SAP® MaxDB™

Expert Session – No-Reorganization Principle

IMS NW MaxDB / liveCache
August 2012

THE BEST-RUN BUSINESSES RUN SAP™

Expert Session

No-Reorganization Principle

IMS NW MaxDB / liveCache
Heike Gursch
Oksana Alekseious
August 14, 2012

THE BEST-RUN BUSINESSES RUN SAP™

Agenda

1. **Preconditions for No-Reorganization**
2. **B* Tree Concept**
   - Structure of a B* Tree
   - Access to Data
   - Examples of Pages
   - Contents of a Data Page
   - B* Tree Balancing
   - What are B* Trees used for?
3. **Concept of Filedirectory**
   - System View FILES
   - System View ROOT

© SAP AG 2012. All rights reserved. / Page 3

Preconditions for No-Reorganization

Space that is no longer used must be available for the database immediately.

The degree of usage of the data blocks must maintained at a consistently high level.

The data storage within the data blocks must be compact; no gaps are allowed.

© SAP AG 2012. All rights reserved. / Page 4

This slide describes the paradigms of a database system that is reorganization free.

As MaxDB does not need to be reorganized, the database can be operated with minimal administrative outlays.

The absence of the need to reorganize also means that the database always works with optimal access structures. That means consistently good performance.

No-Reorganization Principle (1)

No-reorganization

- update in place
- sort by insertion
- delete in place

Storage of data in B* trees

- Tables and indexes
- Data of type LOB (BLOBs, CLOBs, ex Long)

© SAP AG 2012. All rights reserved. / Page 5

To achieve an efficient I/O strategy while maintaining the no-reorganization principle of the database system, a framework of structural and functional prerequisites was developed for MaxDB. These include, on the one hand, the no-reorganization principle itself, which is the result of separate memory management for the secondary storage media and the logical data pages and is primarily based on the following functions:

- Sorting of data records when they are inserted,
- Changing of data records in place,
- Deleting of data records in place,

On the other hand, there are the logical storage structures and terms. We will take a closer look at:

- B* Trees,
- Tables and indexes,
- Primary and secondary keys
- and the storage of LOBs (BLOBs – binary large objects, CLOBS – character large objects)

No-Reorganization Principle (2)

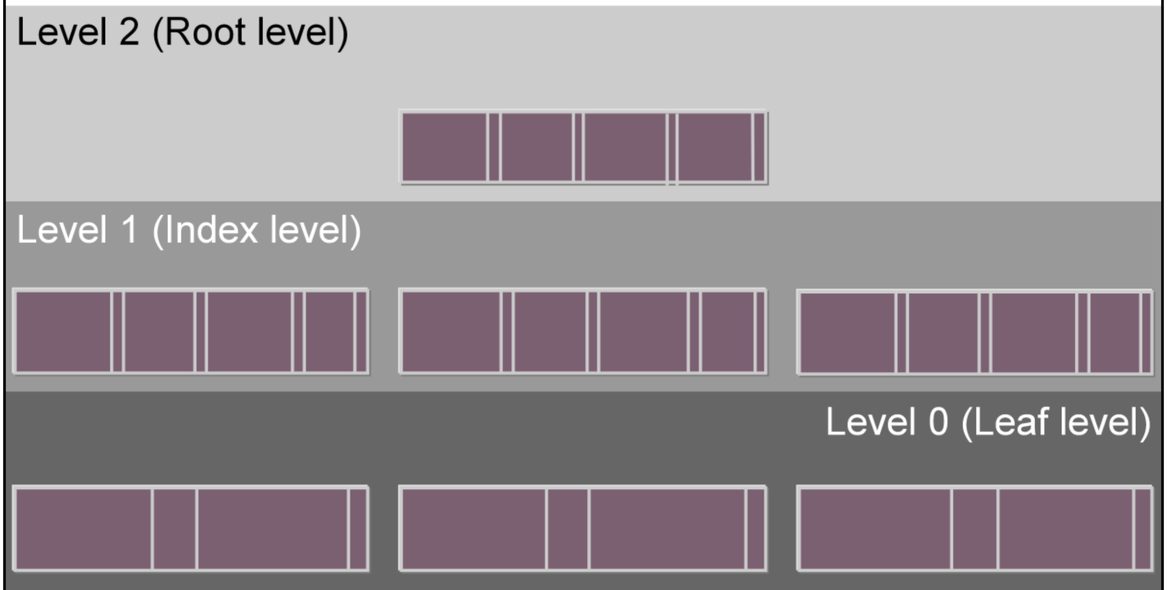
Storage concept

- concept of B* trees
- access to data
- execution of an INSERT
- execution of a DELETE
- execution of an UPDATE
- B* tree balancing
- striping

© SAP AG 2012. All rights reserved. / Page 6

The MaxDB storage concept ensures that data is quickly stored and found on the available disks. It is the key to automatic load balancing by MaxDB and thus guarantees the no-reorganization principle of the database system.

B* Tree Concept



© SAP AG 2012. All rights reserved. / Page 7

In MaxDB, data is stored in B* tree structures.

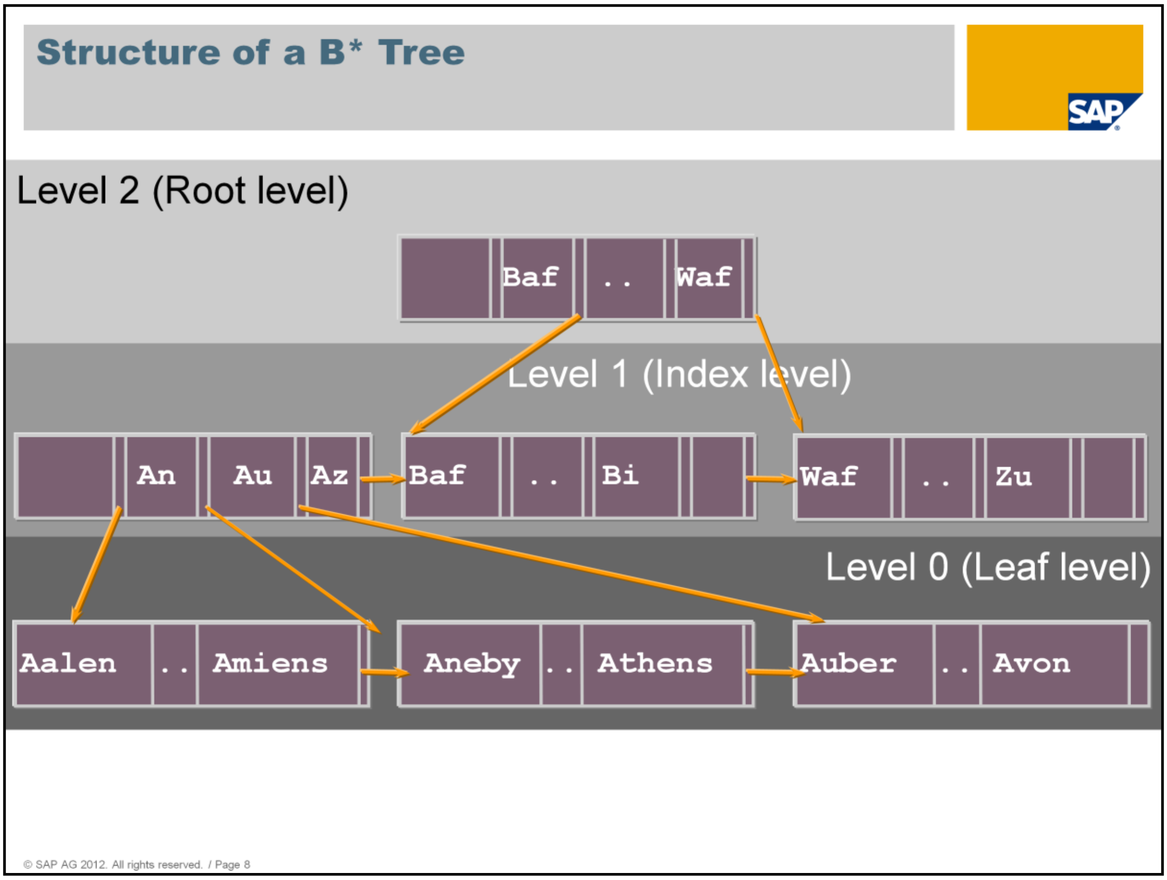
The smallest storage unit is the page. In MaxDB, the size of a page is 8 KB.

A B* tree is created for each table and secondary index.

A B* tree reaches from the highest level, the root level, to the lowest, the leaf level. The data is always on the leaf level.

The primary index of the tables serves as a sorting criterion for the setup of the tree structure.

It can be demonstrated that a B* tree procedure generally requires fewer accesses to find single records than other access methods.



The entries in the data pages are comprised of two parts:

- The fist part is comprised by the contents of the **key fields of a table row**. We shall refer to this as the **separator**.
- The second part is comprised by the remaining data.

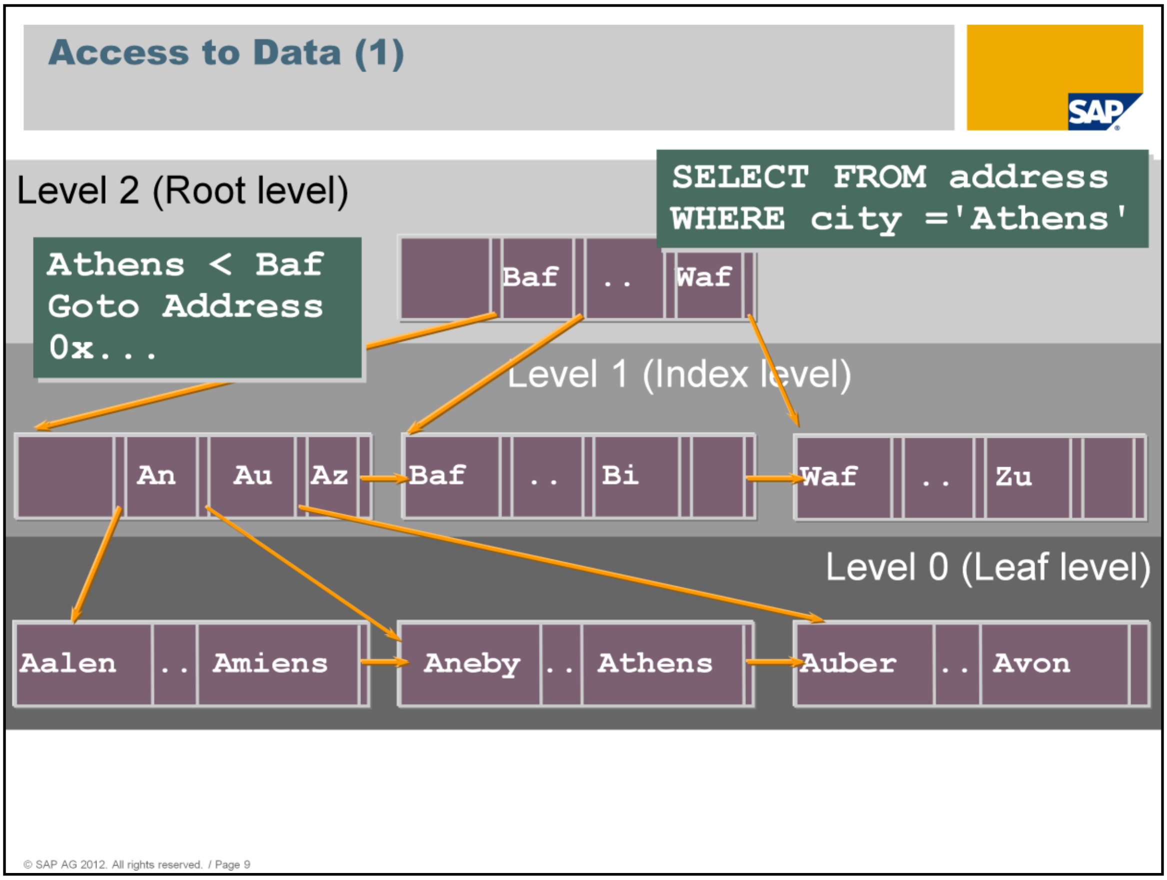
On index pages, every separator is followed by a **logical address** that refers to a page on a lower index level or on the leaf level. The number of entries that fits on an index page depends on the length of the separators.

A node of the B* tree always comprises one page. Thus the number of entries per B* tree level depends on the length of the separators.

In addition to the separator, leaf pages contain the contents of the other columns of the respective table row. The number of entries that fits on an index page depends on the length of the separators.

The amount of memory required for a table depends on the length of the key fields and the total length of a table row.

The procedure described here is supplemented by special treatment of tables that contain LOB columns (**L**arge **Ob**jects). Additional auxiliary trees are created for the purpose of accepting the contents of LOB columns, which can be many times longer than a data page.



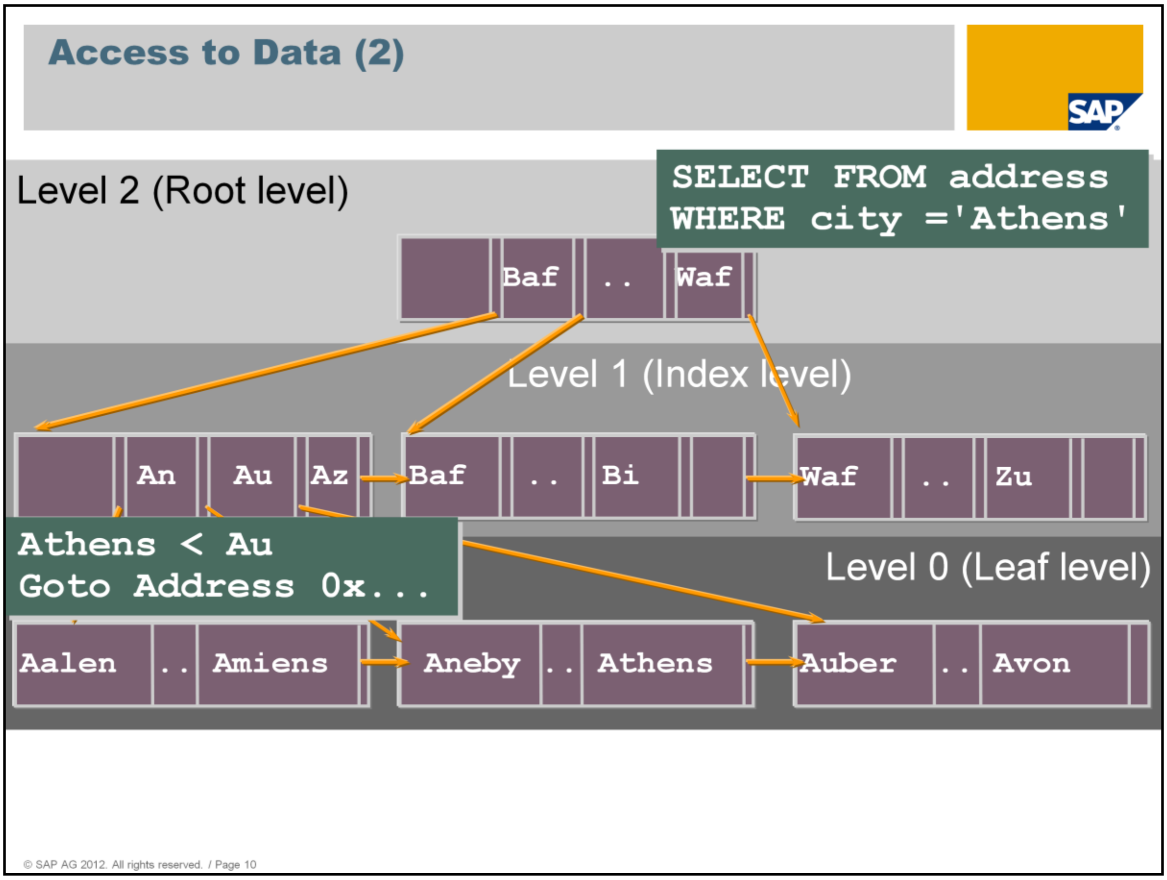
The B* tree procedure makes it possible to find data quickly.

Here's an example of how a data record is found: looking in an address table with the primary index '`city`', you want to find an entry for '`Athens`'.

The search begins on the root level. The comparisons described in the following take place on a character by character basis.

The database system checks if the value '`Athens`' is smaller than the second entry on the root page, '`Baf`'.

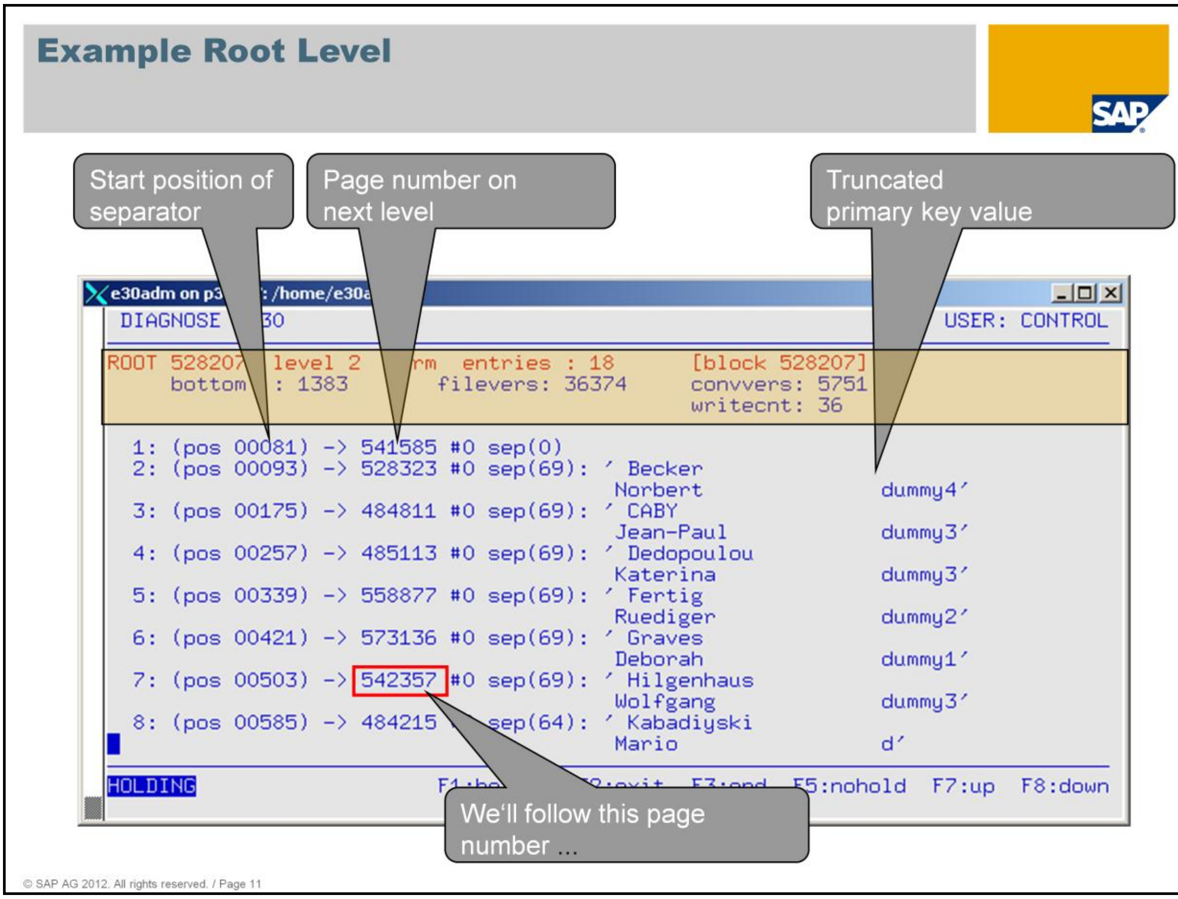
As the value is smaller, the corresponding logical address information from the first branch is evaluated. It points to a page on the next level (index level).



The comparison then continues on the index level. Now the desired value, 'Athens', is smaller than the entry 'Au' on the data page.

So the 'An' branch is evaluated.

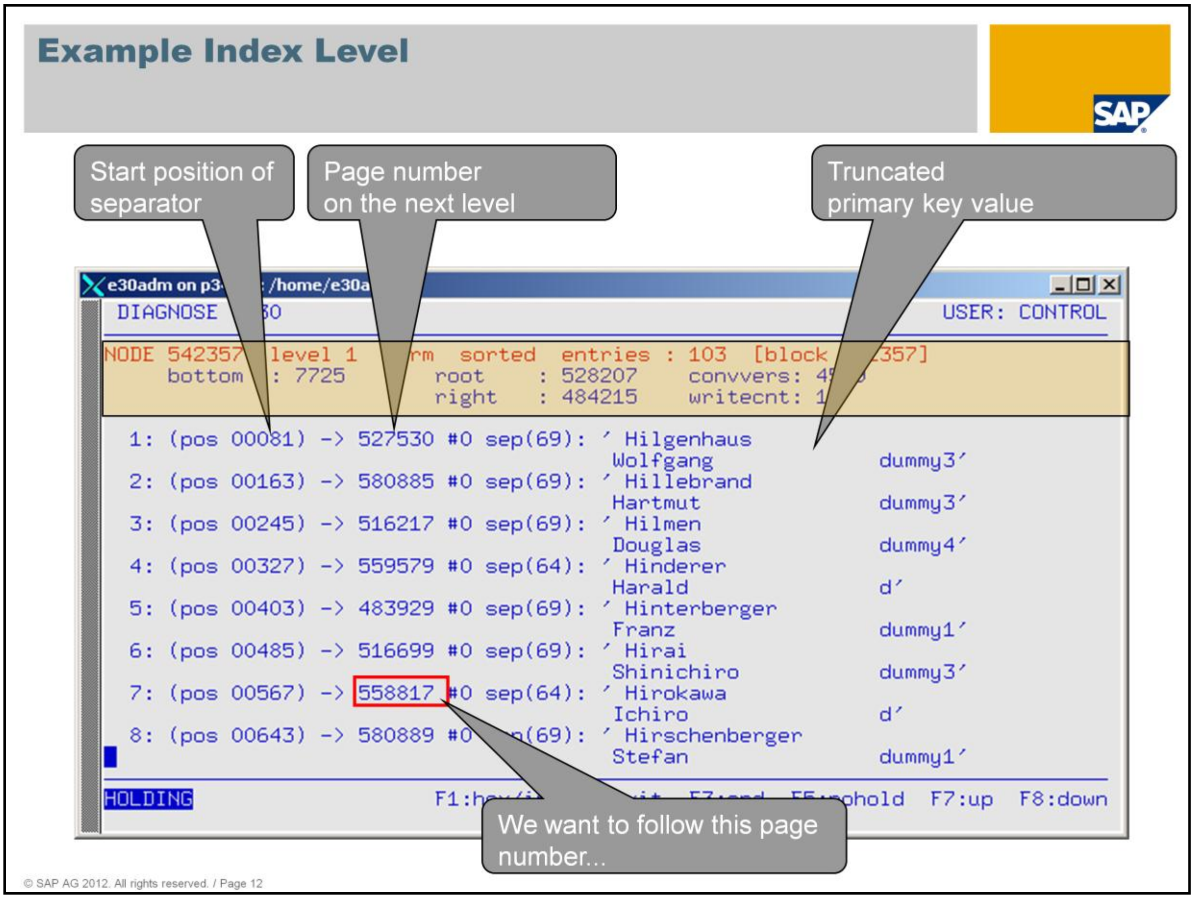
The pointer points to the second page on the leaf level. Now we are on the leaf level (level 0).



This page shows a root page as displayed by the MaxDB Tool x_diagnose.

At the top you see the page header. As the page number and root number on this page are the same, this is a root page. The B* tree has three levels (levels 0 – 2). This page has 18 entries. It was changed 36 times.

The separators are shown in their alphanumeric order. You see the respective start position, the page number on the next page to the separator, the length and the value of the separator.

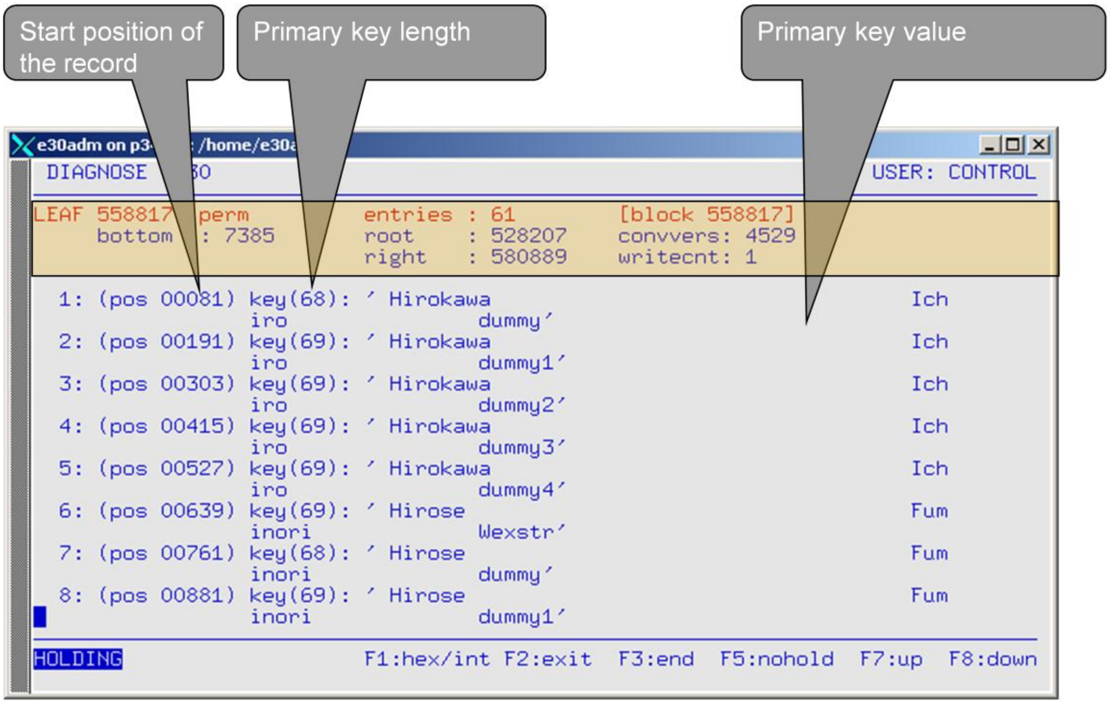


This page is an index page of level 1. The separators refer to pages of the leaf level.

The header contains the known root page. It is checked with each access.

The page has 103 entries, sorted.

Example Leaf Level I



© SAP AG 2012. All rights reserved. / Page 13

In this example the diagnosis tool displays only the primary key values.

This page has 61 entries. The last record ends at position 7385.

Example Leaf Level II

```
e30adm on p34777: /home/e30adm
 DIAGNOSE  E30                                                       USER: CONTROL
  1: (pos 00081) key(68): ' Hirokawa                                        Ich
                        iro                         dummy'
00001       recLen       : 110                        recKeyLen    : 68
00005       recVarcolOff: 7                           recVarcolCnt: 4
      record                   BUFFER FROM 1 TO 110 (FROM 81 TO 190)
      1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
     81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
dec:110  0 68  0  7  0  4  0 32 72105114111107 97119 97 32 32 32
hex: 6E 00 44 00 07 00 04 00 20 48 69 72 6F 6B 61 77 61 20 20 20
chr:  n     D                    H  i  r  o  k  a  w  a

     21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
    101   103   105   107   109   111   113   115   117   119
dec: 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32
hex: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
chr:

     41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
    121   123   125   127   129   131   133   135   137   139
dec: 32 32 32 32 32 32 32 32 32 32 73 99104105114111 32 32 32 32
hex: 20 20 20 20 20 20 20 20 20 20 49 63 68 69 72 6F 20 20 20 20
chr:                                I  c  h  i  r  o

     61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
    141   143   145   147   149   151   153   155   157   159
dec: 32 32 32 32 32 32 32 32 32 32 32100117109109121  0194112  0
hex: 20 20 20 20 20 20 20 20 20 20 20 64 75 6D 6D 79 00 C2 70 00
chr:                                   d  u  m  m  y        p

     81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
    161   163   165   167   169   171   173   175   177   179
dec:  0  0  0  6 32 49 48 49 48 50  1 32  1 32 15 32 70105110 97
hex: 00 00 00 06 20 31 30 31 30 32 01 20 01 20 0F 20 46 69 6E 61
chr:                 1  0  1  0  2                    F  i  n  a

HOLDING                      F1:hex/int F2:exit  F3:end  F5:nohold  F7:up  F8:down
```

© SAP AG 2012. All rights reserved. / Page 14

This graphic shows the first 100 bytes of the first record of page 558817.

Each record begins with a header. This contains the length of the record, the length of the primary key value, the relative start position of the first variable-length value (e.g. VARCHAR) and the number of variable-length fields.

On this page the record begins on position 81. The primary key begins within the record at position 10.

Example: Table ZZTELE

select f.root, f.type, t.tablename, entrycount from files f, tables t where f.fileid = t.tableid and tablename='ZZTELE'

ROOT	TYPE	TABLENAME	ENTRYCOUNT
3034399	TABLE	ZZTELE	114199

Table Definition SAPWB5.ZZTELE

Column Name	Type	Data Type	Code Type	Length	Decimal Places	Access Rights	Default	Position	Key Position
NAME	KEY	VARCHAR	UNICODE	40		SEL+UPD+		1	1
VORNAME	KEY	VARCHAR	UNICODE	20		SEL+UPD+		2	2
STR	KEY	VARCHAR	UNICODE	40		SEL+UPD+		3	3
NR	OPT	NUMBER		10	0	SEL+UPD+	0	4	
PLZ	MAN	VARCHAR	UNICODE	5		SEL+UPD+		5	
ORT	MAN	VARCHAR	UNICODE	25		SEL+UPD+		6	
CODE	MAN	VARCHAR	UNICODE	31		SEL+UPD+		7	
ADDINFO	MAN	VARCHAR	UNICODE	31		SEL+UPD+		8	

	NAME	VORNAME	STR	NR	PLZ	ORT	CODE	ADDINFO
1	Legner	Ina	Dummy	22	26884			kaufmännische Ausbildungsber
2	Legner	Ina	Dummy1	23	26885			kaufmännische Ausbildungsber
3	Legner	Ina	Dummy2	24	26886			kaufmännische Ausbildungsber
4	Legner	Ina	Dummy3	25	26887			kaufmännische Ausbildungsber
5	Legner	Ina	Dummy4	26	26888			kaufmännische Ausbildungsber

© SAP AG 2012. All rights reserved. / Page 15

If the name of the table is known we can find the root of the corresponding B*tree where the table content is stored and how many entries this table has.

The table ZZTELE contains as information the name, surname of a person, their address with the street, house number, postal code, place and some additional information. As we see in the table definition the first three columns NAME, VORNAME and STR build the primary key of this table.

The content of the table shows one distinctive feature that one person often has different addresses so that the primary key differs only in the third column STR. And often the street name differs only in the last character for example „Dummy“ and „Dummy1“. It is important to keep this in mind for a later check of the separators in the B*tree of this table.

Root and Index Level I of the Table ZZTELE

```
ROOT 3034399   level 3   perm  entries : 2      [block 0]
        bottom  : 241            filevers: 46151    convvers: 49448
                                                    writecnt: 1
  1: (pos 00081) -> 3036160 #0 sep(0)
  2: (pos 00093) -> 4201499 #0 sep(135):    Legner Ina dummy4

NODE 3036160   level 2   perm  entries : 28     [block 1]
        bottom  : 3759              root    : 3034399   convvers: 49448
                                    right   : 4201499   writecnt: 1

  1: (pos 00081) -> 3093547 #0 sep(0)
  2: (pos 00093) -> 4150358 #0 sep(135):   Al-Rajhi Liza dummy3
  3: (pos 00241) -> 3080024 #0 sep(135):   Anez Marcela dummy3
  4: (pos 00389) -> 3005515 #0 sep(135):   Baeck Christian dummy2
  5: (pos 00537) -> 3050137 #0 sep(135):   Beck  Bettina dummy2
...
 26: (pos 03335) -> 3080354 #0 sep(135):   Klingels Ulrich dummy4
 27: (pos 03483) -> 2991597 #0 sep(125):   Koyama Emiko d
 28: (pos 03621) -> 3035591 #0 sep(125):   Leblon Pierre  d

NODE 4201499   level 2  perm  entries : 30     [block 2]
        bottom  : 4303           root    : 3034399   convvers: 49448
                                 right   : nil           writecnt: 1

  1: (pos 00081) -> 3064665 #0 sep(135):  Legner Ina dummy4
  2: (pos 00229) -> 3064701 #0 sep(134):  Loehrlein Harald dummy
  3: (pos 04155) -> 3051003 #0 sep(135):  Luedtke Christine dummy2
...
 29: (pos 03565) -> 3095070 #0 sep(135):  Wray Paul Philip  dummy2
 30: (pos 03713) -> 3020557 #0 sep(135):  Zhao Xu Min dummy4
```

© SAP AG 2012. All rights reserved. / Page 16

The root page has only two entries and leads us to the index pages 3036160 and 4201499. This B*tree is rather deep and possesses three administrative levels till we reach the leaf pages (level 0) with the complete content of every table entry. On the root and index pages only the distinctive part of the primary key is stored.

In this case we see the part of the B*tree starting with the second index page 4201499 containing the references to the table data which are the same and bigger than the primary key NAME=„Legner“, VORNAME= „Ina“ and STR=„Dummy4“

Afterwards both index pages from the next level (level 2 on the page) are listed. The first index page 303616 contains the references to the 28 next index pages (level 1 on the page) and the second index page 4201499 contains the references to the next 30 index pages from the last index level before the leaf pages.

We want to descend in the B*tree and will have look on the index page 2991597 which is referenced as the 27th entry of the first index page 3036160.

Index Level II of the Table ZZTELE

```
26: (pos 03335) -> 3080354 #0 sep(135):   Klingels Ulrich dummy4
27: (pos 03483) -> 2991597 #0 sep(125):   Koyama Emiko d
28: (pos 03621) -> 3035591 #0 sep(125):   Leblon Pierre  d

NODE 3080354  level 1   perm  sorted  entries : 58    [block 28]
        bottom  : 7831          root    : 3034399        convvers: 49448
                                right   : 2991597       writecnt: 2

NODE 2991597  level 1   perm  sorted  entries : 67    [block 29]
        bottom  : 7667          root    : 3034399        convvers: 49448
                                right   : 3035591       writecnt: 2


 1: (pos 00081) -> 3094341 #0 sep(125):  Koyama Emiko d
 2: (pos 00219) -> 3019059 #0 sep(9):    Kozi
 3: (pos 00241) -> 3050083 #0 sep(134):  Kraehmer Karin Dummy
...
66: (pos 07371) -> 3080384 #0 sep(135):  LAUREAU Laurence dummy4
67: (pos 07519) -> 2991613 #0 sep(135):  LE GRATIET Gwenola Dummy3

NODE 3035591  level 1   perm  sorted  entries : 62    [block 30]
        bottom  : 7695          root    : 3034399        convvers: 49448
                                right   : 3064665       writecnt: 2
```

© SAP AG 2012. All rights reserved. / Page 17

The first three entries are copied from the first index page 3036160 of the first index level (level 2 on the page). Now we see three pages of the second index level (level 1 on the page). All three have the 3034399 root page and are connected with each other through the reference to its right neighbour.

The page 2991597 (the 27th entry on the first index page of the first index level) is mentioned first as the right neighbour on the page 3080354 which was the 26th entry. This page references itself 67 leaf pages where the full table content is stored.

We want to check the leaf page 3019059 which contains data entries where NAME >= “Kozi” and (NAME <=“Kraehmer” and VORNAME <= “Karin” and STR < “Dummy”)

Leaf Level of the Table ZZTELE

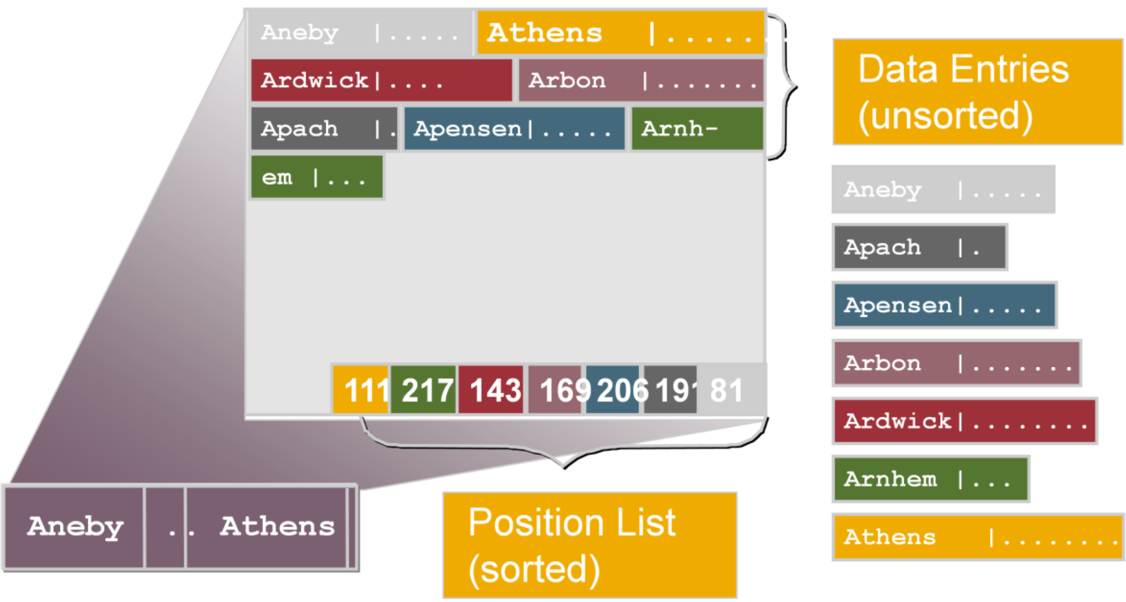
```
 1: (pos 00081) -> 3094341 #0 sep(125):  Koyama Emiko d
 2: (pos 00219) -> 3019059 #0 sep(9):    Kozi
 3: (pos 00241) -> 3050083 #0 sep(134):  Kraehmer Karin Dummy

LEAF 3019059  perm  sorted  entries : 37        [block 1536]
       bottom  : 7193           root    : 3034399   convvers: 49448
                                right   : 3050083   writecnt: 4

00021     ndRoot     : 3034399/1F4D2E00
00025     ndRight    : 3050083/638A2E00
00029     ndLeft     : 3094341/45372F00

 1: (pos 00081) key(133)  Koziaczy Jacek Dummy  34 22306
 2: (pos 00245) key(135)  Koziaczy Jacek Dummy1 35 22307
 3: (pos 00411) key(135)  Koziaczy Jacek Dummy2 36 22308
 4: (pos 00577) key(135)  Koziaczy Jacek Dummy3 37 22309
 5: (pos 00743) key(135)  Koziaczy Jacek Dummy4 38 22310
 6: (pos 00909) key(135)  Koziaczy Jacek Wexstr 39 22311
 7: (pos 01075) key(133)  Kozlov Alexandr Dummy 40 22312 Facilities
Moscow
...
```

© SAP AG 2012. All rights reserved. / Page 18

At the beginning there are again three entries from the previous index page 2991597 from the last index level (level 1 on the page) with the leaf page 3019059 which we want to check. On this leaf page we see the root page and also both neighbour leaf pages from the left and from the right. It is the only page where the left neighbour is referenced because the last leaf page of the table always has NIL as the left neighbour. With it the B*tree is finished.

The 37 entries on this page list the whole data entry in this table and not only the primary key information .

Contents of a Data Page



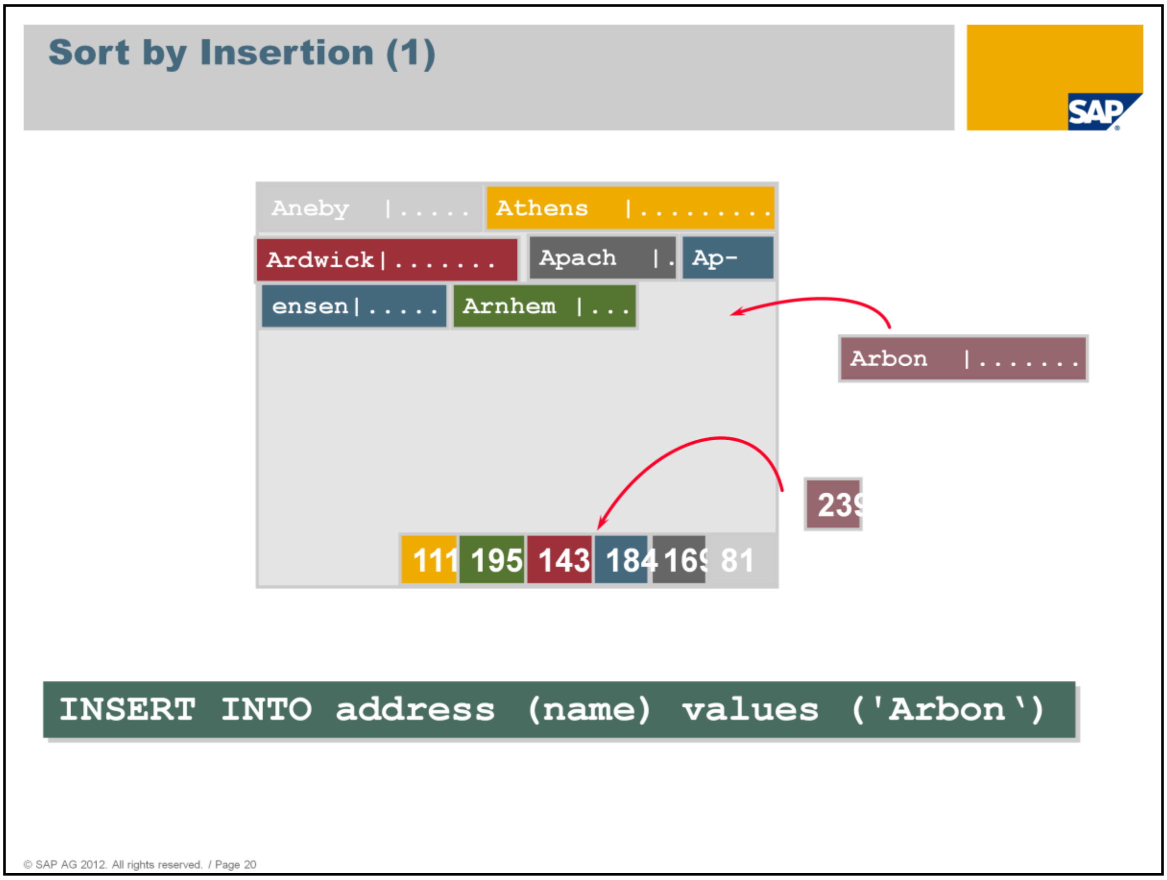
© SAP AG 2012. All rights reserved. / Page 19

The data records are located unsorted in the start area of the target page.

In the end area of the data page, there is a position list that refers to the individual records of the data page. This address list is arranged so that in the case of sequential access via the position list, the data entries can be read sorted.

The database system searches the remaining entries and ultimately returns the requested table row.

The position list and the data record entries start at opposite corners of the page and grow towards each other.



© SAP AG 2012. All rights reserved. / Page 20

If a record is to be inserted into the database or edited, MaxDB first searches for the data page that is changed by the action. This is true for all the actions described in the following. Then, if necessary, the required space is made available by way of clearing operations.

sort by insertion

The records are:

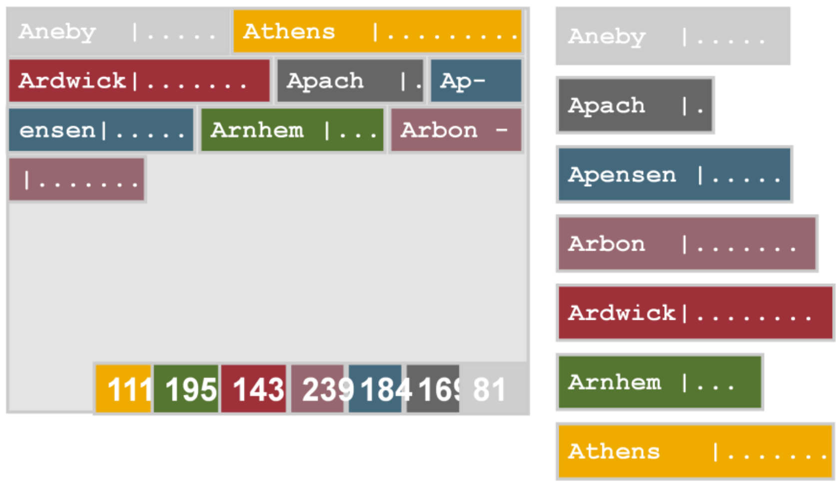
- inserted into the target page at the end of the used data area,
- sorted in the position list via an entry that, in order to minimize the number of moved bytes, contains only references to records.

The records in the data part are only sorted if a clearing operation becomes necessary. If a data page is moved into another one, a sorted block is advantageous as this makes it possible to move whole groups of records rather than copying record by record.

MaxDB data pages are organized such that the data area grows into the page from the beginning and the sorting list from the end.

Let's assume that the record fits on the page. MaxDB simply puts it at the end of the area available on the page...

Sort by Insertion (2)

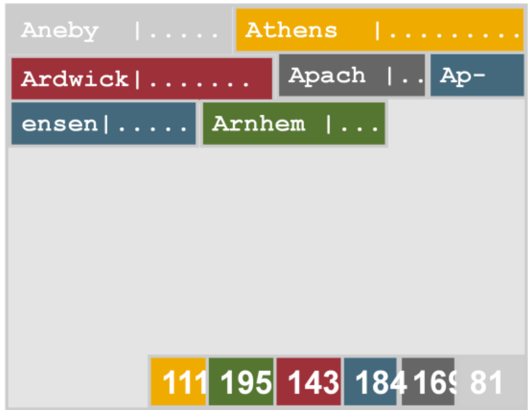



```
INSERT INTO address (name) values ('Arbon')
```

© SAP AG 2012. All rights reserved. / Page 21

... and then the position list is updated. The address of the new entry is written at the correct position in the position list. In our case, the correct position is position 4, which accordingly points to the seventh data record, '`Arbon`'.

Update in Place (1)




```
UPDATE address
SET street = 'AKROPOLIS 1'
KEY city = 'Athens'
```

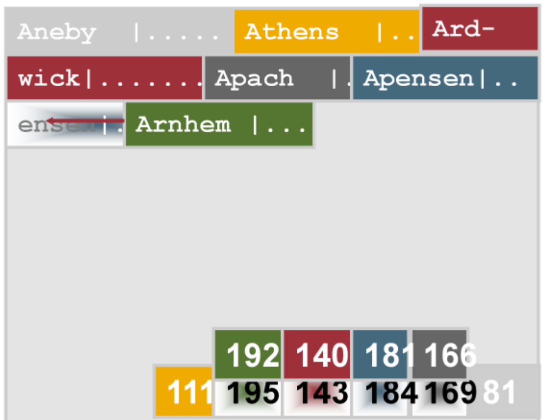
© SAP AG 2012. All rights reserved. / Page 22

update in place

- Records are changed directly on the target page.
- Case 1: length and key remain unchanged.
  If an UPDATE occurs and the separator is unchanged, the contents of the row are changed directly.

- Case 2: the key changes.
  If changes have been made to a key field, the UPDATE is converted into a DELETE with subsequent INSERT. If necessary, clearing operations are carried out.

Update in Place (2)



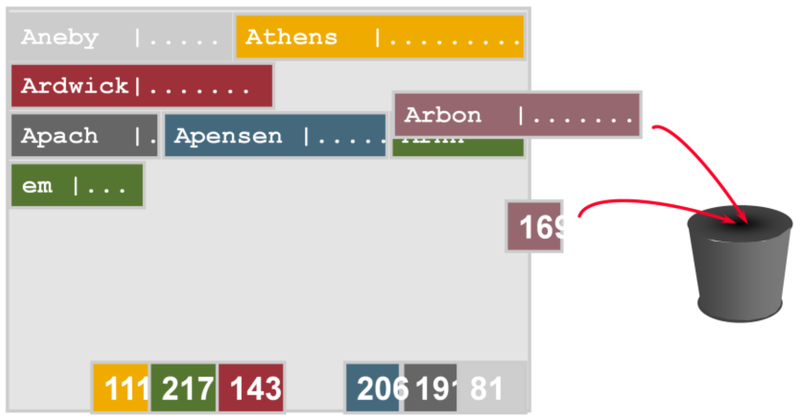

```
UPDATE address
SET street = 'Olymp 27'
KEY city = 'Athens'
```

© SAP AG 2012. All rights reserved. / Page 23

- Case 3: The length is changed, the key remains unchanged.
  The contents of the row are changed directly, but the position of the subsequent entries is different. Thus the subsequent records need to be moved and the address information (of the subsequent records) adjusted in the position list. If necessary, clearing operations are carried out.

If it is necessary to change the tree structure, first the required space is made available by way of B* tree clearing operations or by inserting a new block; then the UPDATE is carried out as described.

Delete in Place (1)



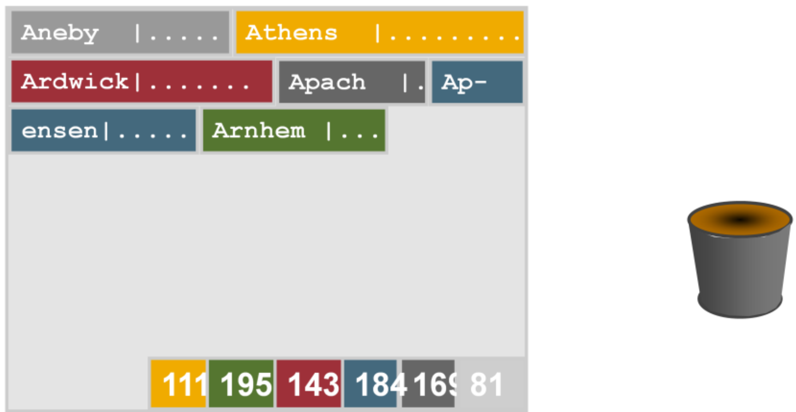

```
DELETE FROM address WHERE name = 'Arbon'
```

© SAP AG 2012. All rights reserved. / Page 24

delete in place

- Records are changed directly on the target page.
- The positions in the sorting list must be changed on the target page for all physically subsequent records
- If a certain usage level is not reached, a B* tree clearing operation is carried out

Delete in Place (2)




```
DELETE FROM address WHERE name = 'Arbon'
```

© SAP AG 2012. All rights reserved. / Page 25

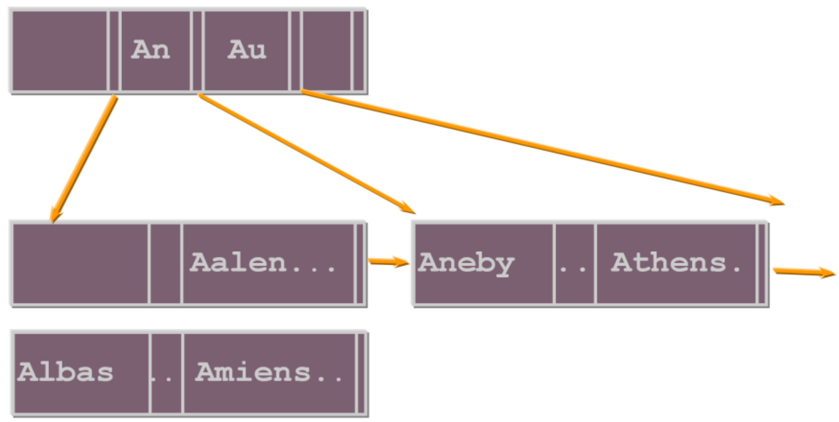
The records and the position list on the page are re-arranged so that the storage space used is contiguous.

All changes to pages are executed in the main memory. That makes them very fast, but also CPU-intensive.

If the fill level of a page falls below a certain mark, the tree structure is rearranged. An example of such a rearrangement will be shown later.

MaxDB offers the possibility of applying the attribute DYNAMIC to tables. Only very simplified clearing operations are carried out on these tables. Such tables require more space, but they offer noticeably higher performance. This attribute is suited to tables that are highly dynamic, in particular through random accesses and large fluctuations in the size of the table.

Inserting a Data Page (Page Split Operation) (1)



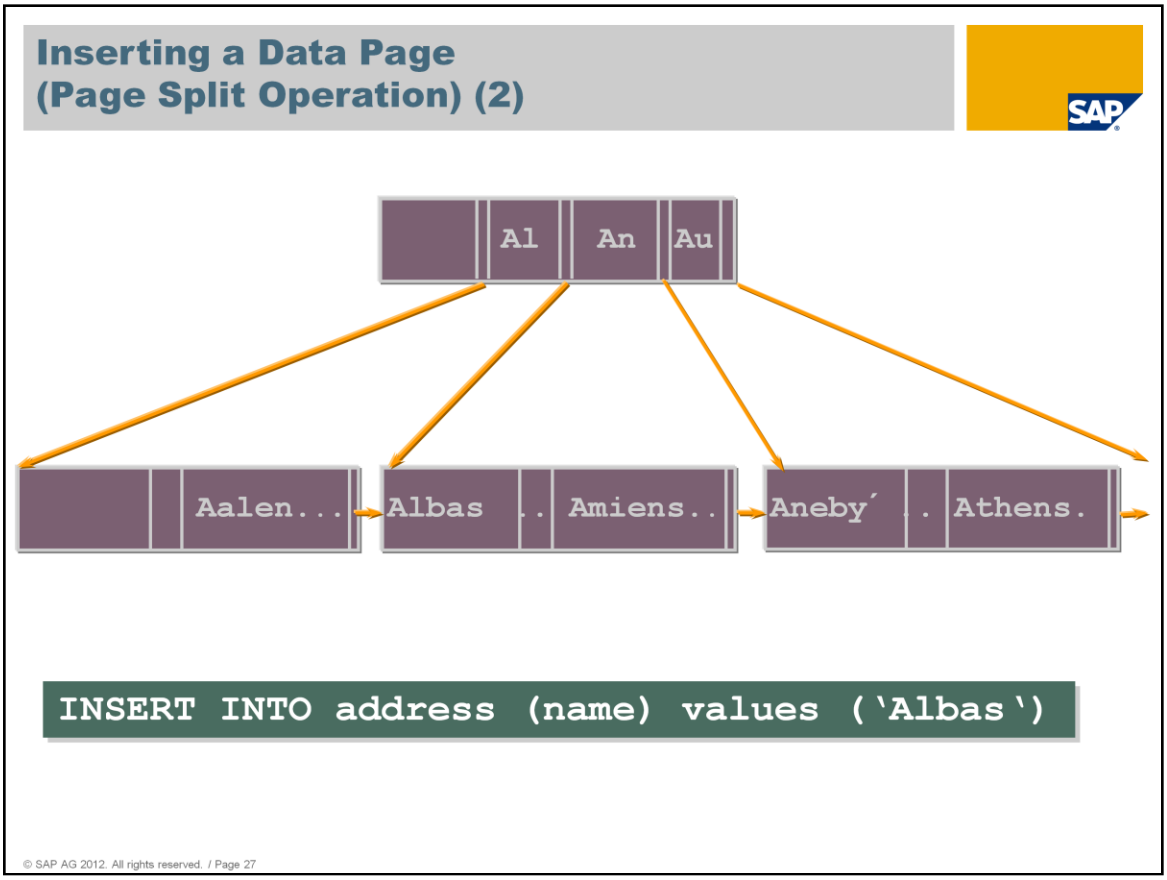

```
INSERT INTO address (name) values ('Albas')
```

© SAP AG 2012. All rights reserved. / Page 26

Now let's have a look at a simple change to the tree structure.

Let's assume that, due to an INSERT, the new data record no longer fits on the corresponding page.

A new page is then created on which the new record and half of the data records from the page that was too small for the INSERT are written. The respective records on the original page are then deleted.



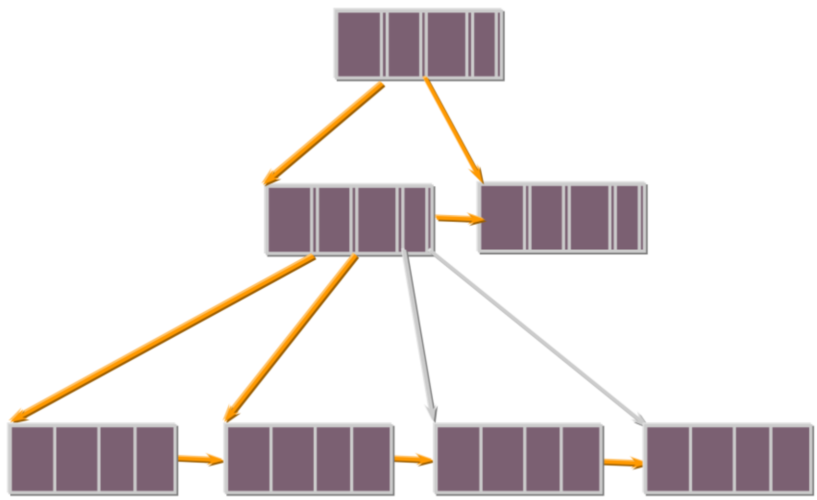
If necessary, the database system updates the pointers to the following pages.

In addition, the address and separator information for the new page is entered in the B* index page above it.

If this also does not fit on the B* index page, a new page has to be inserted.

If the B* tree is no longer able to accept the new page, that is, even in the root page there is no more space available in which to insert a new branch, the entire B* tree has to be expanded by a new level.

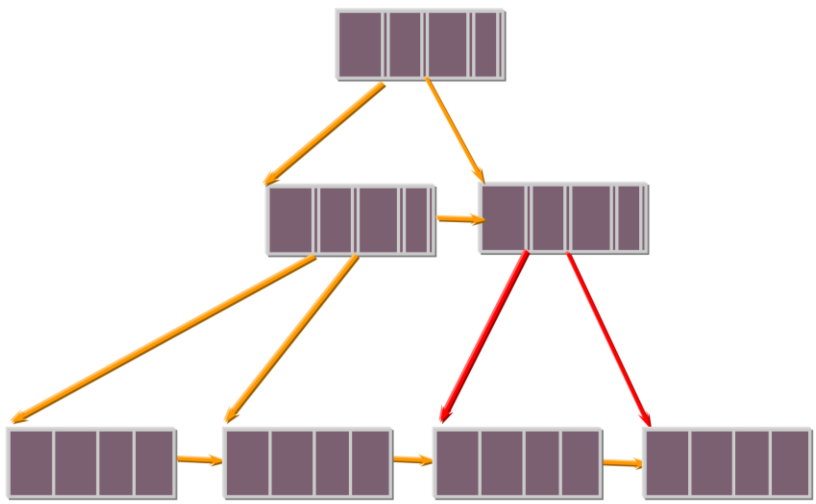
B* Tree Balancing (1)



© SAP AG 2012. All rights reserved. / Page 28

If the distribution of pages in the B* tree is unbalanced, that is, if there is an inordinate amount of pages on certain branches of the tree,...

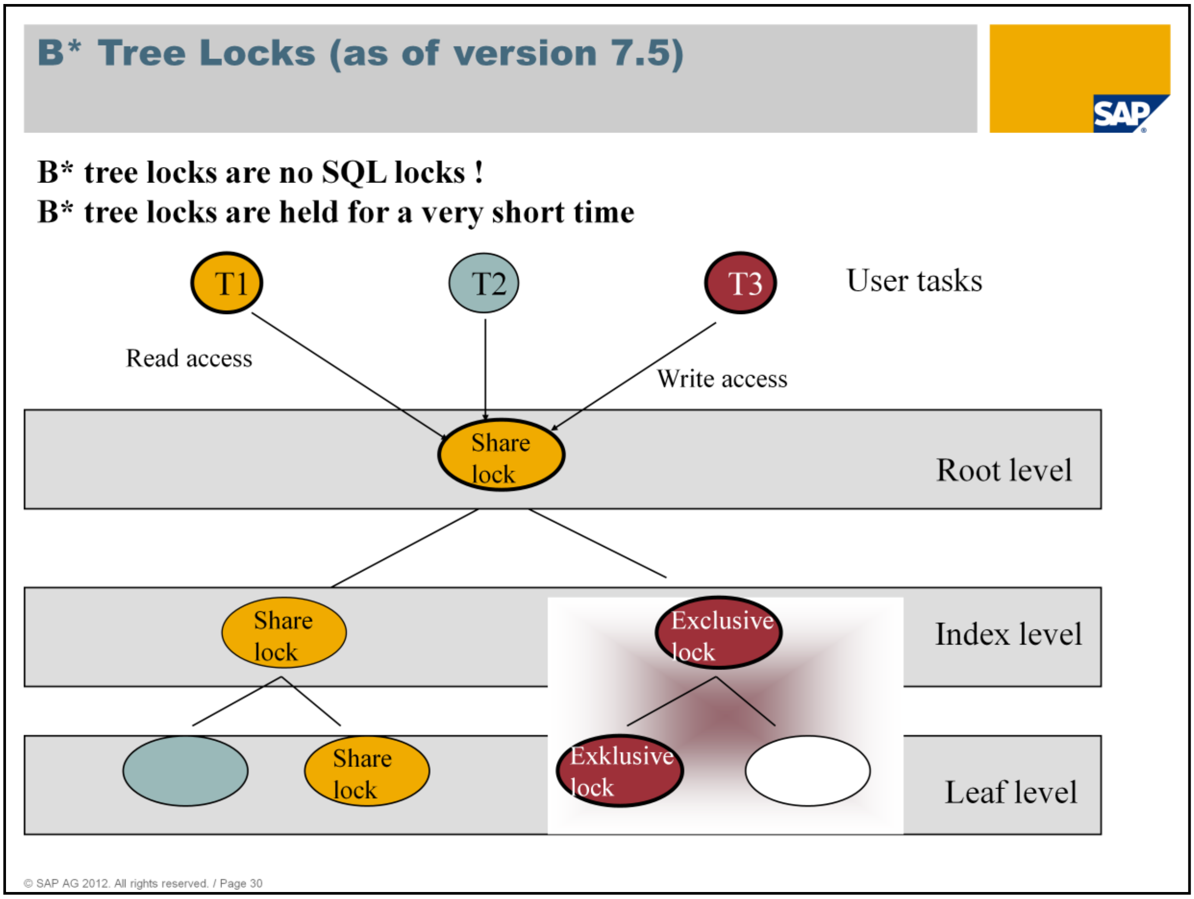
B* Tree Balancing (2)



© SAP AG 2012. All rights reserved. / Page 29

performance suffers because, on average, more accesses are needed to find data records.

Such states are recognized by MaxDB when INSERTs, UPDATEs and DELETEs are processed and the tree is rearranged in the affected subareas. This procedure is known as balancing. This involves moving records back and forth between pages in order to achieve the highest possible utilization of the pages.



© SAP AG 2012. All rights reserved. / Page 30

Each time a B* tree is accessed, the respective page must be locked. As of version 7.5, these locks are no longer managed in separate lock lists but rather directly in the data cache. A lock is requested when the desired page in the data cache is accessed.

Advantages as compared with the lock concept in versions 7.3 and 7.4: a significant characteristic, and thus also the biggest disadvantage of the old concept, was that the locks for the pages B* tree were managed in a separate component, the so-called tree lock list. Heavy parallel access to the list could lead to collisions.

Check Data / Check Table (VERIFY): In contrast to the SAP DB Versions 7.3 and 7.4, from version 7.5 this new concept makes it possible to execute change operations on the B* trees in parallel with Check Data or Check Table.

B* Trees

Exactly one B* tree for:

- **every** table
- **every** table with columns of type LOB for all short LOB values (< ca. 8 KB)
- **every** longer LOB value (> ca. 8 KB)
- **every** index
- subtrees of indexes

© SAP AG 2012. All rights reserved. / Page 31

MaxDB uses B* tree structures for the storage of all its tables.

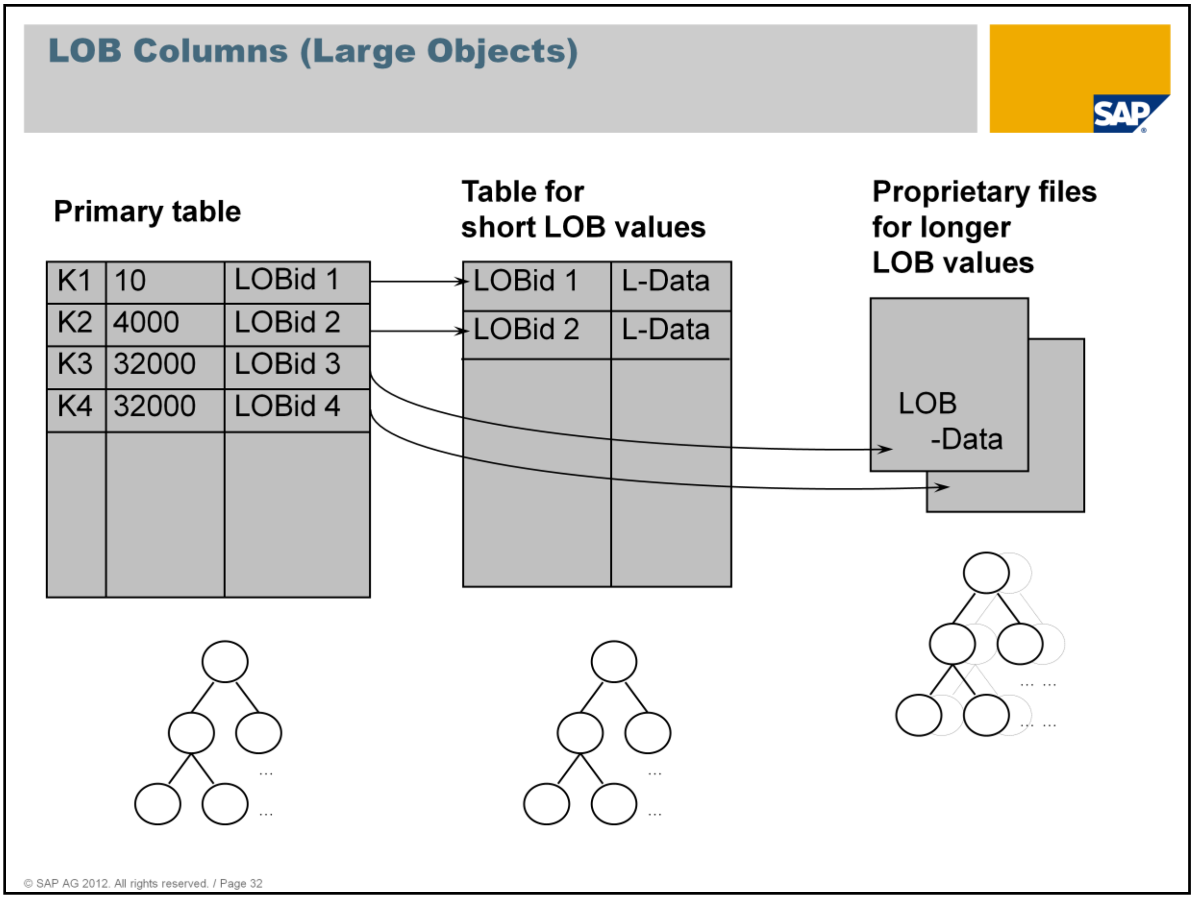
The term "table" includes:

- Primary data, including the associated LOB data (LOB Large Object)
- Secondary data as required for single and multiple secondary keys.

A MaxDB table always has a primary key. This is either a user-defined key or a generated internal key. A user-defined key can be comprised of several columns (multiple key).

The user can define additional secondary keys, which can also consist of one (single index) or several (multiple index) columns.

There is exactly ***one*** B* tree for the primary data of a table and also precisely ***one*** B* tree for each defined index (also known as: secondary key). If a table is defined with LOB columns, **one** additional B* tree is created for the purpose of accepting the LOB values in these columns that do not exceed a certain length. If LOB values are longer than this defined value, a new B* tree is created for every single one of these values.



This illustration shows a table with a LOB column. The number column represents the length of the LOB values. There is a B* tree for primary data, a B* tree for the shorter LOB values and *n* B* trees for *n* longer LOB values.

Irrespective of their length, for LOB values the primary table always has a single entry of a fixed length which refers to the respective storage structure.

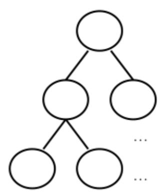
Indexes

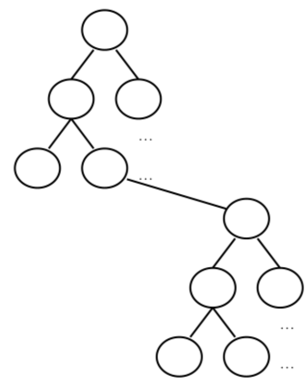
Primary table

K1	10	10	
K2	20	10	
K3	10	10	
K4	20	40	
K5	10	10	
K6	20	40	
K7	40	30	

Index table

1010	K1	K3	K5	...	...
2010	K2				
2040	K4	K6			
4030	K7				





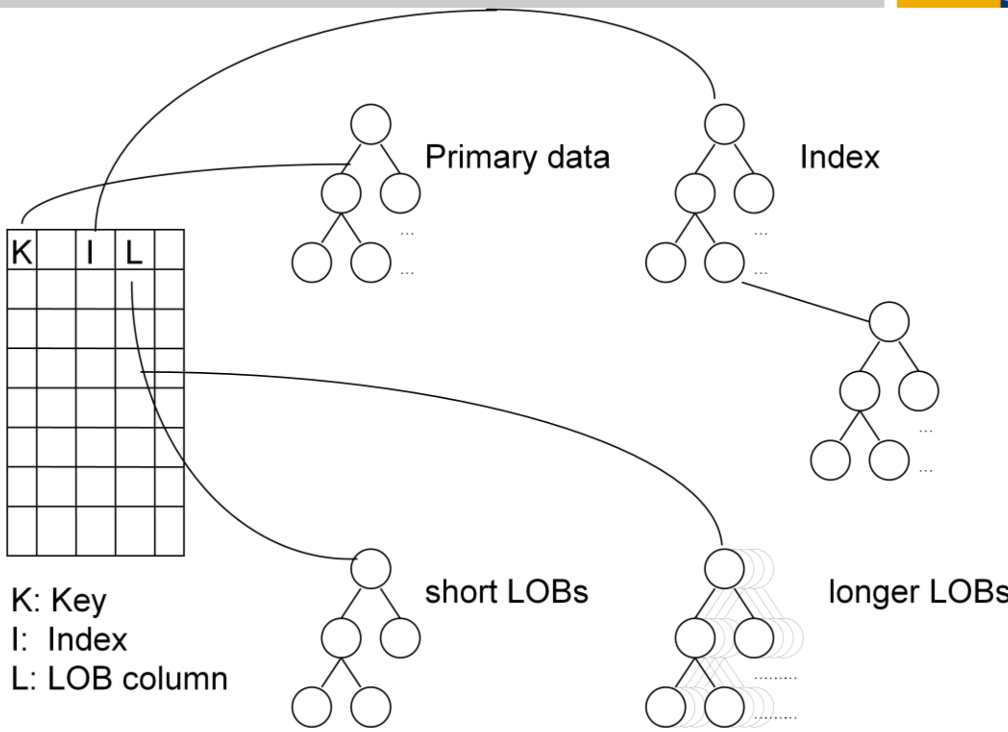
© SAP AG 2012. All rights reserved. / Page 33

This illustration shows an example of a table with a secondary key defined for multiple fields (2 fields). There is one B* tree for the primary data and a second B* tree for the indexed data.

The B* index is not to be confused with the term index as it is commonly used for secondary key definitions!

If the primary key values for a secondary key value cannot be contained on one data page, MaxDB stores the primary key values, sorted, in a separate B* tree. This means that the size of the secondary key tree can be significantly decreased.

B* Trees



© SAP AG 2012. All rights reserved. / Page 34

This illustration, taking the example of a table that contains LOBs and for which a secondary key has been defined, depicts how the assignment to B* tree structures works.

System View ROOTS

File-Directory				File-Directory		
TABLEID	OWNER	TABLENAME	INDEX NAME	TYPE	ROOT	FILE_ID
0000000000000CE1	SAPS13	CUEX	CUEX~1	NAMED INDEX	119047	C2EB5DA3FFFFFFFF7FFFFFFF0000FFFF07D1 01000100000A0701000000000000000CE10000 00000000
0000000000000CE1	SAPS13	CUEX	?	TABLE	119036	39EA5DA3FFFFFFFF7FFFFFFF0000FFFFFCD0 01000100000A0D00000000000000000CE10000 00000000
0000000000000CE1	SAPS13	CUEX	?	SHORT STRING FILE	119030	33EA5DA3FFFFFFFF7FFFFFFF0000FFFFF6D0 01000100000A1200000000000000000CE10000 00000000
0000000000000290	?	?	?	LONG COLUMN	4311	122A5CA3FFFFFFFF7FFFFFFF0000FFFFD710 00000100000C010000000000000000002900000 00000000
00000000000004BD	?	?	?	LONG COLUMN	4398	EB2B5CA3FFFFFFFF7FFFFFFF0000FFFF2E11 00000100000C01000000000000000004BD0000 00000000
Datenbank-Katalog						Datenbank-Katalog

Tables are internally administered by a 'tableid'

Mapping to B* trees via an entry in the file directory.

© SAP AG 2012. All rights reserved. / Page 35

A table, which is known to the user by a name, is internally administered with a 'tableid'. The correlation between the names and **tableid**s is registered in the database system dictionary (catalog).

There is also the database **file directory**, which contains the assignments of the root nodes of the B* trees to the tableids of the database objects. The tableids are stored in the file directory along with a type flag which indicates what contents the underlying B* tree has.

Thus a single tableid, in combination with the type flag, can be used to administer a table with all its associated B* tree entries in the file directory.

The system table ROOTS contains information from the file directory and the database catalog.

As of version 7.8 the system table ROOTS is no longer available. The view FILES (from the next slide) has to be used instead.

System View FILES

SQL Dialog 1

```sql
select f.*, t.tablename from files f, tables t
where f.primaryfileid = t.tableid or f.fileid = t.tableid
and t.owner = user
```

INS Ln 3, Col 19

FILEID	SESSIONID	ROOT	TYPE	PRIMARYFILEID	FILESTATE
000000000000048C	?	75476	TABLE	?	OK
000000000000048D	?	105246	INDEX	000000000000048C	OK
000000000000048E	?	30812	INDEX	000000000000048C	OK
000000000000048F	?	75477	INDEX	000000000000048C	OK
0000000000000490	?	15922	INDEX	000000000000048C	OK
0000000000000491	?	60587	INDEX	000000000000048C	OK

ENTRYCOUNT	TREEINDEXSIZE	TREELEAVESSIZE	LOBSIZE	TABLENAME
114199	144	14400	0	ZZTELE
2	104	9240	?	ZZTELE
513	8168	20504	?	ZZTELE
20001	48	10584	?	ZZTELE
5156	3144	14512	?	ZZTELE
10	48	9320	?	ZZTELE

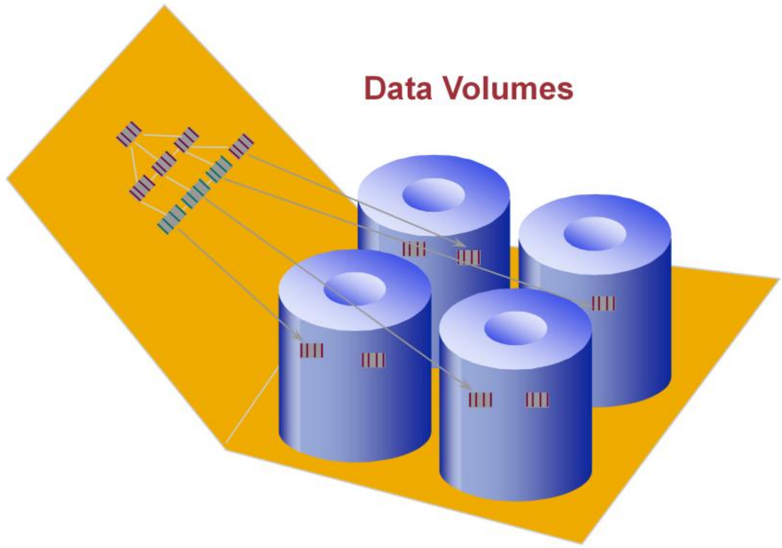
© SAP AG 2012. All rights reserved. / Page 36

As of Version 7.6, the FILES system view displays all information in the new file directory.

The user can specify the route to the database catalog in his SQL query. The columns of the FILES view mean the following:

FILEID	Corresponds to ID for tables, indexes, etc. in the catalog
SESSIONID	Creator session for temporary trees
ROOT	Root page number of the B* tree
TYPE	TABLE | INDEX | FIXED OBJECT | VARIABLE OBJECT | KEYED OBJECT | KEYED OBJECT INDEX | SHORT COLUMN FILE | internal file type for temporary files
PRIMARYFILEID	FILEID of the B* tree of the table
FILESTATE	OK | DELETED | BAD | READ ONLY
ENTRYCOUNT	Number of entries in the tree. For indexes, entries in subtrees are not included. NULL: Value was not yet determined for migrated systems.
TREEINDEXSIZE	Size of index level in KB
TREELEAVESIZE	Size of leaf level in KB
LOBSIZE	Size of all BLOB values of the table

Disk Striping



© SAP AG 2012. All rights reserved. / Page 37

An important role in the access performance of the database is playing by the MaxDB striping mechanism, which distributes the data pages evenly on the disks. Additional striping can be performed by the hardware.

Striping guarantees even distribution of the I/O load on the available disks.

Even load balancing of all the data areas in the database also prevents individual data areas from overflowing. A table can be larger than a single data area without the need for maintenance tasks to be carried out.

Questions and Answers

Thank You!
Bye, Bye – And Remember Next Sessions

August 28 and 29, 2012	Session 16: MaxDB SQL Query Optimization – Part I Session 17: MaxDB SQL Query Optimization – Part II

© SAP AG 2012. All rights reserved.