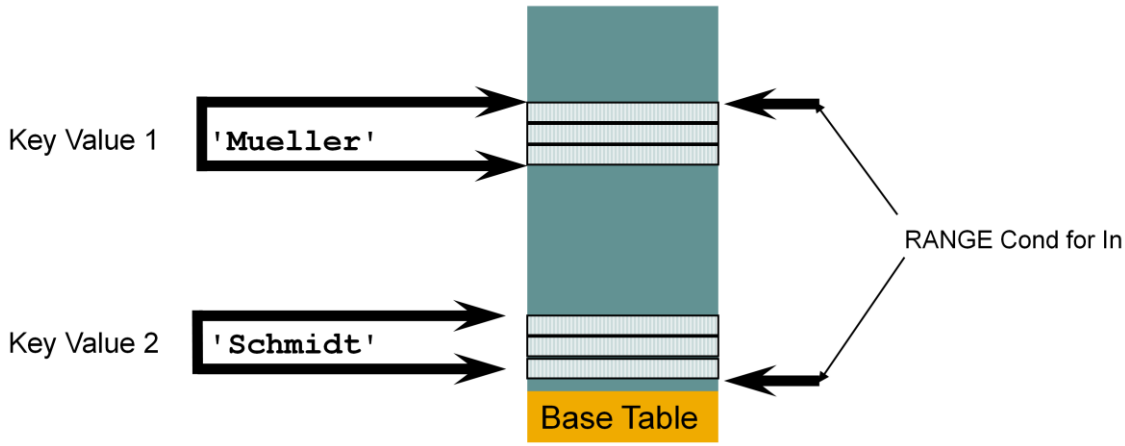
| SCHEMANAME | TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|---|
| SAPWB5 | ZZTELE | | TABLE SCAN | 3212 |
| | JDBC_CURSOR_29 | | RESULT IS NOT COPIED , COSTVALUE IS | 3212 |
| | JDBC_CURSOR_29 | | QUERYREWRITE : APPLIED RULES: | |
| | JDBC_CURSOR_29 | | DistinctPullUp | 1 |



primary key order

# IN CONDITION FOR KEY

```
SELECT * FROM zztele
WHERE Name IN ('Mueller', 'Schmidt')
```






The IN condition can be placed on each field of a primary key.

Only one IN condition is taken into account.

The primary key fields that precede the field with the IN condition may only be specified in an EQUAL condition.

An intermediate result set is generated. The result set is sorted according to the primary key.

As of version 7.4, the optimizer checks whether the RANGE CONDITION FOR KEY is advantageous. This happens if the values in the IN condition are close to each other. Example:

```
SELECT *
 FROM zztele
 WHERE name IN  ( 'Scheu' , 'Schmidt')
```

There are additional names in the table that are located between the values 'Scheu' and 'Schmidt'. Thus, using this search condition, records are also included that do not belong to the results set. However, the strategy is more favorable since only one start and stop key have to be determined.

# IN CONDITION FOR KEY - Example

```
SELECT * FROM zztele
WHERE Name IN ('Mueller', 'Schmidt')
```

| SCHEMANAME | TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|---|
| SAPWB5 | ZZTELE | | IN CONDITION FOR KEY | 3212 |
| | | NAME | (USED KEY COLUMN) | |
| | JDBC_CURSOR_45 | | RESULT IS COPIED , COSTVALUE IS | 69 |
| | JDBC_CURSOR_45 | | QUERYREWRITE : APPLIED RULES: | |
| | JDBC_CURSOR_45 | | DistinctPullUp | 1 |

```
SELECT * FROM zztele
WHERE Name IN ('Scheu', 'Schmidt')
```

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZTELE | | RANGE CONDITION FOR KEY | 3212 |
| | NAME | (USED KEY COLUMN) | |
| JDBC_CURSOR_84 | | RESULT IS NOT COPIED , COSTVALUE IS | 32 |
| JDBC_CURSOR_84 | | QUERYREWRITE : APPLIED RULES: | |
| JDBC_CURSOR_84 | | DistinctPullUp | 1 |



An intermediate result is created for IN CONDITIONs.

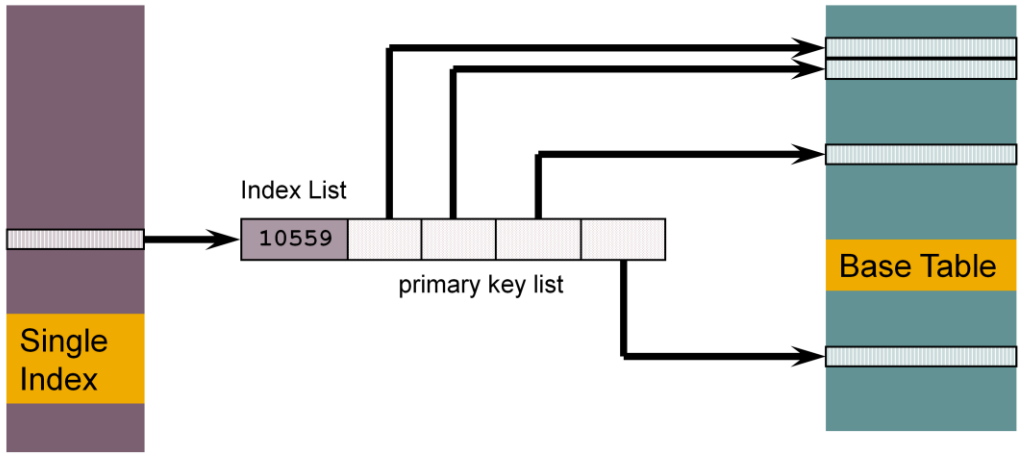
No intermediate result is created for RANGE CONDITIONs.

Primary key order

# EQUAL CONDITION FOR INDEX

```
CREATE INDEX "ZZTELE~3" ON ZZTELE(PLZ)

SELECT * FROM zztele
  WHERE plz = '12345'
```






When determining the strategy, additional costs (index_overhead) for accessing the base data via the index are taken into account.

The optimizer also selects the strategy EQUAL CONDITION FOR INDEX, if all fields of a multiple index in the WHERE condition are specified with an equal condition.

An intermediate result set is not generated.

# EQUAL CONDITION FOR INDEX - Example

```
CREATE INDEX "ZZTELE~3" ON ZZTELE(PLZ)


SELECT * FROM zztele
  WHERE plz = '12345'
```

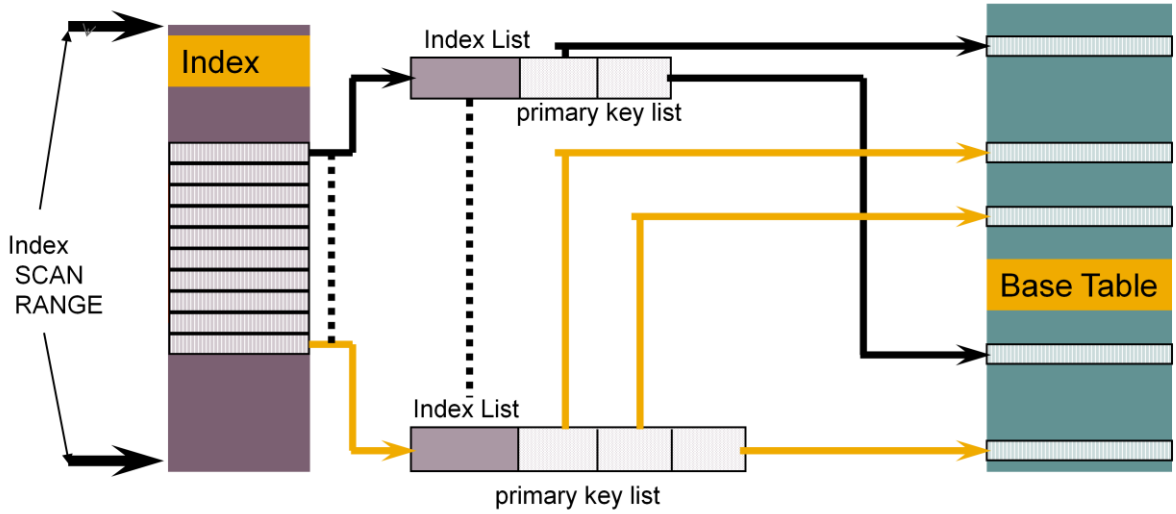
| SCHEMANAME | TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|---|
| SAPWB5 | ZZTELE | ZZTELE~3 | EQUAL CONDITION FOR INDEX | 2105 |
| | | PLZ | (USED INDEX COLUMN) | |
| | JDBC_CURSOR_55 | | RESULT IS NOT COPIED , COSTVALUE IS | 2 |
| | JDBC_CURSOR_55 | | QUERYREWRITE : APPLIED RULES: | |
| | JDBC_CURSOR_55 | | DistinctPullUp | 1 |

| NAME | VORNAME | STR | NR | PLZ | ORT | CODE | ADDINFO |
|---|---|---|---|---|---|---|---|
| Antao | Sanjiv | Wexstr | 51 | 12345 | | | Consulting-Bombay |
| Dry | Marie | Dummy2 | 162 | 12345 | | | Central Registration |
| Huebel | Ralf | Wexstr | 18 | 12345 | | | FI Dev. International |
| Marek | Peter | Dummy2 | 129 | 12345 | | | RIVA Entwicklung R/2 |
| Ringling | Sven | Wexstr | 240 | 12345 | | | Test R/3-HR |
| Tsuchiyama | Takayoshi | Dummy2 | 96 | 12345 | | | Log - 2 |

## RANGE CONDITION FOR INDEX

```
CREATE INDEX "ZZTELE~3" ON ZZTELE(PLZ)

SELECT * FROM zztele
  WHERE plz BETWEEN '10100' AND '10967'
```






If a part of the start of the secondary key is specified in the WHERE condition, the strategy RANGE CONDITION FOR INDEX will be executed.

An intermediate result set is not generated. The result is sorted by the secondary key.

The index scan is a special index range with start key at the beginning of the index and stop key and the end of the index.
The index scan is only used for ORDER BY.

During an INDEX SCAN, all entries are read via the index in the order of the secondary key. An intermediate result set is not generated.

As of version 7.4, NULL values are also included in single indexes.  Thus, this strategy can be used on all indexes.

## RANGE CONDITION FOR INDEX - Example

```
CREATE INDEX "ZZTELE~3" ON ZZTELE(PLZ)

SELECT * FROM zztele
WHERE plz BETWEEN '10100' AND '10967'
```

| SCHEMANAME | TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|---|
| SAPWB5 | ZZTELE | ZZTELE~3 | RANGE CONDITION FOR INDEX | 2105 |
| | | PLZ | (USED INDEX COLUMN) | |
| | JDBC_CURSOR_72 | | RESULT IS NOT COPIED , COSTVALUE IS | 245 |
| | JDBC_CURSOR_72 | | QUERYREWRITE : APPLIED RULES: | |
| | JDBC_CURSOR_72 | | DistinctPullUp | 1 |

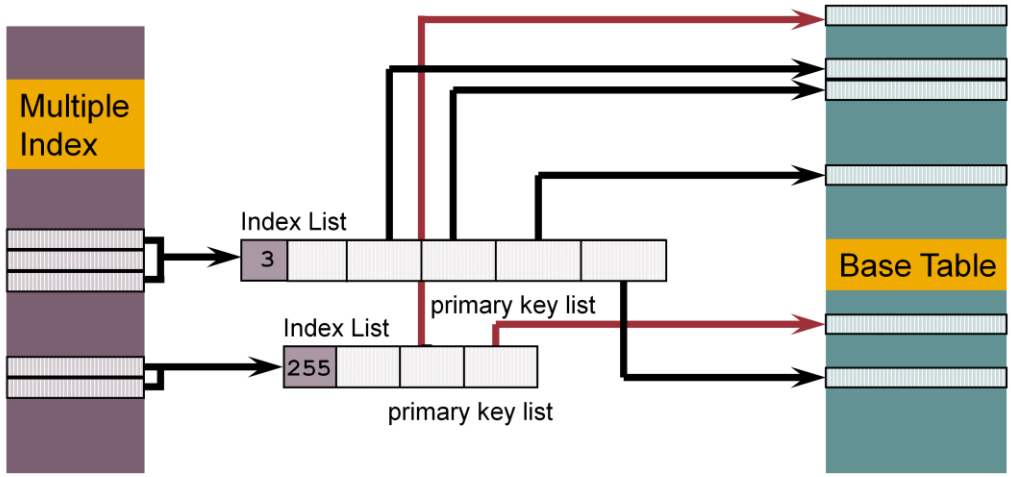
| NAME | VORNAME | STR | NR | PLZ | ORT | CODE | ADDINFO |
|---|---|---|---|---|---|---|---|
| ALZIATI | DORIANO | Dummy4 | 101 | 10100 | Berlin | | ISU Finance Sector |
| Del Rio | Marcela | Dummy1 | 212 | 10100 | Berlin | | Presales ARG |
| Hirn | Manfred | Dummy4 | 68 | 10100 | Berlin | | Log.Entw. Auftr/Vers/Faktura |
| Luengen | Eric | Dummy1 | 179 | 10100 | Berlin | | Log.Entw. Grunddaten |
| Rasanayagam | Rajiv | Dummy4 | 35 | 10100 | Berlin | | Dallas - Early Watch |
| Teusch | Patricia | Dummy1 | 146 | 10100 | Berlin | | Communications Media - Train |
| ALZIATI | DORIANO | Wexstr | 102 | 10101 | Berlin | | ISU Finance Sector |
| Del Rio | Marcela | Dummy2 | 213 | 10101 | Berlin | | Presales ARG |
| Hirn | Manfred | Wexstr | 69 | 10101 | Berlin | | Log.Entw. Auftr/Vers/Faktura |
| Luengen | Eric | Dummy2 | 180 | 10101 | Berlin | | Log.Entw. Grunddaten |



The index scan is used if an ORDER BY is added to the SQL statement.

# IN CONDITION FOR INDEX

```
CREATE INDEX "ZZTELE~2" ON ZZTELE ( STR, NR )
SELECT * FROM zztele
  WHERE str = 'Wexstr' AND nr IN (3, 255)
```






A secondary key can be taken into account for an IN CONDITION. Only one IN CONDITION is taken into account.

The secondary key fields that precede the field with the IN condition may only be specified in an EQUAL CONDITION.

The result set is sorted according to the secondary key.

An intermediate result set is generated.

# IN CONDITION FOR INDEX - Example

```
CREATE INDEX "ZZTELE~2" ON ZZTELE ( STR, NR )
SELECT * FROM zztele
  WHERE str = 'Wexstr' AND nr IN (3, 255)
```

| SCHEMANAME | TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|---|
| SAPWB5 | ZZTELE | ZZTELE~2 | IN CONDITION FOR INDEX | 2048 |
| | | STR | (USED INDEX COLUMN) | |
| | | NR | (USED INDEX COLUMN) | |
| | JDBC_CURSOR_24 | | RESULT IS COPIED , COSTVALUE IS | 52 |
| | JDBC_CURSOR_24 | | QUERYREWRITE : APPLIED RULES: | |
| | JDBC_CURSOR_24 | | DistinctPullUp | 1 |

| NAME | VORNAME | STR | NR | PLZ | ORT | CODE | ADDINFO |
|---|---|---|---|---|---|---|---|
| Abe | Etsuko | Wexstr | 3 | 10257 | Berlin | | Best Practice Library |
| Agnes | Darci | Wexstr | 3 | 10767 | Berlin | | Pre-Sales-Atlanta |
| Alexander | William | Wexstr | 3 | 11277 | | | Performance & Tuning - ATAC |
| Amidei | Lester | Wexstr | 3 | 11787 | | | O&G Sales |
| Annweiler | Denise | Wexstr | 3 | 12297 | | | Empfang EVZ |
| Armstrong | Patricia | Wexstr | 3 | 12807 | | | HR Operations |
| Axmann | Bernd | Wexstr | 3 | 13317 | | | Vertrieb München |
| BORTOLAN | EGIDIO | Wexstr | 3 | 13827 | | | Training Administration |
| Baethke | David | Wexstr | 3 | 14337 | | | Atlanta - Consulting - Malon |
| Bandula | Diane | Wexstr | 3 | 14847 | | | Sales Health care |
| Bartel-Moufang | Diana | Wexstr | 3 | 15357 | | | Communications Media - Knowl |

## Properties which influence the Optimizer

- Only Index
- Create intermediate result Yes / NO ( e.g. order by)
- Min / Max Optimization
- Distinct



The list above shows the properties which are additionally checked by the optimizer and therefore lower the costs of an SQL statement.

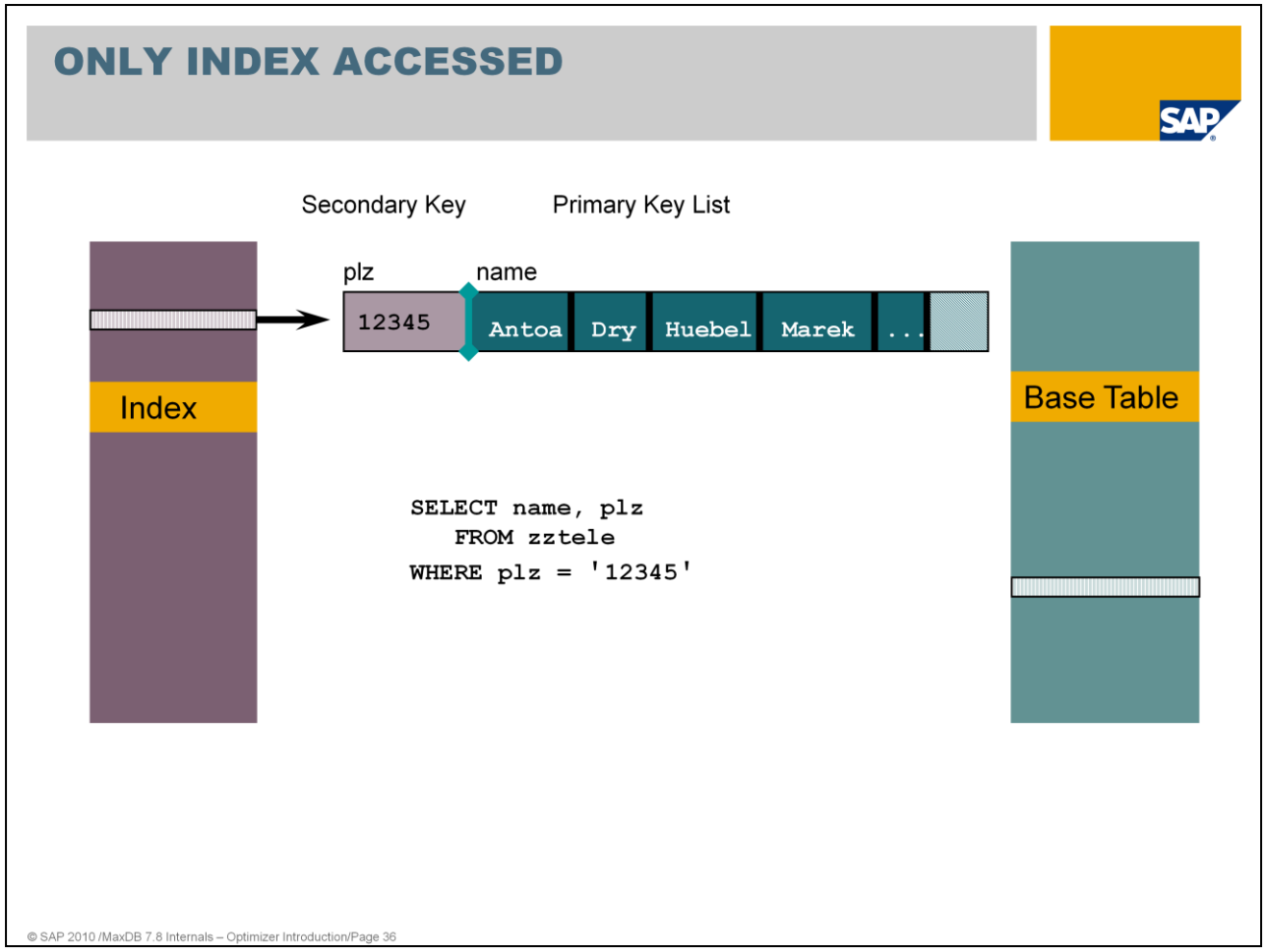
Examples:

*Index only* strategy can be used if a SELECT statement only addresses columns that are also contained in an index (SELECT list and WHERE condition)

An intermediate result is not necessary to create if the ORDER BY is in key order or in order of an index

If the SQL statement specifies a MIN or MAX operation of a key or index column no intermediate result has to be created.

If the SQL statement specifies a DISTINCT on a key or index column the distinct can be verified directly on the key or index.




Remember: in each index the primary key is also part of the data. The primary key is used as separator in the index B* tree.

If a SELECT statement only addresses columns that are also contained in an index (SELECT list, WHERE clause), then only this index will be accessed for the execution of the command.

Advantage:

- In some cases, significantly fewer pages that have to be read
- Optimal usage of sorting of secondary and primary keys in the index
- No additional access to the base table

# ONLY INDEX ACCESSED - Example

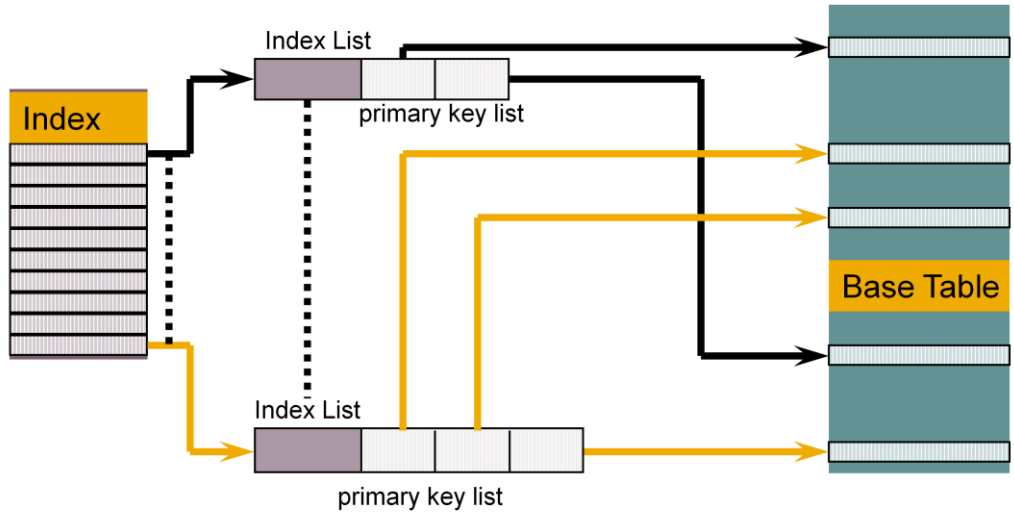
```
SELECT name, plz
   FROM zztele
WHERE plz = '12345'
```

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZTELE | ZZTELE~3 | EQUAL CONDITION FOR INDEX | 2105 |
| | | ONLY INDEX ACCESSED | |
| | PLZ | (USED INDEX COLUMN) | |
| JDBC_CURSOR_108 | | RESULT IS NOT COPIED , COSTVALUE IS | 1 |

| NAME | PLZ |
|---|---|
| Aaron | 10559 |
| Adams | 10559 |
| Dettmann | 10559 |
| Hoehn | 10559 |
| Lyngstad | 10559 |
| Redmond | 10559 |
| Thomas | 10559 |

## ORDER BY Index column(s)

```
CREATE INDEX "ZZTELE~2" ON zztele ( str, nr )
SELECT * FROM zztele WHERE name BETWEEN 'A' and 'T'
   ORDER BY str, nr
```






An ORDER BY specification influences the choice of the Optimizer strategy.

The above SQL statement without the ORDER BY specification would be executed via a key range directly on the table. But with the ORDER BY specification (by secondary key - zztele~2) the strategy will change to an INDEX SCAN.

It is more expensive to read the data via key range and do the sort afterwards then to use an index which is already sorted like the ORDER BY specification and access the rows in the specified order in the primary table.

During the INDEX SCAN, all entries are read via the index in the order of the secondary key. An intermediate result set is not generated.

# ORDER BY Index column(s) - Example

```
SELECT * FROM zztele WHERE name
      BETWEEN 'A' and 'T'
         ORDER BY str, nr
```

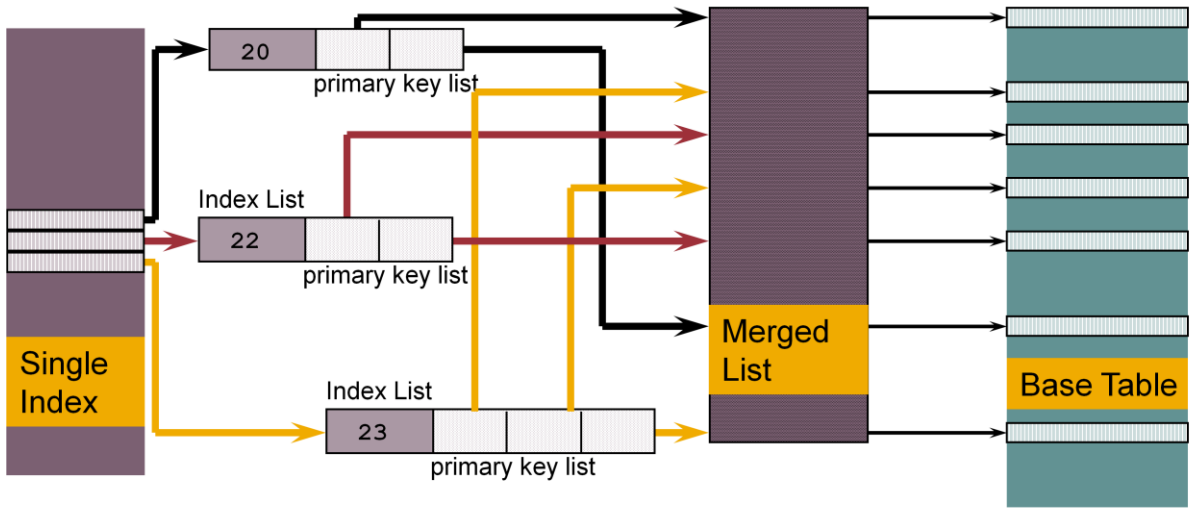
| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZTELE | ZZTELE~2 | INDEX SCAN | 2048 |
| | NAME | (USED KEY COLUMN) | |
| JDBC_CURSOR_40 | | RESULT IS NOT COPIED , COSTVALUE IS | 5260 |
| JDBC_CURSOR_40 | | QUERYREWRITE : APPLIED RULES: | |
| JDBC_CURSOR_40 | | DistinctPullUp | 1 |

| NAME | VORNAME | STR | NR | PLZ | ORT | CODE | ADDINFO |
|---|---|---|---|---|---|---|---|
| Aaron | Anton | Alt Moabit | 96 | 10559 | Berlin | X | Testperson |
| A-J de Groot | Hugo | Dummy | 1 | 10000 | | | Growth Mkt-Sapient/CA |
| Adametz | Hannelore | Dummy | 1 | 10510 | Berlin | | Sekretariat Geschf. |
| Akin | Nursen | Dummy | 1 | 11020 | | | Beratung RW3 |
| Alper | Carol | Dummy | 1 | 11530 | | | Pre-Sales-Boston |
| Andersson | Lisa | Dummy | 1 | 12040 | | | Consulting Basis |
| Aprisnik | Thorsten | Dummy | 1 | 12550 | | | BA-Industrie |
| Asik | Isiltan | Dummy | 1 | 13060 | | | Verrechnungsk. Tuerkei |
| BENBICH | Mohamed | Dummy | 1 | 13570 | | | AUTOMOTIVE |

## ORDER BY key column(s)

```
CREATE INDEX "ZZTELE~2" ON ZZTELE ( STR, NR )

SELECT * FROM zztele
      WHERE str = 'Wexstr' AND nr BETWEEN 20 AND 23
      ORDER BY name, vorname, str
```




**Kernel parameter: IndexlistsMergeThreshold**



The WHERE condition specifies columns of index ZZTELE~2.

The result should be sorted according to the primary key.

Using the additional strategy TEMPORARY INDEX CREATED, the primary keys are sorted in a merge list. The optimum cache usage is guaranteed using access to the base data in the order of the primary keys.

Note: During index merge the index is locked therefore it is useful to define a limit.

The maximum size of the merge lists that are generated can be configured using the parameter IndexlistsMergeThreshold (OPTIM_MAX_MERGE). When the number of index pages involved is less than or equal to IndexlistsMergeThreshold ( Default 500 Pages) the strategy eg. RANGE CONDITION FOR INDEX is used.
As an alternative the strategy which only works on the base table is used.

An intermediate result set is generated (merge list)  but result is not copied.

# ORDER BY key column(s) - Example

```
SELECT * FROM zztele
      WHERE str = 'Wexstr' AND nr
BETWEEN 20 AND 23
      ORDER BY name, vorname, str
```

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZTELE | ZZTELE~2 | RANGE CONDITION FOR INDEX | 2048 |
| | | TEMPORARY INDEX CREATED | |
| | STR | (USED INDEX COLUMN) | |
| | NR | (USED INDEX COLUMN) | |
| JDBC_CURSOR_76 | | RESULT IS NOT COPIED , COSTVALUE IS | 22 |
| JDBC_CURSOR_76 | | QUERYREWRITE : APPLIED RULES: | |
| JDBC_CURSOR_76 | | DistinctPullUp | 1 |

## MIN/MAX Optimization - Example

```
CREATE INDEX "ZZTELE~2" ON zztele ( str, nr )
```

```
SELECT MAX (nr)
  FROM zztele
 WHERE str = 'Wexstr'
```

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZTELE | ZZTELE~2 | RANGE CONDITION FOR INDEX | 2048 |
| | | ONLY INDEX ACCESSED | |
| | | MIN/MAX OPTIMIZATION | |
| | STR | (USED INDEX COLUMN) | |
| JDBC_CURSOR_19 | | RESULT IS COPIED , COSTVALUE IS | 3 |



If possible an index is used to compute the min max value of the specified column in the select list.

In this example the multiple index on columns str and nr is used to find the highest number of street Wexstr.

Access on table zztele is not necessary to deliver the result.

## DISTINCT Optimization - Example

```
CREATE INDEX "ZZTELE~1" ON zztele ( ort, str )
```

```
select distinct (ort)
    from zztele
```

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZTELE | ZZTELE~1 | INDEX SCAN | 2273 |
| | | ONLY INDEX ACCESSED | |
| | | DISTINCT OPTIMIZATION (P) | |
| JDBC_CURSOR_30 | | RESULT IS NOT COPIED , COSTVALUE IS | 2273 |

| | ORT |
|---|---|
| 1 | |
| 2 | Berlin |



DISTINCT eliminates Duplicates.

Internally MaxDB creates a temporary b*tree. The key of this temporary B* tree is defined on the DISTINCT fields of the SELECT list.

To compute the result each record which is read is copied into this temporary table. When a record with the same distinct value will be inserted twice an error avoids the insert.

This expensive procedure is not necessary if an index exists on the columns of the select list or parts of the select list. If this optimization can be used only the keys of the secondary index will be checked but not the primary key list.

In this example there is no access to the primary table necessary; the DISTINCT can be provided with the index zztele~1by the secondary key.

# NO STRATEGY NOW (ONLY AT EXECUTION TIME)

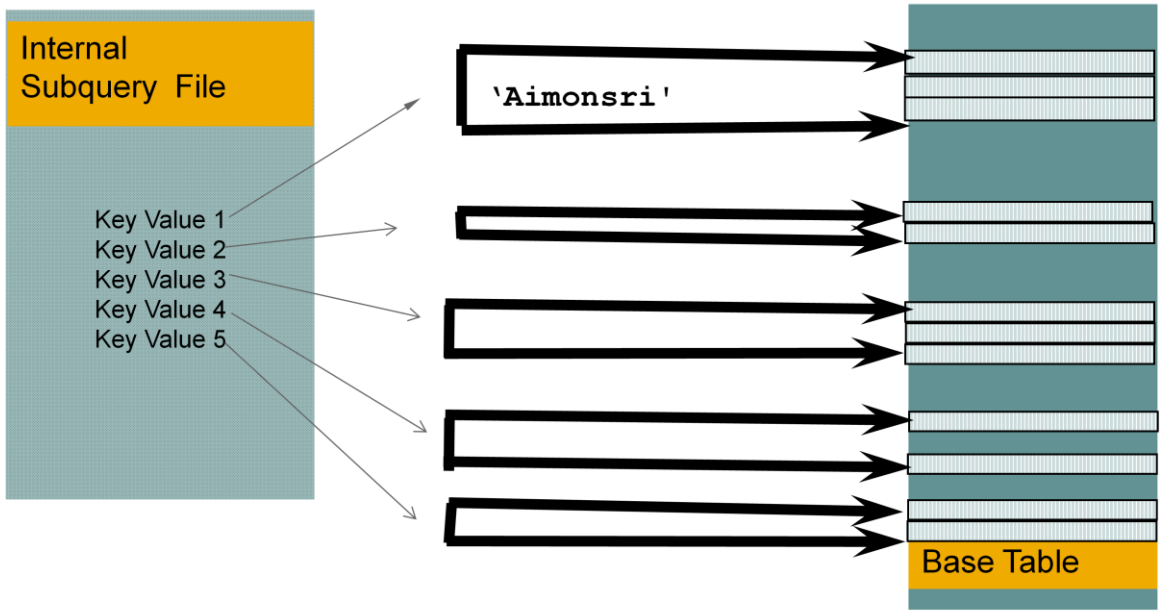
Strategy will be determined first during execution of the command

This is valid for queries if the access path will be determined first when they are executed.

Usually this output is used for queries containing sub-queries or correlated sub-queries: strategy will first be determined when interim results become available.

## SUBQUERY allows CONDITION FOR KEY

**SELECT * FROM zztele**
**WHERE name IN (SELECT name FROM zzmaster WHERE year = 2000)**






With a subquery you can generate the input values for WHERE condition of the query.

In this example all names of persons who made the master in year 2000 should be listed.

First the subquery will be executed to get the results as input for the IN condition.

The strategy which is used for the ‚external' SELECT on ZZTELE can only be chosen during execution. The values of the IN-clause are not known yet.

An intermediate result set is generated. The result set contains key values, in this example the first key column name, ordered by the primary key of the base table. The Optimizer is doing an IN or RANGE STRATEGY FOR KEY COLUMN on table zztele to find those entries which belong to the result with PLZ = 10967.

If a subquery returns values which can be compared with key values, EQUAL CONDITION FOR KEY , IN CONDITION FOR KEY or RANGE CONDITION FOR KEY is used on the base table. The result set is sorted according to primary key values.

If a subquery returns values which can be compared with index values, EQUAL CONDITION FOR Index , IN CONDITION FOR Index or RANGE CONDITION FOR Index is used.

# SUBQUERY – Example (1)

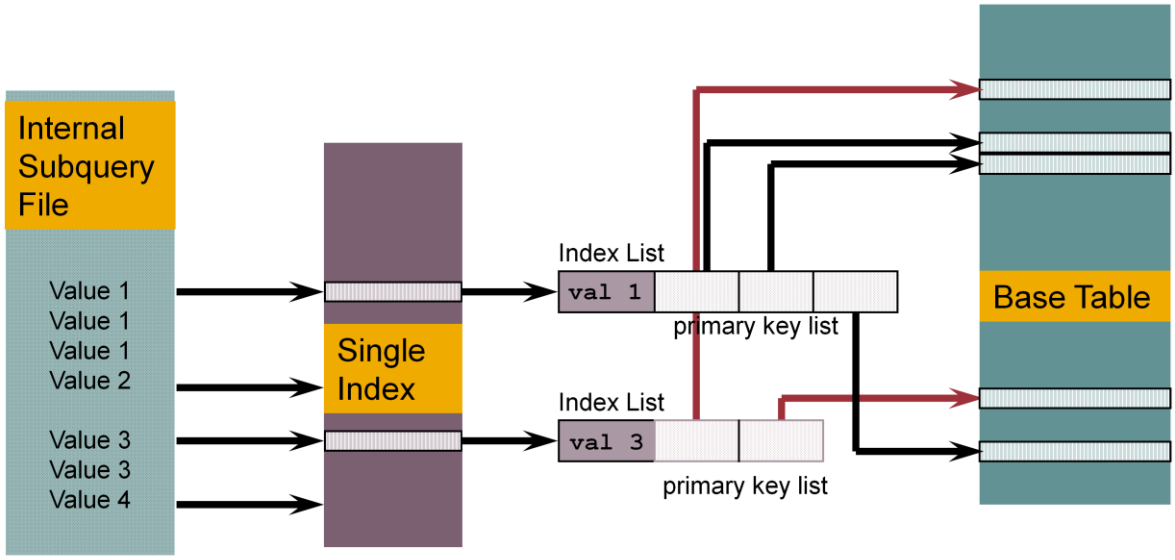
**EXPLAIN SELECT * FROM zztele**
WHERE **name IN**
(**SELECT name FROM zzmaster**
**WHERE year = 2000)**

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZMASTER | | RANGE CONDITION FOR KEY | 1 |
| | YEAR | (USED KEY COLUMN) | |
| ZZTELE | | NO STRATEGY NOW (ONLY AT EXECUTION TIME) | |
| JDBC_CURSOR_36 | | RESULT IS COPIED , COSTVALUE IS | |
| JDBC_CURSOR_36 | | QUERYREWRITE : APPLIED RULES: | |
| JDBC_CURSOR_36 | | DistinctPullUp | 1 |
| JDBC_CURSOR_36 | | DistinctPushDown | 1 |

## SUBQUERY allows CONDITION FOR INDEX

```
CREATE INDEX "ZZTELE~3" ON ZZTELE(PLZ)
```

**SELECT * FROM zztele WHERE plz IN**
**(SELECT plz FROM zzstadtteil WHERE stadtteil = 'Kreuzberg' )**






The result set is sorted according to the secondary key sequence. If only values from the index are queried, the Only Index strategy is used.

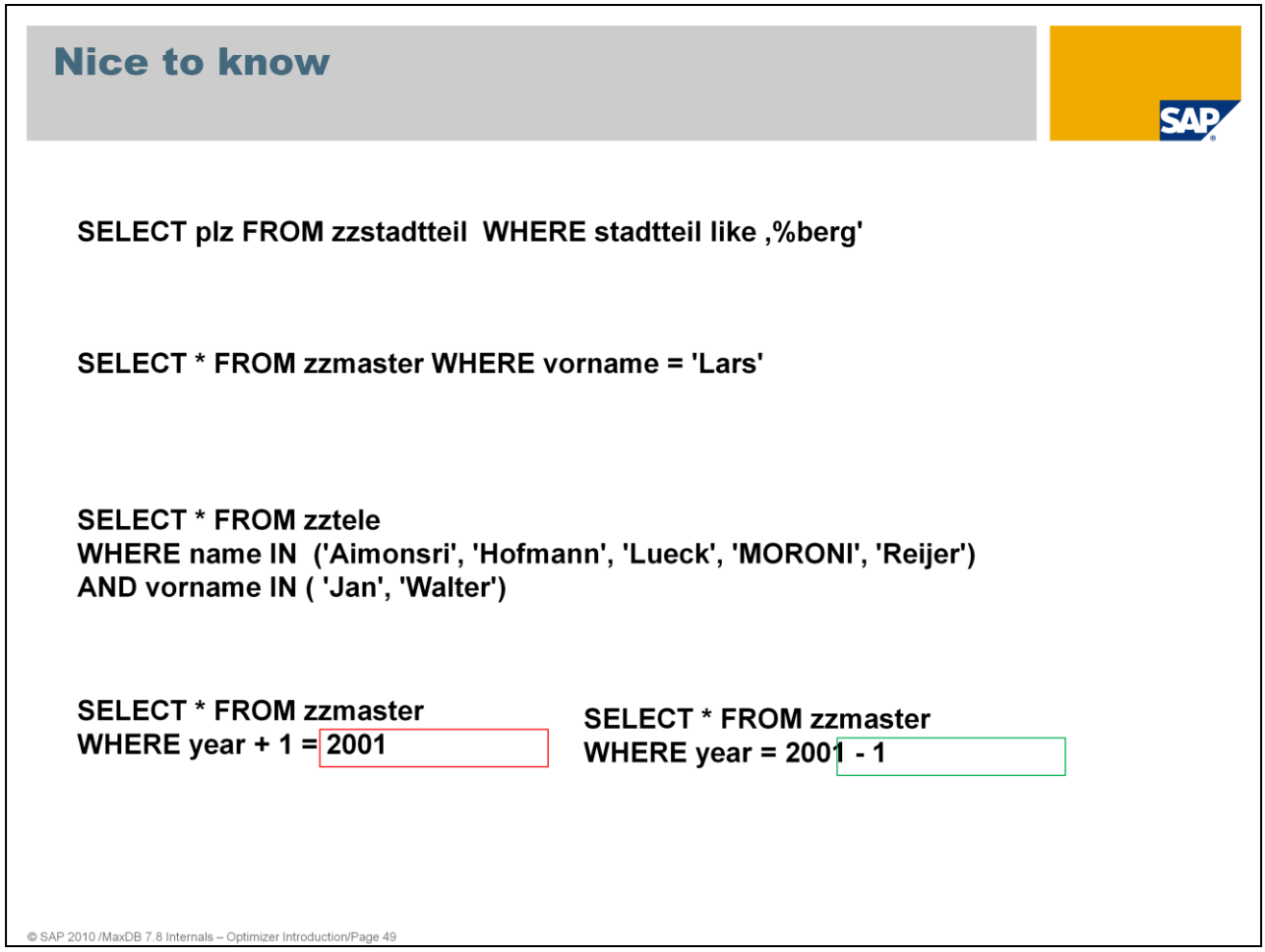
An intermediate result set is generated.

# SUBQUERY – Example (2)

```
CREATE INDEX "ZZTELE~3" ON ZZTELE(PLZ)
```

```
SELECT * FROM zztele
WHERE plz IN
(SELECT plz FROM zzstadtteil
WHERE stadtteil = 'Kreuzberg' )
```

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZSTADTTEIL | ZZSTADTTEIL~1 | EQUAL CONDITION FOR INDEX | 41 |
| | STADTTEIL | (USED INDEX COLUMN) | |
| ZZTELE | | NO STRATEGY NOW (ONLY AT EXECUTION TIME) | |
| JDBC_CURSOR_47 | | RESULT IS COPIED , COSTVALUE IS | |
| JDBC_CURSOR_47 | | QUERYREWRITE : APPLIED RULES: | |
| JDBC_CURSOR_47 | | DistinctPullUp | 1 |
| JDBC_CURSOR_47 | | DistinctPushDown | 1 |




**Wildcards at the beginning of a column specification**
Using wildcards at the beginning of a column specification cannot be optimized. All rows of the table can be part of the result.

Such queries will be processed with table scans and can result in terrible performance.

Solution: Teach your end users not to start the specification with %

**WHERE qualification specifies only some columns at the end of an index or the primary key**

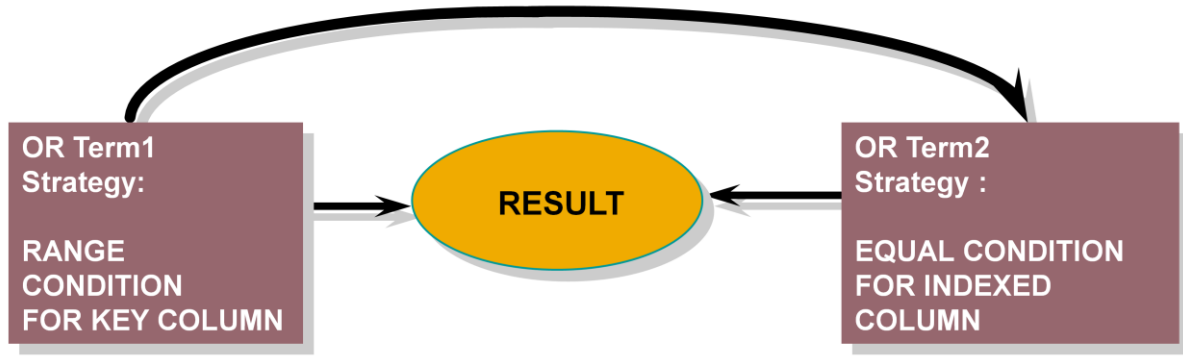
As the first key filed (Name) was not specified, MaxDB is not able to use the primary key of ZZTELE and performs a table scan which can result in terrible performance.
Solution: Teach your end users to specify as many values as possible
Create secondary index

**Only one IN condition can be optimized, if there are more than one IN qualifications the first is optimized.** All followed IN conditions will be processed via range.

**Do not use functions in where column qualification.** This cannot be optimized. Always try to use the function in the value specification.

# DIFFERENT STRATEGIES FOR OR-TERMS

```
CREATE INDEX "ZZTELE~3" ON ZZTELE(PLZ)
SELECT * FROM zztele
 WHERE name= 'Aaron'
  OR    plz    = '12345'
```



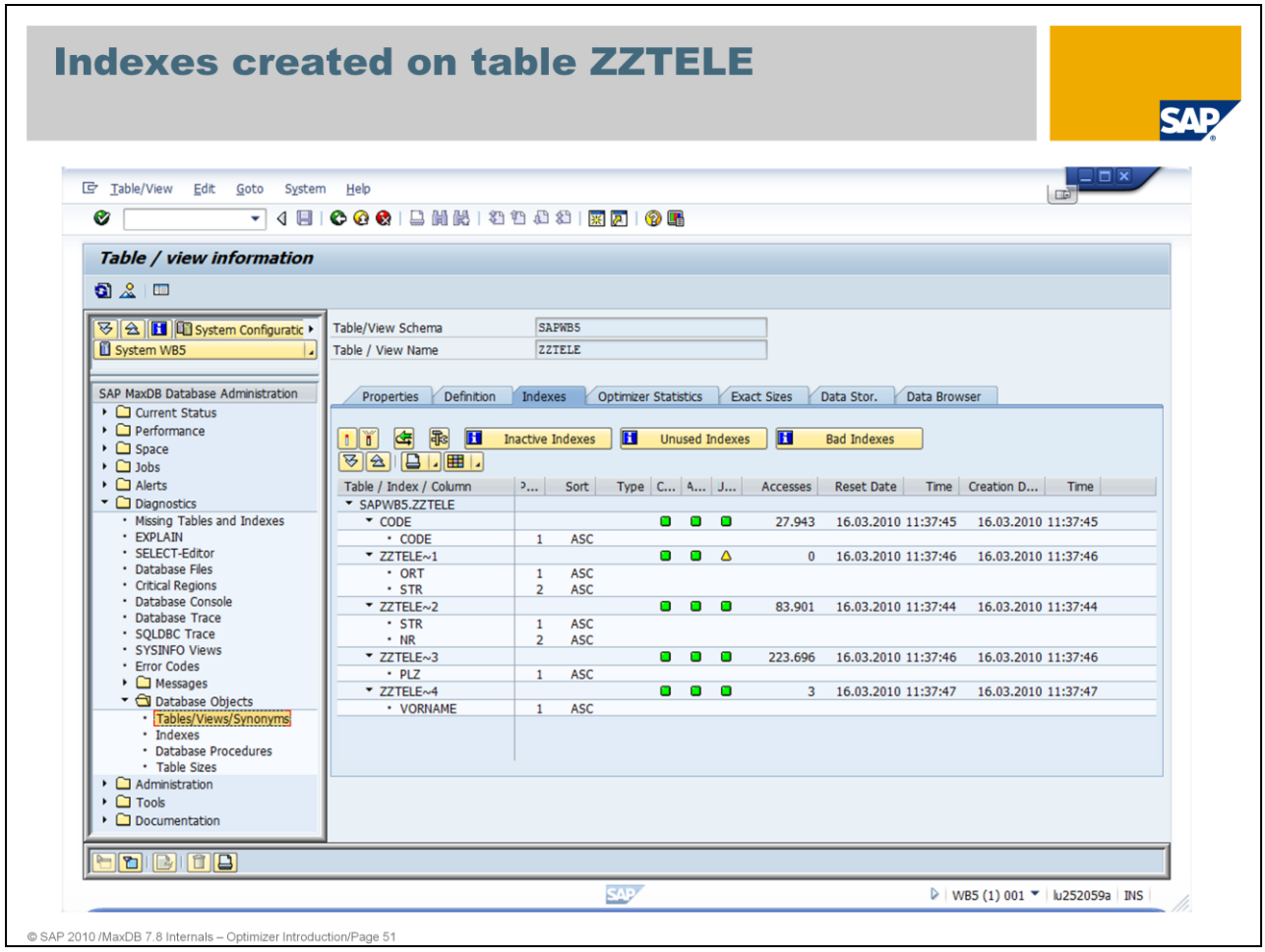



Nested OR terms are analyzed down to the third level.

If the costs of the strategy search exceed the costs determined for the highest level, the strategy search is discontinued.

An intermediate result set is generated.

Within the SAP environment, similar statements are also generated by SELECTS with RANGES.




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>*

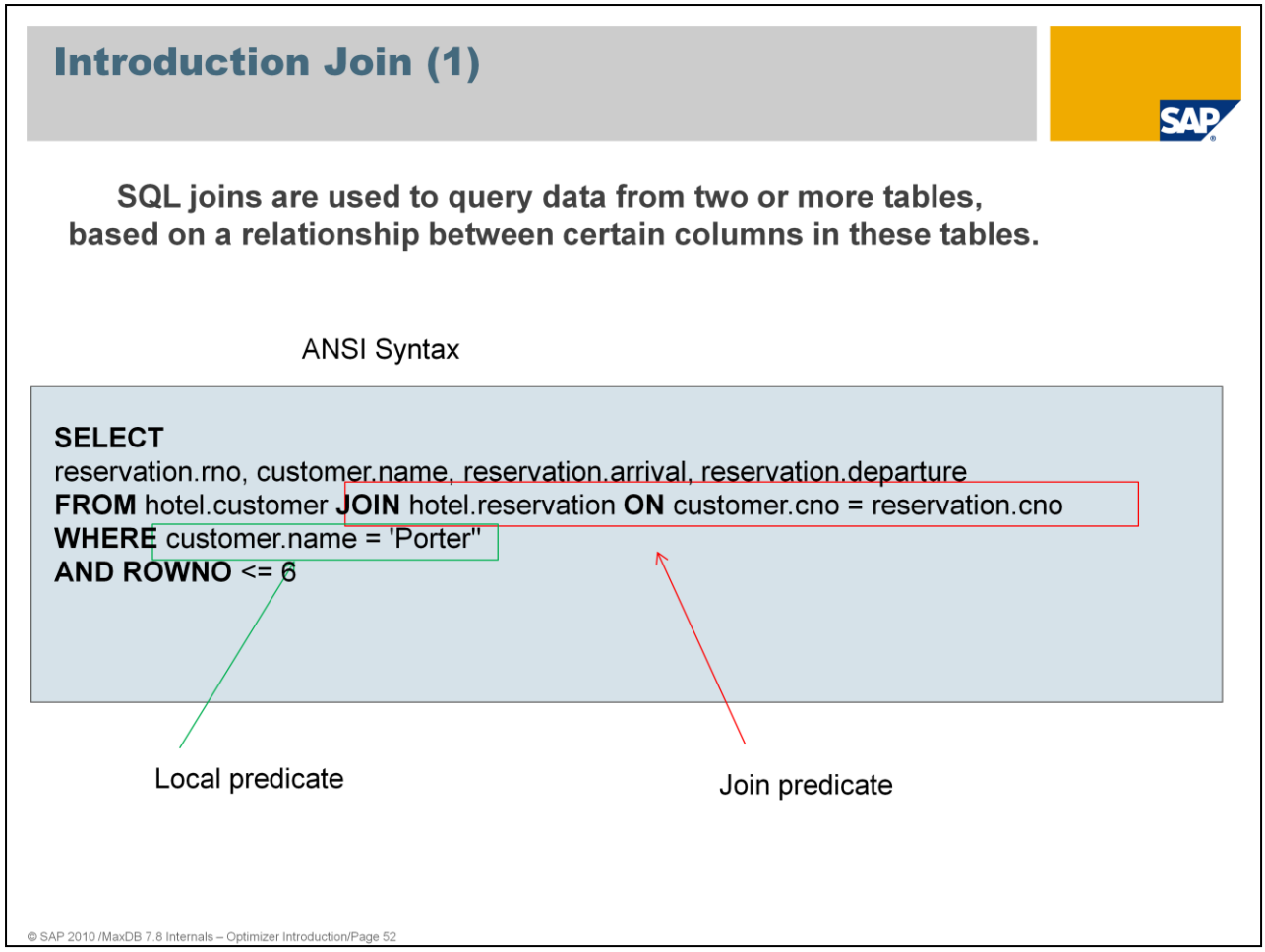
*AND schemaname = <schema>*

*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.

## Introduction Join (2)

```
EXPLAIN SELECT
reservation.rno, customer.name, reservation.arrival, reservation.departure
FROM hotel.customer JOIN hotel.reservation ON customer.cno = reservation.cno
WHERE customer.name = 'Porter''
AND ROWNO <= 6
```

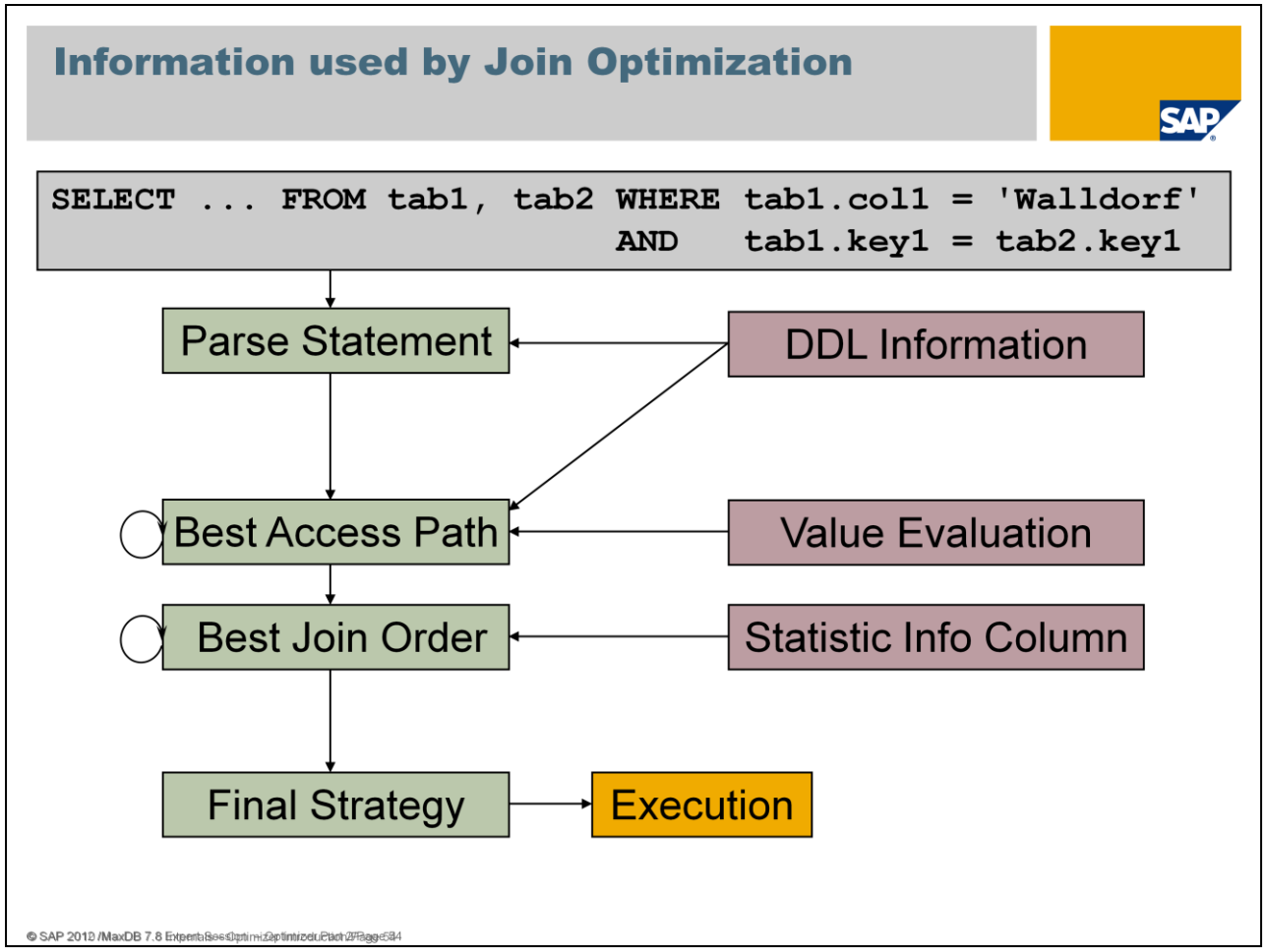
| SCHEMANAME | TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|---|
| HOTEL | CUSTOMER | FULL_NAME_INDEX | RANGE CONDITION FOR INDEX | 1 |
| | | | ONLY INDEX ACCESSED | |
| | | NAME | (USED INDEX COLUMN) | |
| HOTEL | RESERVATION | | JOIN VIA KEY RANGE | 1 |
| | | | TABLE TEMPORARY SORTED | |
| | | CNO | (USED COLUMN) | |
| | JDBC_CURSOR_15 | | RESULT IS COPIED , COSTVALUE IS | 3 |
| | JDBC_CURSOR_15 | | QUERYREWRITE : APPLIED RULES: | |
| | JDBC_CURSOR_15 | | PushDownPredicates | 1 |



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 executation.
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 STAT[ISTICS]  [<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. Additonally 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 in 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

## Update Statistics (2)

```
ALTER TABLE <table_name>
  SAMPLE <unsigned_integer> <PERCENT,ROWS>
```

The default value for the number of rows to be included when determining the statistics is stored in the database catalog.

This value can be changed either directly with ALTER TABLE or using transaction DBACOCKPIT -> Diagnostics-> Database Objects ->Tables/Views/Synonyms

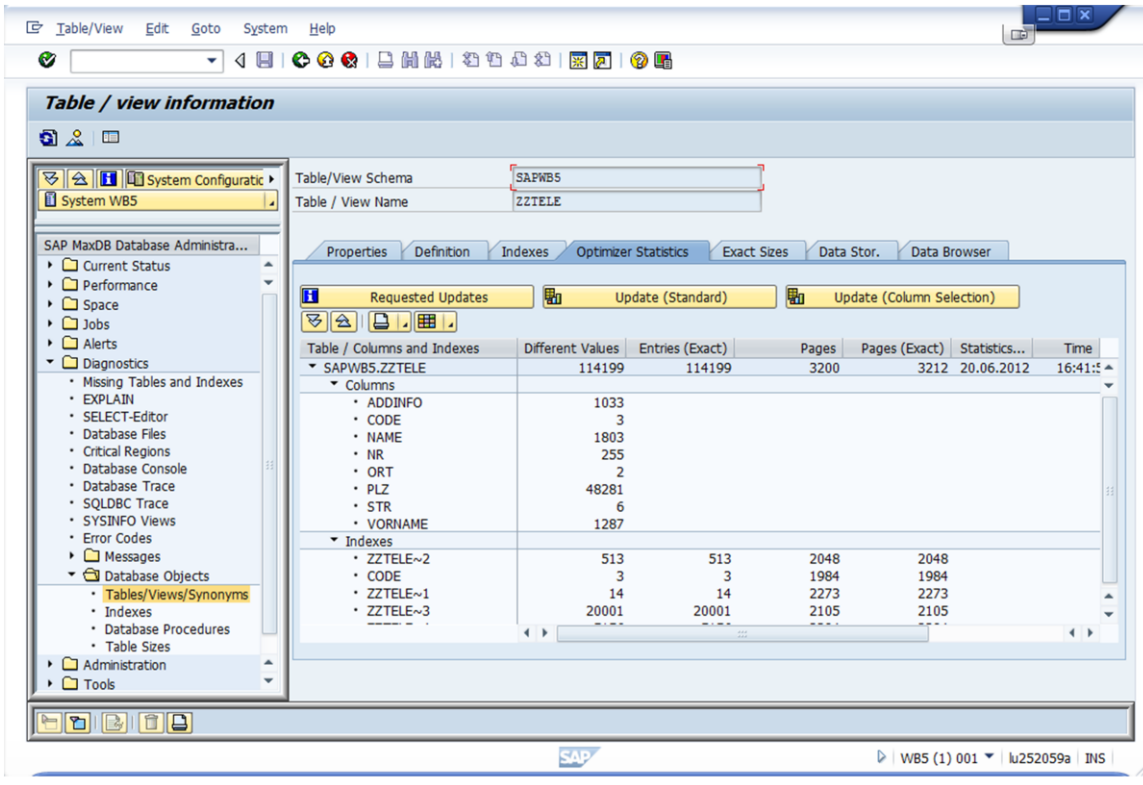


For tables that grow and shrink very quickly, such as spool tables, for example, it is a good idea to set the sampling rate to 0. This prevents Update Statistics from being requested and executed for these tables.

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

sql_updatestat SAP<SID>.* estimate

# Update Statistics (3)






*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.

# Update Statistics (4)

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

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

| TABLENAME | INDEXNAME | COLUMNNAME | DISTINCTVALUES | PAGECOUNT |
|---|---|---|---|---|
| ZZTELE | ? | ADDINFO | 1969 | ? |
| ZZTELE | ? | CODE | 2 | ? |
| ZZTELE | ? | NAME | 13363 | ? |
| ZZTELE | ? | NR | 255 | ? |
| ZZTELE | ? | ORT | 2 | ? |
| ZZTELE | ? | PLZ | 20001 | ? |
| ZZTELE | ? | STR | 8 | ? |
| ZZTELE | ? | VORNAME | 5156 | ? |
| ZZTELE | CODE | ? | ? | 1155 |
| ZZTELE | ZZTELE~1 | ? | ? | 1165 |
| ZZTELE | ZZTELE~3 | ? | ? | 1112 |
| ZZTELE | ZZTELE~4 | ? | ? | 1334 |
| ZZTELE | ZZTELE~2 | ? | ? | 1548 |
| ZZTELE | ? | TABLE STATISTICS | 114199 | 1800 |



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*.

## Counters in the File Directory

```
SELECT  f.type, r.tablename, r.indexname, f.entrycount,
        f.treeindexsize, f.treeleavessize, f.lobsize
  FROM    files f, roots r
  WHERE   f.fileid = r.tableid
  AND     r.tablename = ('ZZTELE' )
```

- Displays the current counter values in the file directory.

| TYPE | TABLENAME | INDEXNAME | ENTRYCOUNT | TREEINDEXSIZE | TREELEAVESSIZE | LOBSIZE |
|---|---|---|---|---|---|---|
| TABLE | ZZTELE | ? | 114199 | 144 | 14400 | 0 |
| INDEX | ZZTELE | CODE | 2 | 9240 | 9240 | ? |
| INDEX | ZZTELE | ZZTELE~1 | 10 | 9320 | 9320 | ? |
| INDEX | ZZTELE | ZZTELE~3 | 20001 | 8896 | 8896 | ? |
| INDEX | ZZTELE | ZZTELE~4 | 5156 | 10672 | 10672 | ? |
| INDEX | ZZTELE | ZZTELE~2 | 513 | 12384 | 12384 | ? |



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.

## Explain (1)

Input : EXPLAIN <SELECT-Command>

Output : Description of search strategy

- EXPLAIN is used with SELECT commands that access tables and views
- EXPLAIN does not execute the specified SELECT command.



An executation 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 JOIN -> http://wiki.sdn.sap.com/wiki/pages/viewpage.action?pageId=13230&bc=true

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

## Explain (2)

| SCHEMANAME | TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|---|
| Schema | Table 1 | Names of key or index columns | Name of chosen strategy for this table | Number of pages In system table Optimizerstatistics |
| Schema | Table 2 | Names of key or index columns | Name of chosen strategy for this table | Number of pages in system table Optimizerstatistics |
| | Result name | | RESULT IS (NOT) COPIED, COSTVALUE IS | Estimated costs |
| | | | Applied Query Rewrite rules | 1 |



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

## Which condition will be evaluated ?

Join select

- Table1_column = table2_column
- Condition has to be on the ‚Top-AND-Level' of the <search condition>,(Or terms are not relevant)

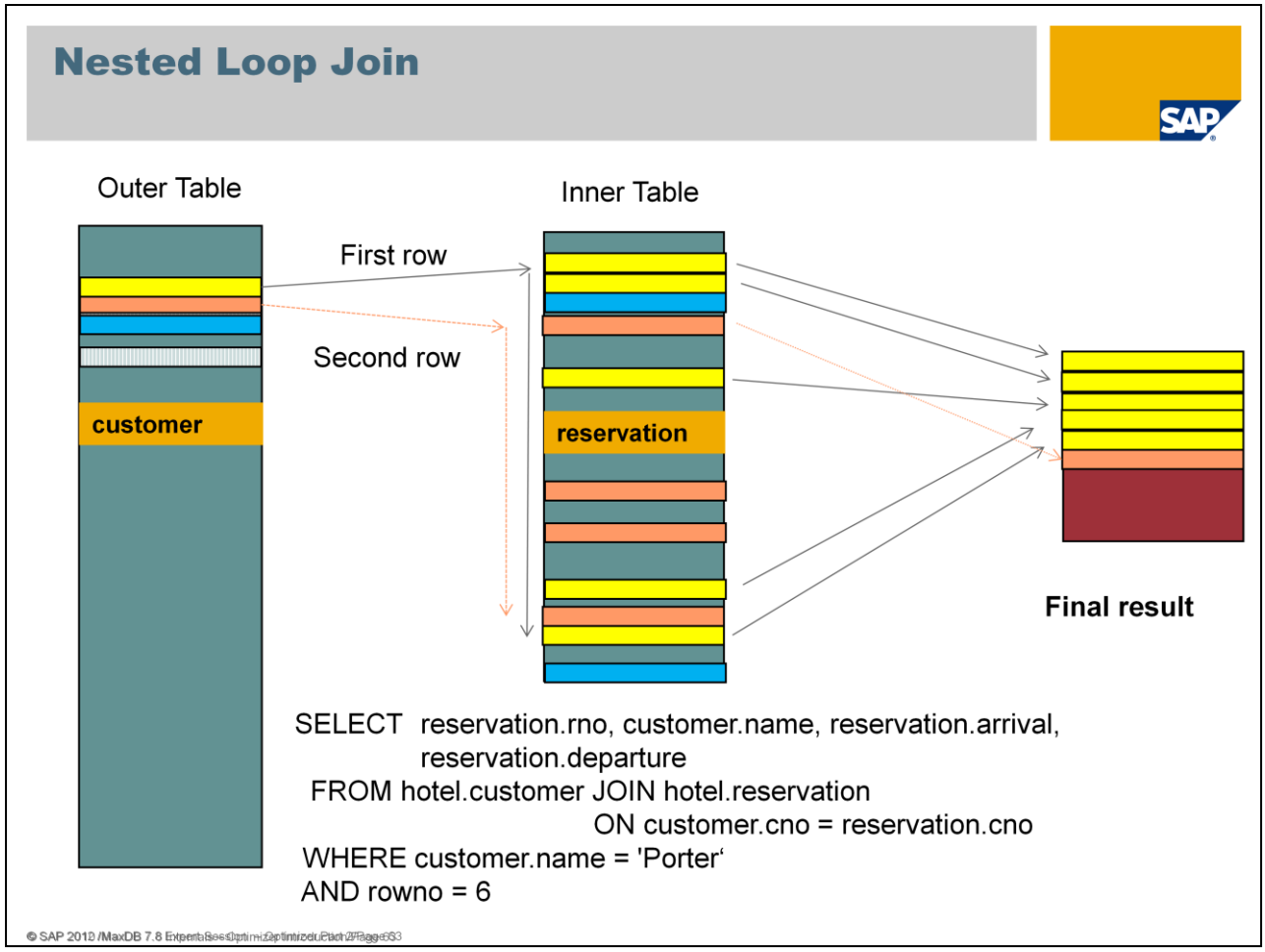


Search conditions used by the optimizer to determine the optimal search strategy are:

- 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.*

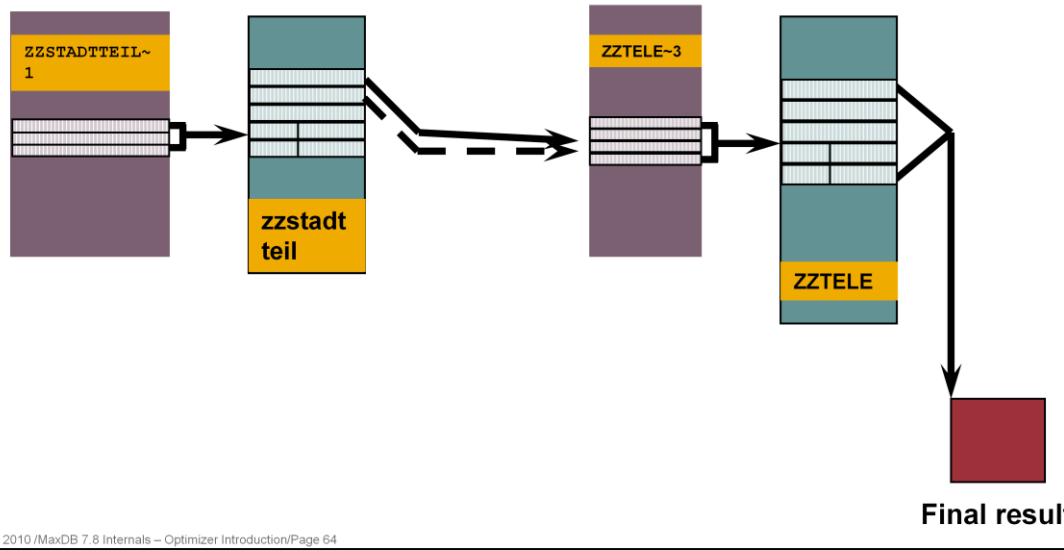
For each hit in customer (outer table) the inner table *reservation* is read. Each hit in reservation is inserted immediately into the final result.

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.

## Join Across two Tables (Nested Loop)

```
CREATE INDEX "ZZTELE~3" ON ZZTELE(PLZ)
CREATE INDEX "ZZSTADTTEIL~1" ON ZZSTADTTEIL(STADTTEIL)

SELECT * FROM zztele JOIN zzstadtteil
ON zztele.plz = zzstadtteil.plz
WHERE  zzstadtteil.stadtteil = 'Moabit'
```






Here is an example for nested loop join processed via index strategies.

## Join Key Strategies

```
SELECT * FROM scantab JOIN jointab ON scantab.A = jointab.Col1
AND   scantab.B = jointab.Col2
```

| Join Strategy | Meaning |
|---|---|
| JOIN VIA KEY COLUMN | Join table has a single key column<br>key column is part of the join |
| JOIN VIA MULTIPLE KEY COLUMNS | Join table has multiple key columns<br>all key columns are part of the join |
| JOIN VIA KEY RANGE | Join table has multiple key columns<br>the first key column is part of the join |
| JOIN VIA RANGE OF MULTIPLE KEY COLUMNS | Join table has multiple key columns<br>some key columns are part of the join |



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 acess 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.

## Join via key column (1)

**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'
```

| ZZTELE | | RANGE CONDITION FOR KEY | 3200 |
|---|---|---|---|
| | NAME | (USED KEY COLUMN) | |
| ZZSTADTTEIL | PLZ | JOIN VIA KEY COLUMN | 98 |
| | | NO TEMPORARY RESULTS CREATED | |
| JDBC_CURSOR_15 | | RESULT IS COPIED , COSTVALUE IS | 79 |
| JDBC_CURSOR_15 | | QUERYREWRITE : APPLIED RULES: | |
| JDBC_CURSOR_15 | | DistinctPullUp | 1 |
| JDBC_CURSOR_15 | | PushDownPredicates | 1 |



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'
```

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZMASTER | | RANGE CONDITION FOR KEY | 1 |
| | YEAR | (USED KEY COLUMN) | |
| ZZTELE | | JOIN VIA MULTIPLE KEY COLUMNS | 3200 |
| | NAME | (USED KEY COLUMN) | |
| | VORNAME | (USED KEY COLUMN) | |
| | STR | (USED KEY COLUMN) | |
| | | NO TEMPORARY RESULTS CREATED | |
| JDBC_CURSOR_71 | | RESULT IS COPIED , COSTVALUE IS | 2 |



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.*

## Join via key range

**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'
```

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZMASTER | | RANGE CONDITION FOR KEY | 1 |
| | YEAR | (USED KEY COLUMN) | |
| ZZTELE | NAME | JOIN VIA KEY RANGE | 3200 |
| | | NO TEMPORARY RESULTS CREATED | |
| JDBC_CURSOR_25 | | RESULT IS COPIED , COSTVALUE IS | 13 |



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.*

## Join via key range of multiple key columns

**zztele.name** is the **first** primary key column
**zztele.vorname** is the **second** primary key column

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

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZMASTER | | RANGE CONDITION FOR KEY | 1 |
| | YEAR | (USED KEY COLUMN) | |
| ZZTELE | | JOIN VIA RANGE OF MULTIPLE KEY COLUMNS | 3200 |
| | NAME | (USED KEY COLUMN) | |
| | VORNAME | (USED KEY COLUMN) | |
| | | NO TEMPORARY RESULTS CREATED | |
| JDBC_CURSOR_16 | | RESULT IS COPIED , COSTVALUE IS | 13 |



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 columnes of the joined table and has historical reasons too.

## Indexes on table ZZTELE

| OWNER | TABLENAME | INDEXNAME | COLUMNNAME |
|---|---|---|---|
| SAPWB5 | ZZTELE | ZZTELE~2 | STR |
| SAPWB5 | ZZTELE | ZZTELE~2 | NR |
| SAPWB5 | ZZTELE | CODE | CODE |
| SAPWB5 | ZZTELE | ZZTELE~1 | ORT |
| SAPWB5 | ZZTELE | ZZTELE~1 | STR |
| SAPWB5 | ZZTELE | ZZTELE~3 | PLZ |
| SAPWB5 | ZZTELE | ZZTELE~4 | VORNAME |



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

## Join Index Strategies

```
SELECT * FROM scantab JOIN jointab ON scantab.A = jointab.Col1
AND   scantab.B = jointab.Col2
```

| Join Strategy | Meaning |
|---|---|
| JOIN VIA INDEXED COLUMN | Join table has a single index column<br>Single Index column is part of the join |
| JOIN VIA MULTIPLE INDEXED COLUMNS | Join table has multiple index columns<br>all index columns are part of the join |
| JOIN VIA RANGE OF MULTIPLE INDEXED COLUMNS | col1 is the first column of a multiple index<br>col2 is the second column of a multiple index |



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.

## Join via indexed column

**zztele.plz** is a **single** index column of zztele~3

zztele.ort is a standard column

```
SELECT *
FROM zztele JOIN zzstadtteil ON zzstadtteil.plz = zztele.plz
AND zzstadtteil.ort = zztele.ort
 WHERE zzstadtteil.stadtteil = 'Kreuzberg'
```

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZSTADTTEIL | | TABLE SCAN | 98 |
| ZZTELE | ZZTELE~3 | JOIN VIA INDEXED COLUMN | 3200 |
| | PLZ | (USED INDEX COLUMN) | |
| | | NO TEMPORARY RESULTS CREATED | |
| JDBC_CURSOR_265 | | RESULT IS COPIED , COSTVALUE IS | 6696 |



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*.

## Join via multiple indexed columns

**zztele.ort** is the first index column of index ZZTELE~1
**zztele.nr** is the second index column of index ZZTELE~1

```
SELECT *
FROM zztele JOIN zzcode ON zztele.str = zzcode.str
AND  zztele.ort = zzcode.ort
WHERE zzcode.code = 'Cable TV'
```

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZCODE | | TABLE SCAN | 2776 |
| ZZTELE | ZZTELE~1 | JOIN VIA MULTIPLE INDEXED COLUMNS | 3200 |
| | ORT | (USED INDEX COLUMN) | |
| | STR | (USED INDEX COLUMN) | |
| | | NO TEMPORARY RESULTS CREATED | |
| JDBC_CURSOR_14 | | RESULT IS COPIED , COSTVALUE IS | 93483620 |
| JDBC_CURSOR_14 | | QUERYREWRITE : APPLIED RULES: | |
| JDBC_CURSOR_14 | | PushDownPredicates | 1 |



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.

## Join via range of multiple indexed columns

**zztele.str** is the **first** index column of index ZZTELE~2
the second index column (NR) of index ZZTELE~2 is not qualified

```
SELECT *
FROM zztele JOIN zzcode ON zztele.str = zzcode.str
WHERE zzcode.code = 'Cable TV'
```

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZCODE | | TABLE SCAN | 1 |
| ZZTELE | ZZTELE~2 | JOIN VIA RANGE OF MULTIPLE INDEXED COL. | 3200 |
| | STR | (USED INDEX COLUMN) | |
| | | NO TEMPORARY RESULTS CREATED | |
| JDBC_CURSOR_185 | | RESULT IS COPIED , COSTVALUE IS | 6378 |



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.

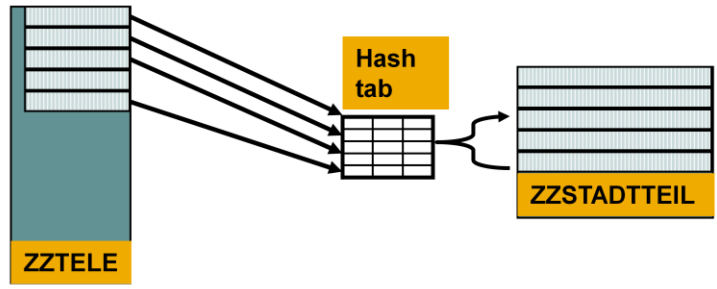
If there is more than one index column part of the join predicates but not all secondary key columns are qualified then we are talking about the Join strategy

*JOIN VIA RANGE OF MULTIPLE INDEXED COLUMNS*.

## Hash Join (2)

```
CREATE INDEX "ZZTELE~3" ON ZZTELE(PLZ)

SELECT zzstadtteil.* FROM zztele JOIN zzstadtteil
ON zztele.plz = zzstadtteil.plz
```




**Kernel parameter:** **EnableJoinHashTableOptimization**
**JoinHashMinimalRatio**
**HashJoinSingleTableMemorySize**
**HashJoinTotalMemorySize**



**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

| TABLENAME | COLUMN_OR_INDEX | STRATEGY | PAGECOUNT |
|---|---|---|---|
| ZZTELE | ZZTELE~3 | INDEX SCAN | 3200 |
| | | ONLY INDEX ACCESSED | |
| ZZSTADTTEIL | PLZ | JOIN VIA KEY COLUMN | 98 |
| | | TABLE HASHED | |
| | | NO TEMPORARY RESULTS CREATED | |
| JDBC_CURSOR_248 | | RESULT IS COPIED , COSTVALUE IS | 3801 |

## Hints

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.

# Thank you!