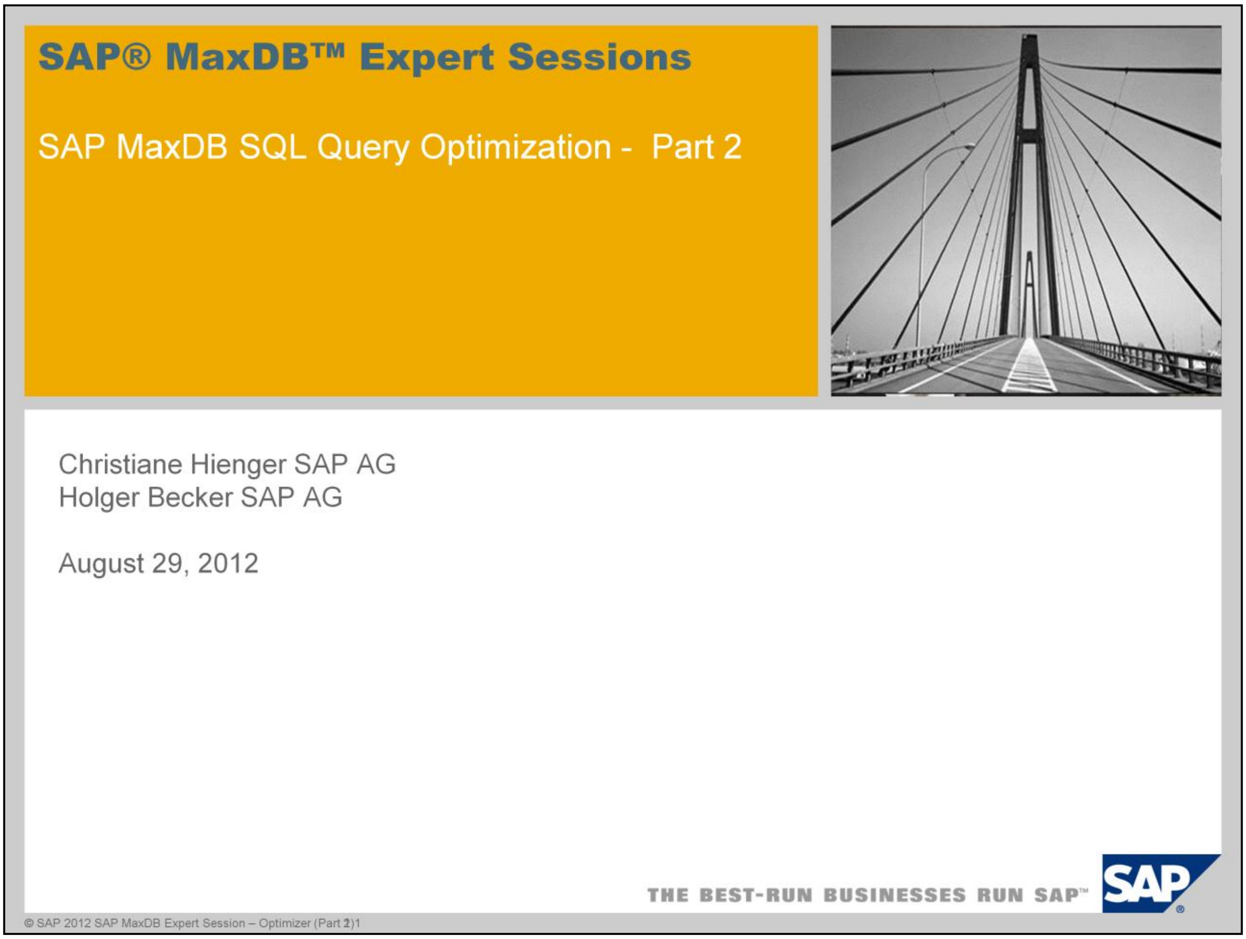
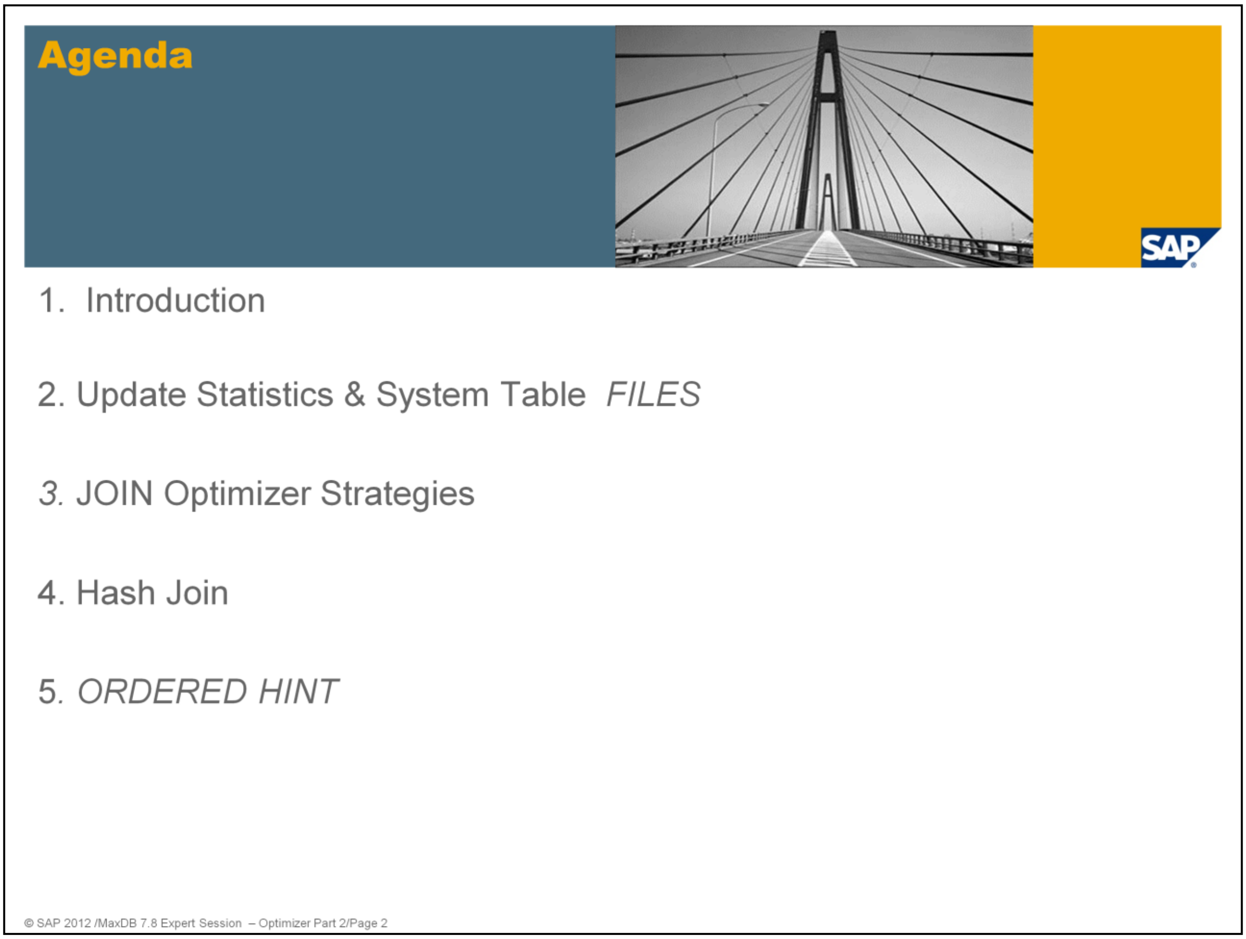


SAP® MaxDB™ Expert Sessions

SAP MaxDB SQL Query Optimization - Part 2

Christiane Hienger SAP AG
Holger Becker SAP AG

August 29, 2012



Agenda

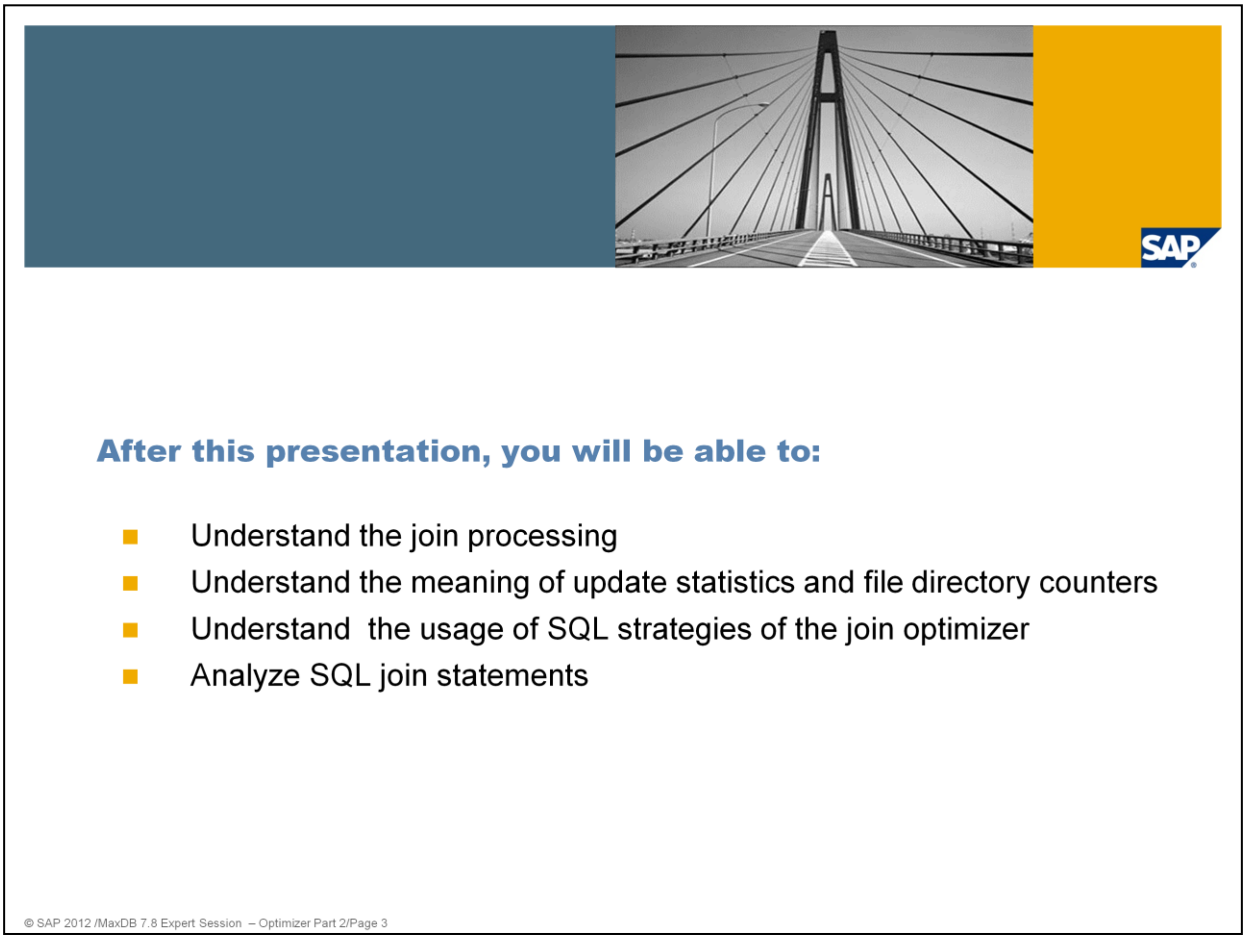
1. Introduction

2. Update Statistics & System Table *FILES*

3. JOIN Optimizer Strategies

4. Hash Join

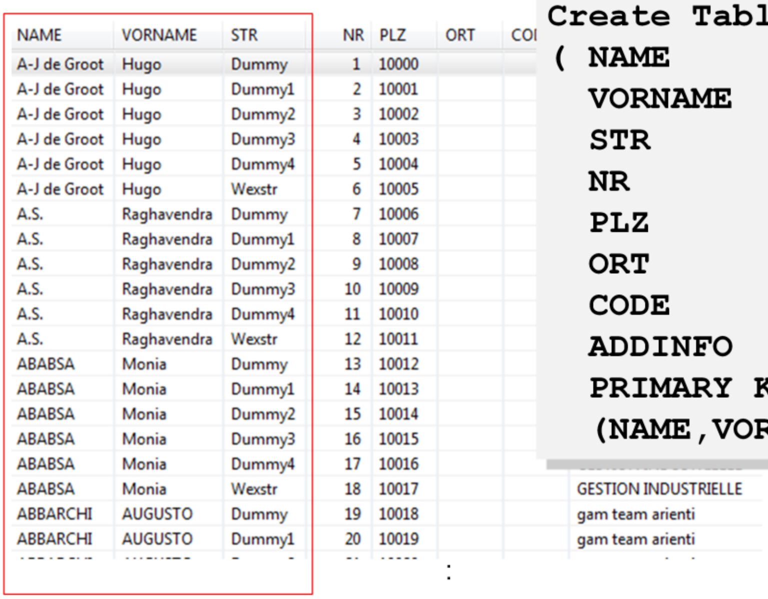
5. *ORDERED HINT*



After this presentation, you will be able to:

- Understand the join processing
- Understand the meaning of update statistics and file directory counters
- Understand the usage of SQL strategies of the join optimizer
- Analyze SQL join statements

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, vorname, str

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

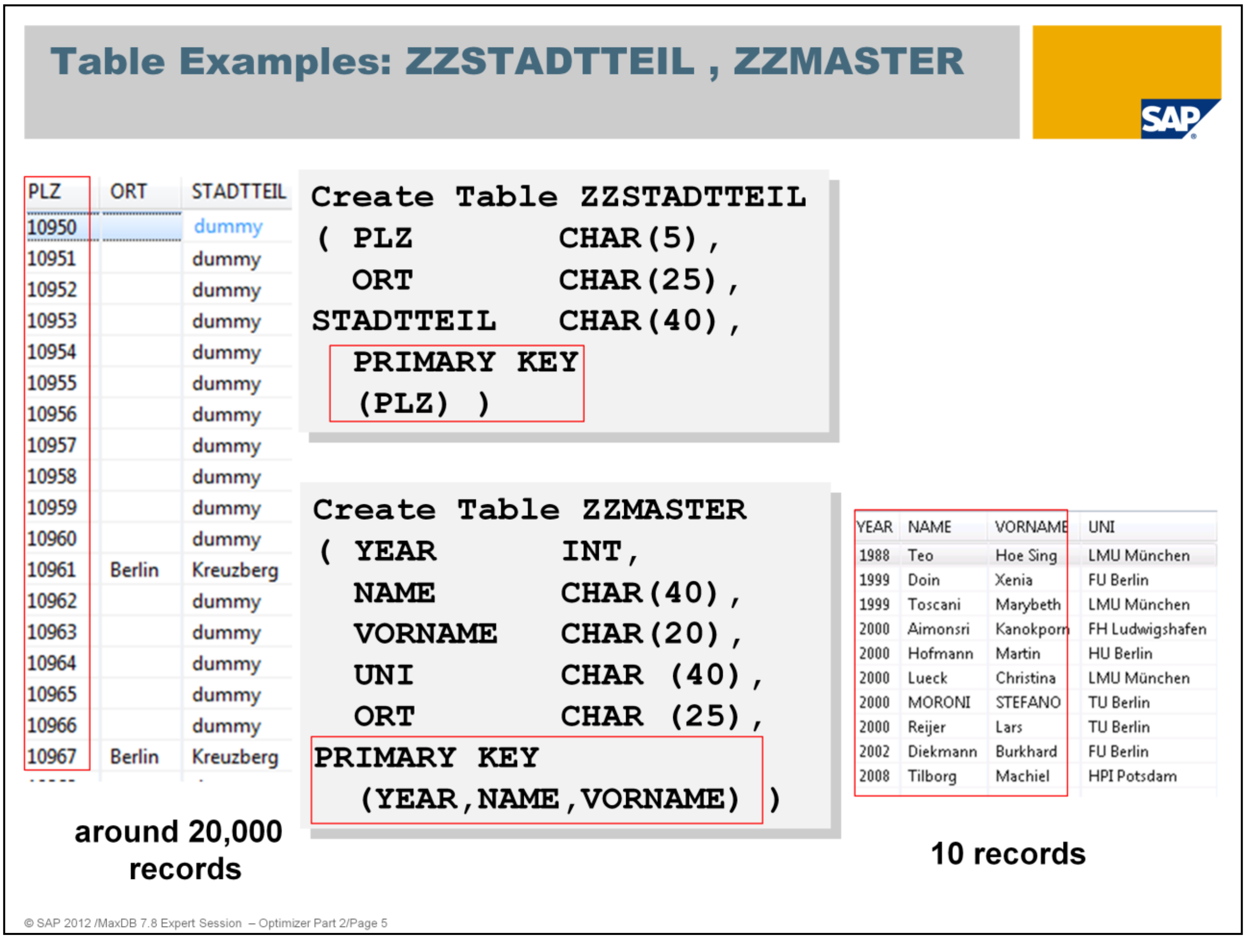


Table Examples: ZZSTADTTEIL , ZZMASTER

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

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

around 20,000 records

```
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

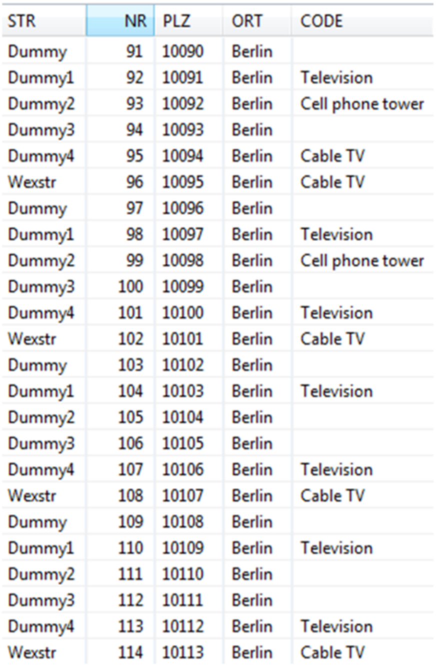
To explain strategies which can be used for joins, the examples also refer to the table ZZSTADTTEIL with approx. 20000 records and table ZZMASTER.

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

The table is sorted via zip code.

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

Table Examples: ZZCODE

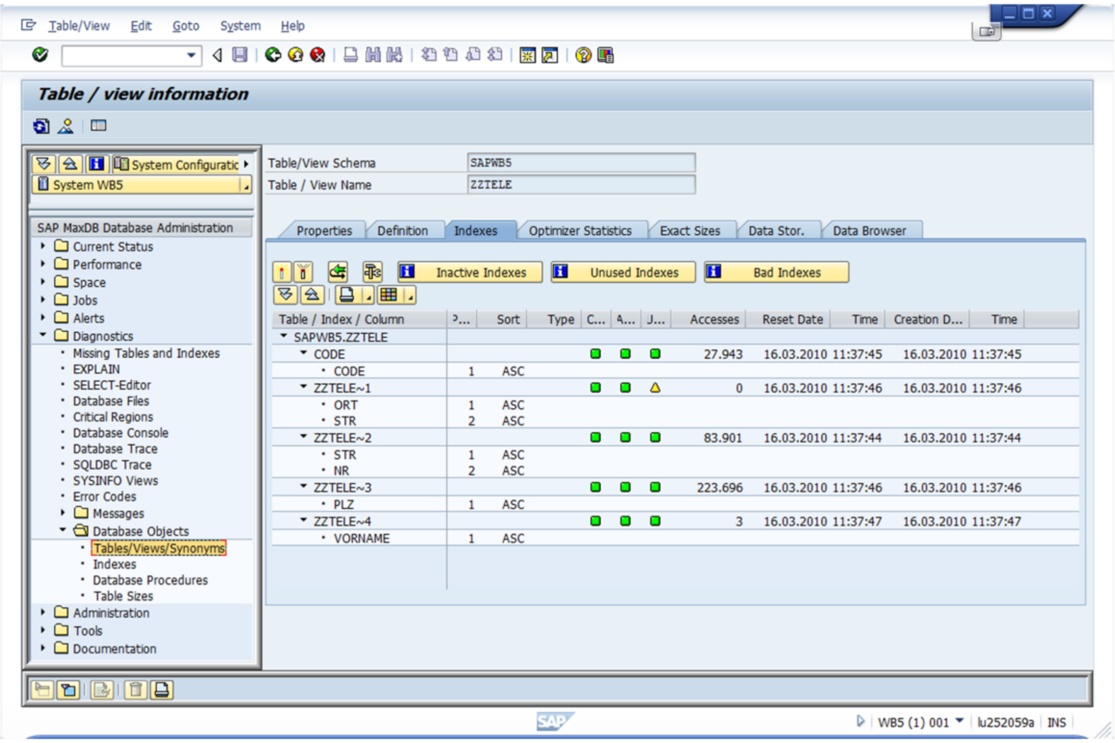


STR	NR	PLZ	ORT	CODE
Dummy	91	10090	Berlin	
Dummy1	92	10091	Berlin	Television
Dummy2	93	10092	Berlin	Cell phone tower
Dummy3	94	10093	Berlin	
Dummy4	95	10094	Berlin	Cable TV
Wexstr	96	10095	Berlin	Cable TV
Dummy	97	10096	Berlin	
Dummy1	98	10097	Berlin	Television
Dummy2	99	10098	Berlin	Cell phone tower
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



Table/View Schema: SAPWB5

Table / View Name: ZZTELE

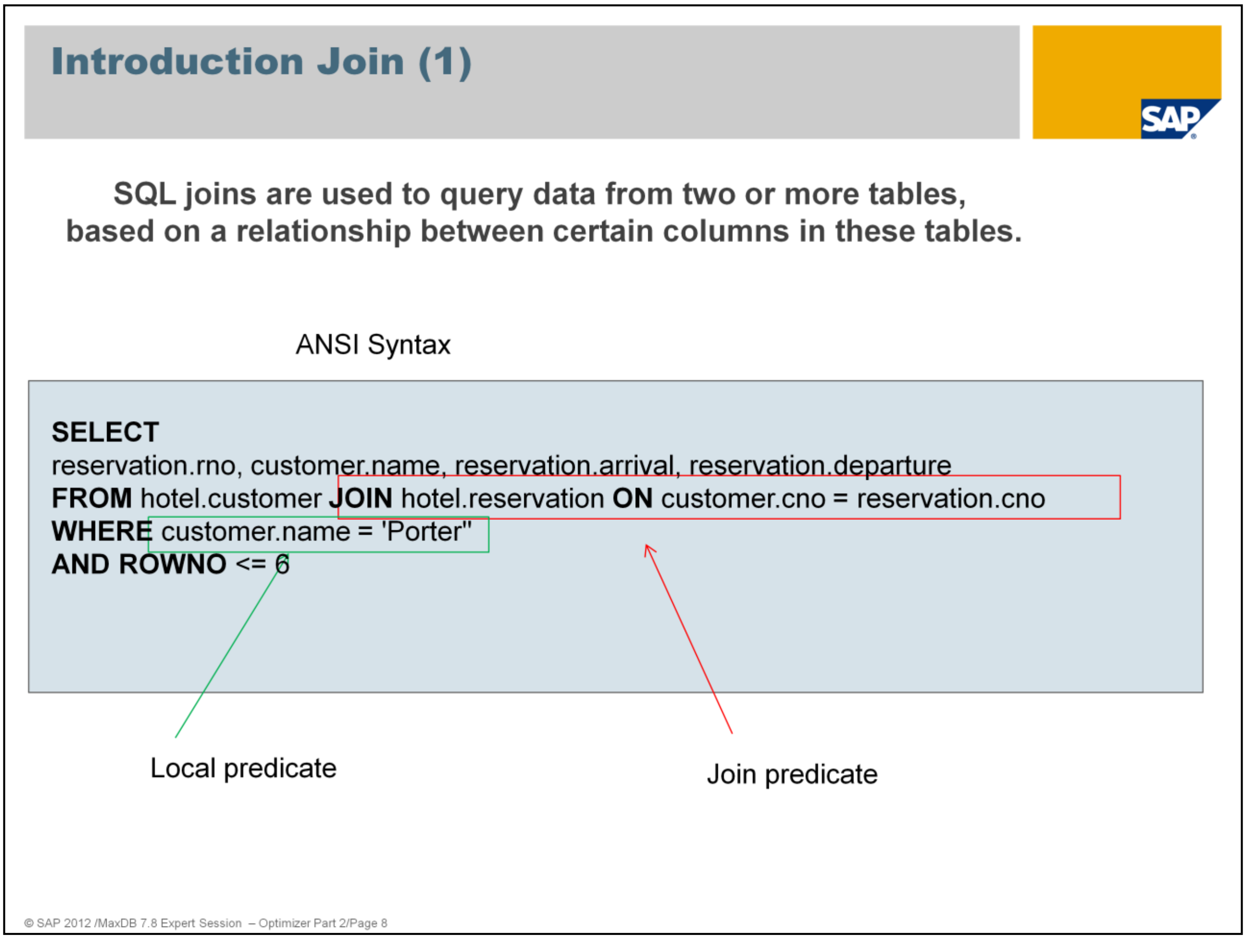
Table / Index / Column	P...	Sort	Type	C...	A...	J...	Accesses	Reset Date	Time	Creation D...	Time
SAPWB5.ZZTELE											
CODE				🟩	🟩	🟩	27.943	16.03.2010	11:37:45	16.03.2010	11:37:45
• CODE	1	ASC									
ZZTELE~1				🟩	🟩	△	0	16.03.2010	11:37:46	16.03.2010	11:37:46
• ORT	1	ASC									
• STR	2	ASC									
ZZTELE~2				🟩	🟩	🟩	83.901	16.03.2010	11:37:44	16.03.2010	11:37:44
• STR	1	ASC									
• NR	2	ASC									
ZZTELE~3				🟩	🟩	🟩	223.696	16.03.2010	11:37:46	16.03.2010	11:37:46
• PLZ	1	ASC									
ZZTELE~4				🟩	🟩	🟩	3	16.03.2010	11:37:47	16.03.2010	11:37:47
• VORNAME	1	ASC									

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



Introduction Join (1)

SQL joins are used to query data from two or more tables, based on a relationship between certain columns in these tables.

ANSI Syntax

```
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
```

Local predicate

Join predicate

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**.

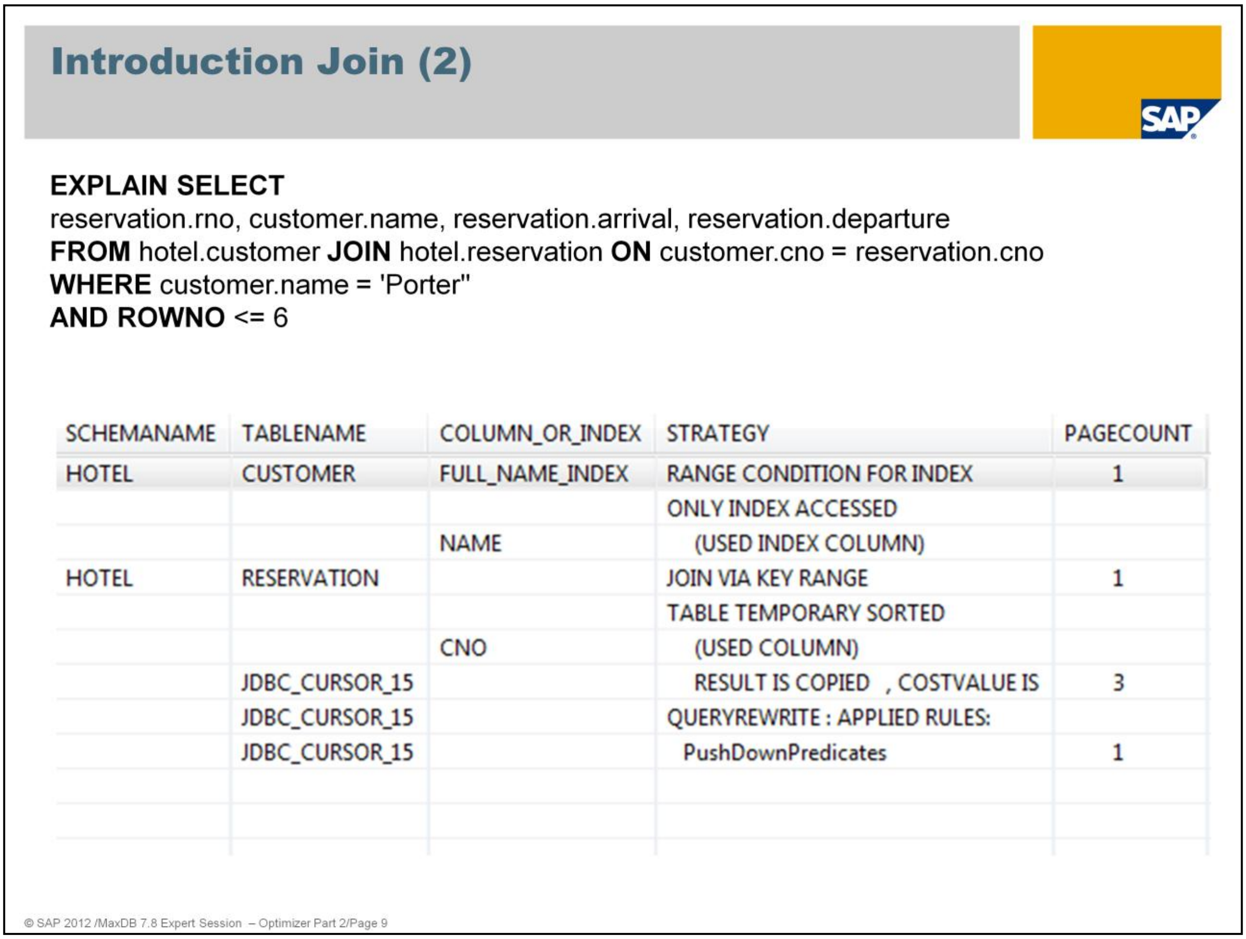
The join predicate is defined in an ON clause and specifies a comparison between two values or lists of values of both tables.

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
```

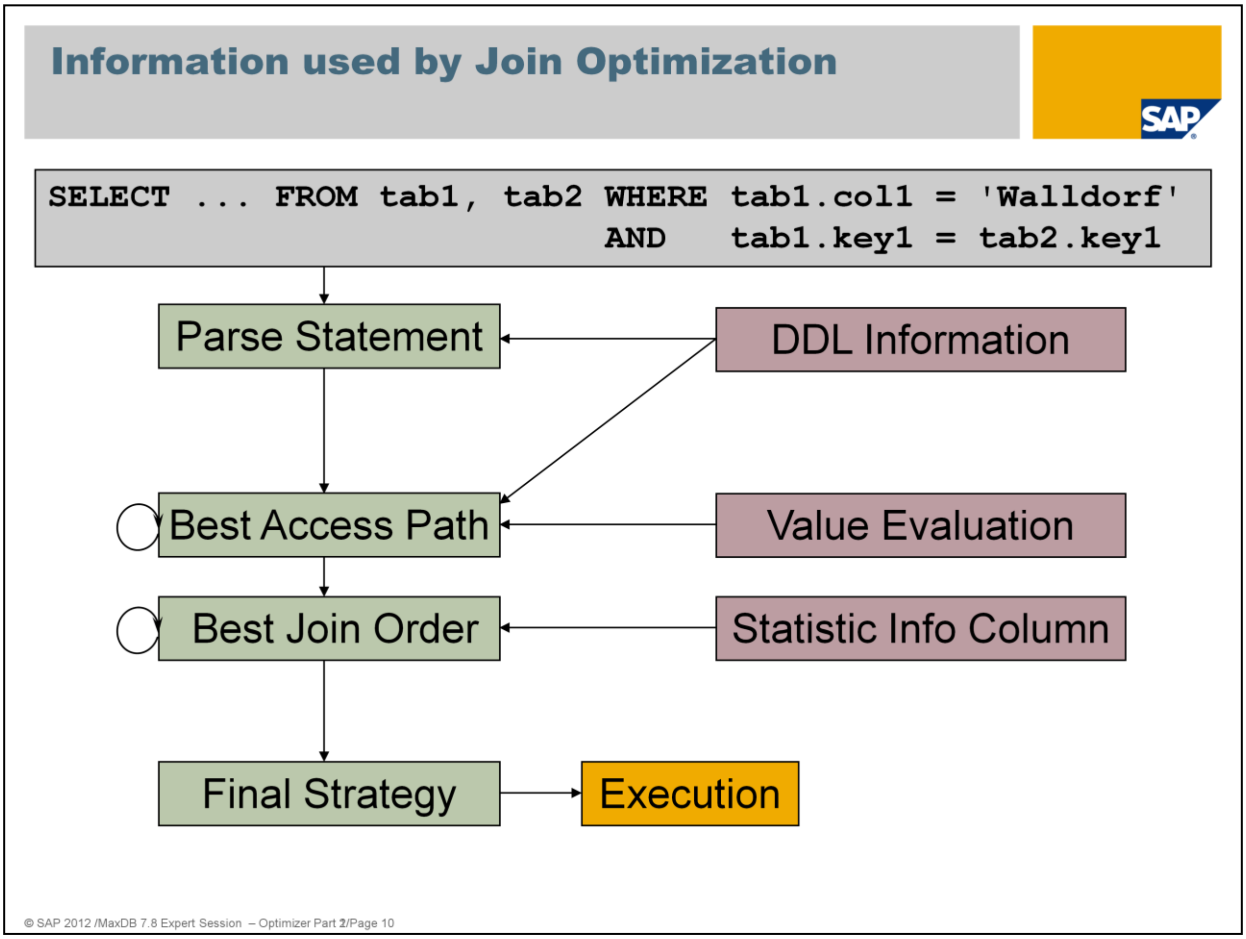
SCHEMANAME	TABLENAME	COLUMN_OR_INDEX	STRATEGY	PAGECOUNT
HOTEL	CUSTOMER	FULL_NAME_INDEX	RANGE CONDITION FOR INDEX	1
			ONLY INDEX ACCESSED	
		NAME	(USED INDEX COLUMN)	
HOTEL	RESERVATION		JOIN VIA KEY RANGE	1
			TABLE TEMPORARY SORTED	
		CNO	(USED COLUMN)	
	JDBC_CURSOR_15		RESULT IS COPIED , COSTVALUE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

© SAP 2012 /MaxDB 7.8 Expert Session – Optimizer Part 2/Page 9

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.



Information used by Join Optimization

```
SELECT ... FROM tab1, tab2 WHERE tab1.col1 = 'Walldorf'
                             AND   tab1.key1 = tab2.key1
```

Parse Statement

DDL Information

Best Access Path

Value Evaluation

Best Join Order

Statistic Info Column

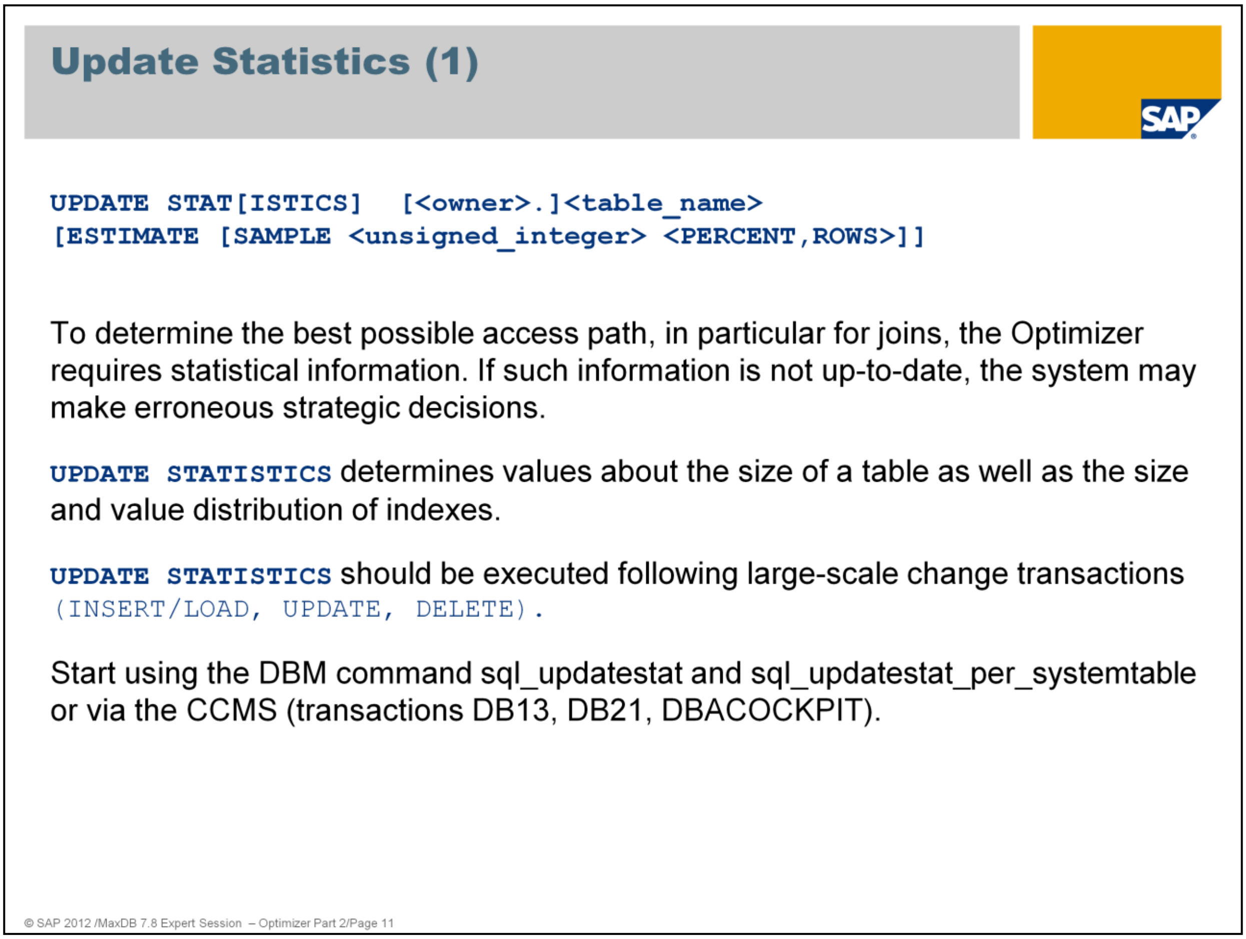
Final Strategy

Execution

© SAP 2012 /MaxDB 7.8 Expert Session – Optimizer Part 2/Page 10

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.

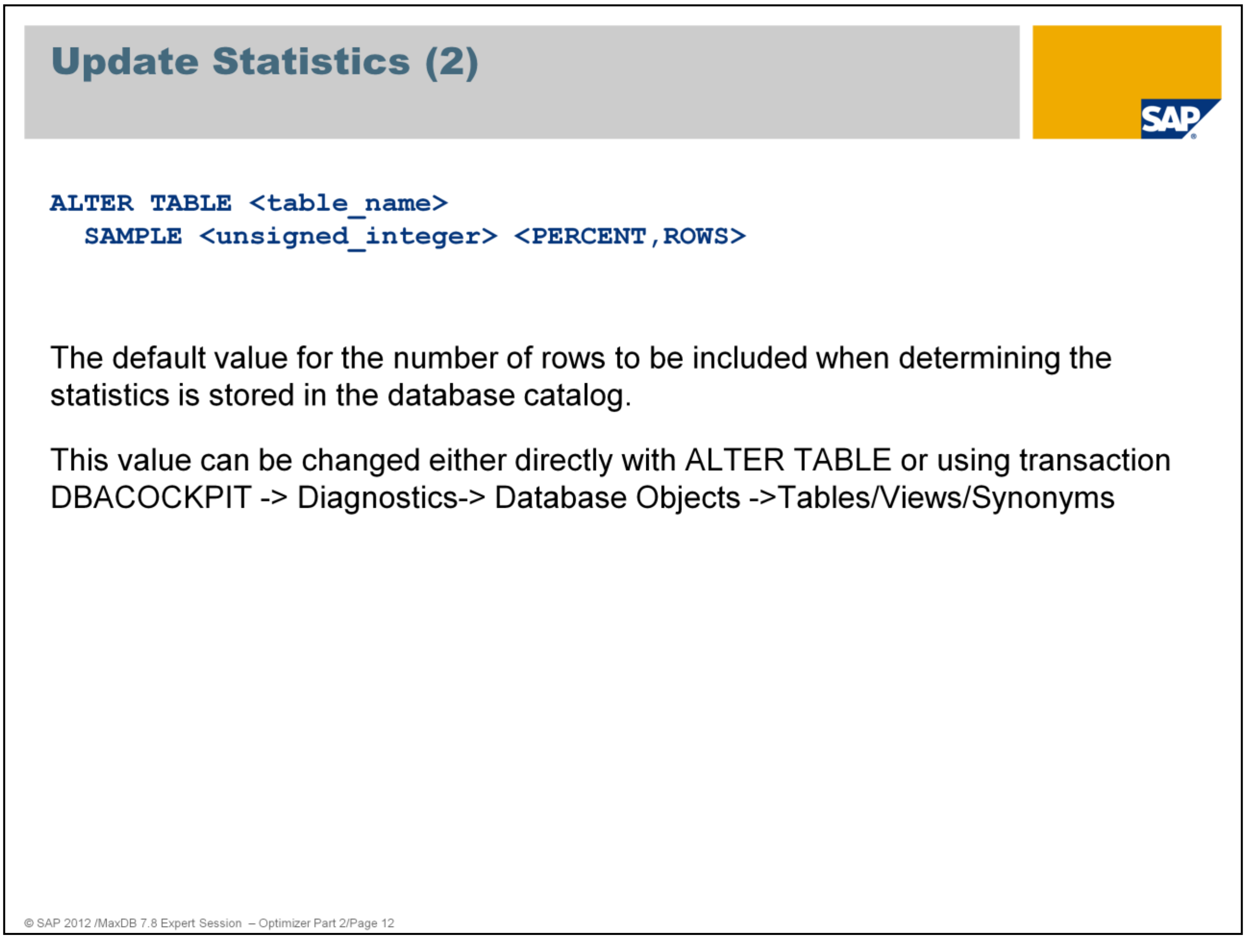
The DBM command sql_updatestat executes Update Statistics for all tables in the database.

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.

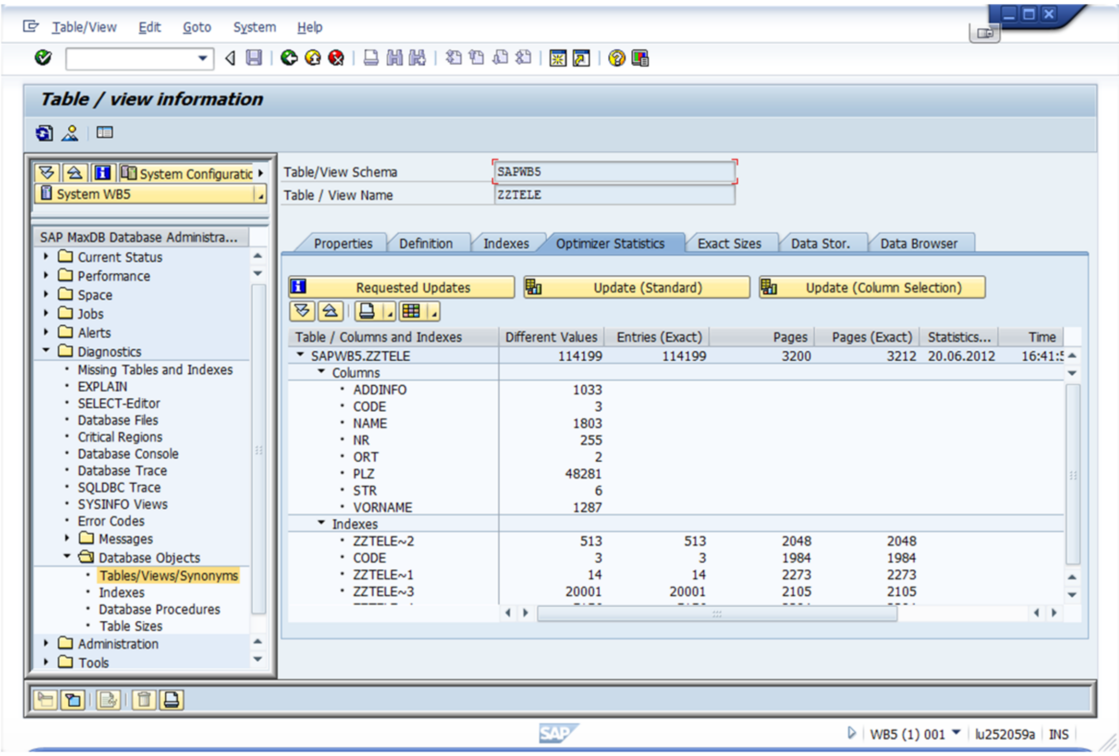
This value can be changed either directly with ALTER TABLE or using transaction DBACOCKPIT -> Diagnostics-> Database Objects ->Tables/Views/Synonyms

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)



Table/View Schema: SAPWB5
Table / View Name: ZZTELE

Table / Columns and Indexes	Different Values	Entries (Exact)	Pages	Pages (Exact)	Statistics...	Time
SAPWB5.ZZTELE	114199	114199	3200	3212	20.06.2012	16:41:5
Columns						
ADDINFO	1033					
CODE	3					
NAME	1803					
NR	255					
ORT	2					
PLZ	48281					
STR	6					
VORNAME	1287					
Indexes						
ZZTELE~2	513	513	2048	2048		
CODE	3	3	1984	1984		
ZZTELE~1	14	14	2273	2273		
ZZTELE~3	20001	20001	2105	2105		

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.

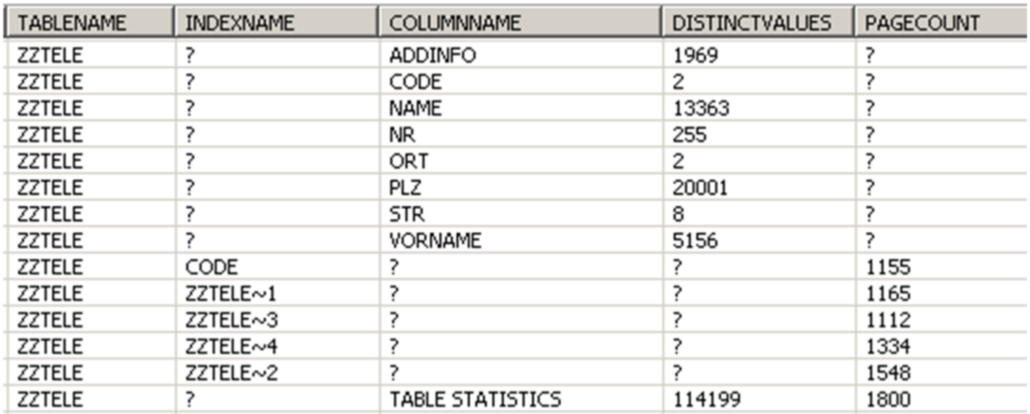
You can use *Update (Column Statistics)* to create column statistics for specified columns.

In the *Optimizer Statistics* view the column and table statistics are listed.

Update Statistics (4)

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

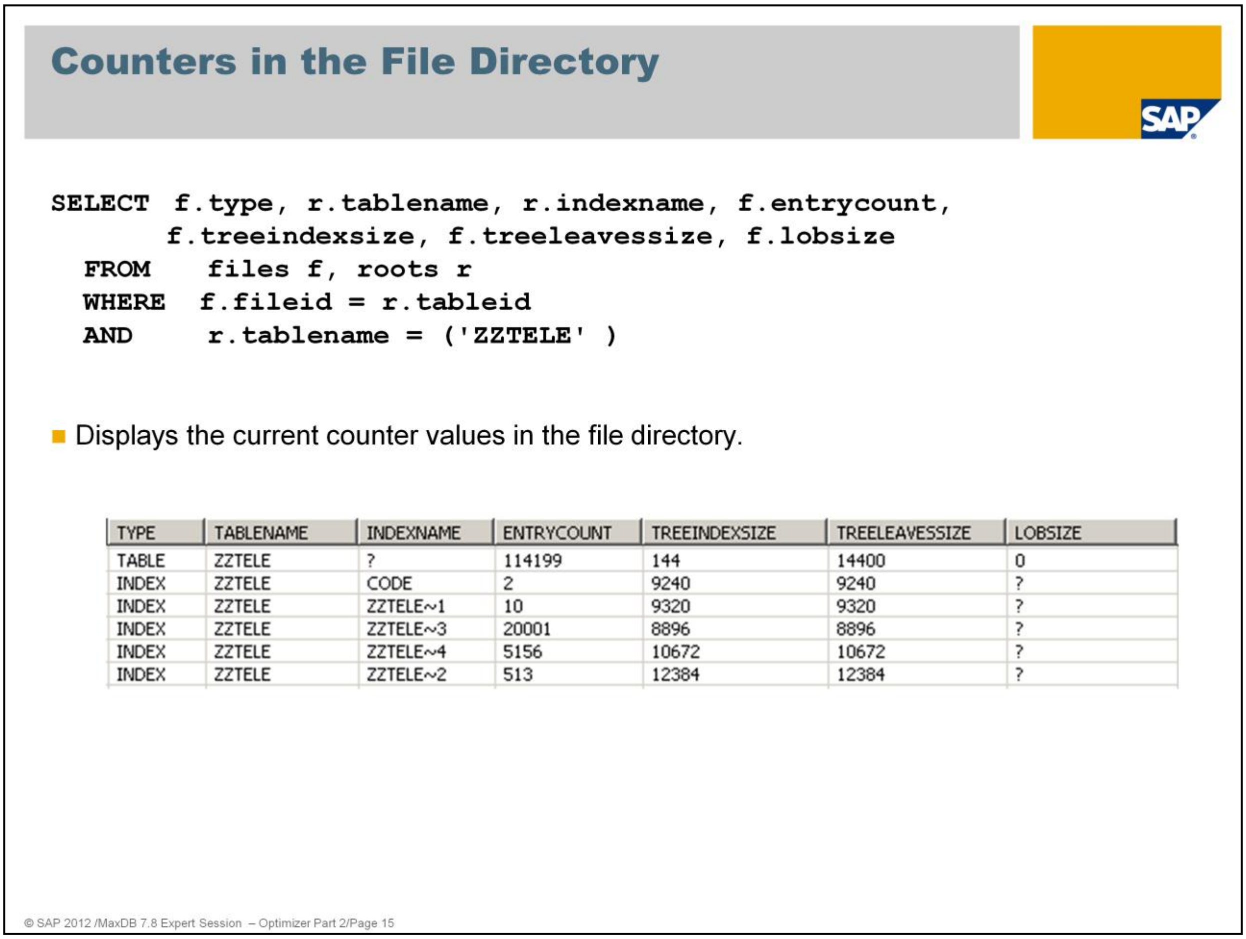
- Shows the current statistic values that will be used by the optimizer to determine the strategy.



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.

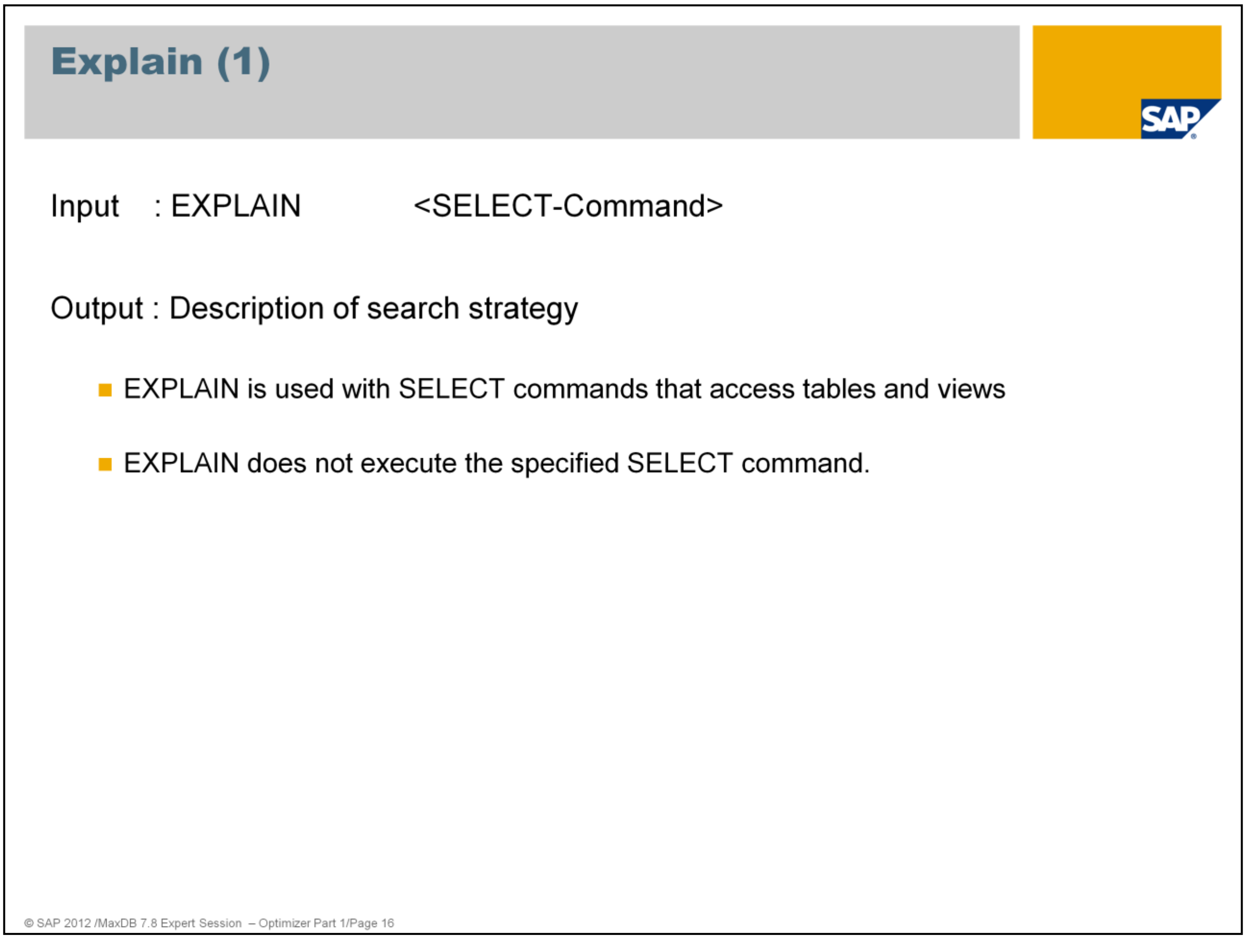
TYPE	TABLENAME	INDEXNAME	ENTRYCOUNT	TREEINDEXSIZE	TREELEAVESSIZE	LOBSIZE
TABLE	ZZTELE	?	114199	144	14400	0
INDEX	ZZTELE	CODE	2	9240	9240	?
INDEX	ZZTELE	ZZTELE~1	10	9320	9320	?
INDEX	ZZTELE	ZZTELE~3	20001	8896	8896	?
INDEX	ZZTELE	ZZTELE~4	5156	10672	10672	?
INDEX	ZZTELE	ZZTELE~2	513	12384	12384	?

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.

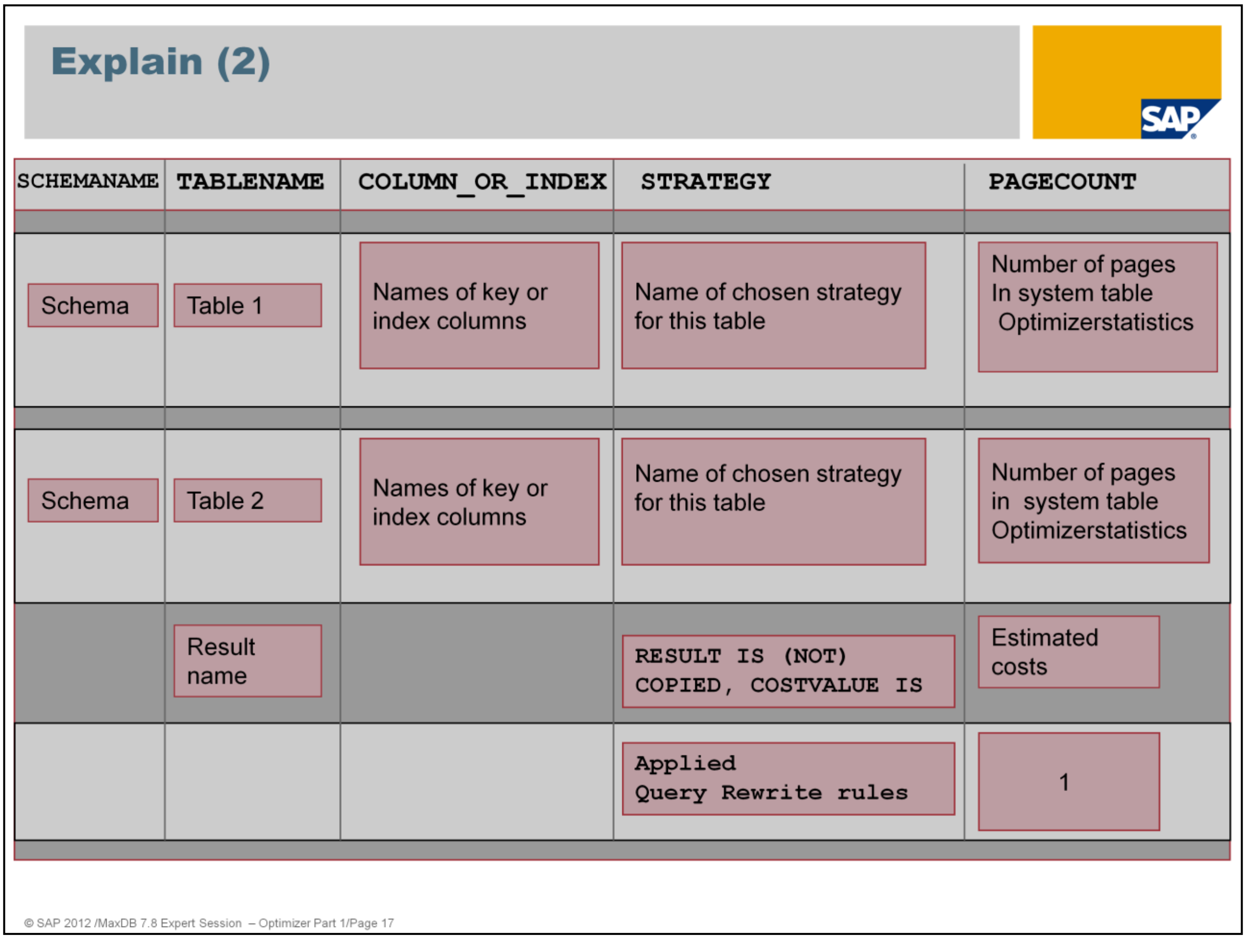
In the ABAP-based SAP application server, EXPLAIN is available in transactions ST05, DB50 and DBACockpit (in the command monitor). The SQL editor of the Database Studio can send an EXPLAIN via context menu (right mouse click) to the database. The output is shown in a separate window.

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



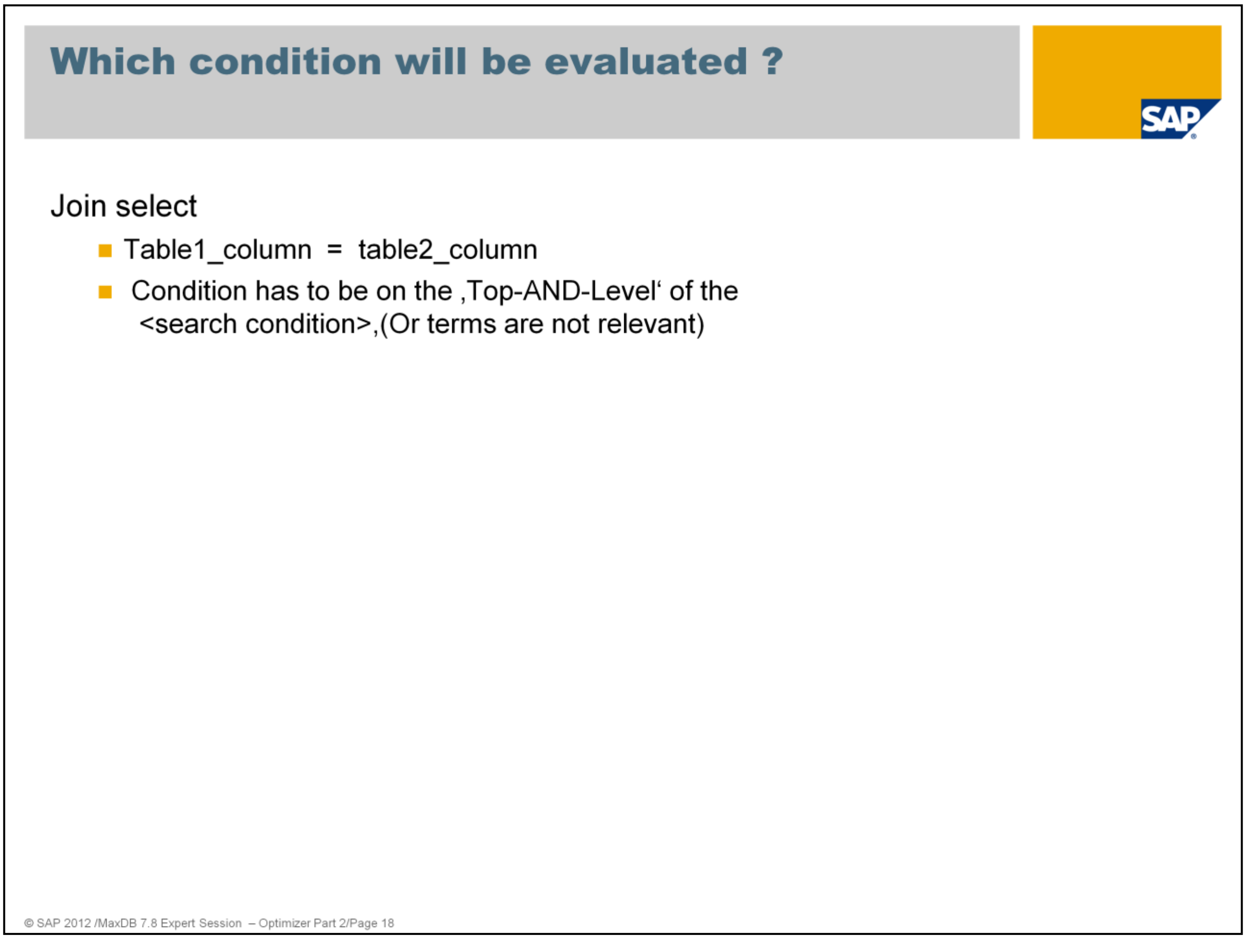
Explain (2)

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

© SAP 2012 /MaxDB 7.8 Expert Session – Optimizer Part 1/Page 17

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

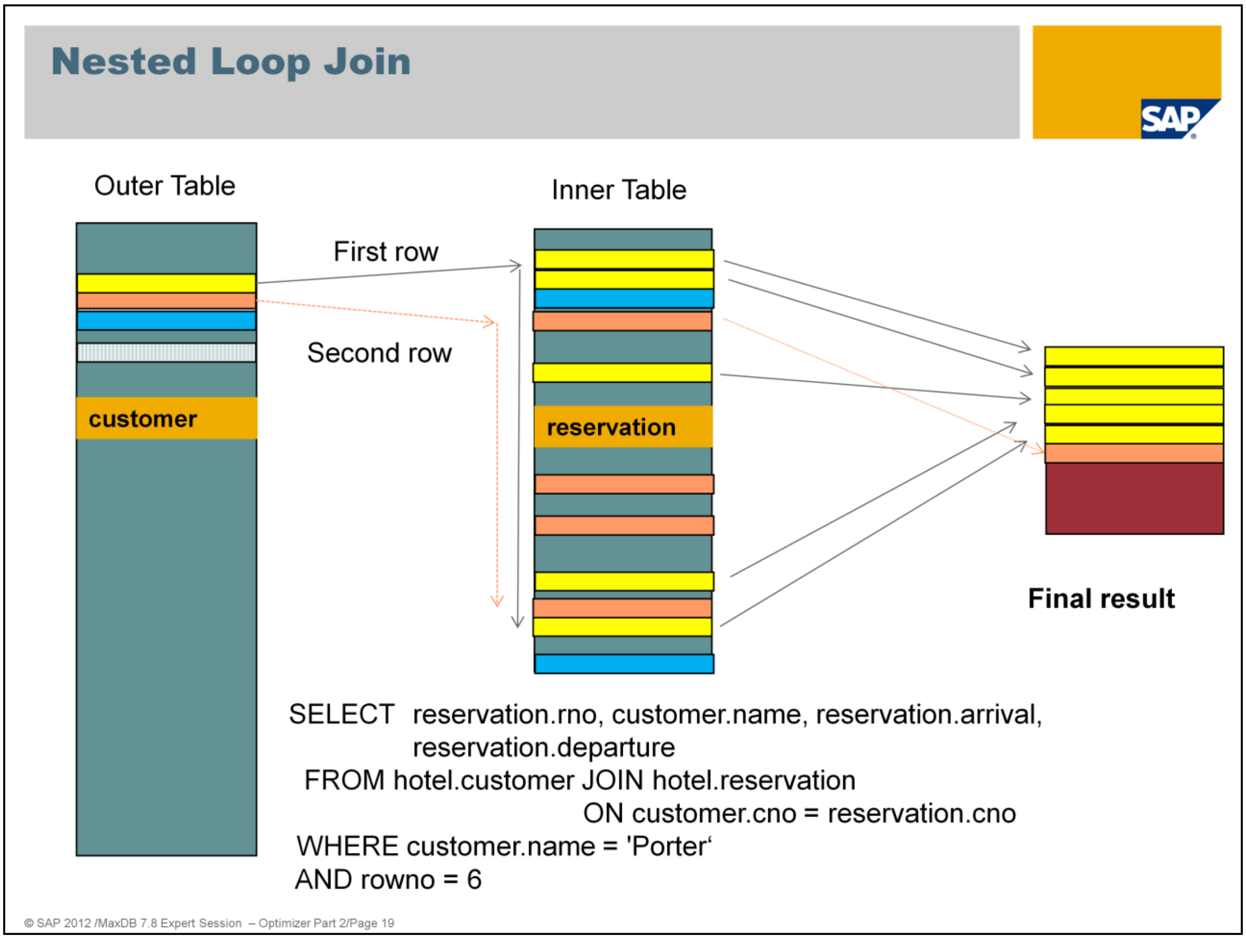
- Table1_column = table2_column
- Condition has to be on the ‚Top-AND-Level‘ of the <search condition>,(Or terms are not relevant)

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.



Nested Loop Join

```
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
```

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

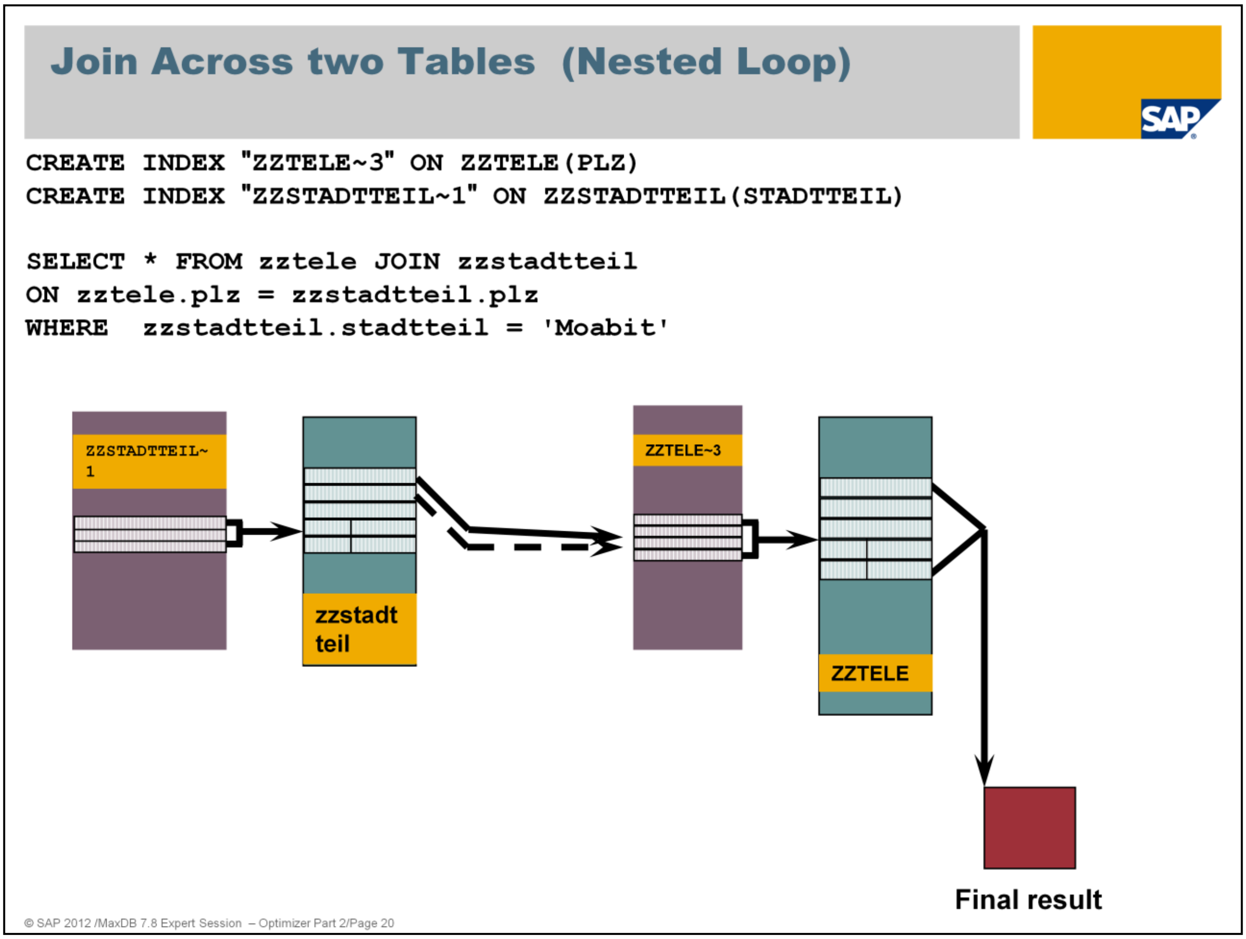
As of version 7.7 there is no more possibility to choose between **Sorted Merge** or **Nested Loop** by a parameter setting (JOIN_OPERATOR_IMPLEMENTATION). There are only marginal disadvantages concerning CPU usage for Nested Loop with the current algorithms. Therewith the Nested Loop can deliver the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

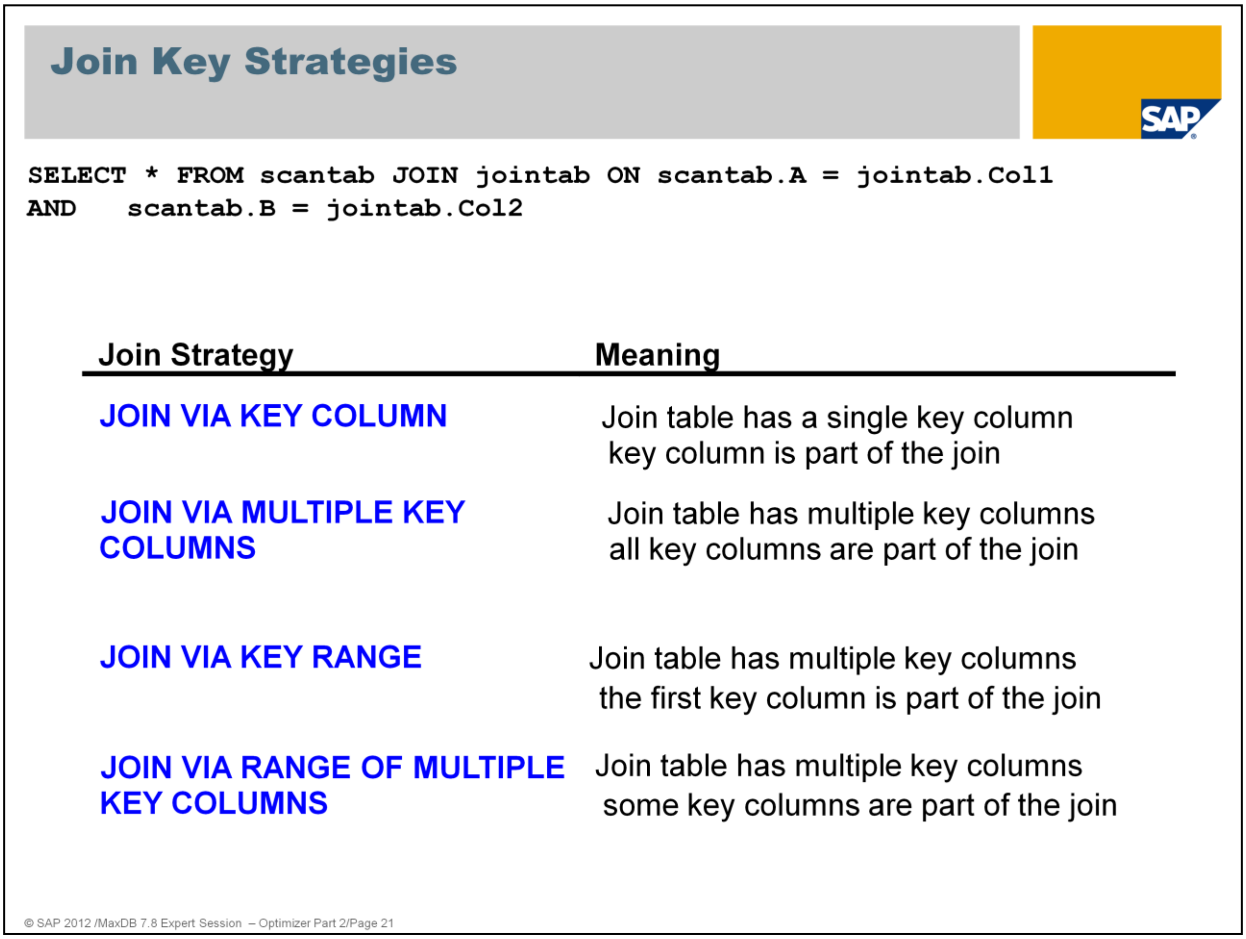


Join Across two Tables (Nested Loop)

```
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

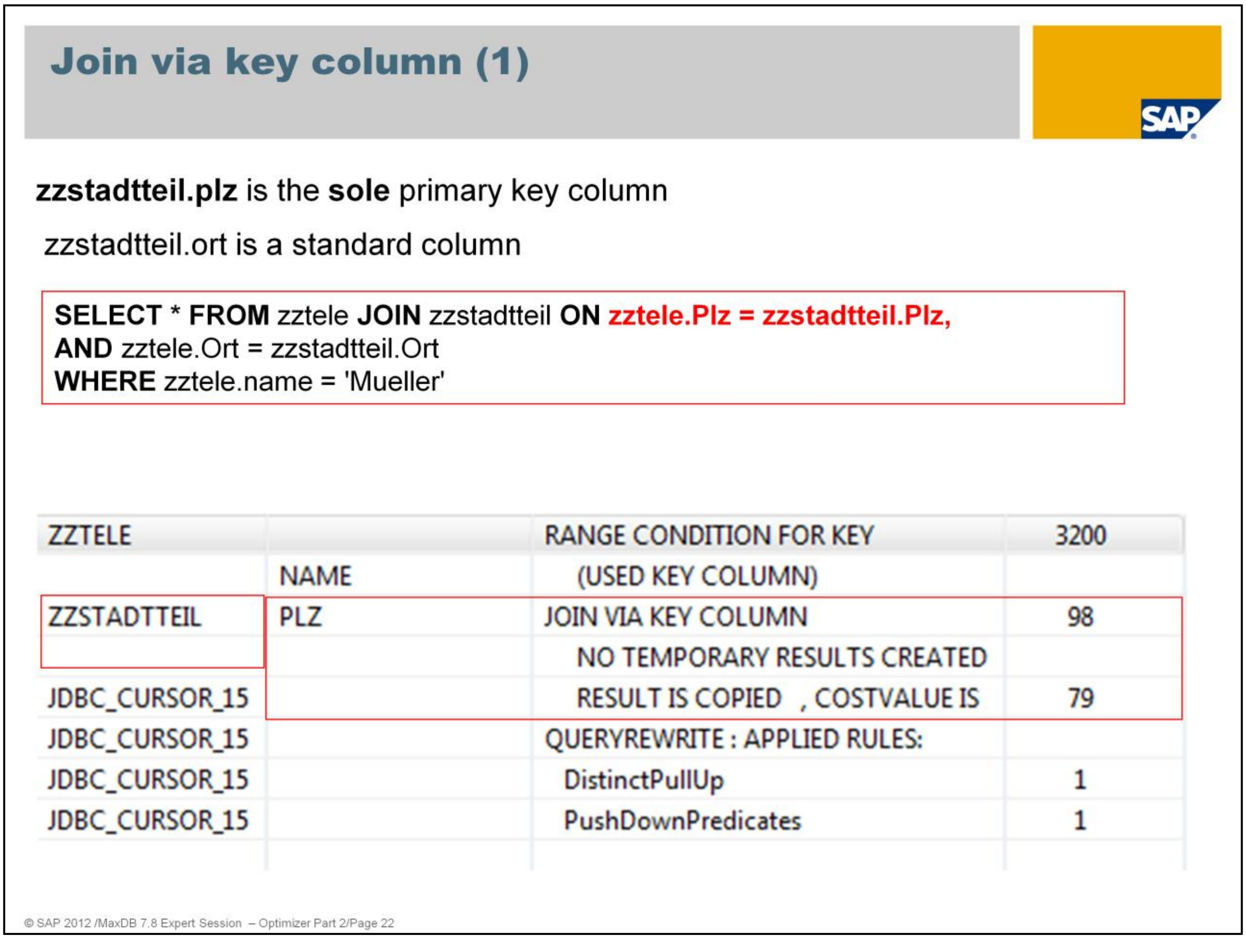
```
SELECT * FROM scantab JOIN jointab ON scantab.A = jointab.Col1
AND   scantab.B = jointab.Col2
```

Join Strategy	Meaning
JOIN VIA KEY COLUMN	Join table has a single key column key column is part of the join
JOIN VIA MULTIPLE KEY COLUMNS	Join table has multiple key columns all key columns are part of the join
JOIN VIA KEY RANGE	Join table has multiple key columns the first key column is part of the join
JOIN VIA RANGE OF MULTIPLE KEY COLUMNS	Join table has multiple key columns 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)

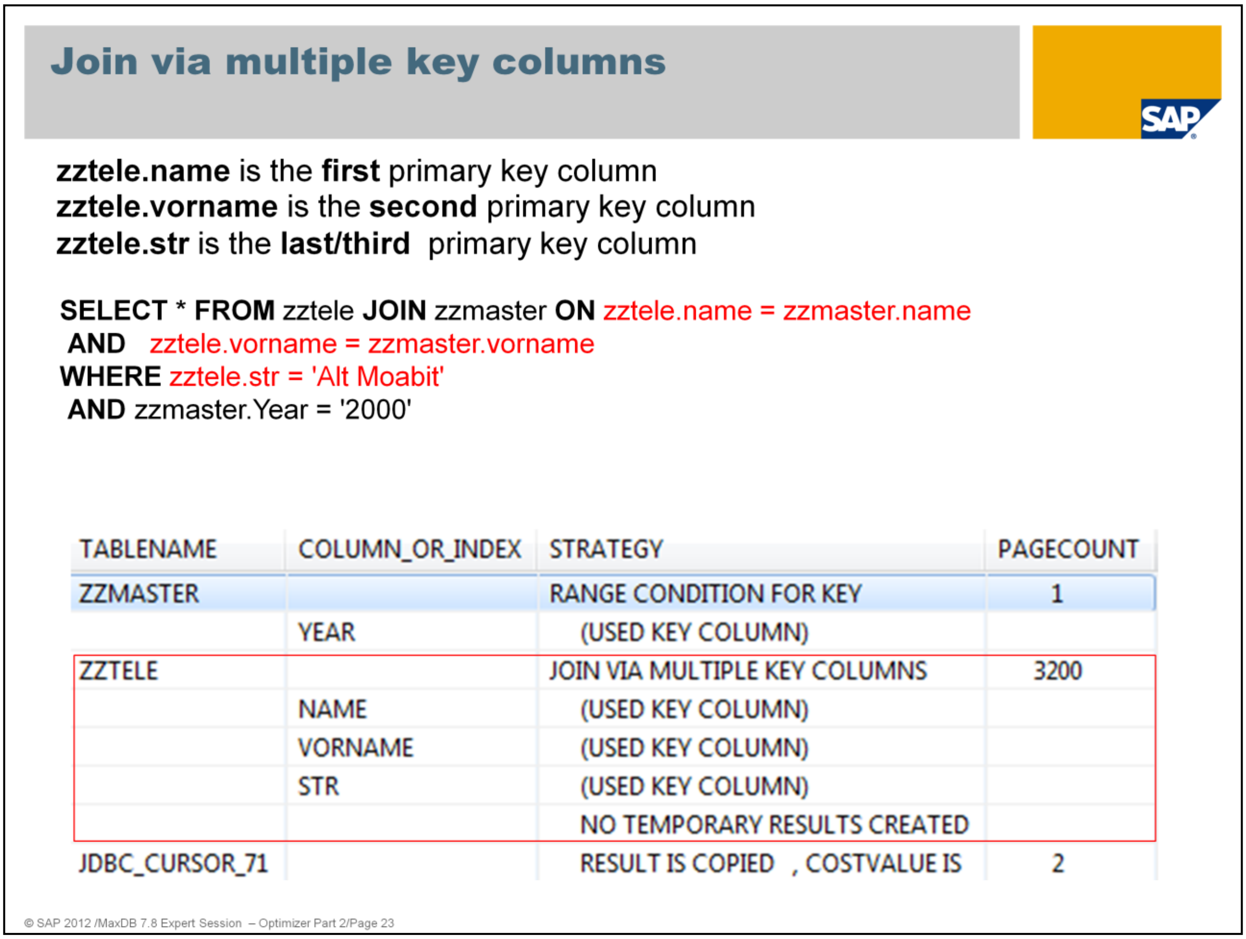
zzstadtteil.plz is the **sole** primary key column

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

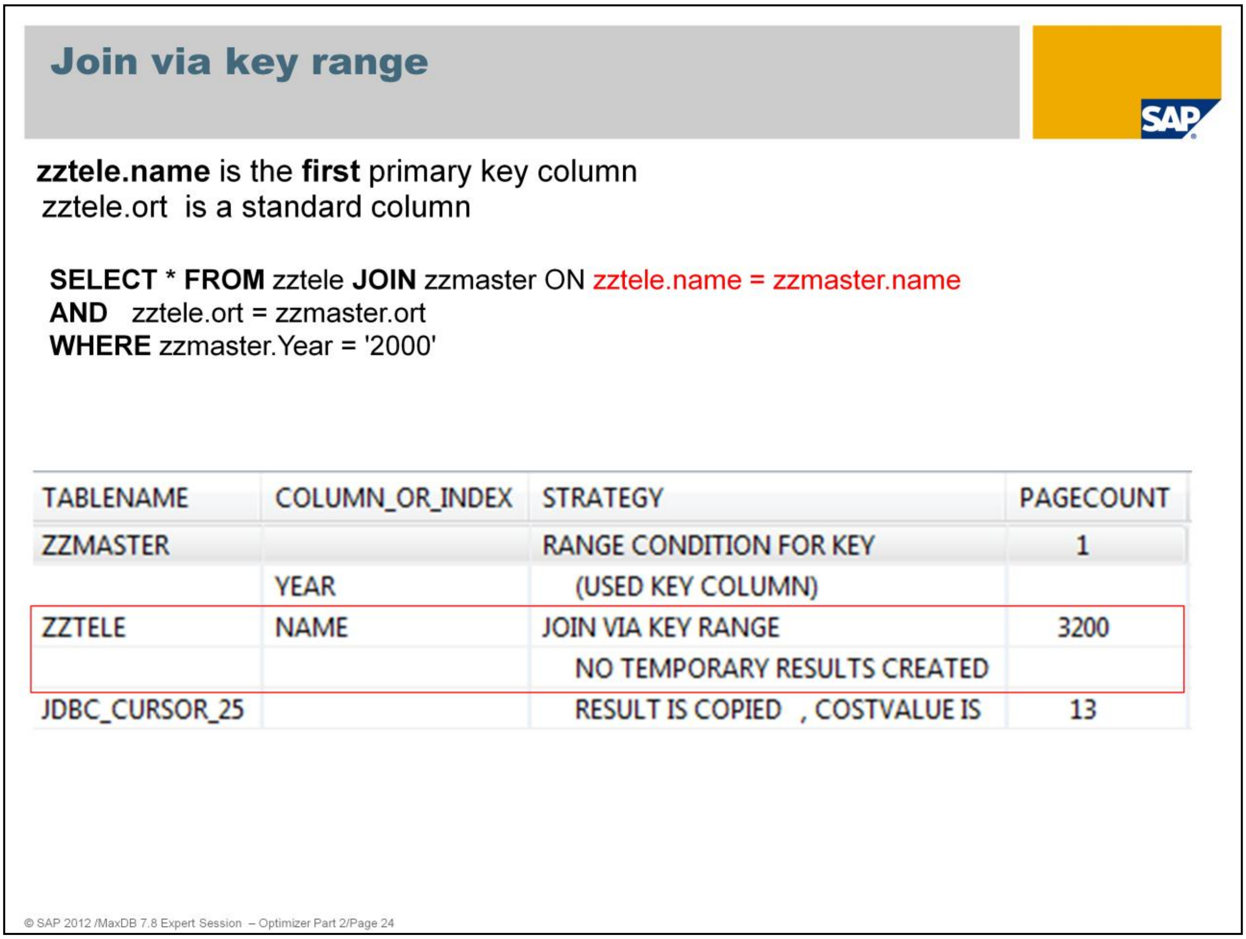
```
SELECT * FROM zztele JOIN zzmaster ON zztele.name = zzmaster.name
 AND   zztele.vorname = zzmaster.vorname
WHERE zztele.str = 'Alt Moabit'
 AND zzmaster.Year = '2000'
```

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

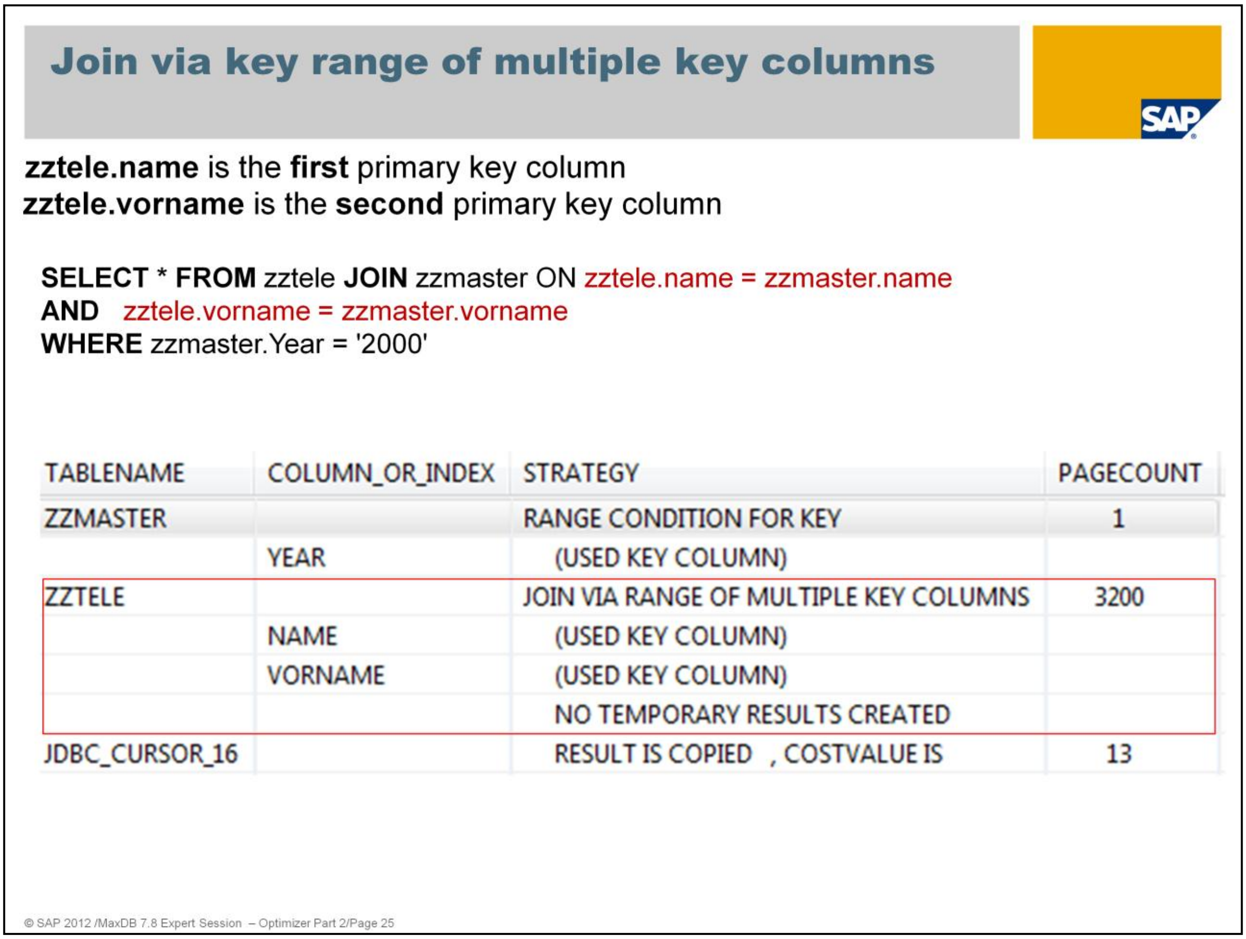
zztele.name is the **first** primary key column
zztele.ort is a standard column

```
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

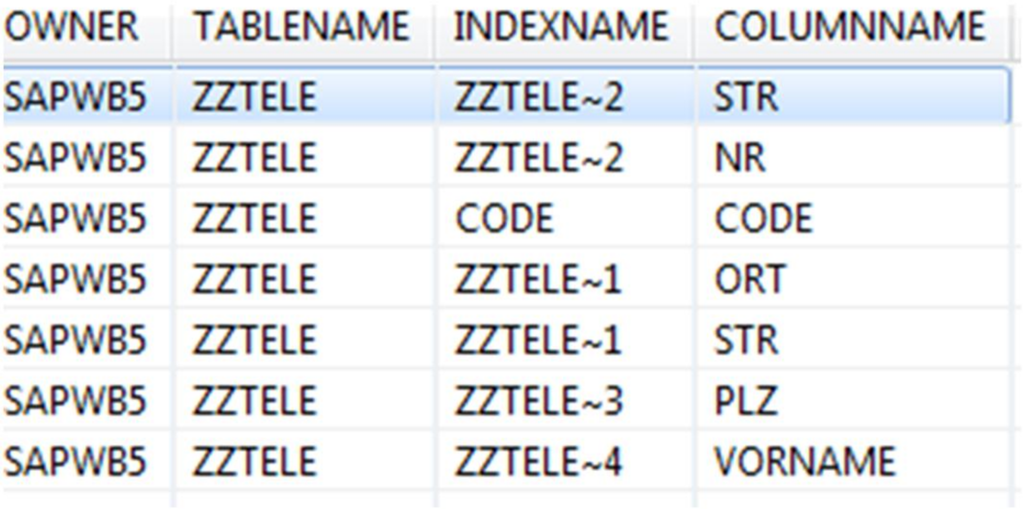
If the key of a joined table exists of more than one column and only a part of the multiple key is qualified the join transition is done via a key range.

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

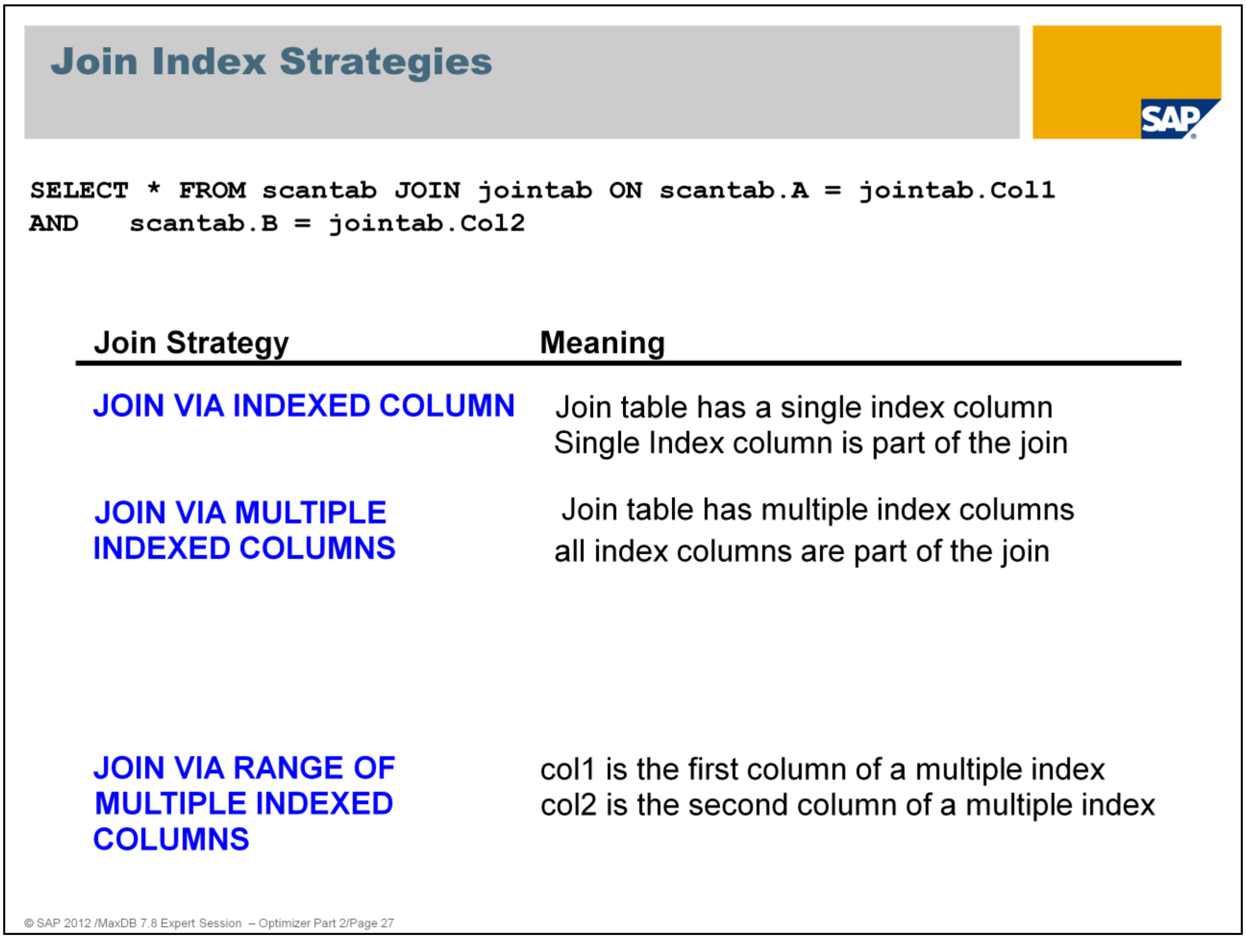
For the next examples about *JOIN VIA INDEX* accesses the tables *ZZTELE, ZZCODE,ZZMASTER* and *ZZSTADTTEIL* are used.

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

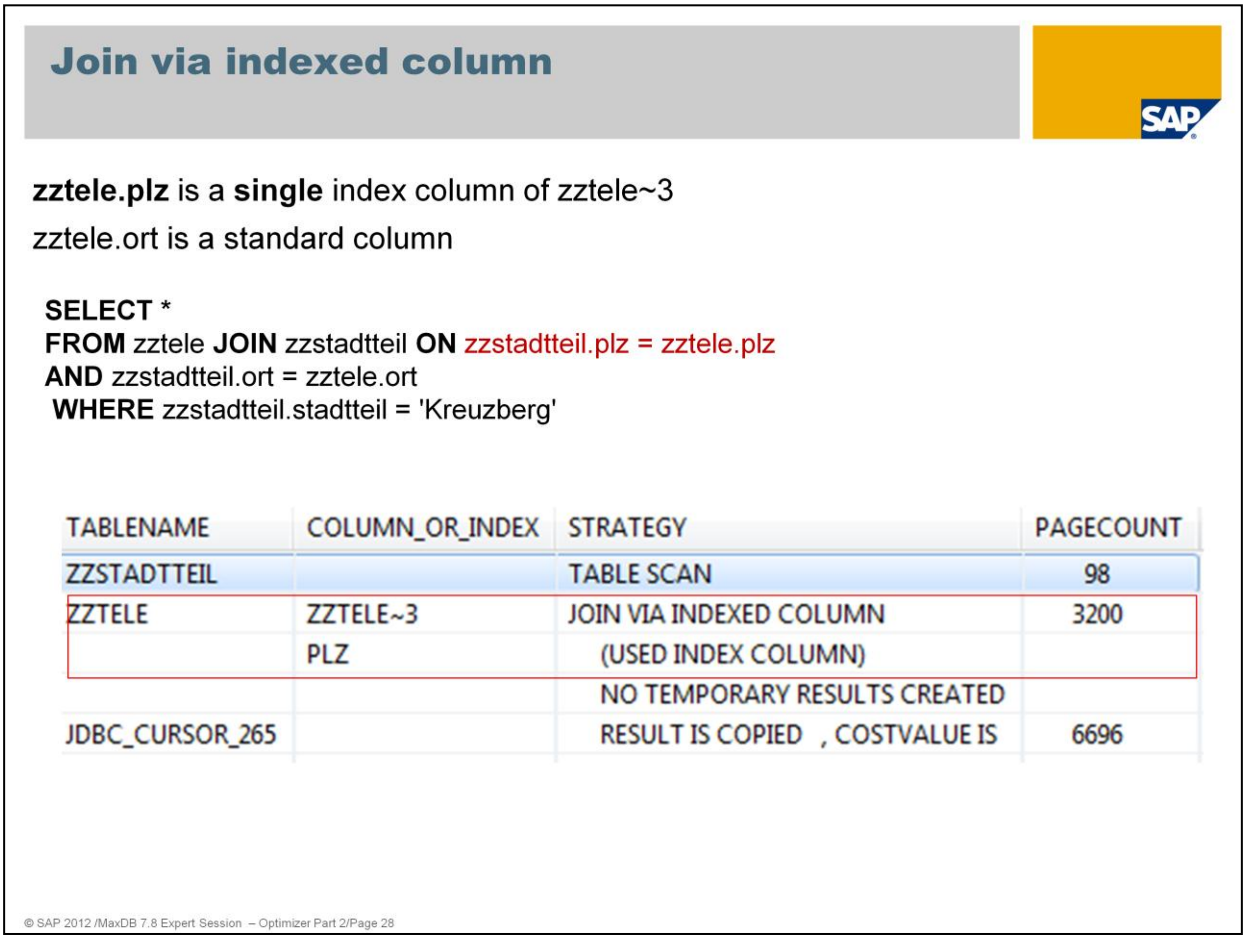
```
SELECT * FROM scantab JOIN jointab ON scantab.A = jointab.Col1
AND   scantab.B = jointab.Col2
```

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

© SAP 2012 /MaxDB 7.8 Expert Session – Optimizer Part 2/Page 27

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

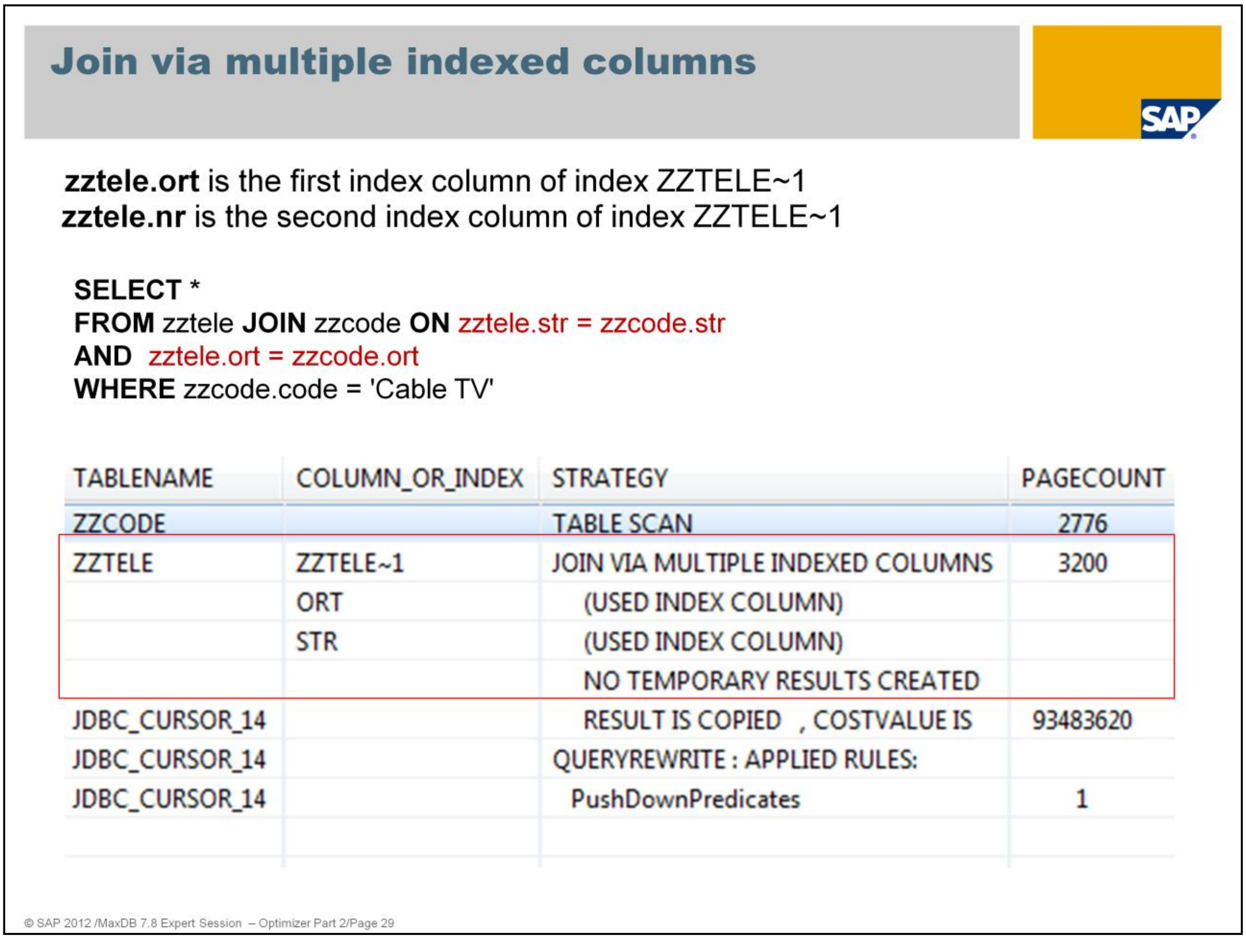
```
SELECT *
FROM zztele JOIN zzstadtteil ON zzstadtteil.plz = zztele.plz
AND zzstadtteil.ort = zztele.ort
 WHERE zzstadtteil.stadtteil = 'Kreuzberg'
```

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.

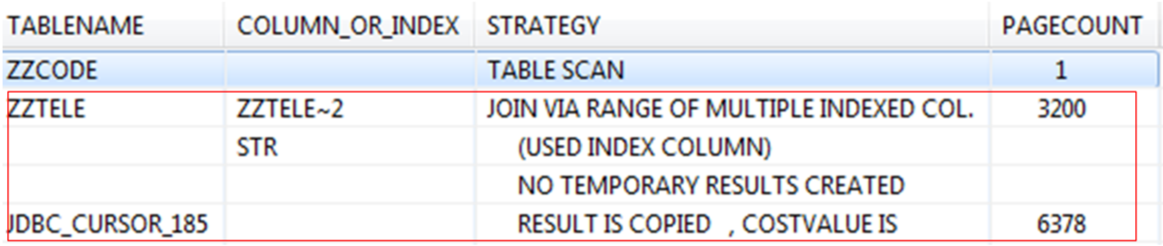
For the join transition a strategy called *JOIN VIA MULTIPLE INDEXED COLUMNS* can be used.

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'
```

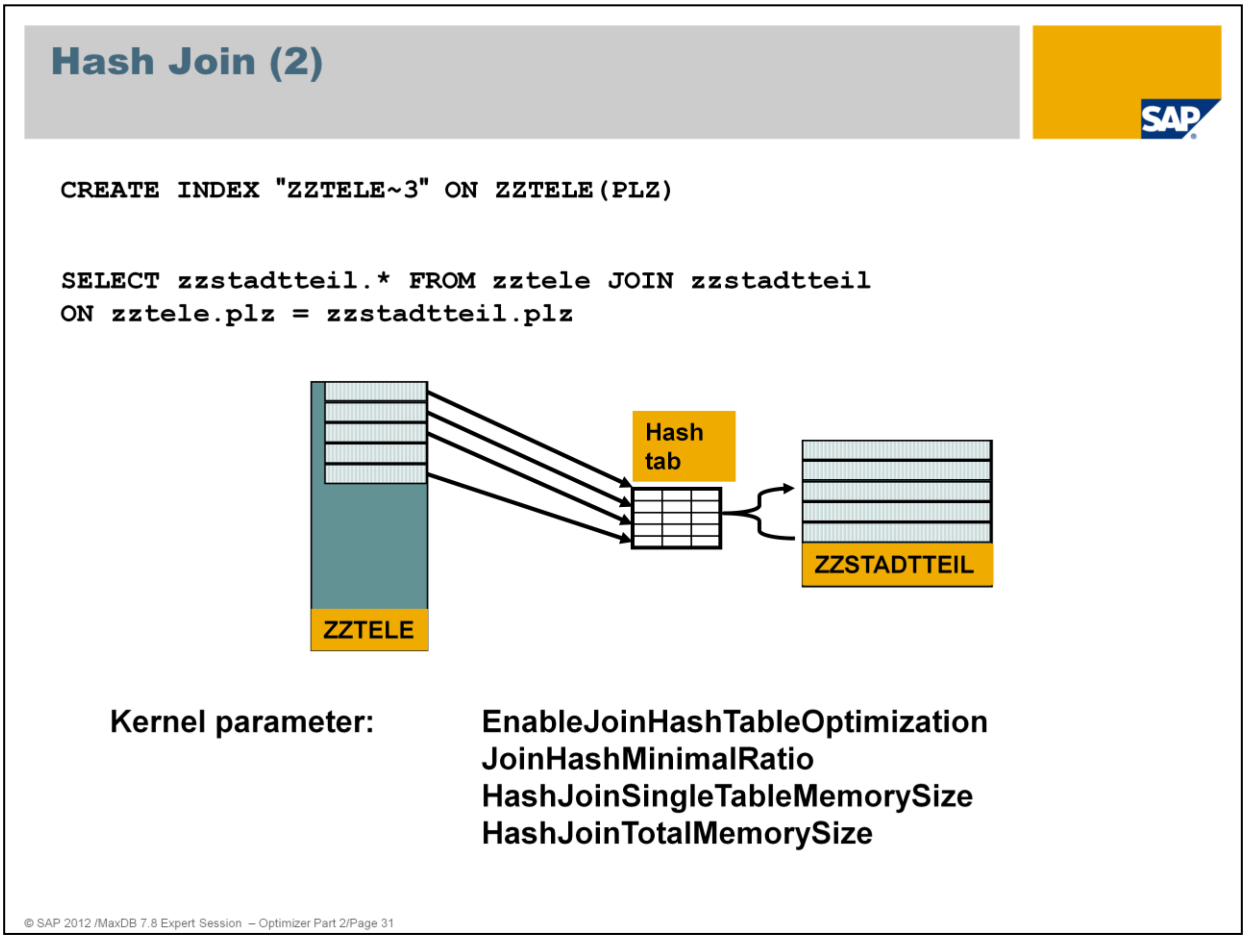


TABLENAME	COLUMN_OR_INDEX	STRATEGY	PAGECOUNT
ZZCODE		TABLE SCAN	1
ZZTELE	ZZTELE~2	JOIN VIA RANGE OF MULTIPLE INDEXED COL.	3200
	STR	(USED INDEX COLUMN)	
		NO TEMPORARY RESULTS CREATED	
JDBC_CURSOR_185		RESULT IS COPIED , COSTVALUE IS	6378

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)

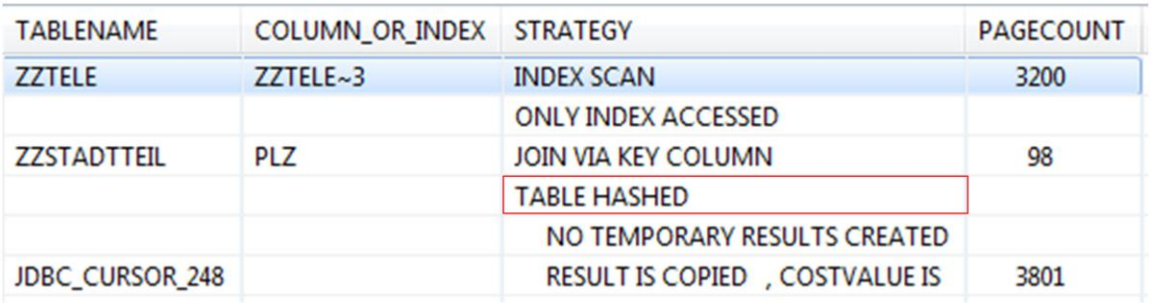
The maximum table size in KB for which hash joins will be executed. If HashJoinSingleTableMemorySize = 0 then no hash tables will be created during join execution.

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.

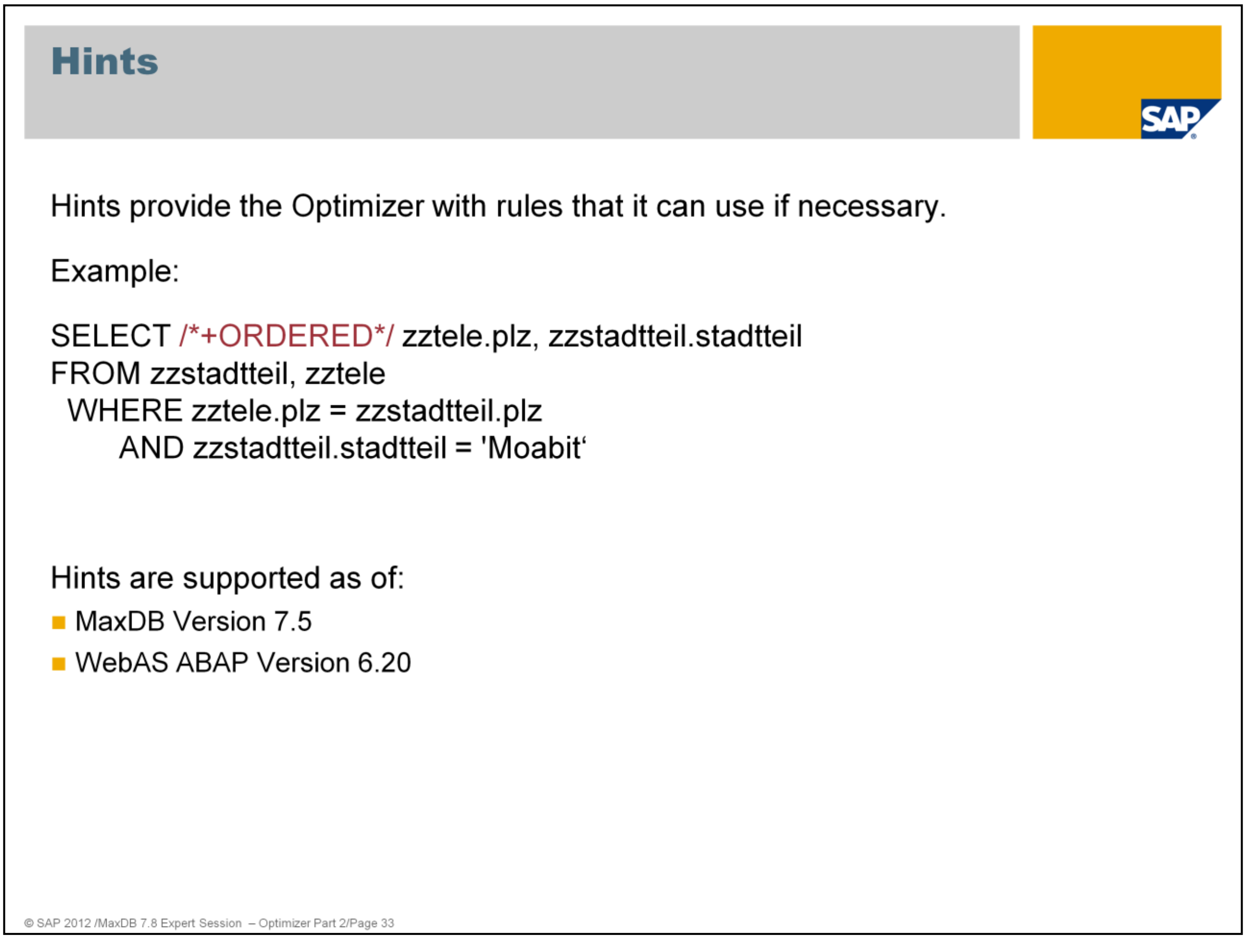
If HashJoinTotalMemorySize = 0 then no hash joins will be executed.

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:

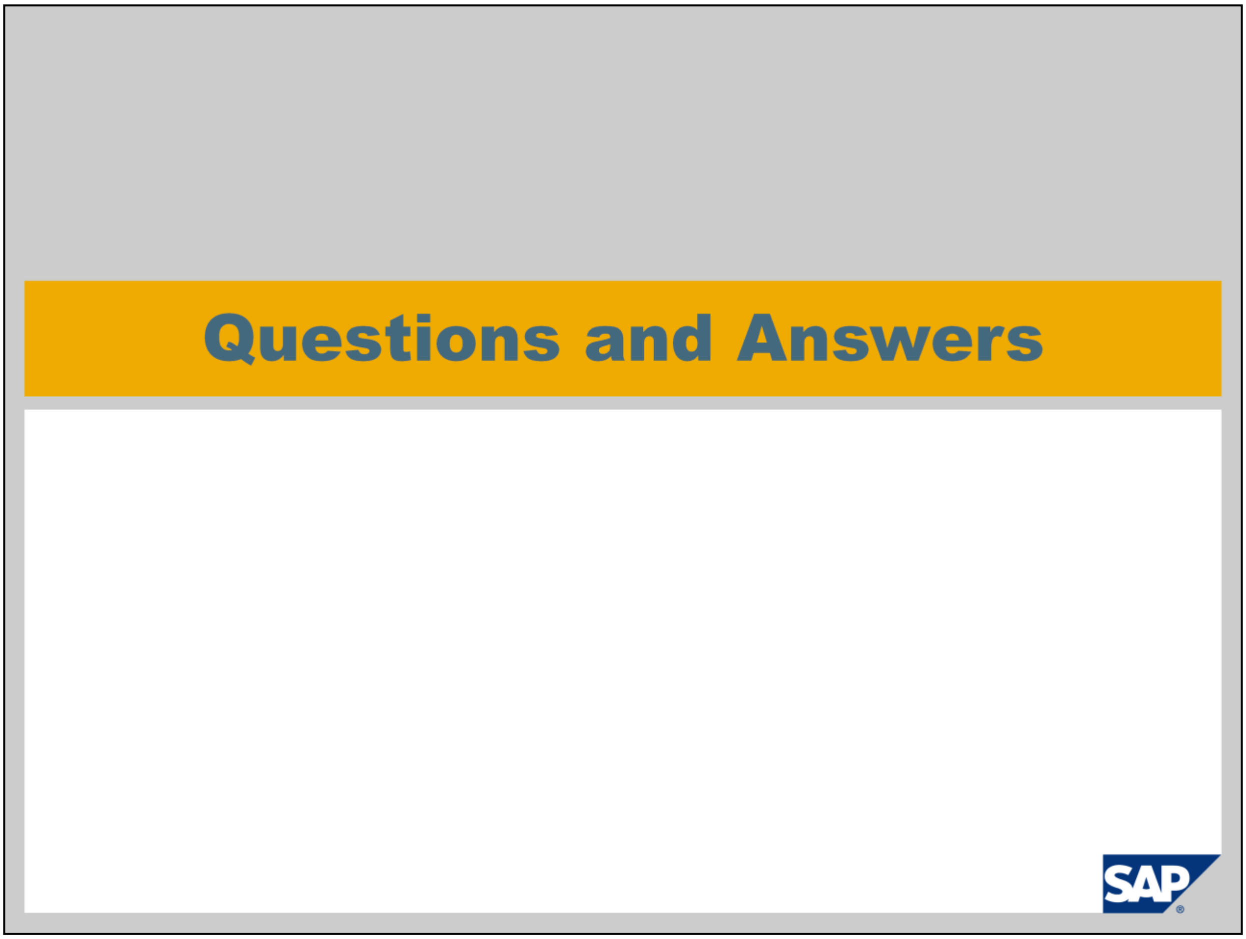
```
SELECT /*+ORDERED*/ zztele.plz, zzstadtteil.stadtteil
FROM zzstadtteil, zztele
  WHERE zztele.plz = zzstadtteil.plz
        AND zzstadtteil.stadtteil = 'Moabit‘
```

Hints are supported as of:

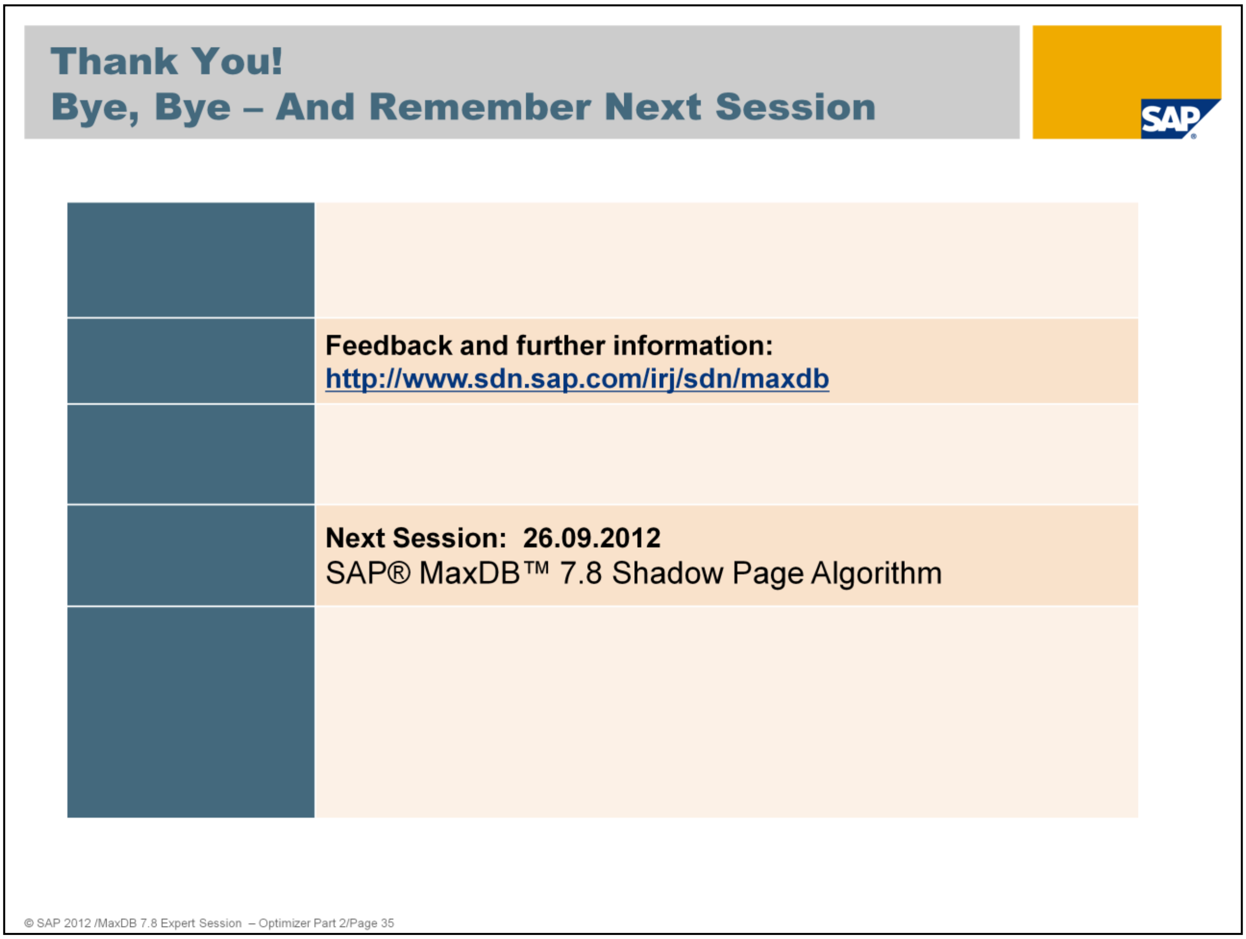
- MaxDB Version 7.5
- WebAS ABAP Version 6.20

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.



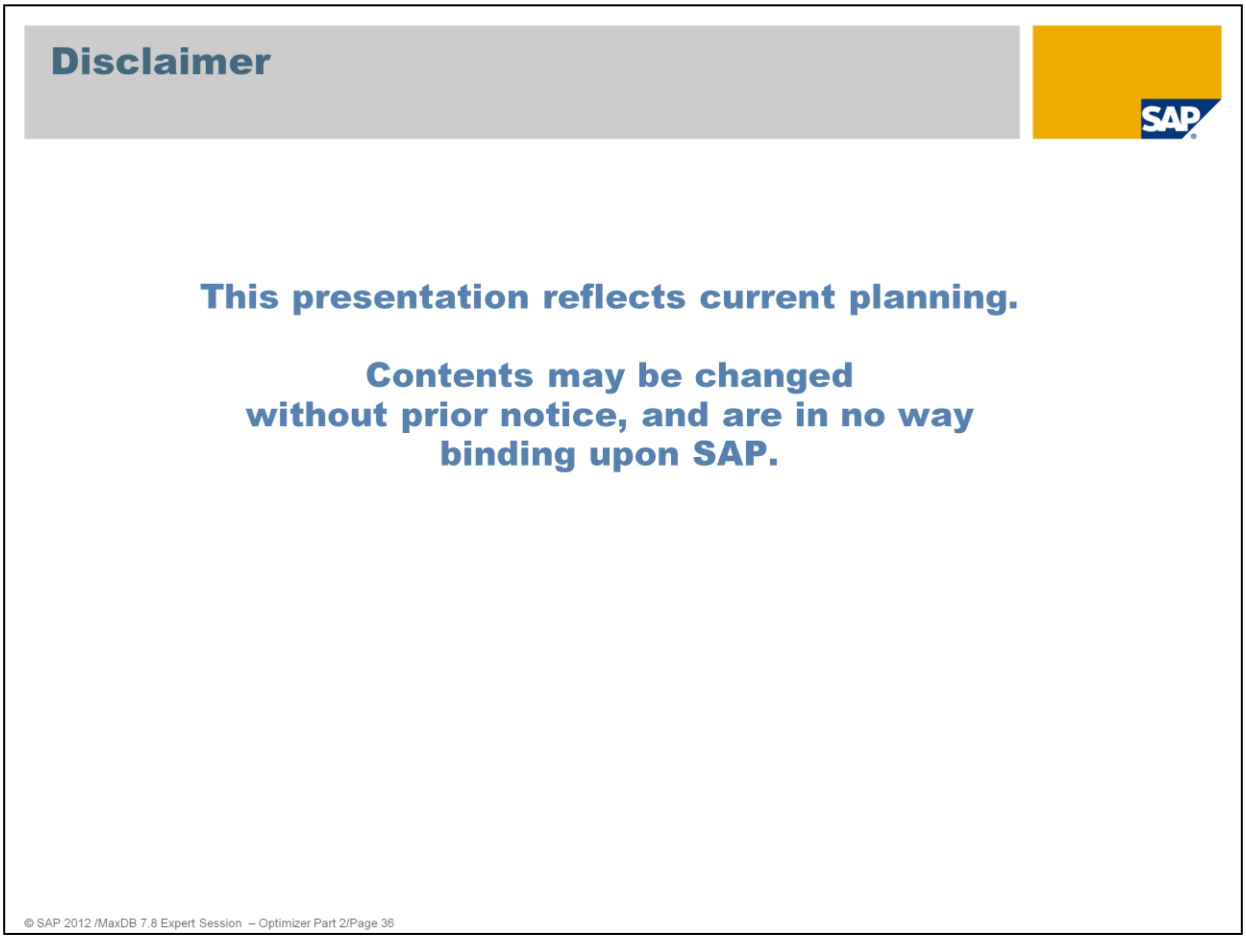
Questions and Answers



Thank You!
Bye, Bye – And Remember Next Session

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

Next Session: 26.09.2012
SAP® MaxDB™ 7.8 Shadow Page Algorithm



Disclaimer

This presentation reflects current planning.

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