SAP® MaxDB™ Expert Sessions

SAP MaxDB SQL Query Optimization - Part 1

Christiane Hienger SAP AG
Holger Becker SAP AG

August 28, 2012

THE BEST-RUN BUSINESSES RUN SAP™

Agenda

1. Introduction

2 . EXPLAIN
   - 2.1. EXPLAIN
   - 2.2. EXPLAIN QUERYREWRITE

3. Single Table Optimizer Strategies

After this presentation, you will be able to:

- Use several EXPLAIN commands to analyze SQL queries
- Understand and interpret the EXPLAIN output
- Understand the usage of SQL strategies of the single table optimizer

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.

Primary and Secondary keys (indexes)

Primary key

- The primary key is stored in the data tree (clustered)
- No separate tree for primary key !
- Parts of the primary keys are used as separator in B*trees
- The records are stored in primary key order

Secondary key (index)

- Construction of a separate B*tree for the secondary key values
- A secondary key does not contain physical addresses pointing to the base data but logical addresses in terms of primary keys

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.

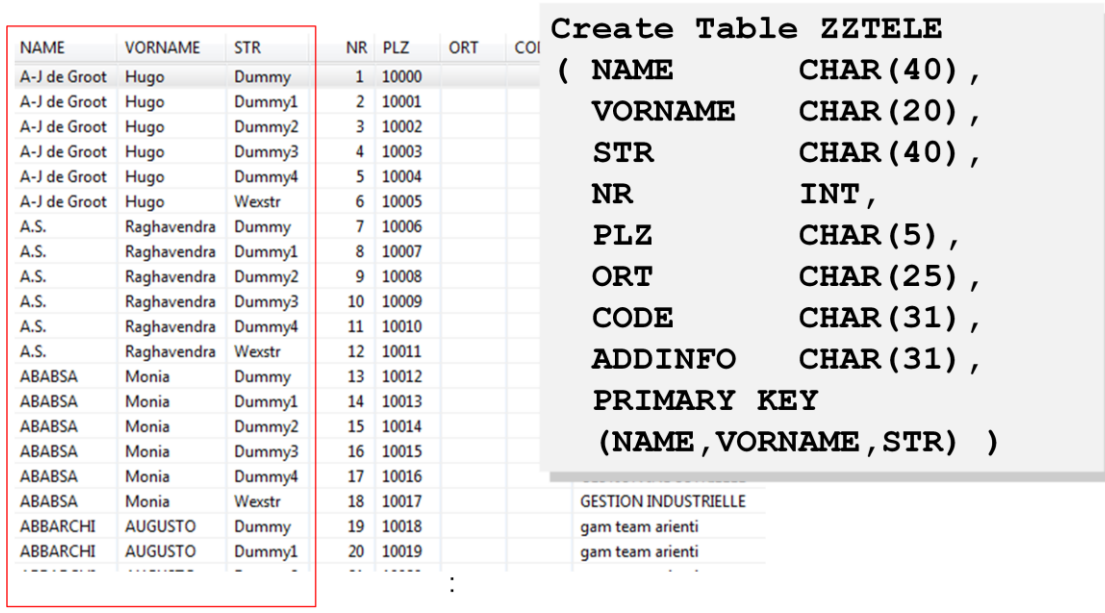
A separate B* tree is created for a secondary key (or index). The secondary key (index) contains no physical addresses on the data tree, instead it contains logical addresses in the form of primary keys. An index is a database object that can be created for an individual column or a series of columns 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		
ABBARCHI	AUGUSTO	Dummy	19	10018		
ABBARCHI	AUGUSTO	Dummy1	20	10019		

```
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) )
```

GESTION INDUSTRIELLE
gam team arienti
gam team arienti

# 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

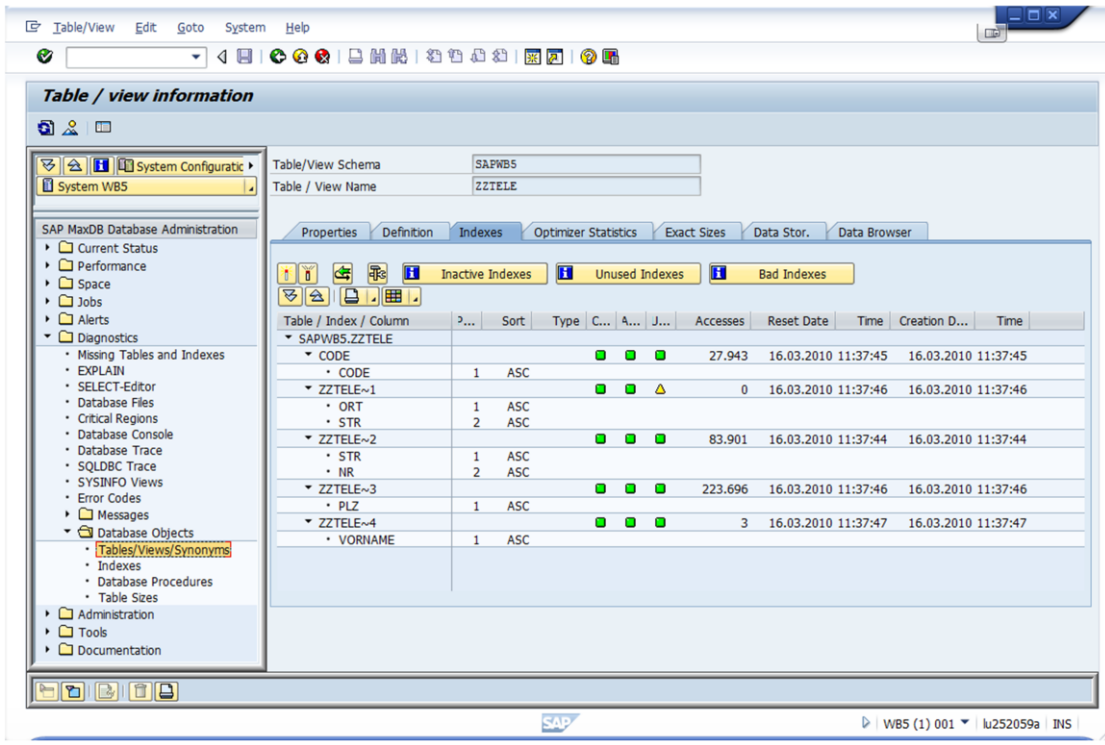
To explain strategies which can be used for subqueries, the examples also refer to the table ZZSTADTTEIL with approx. 20000 records and table ZZMASTER.

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.

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

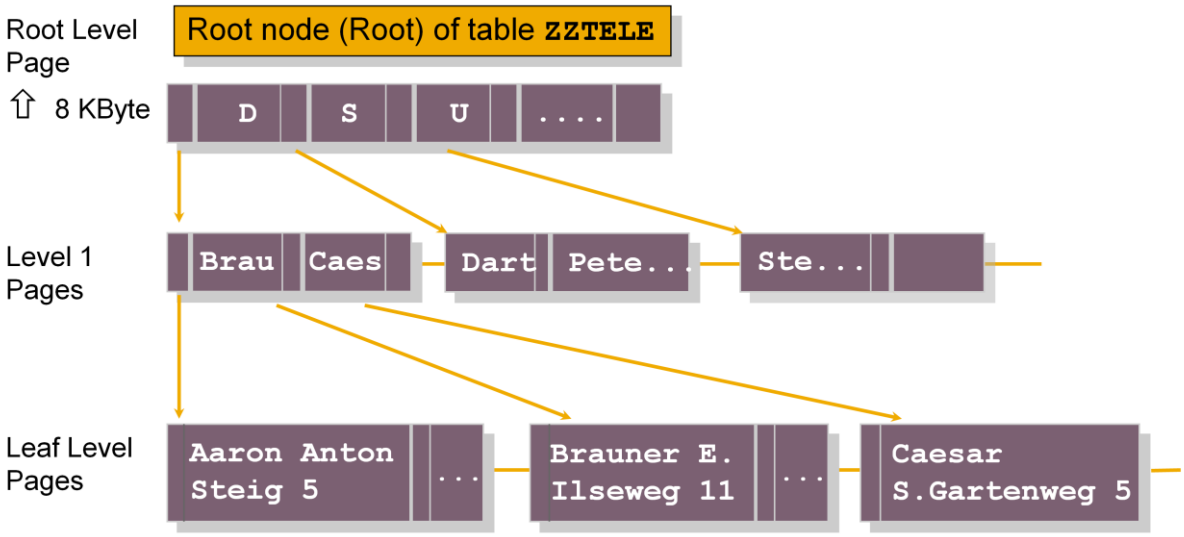
WHERE owner = <owner>

AND schemaname = <schema>

AND tablename = <table_name>

ORDER BY owner, tablename, indexname, columnno

Storing data in a B*tree

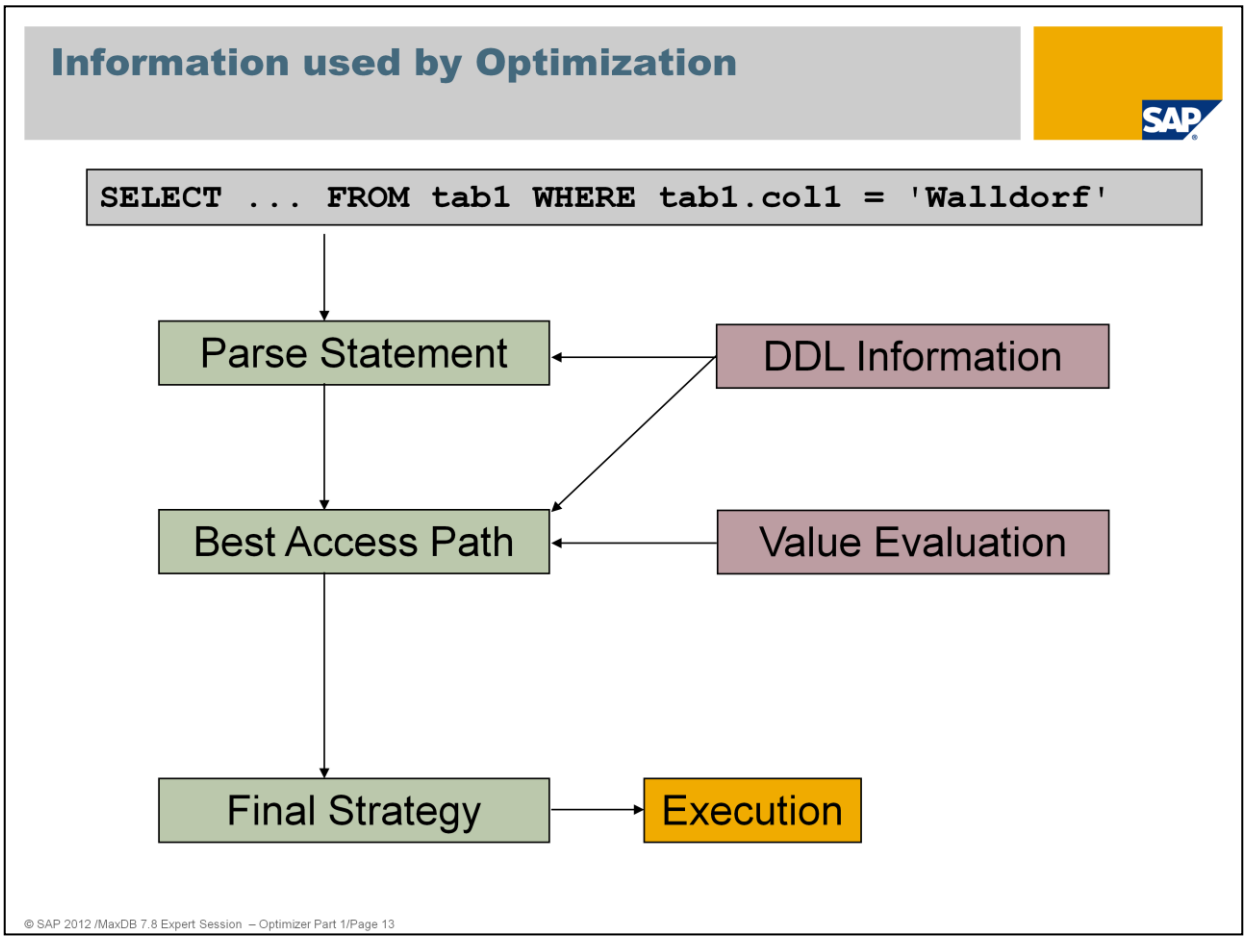


Base data as well as index data (secondary key) is stored in B*tree structures.

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
SET ADDINFO = 'ledig'
WHERE NAME = 'Mueller'
AND   VORNAME = ' Egon‘
AND   STR = ' Wexstraße‘
```

```
SELECT * FROM ZZTELE
WHERE NAME = 'Mueller'
 AND   VORNAME = ' Egon'
 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

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 "SAPA1S"."ZZTELE" AS "_T1"

The view MONITOR indicates how often rules were applied.

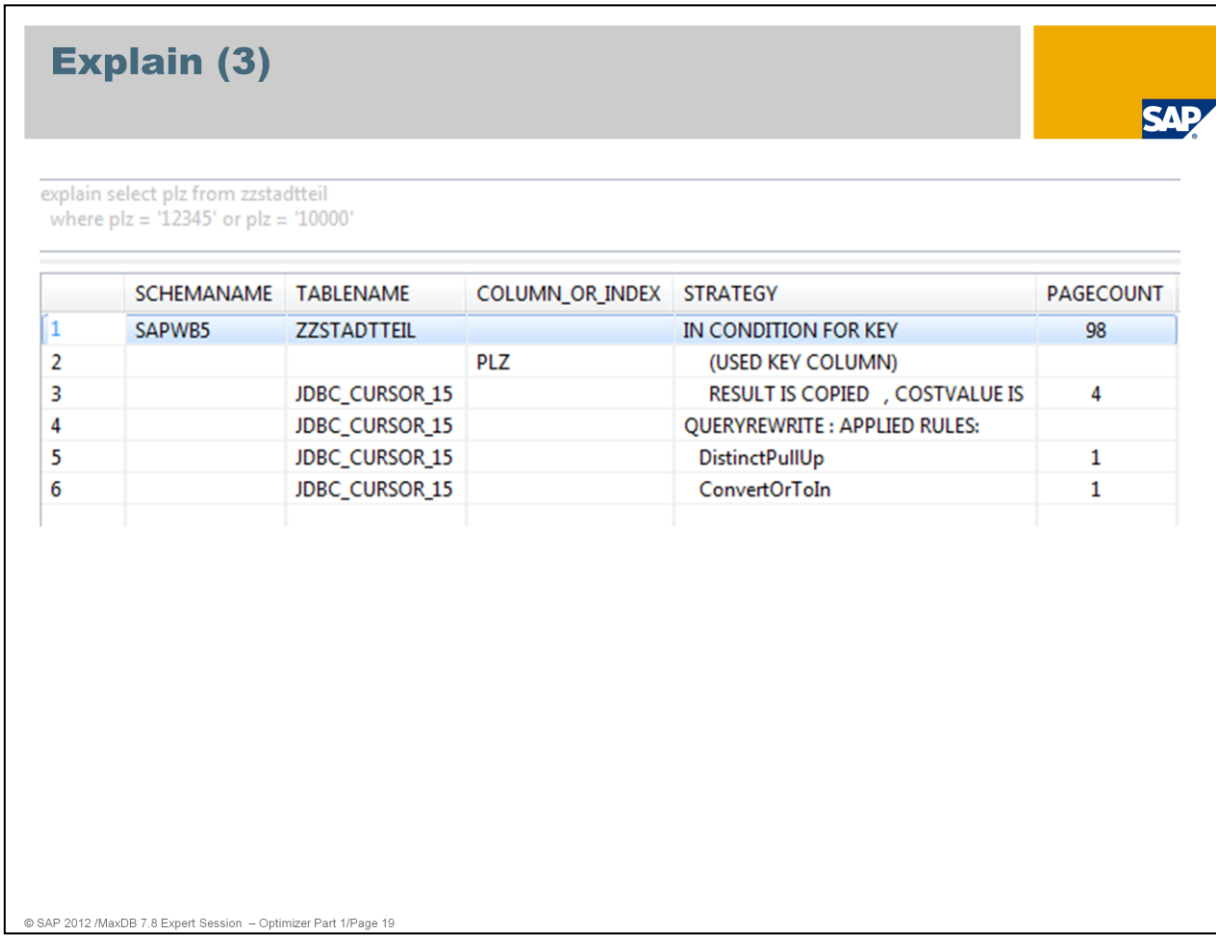
```
select * from monitor
where type = 'REWRITE'
```

	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‘

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.

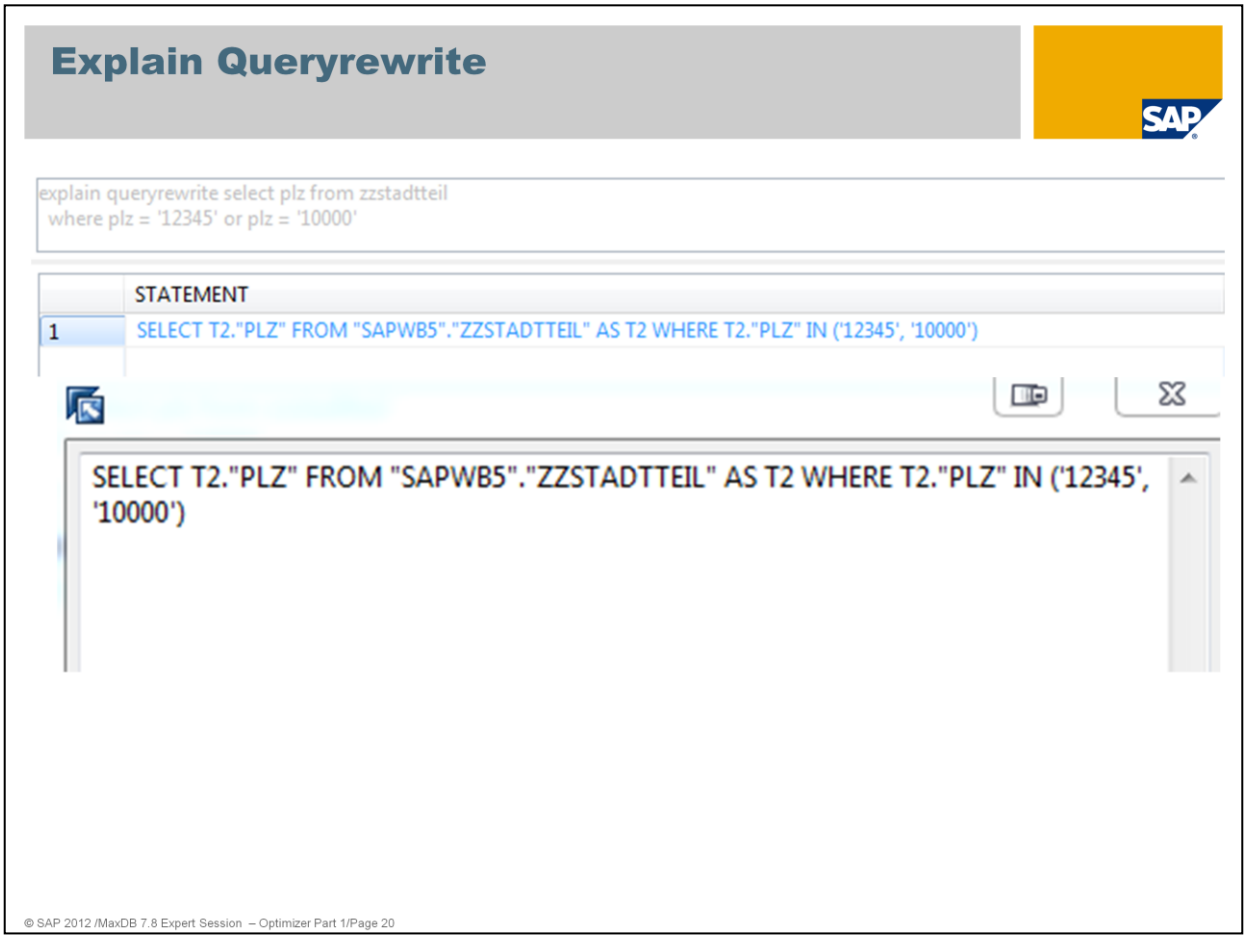
Select plz from zzstadtteil where plz = '12345' or plz = '10000'

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.

Explain Queryrewrite



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

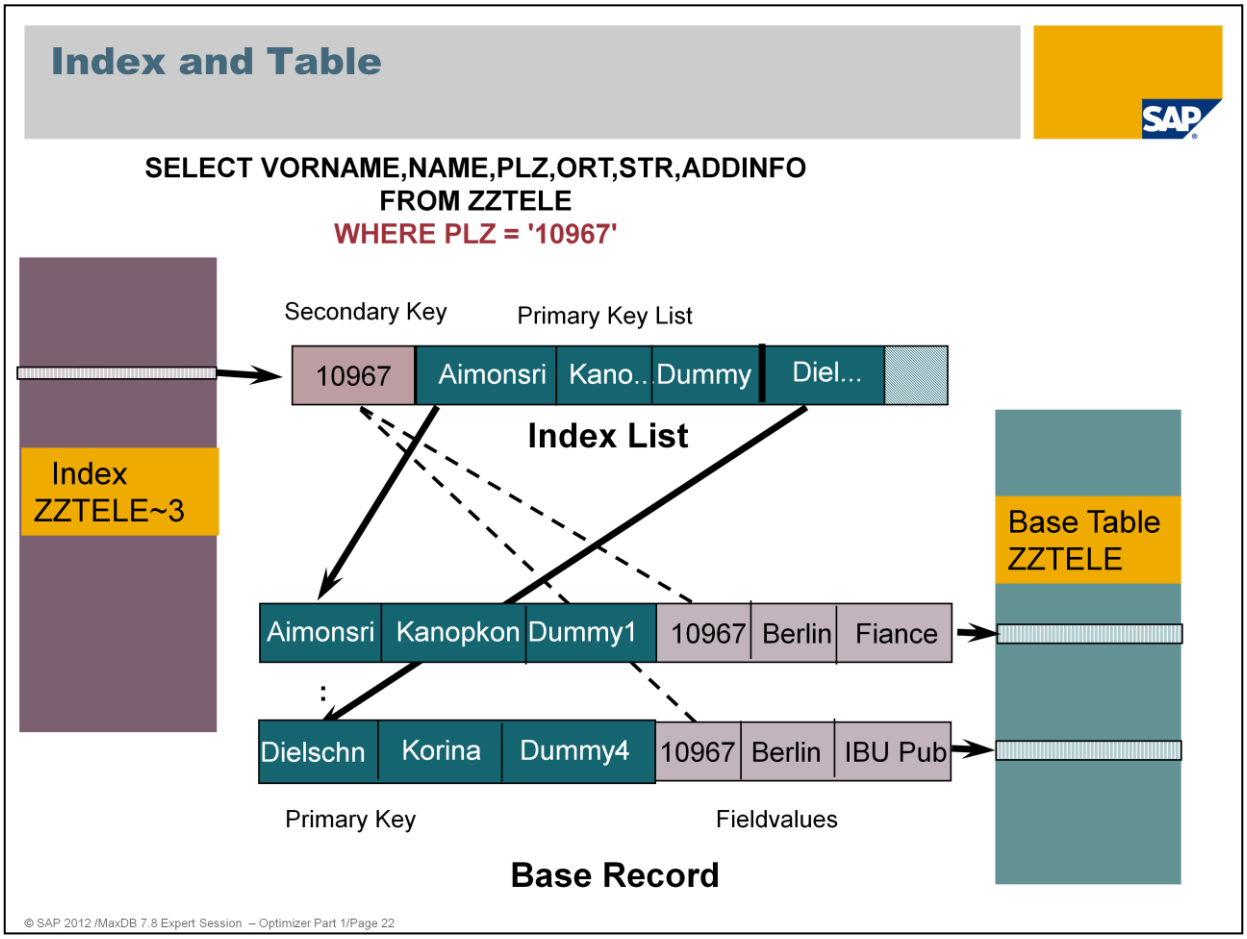
STRATEGY	PAGECOUNT
TABLE SCAN	3212
RESULT IS NOT COPIED , COSTVALUE IS	3212
QUERYREWRITE : APPLIED RULES:	
DistinctPullUp	1

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.

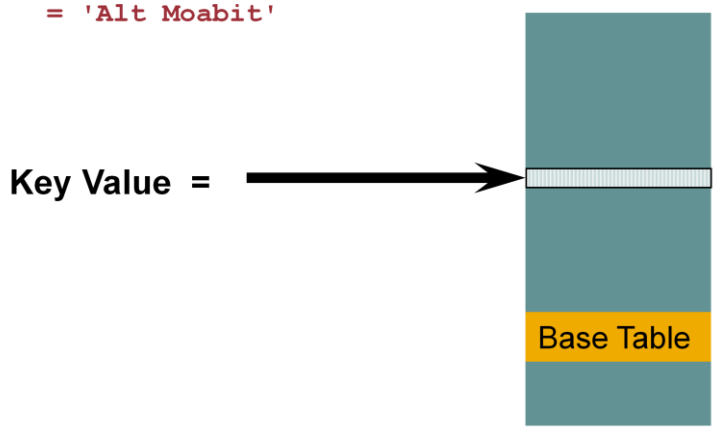
Searching via an index is relatively costly.

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

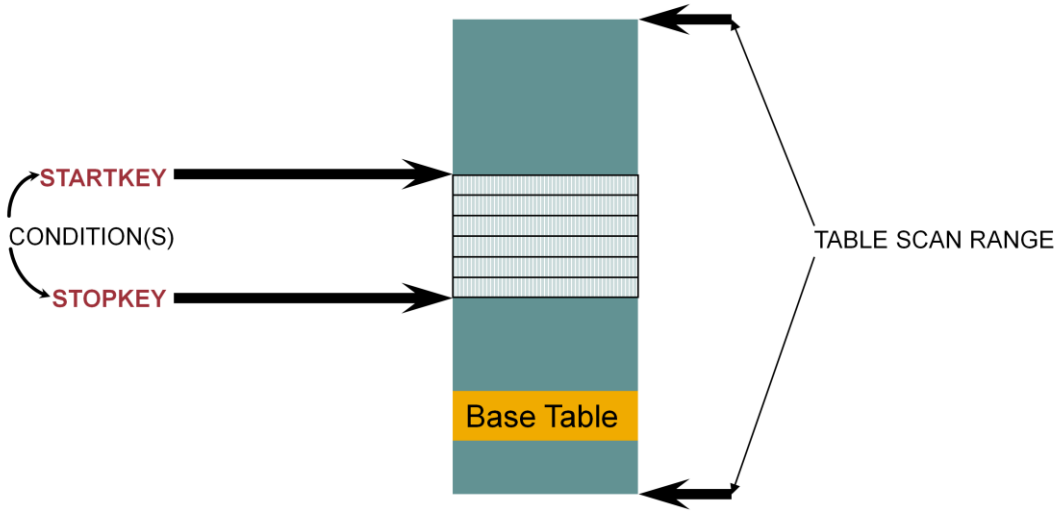
```
SELECT * FROM zztele
  WHERE Name     = 'Aaron'
  AND   Vorname  = 'Anton'
  AND   Str      = 'Alt Moabit'
```

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

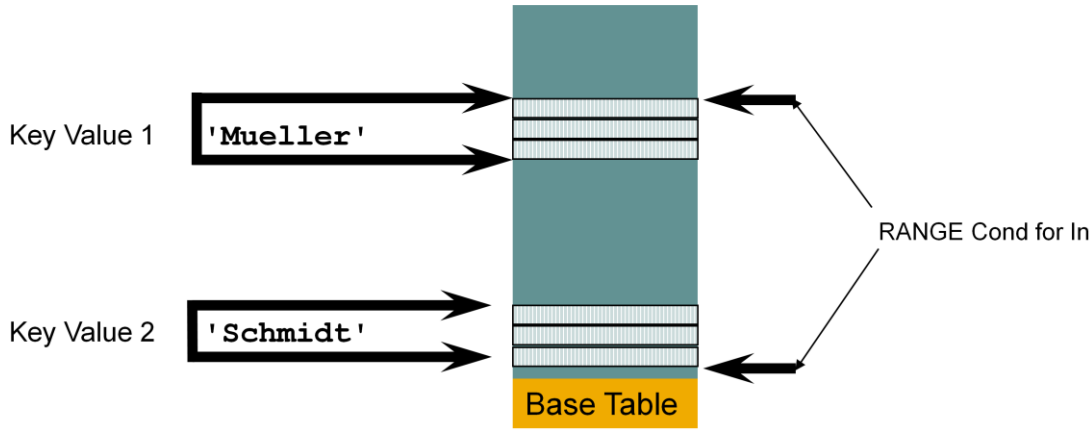
```
SELECT * FROM zztele
```

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.

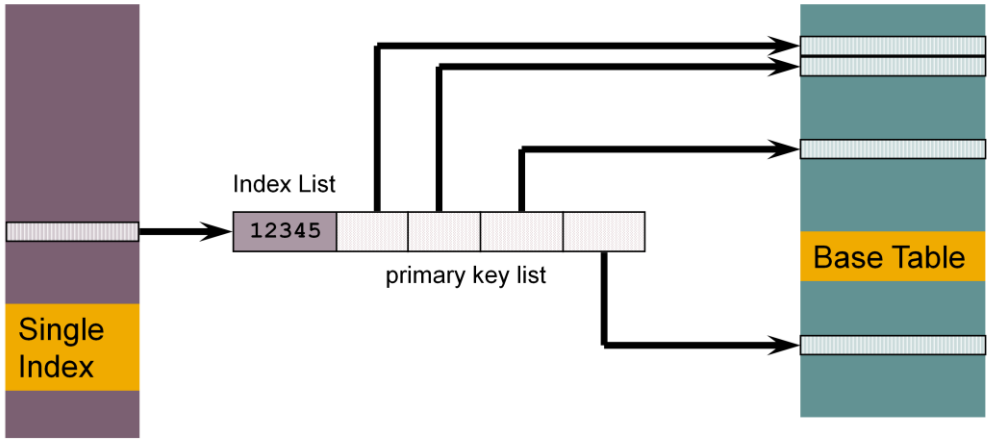
No intermediate result is created for RANGE CONDITIONs.

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'
```

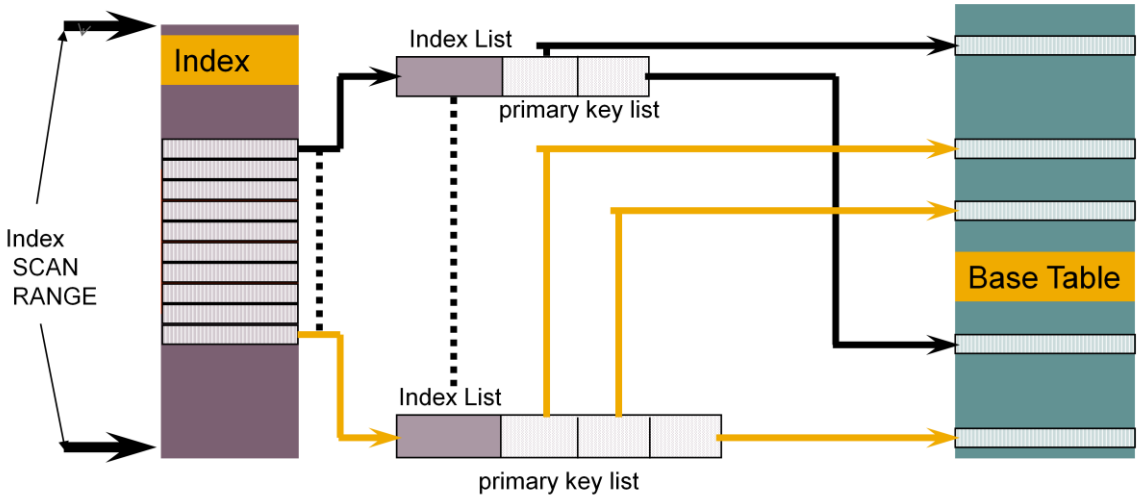
SCHEMANAME	TABLENAME	COLUMN_OR_INDEX	STRATEGY	PAGECOUNT
SAPWB5	ZZTELE	ZZTELE~3	EQUAL CONDITION FOR INDEX	2105
		PLZ	(USED INDEX COLUMN)	
	JDBC_CURSOR_55		RESULT IS NOT COPIED , COSTVALUE IS	2
	JDBC_CURSOR_55		QUERYREWRITE : APPLIED RULES:	
	JDBC_CURSOR_55		DistinctPullUp	1

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'
```

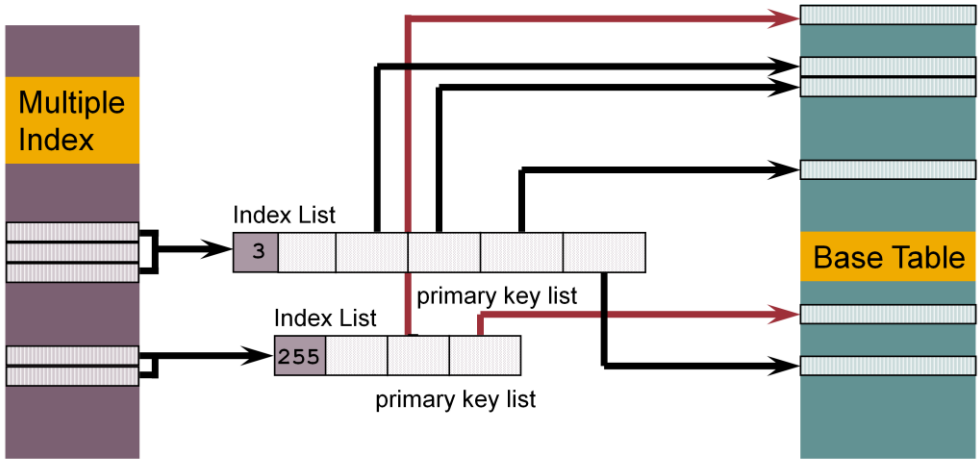
SCHEMANAME	TABLENAME	COLUMN_OR_INDEX	STRATEGY	PAGECOUNT
SAPWB5	ZZTELE	ZZTELE~3	RANGE CONDITION FOR INDEX	2105
		PLZ	(USED INDEX COLUMN)	
	JDBC_CURSOR_72		RESULT IS NOT COPIED , COSTVALUE IS	245
	JDBC_CURSOR_72		QUERYREWRITE : APPLIED RULES:	
	JDBC_CURSOR_72		DistinctPullUp	1

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:

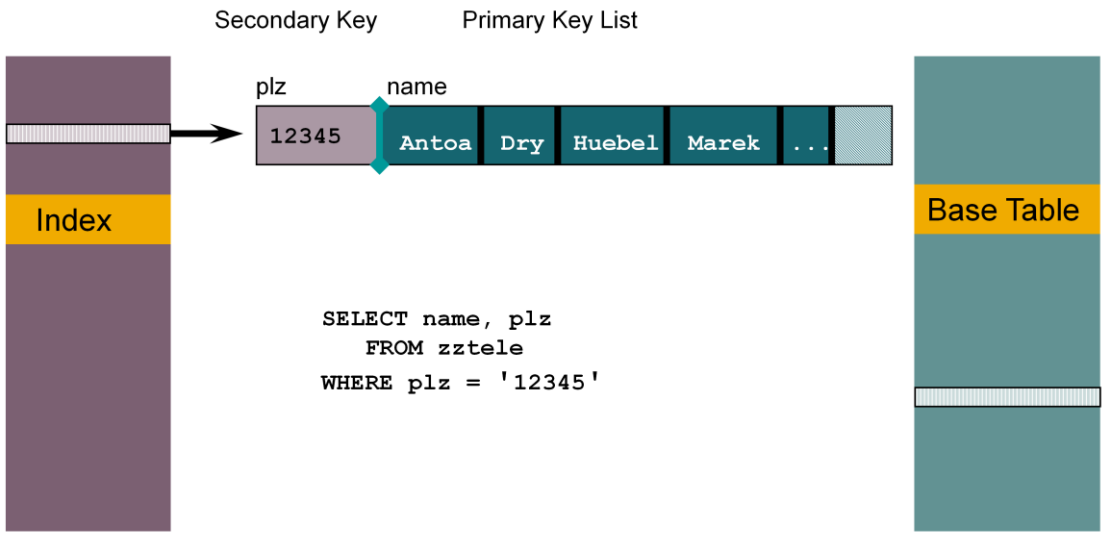
Index only strategy can be used if a SELECT statement only addresses columns that are also contained in an index (SELECT list and WHERE condition)

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.

ONLY INDEX ACCESSED



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

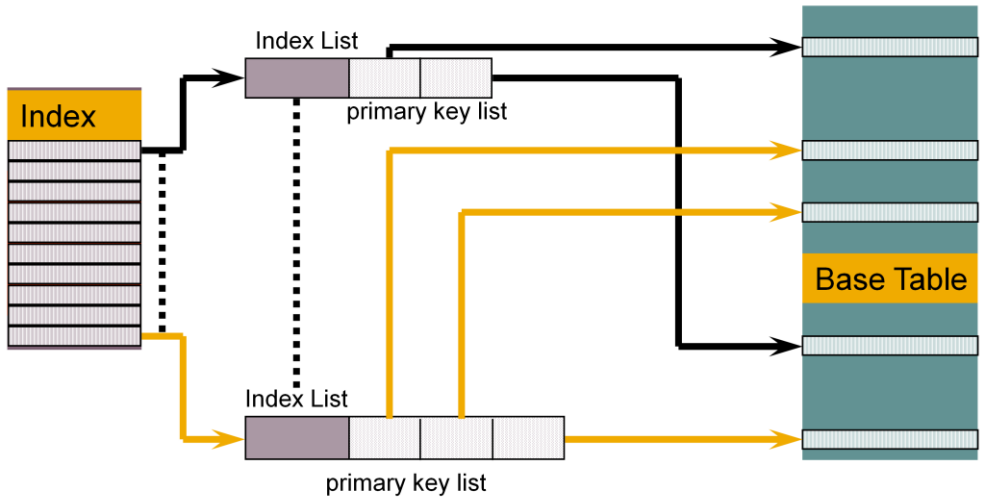
```
SELECT name, plz
   FROM zztele
WHERE plz = '12345'
```

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
```

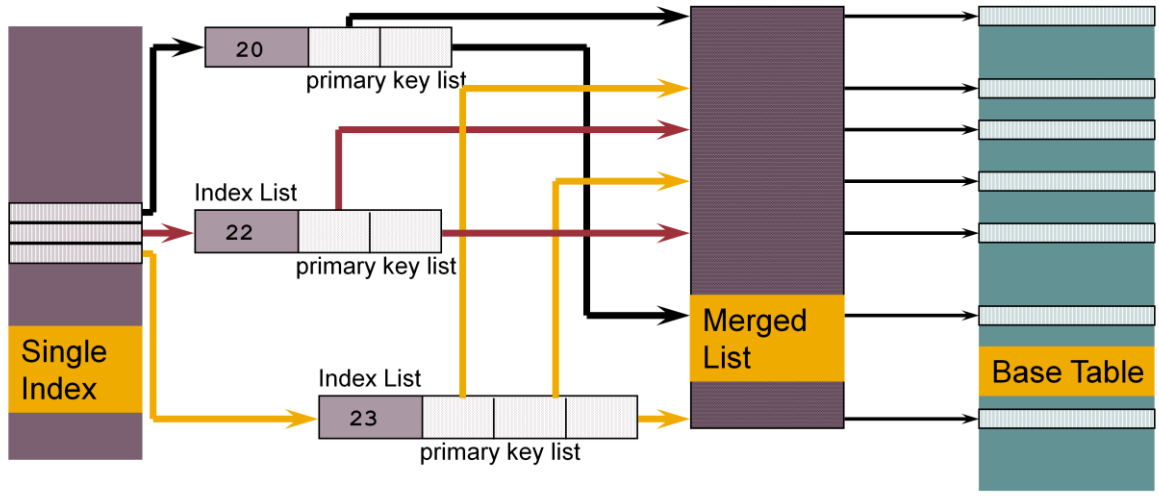
TABLENAME	COLUMN_OR_INDEX	STRATEGY	PAGECOUNT
ZZTELE	ZZTELE~2	INDEX SCAN	2048
	NAME	(USED KEY COLUMN)	
JDBC_CURSOR_40		RESULT IS NOT COPIED , COSTVALUE IS	5260
JDBC_CURSOR_40		QUERYREWRITE : APPLIED RULES:	
JDBC_CURSOR_40		DistinctPullUp	1

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)

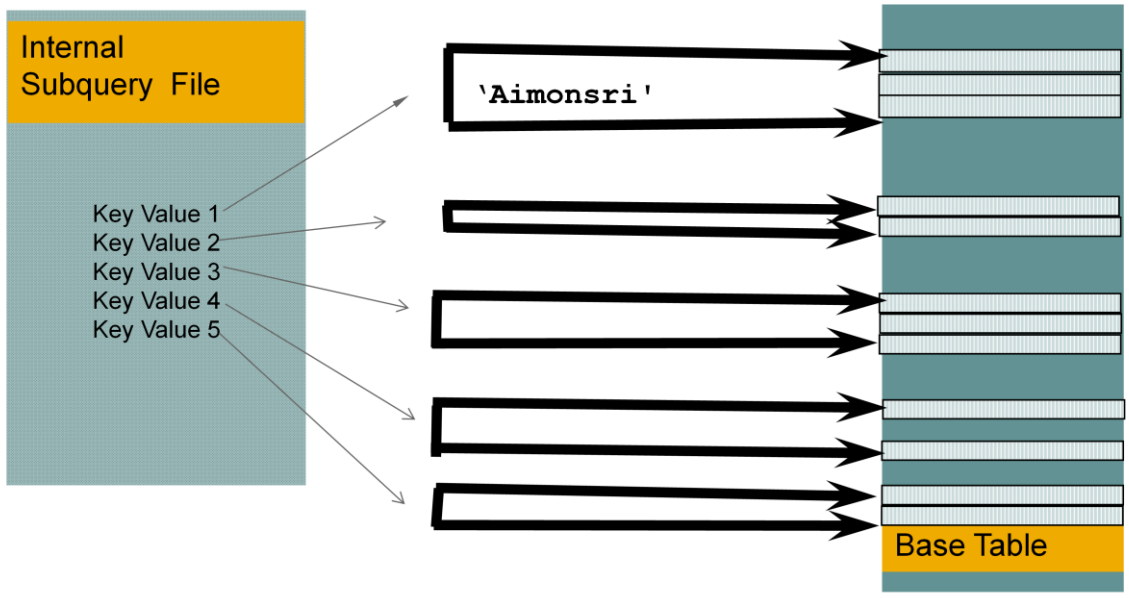
Strategy will be determined first during execution of the command

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)

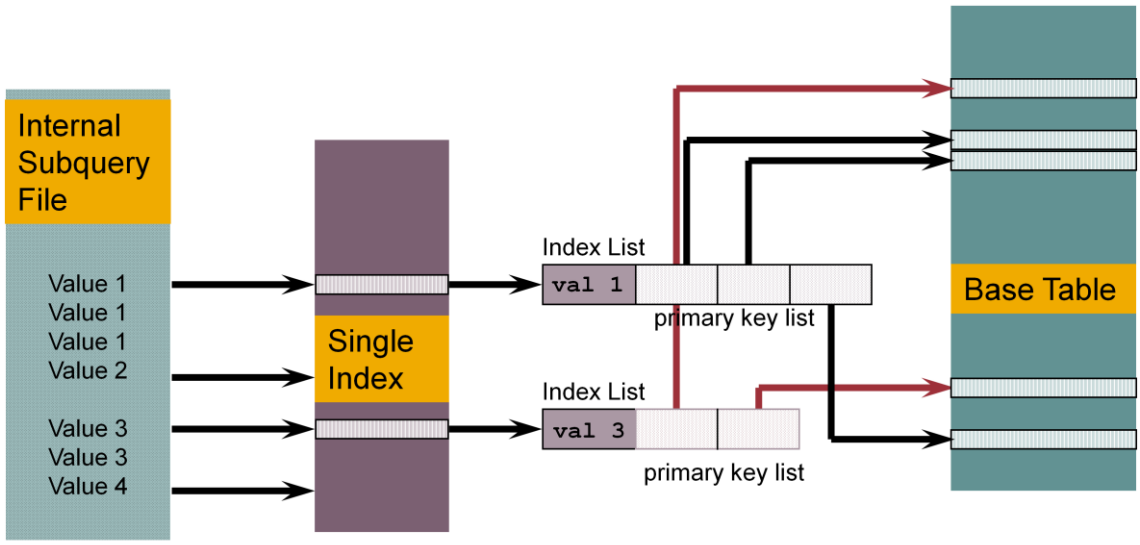
EXPLAIN SELECT * FROM zztele
WHERE **name IN**
(**SELECT name FROM zzmaster**
WHERE year = 2000)

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.

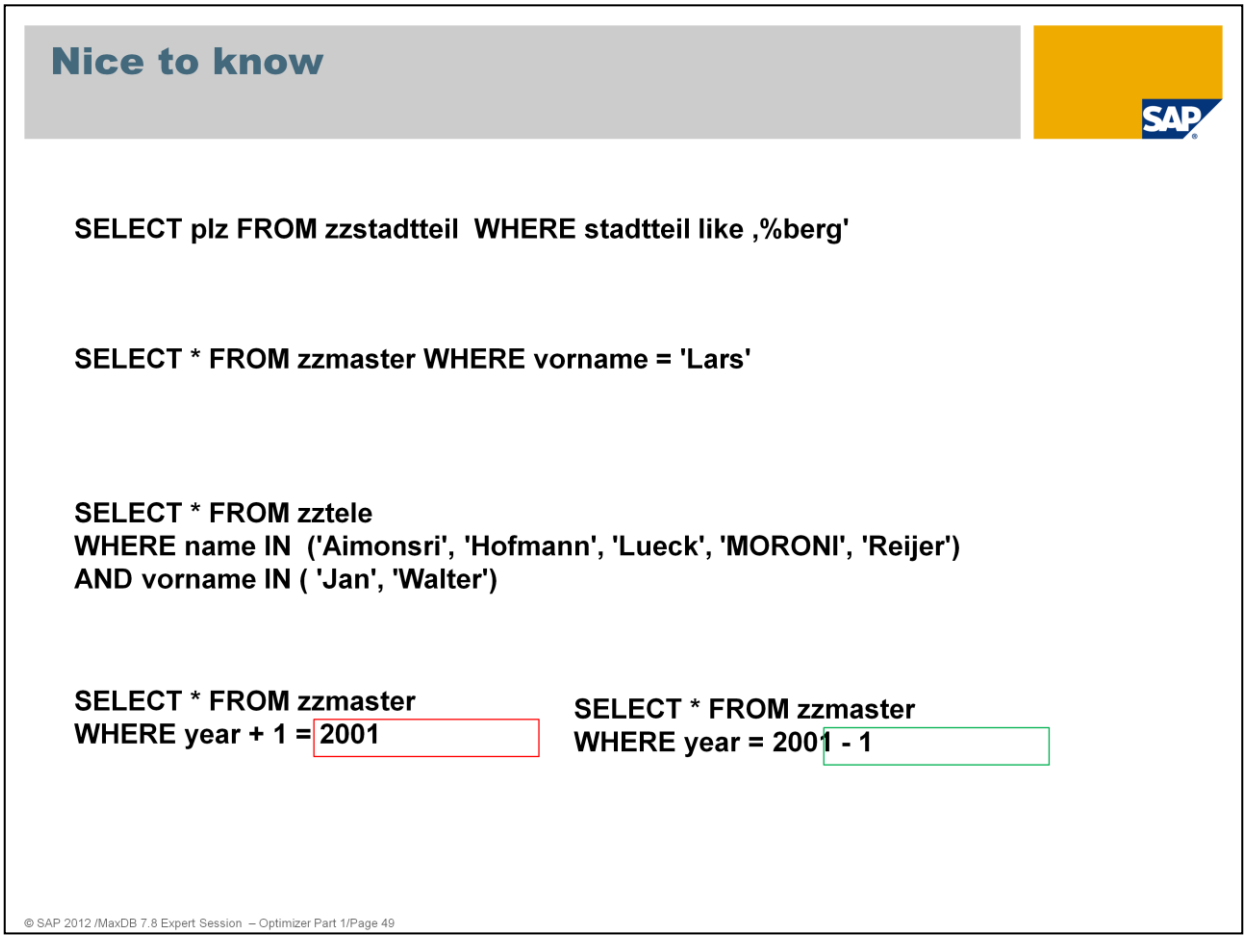
An intermediate result set is generated.

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.

Questions and Answers

Thank You!
Bye, Bye – And Remember Next Session

Feedback and further information:
http://www.sdn.sap.com/irj/sdn/maxdb

Next Session: 29.08.2012
SAP® MaxDB SQL Query Optimization - Part 2

Disclaimer

This presentation reflects current planning.

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