Dresden Database Systems Group

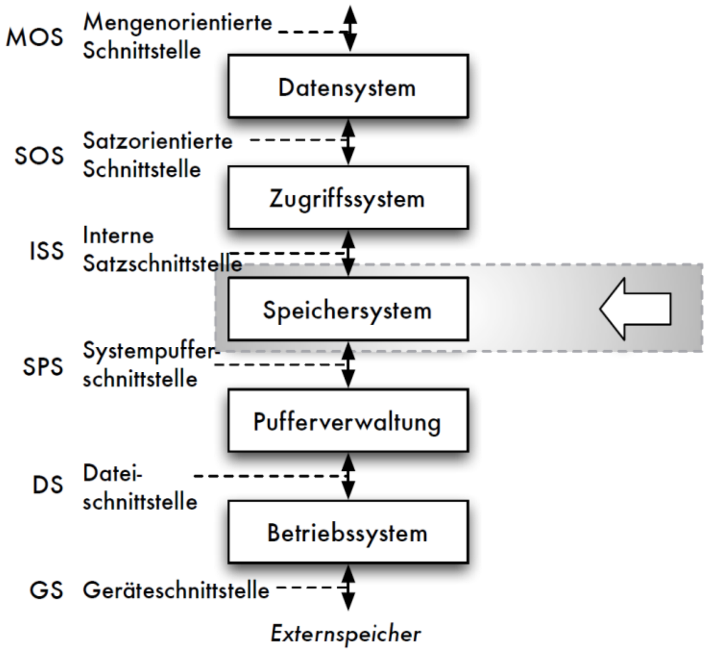
4. Row-based Record Management

Architecture of Database Systems

Classification in 5-Tier-Architecture

Dresden Database Systems Group

5-Tier-Architecture



Tasks

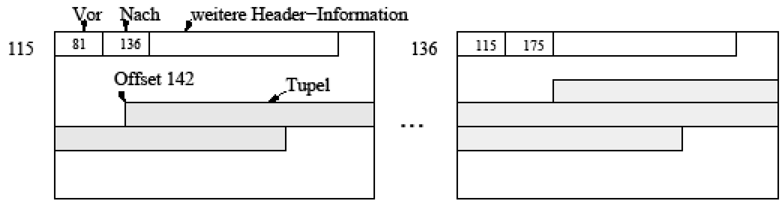
- **Storage system** requests pages through system buffer interface (Systempufferschnittstelle)
- Storage system interprets them as internal records
- Internal realisation of logical records via pointers, special index entries and further falsework
- Access system abstracts from the concrete implemention (further details in ADBS II)

Organisation of Sides (Review)

Dresden Database Systems Group

Concatenation

- Pages are linked together by double linked lists
- Recording of free pages: heap management (Freispeicherverwaltung)



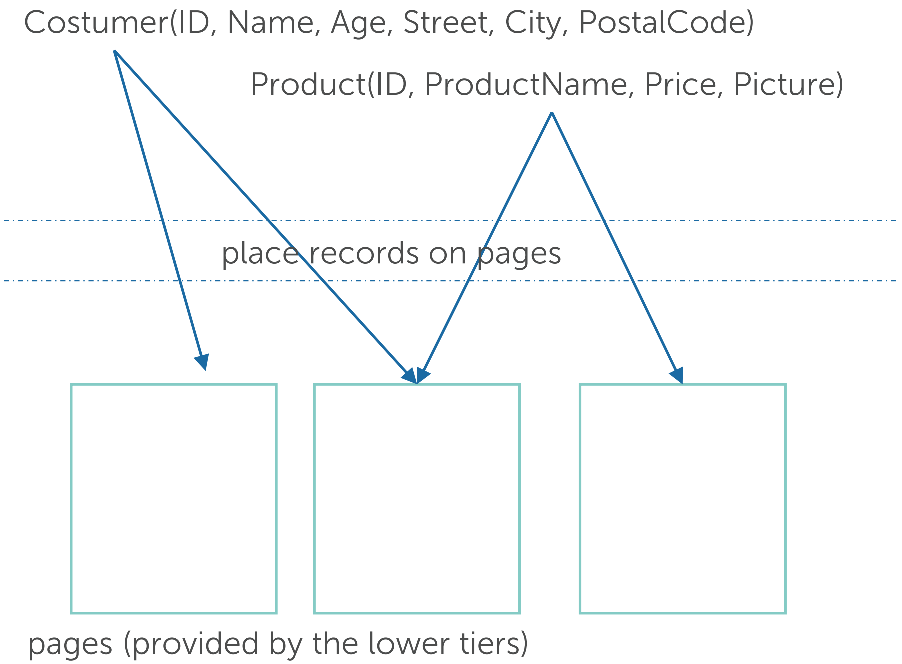
L_S = page size
L_{SK} = length of page header

Page Header

- Information about previous and and following page
- Optionally also number of the page itself
- Information about the type of record (Table Directory)
- Information about free space

Storage System-Requirement

Graphical Presentation



Key Aspects

- Representation of complete records on pages
- Organisation of structure formats for records
- Adressing of records
  - Allocation table
  - TID-concept
- Heap management (Freispeicherverwaltung)

Comparison to Key/Value-Store

Key/ Value Approach

- Get(Key) → (object, context)
  - Returns a list of data objects associated with key
  - More than one object only if there was a conflict
  - Returns a context
- Put(key, context, object)
  - Determine where replicas should be placed for associated key
  - Write the replicas to the disk
- Context
  - Encodes system metadata that the caller is not aware of
  - Includes versioning information

Differentiation

- Key – either an internal key (TID/RID) or an application key (in the worst case: composite key)
- Values – either known to the system or only known to the application

Assumption for now

- Key is “internal” / value structure is known to the system
- Value structure is preserved as a whole

Processing Unit ‘Record’

Definition “Record”

- Summary of data which belong to an object, a person, a circumstance etc. of an application and which describe the properties of the object
- Records are put together by fields
  (components of a struct in C).

Comment

- The structure of a record is irrelevant for the storage management on that level
  - every records has an arbitrary number of attributes → every record may have a variable length
- In this tier, a record is only a byte sequence which length is determined not any longer by the system but by the application!
- On this level of abstraction, a file is a (linear) sequence of records with a fixed or a variable length.

Tasks of the Record-Manager (Storage System)

- Physical storage / records organization on pages
- Operations: read, insert, update, delete

Mapping of Records on Sides

IMPORTANT

- **Until now:** page of fixed length as a processing unit
- **Now**: data record of variable length as a processing unit

DECOUPLING OF

- System given processing units (pages) (physical record)
- Data structure of an application (records) (logical record)

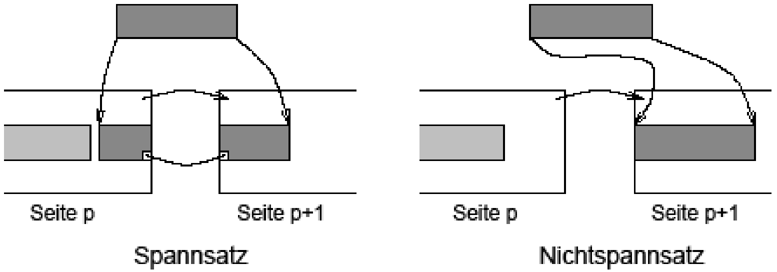
Mapping function:

Record
(logical record)

(physical record)
Page in Main Memory

REMEMBER: NO SPANNING RECORDS! (SPANNSATZ)

- The mapping of a single record may not cross the boundaries of a single page
- Large records are split into multiple smaller pieces at the logial level and connected via explicit record address



Record Formats

Format of Records

Dresden Database Systems Group

Requirements for Processing Efficiency and Flexibility

- Every record is identified by a **record indentfier** (SKZ (= Satzkennzeichen) or OID (=Object Identifier))
- Storage as space-saving as possible (storage economy)
- Extensibility of record type should be possible any time during the operation (requirement of application)
- Simple calculation of an internal record address of the n-th field (when access to only a part of records)

Record Description

- Description of record and access path in catalogue
- Special methods of storage
  - Blank-/zero suppression
  - Character compression
  - Cryptographic encryption
  - Symbol for undefined values (NULL values)
- Organization
  - n record types per segment
  - m records of different types per page
  - Record length < page length

Format of Field

Dresden Database Systems Group

Description of Attributes/Fields

- Name (often a difference between internal name of field and external name of type of attribute)
- Characteristic (fixed, variable, multiple)
- Length
- Type (alpha-numeric, numeric, packed, ...)
- Special methods for storage (zero suppression, character compression, cryptography etc.)
- If necessary symbol for a undefined value (if not globally defined as a segment or system constant)

The format description manages all operations on records.

Record Types

- Typically many records with the same structure/same arrays have a unique description in data dictionary for all
- **Record type**: set of records with the same structure gets a name (usually table name)
- Every records needs to be allocated to a record type when taken into storage
  (records without type are not allowed).

Record Types

Record Length of a Record Type

- Fixed, when all fields have a fixed length or when the maximal length is reserved for fields with flexible length
- Otherwise variable

Problem

- On which page is a record deposited and how can this record be found again, even if meanwhile several other records have been deleted and inserted?
- See record addressing!

Assumption

- Length of records is variable (general cases)
- Order of storage doesn't have to be the order of inserting
- Direct access to single records through their record address
- A record should be stored within a single page: $L_R \le L_S - L_{SK}$ (Standard)
- Several record types per page should be possible.

Storage Structure for Records

Example

- Record type PERS with 6 fields
  (f_i = fixed, length i; v = variable length)

PERS(PNR: f_5;
Name: v;
Job: v,
Salary: f_6,
City: f_2,
DName: v
)

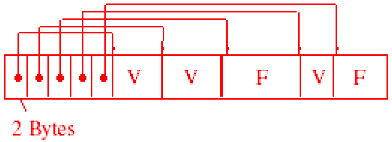
Option A)

- Concatenation of fields with fixed length

- Storage extensive
- Inflexible

Option B)

- Pointer in prefix

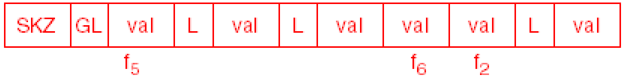


- Inflexible

Storage Structure for Records

Option C)

- Embedded length arrays

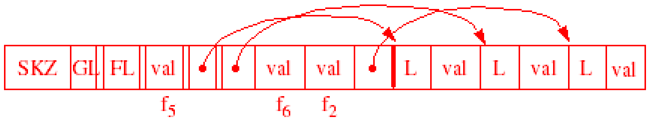


GL = Overall lenght

- Dynamic extension possible
- But: additional knowledge necessary:
  f_5 | v | v | f_6 | f_2 | v |
- Disadvantage:
  - Address of the n-th attribute cannot be calculated

Option D)

- Embedded length arrays with pointers
- Extension of C)



FL = lenght of fixed structure

- Address of the n-th attribute can be calculated

Variable Storages Structures

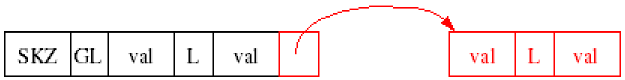
Dresden Database Systems Group

Problem

- Dynamic growth/variable length
  - Extension and contraction within a page
  - Overflow management
  - Garbage Collection

- Strictly connected storage of records
  - Possibly often rearrangement at a high change frequency
  - Advantage for indirect addressing schemata

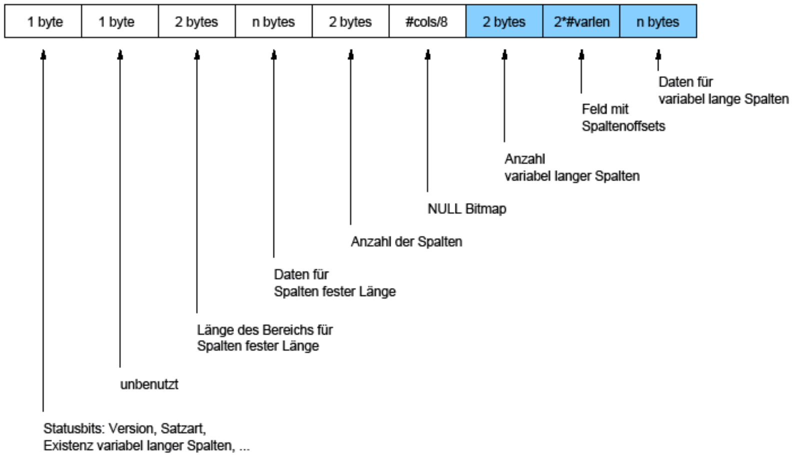
Split-up of a Record (Approach)



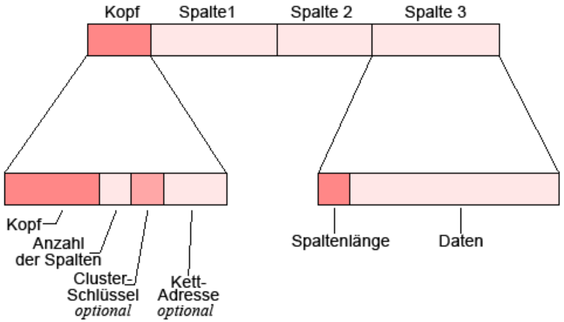
- Order according to frequency of reference
- Improvement of cluster formation
- Repeated overflow possible
- Unavoidable, when attributes TEXT or IMAGE are involved (except when saved as BLOB/CLOB)

Setup in Commercial Products

Microsoft SQL-Server



Oracle



Blocking Factor

Dresden Database Systems Group

Typically Several Records Fit into a single page (Record Length 100 – 1000 Bytes)

- Blocking factor: number of records per page

Assumption: No Spanning Records, i.e. Every Record is Put Completely within a single page

  - Blocking factor computable of page size and record length
  - Mostly unused storage space at the end of a page

- Variable record length
  - Blocking factor changes from page to page

Record Addressing

Record Addressing

Problem

- Long-term storage of data records
- Avoiding dependency on technologies
- Support of migration.

Record Addresses

- Record addresses are generated when inserting records and can be used later to get access to the records

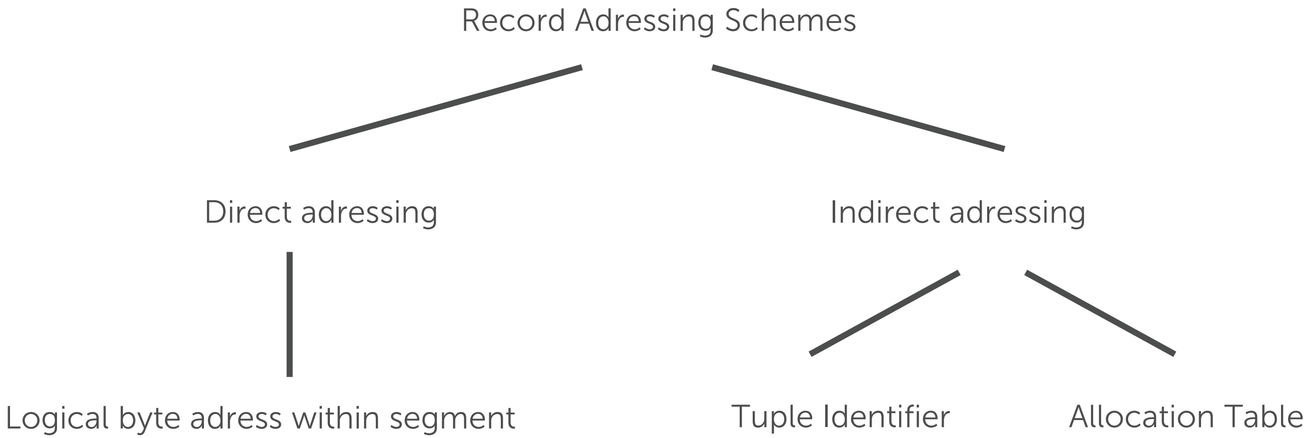
Goals of Addressing Technique

- Fast and preferably direct access to records
- Acceptably stable against slight displacements (displacements inside of a page without consequences)
- Rarely or no reorganisations

Common Form of a Record Address

- DBID, SID, TID and possibly qualification of relations (RID)
- Relation saved completely in a segment: TID DBID, SID in DB-catalogue
- Relation in several segments: SID, TID

Survey of Addressing Procedures



Direct Record Addressing

Dresden Database Systems Group

Running Number of Record

- Instable!
  The running number and therefore the record address changes while inserte and delete operations

Blocking Number and Byte-Position Inside of a Block

- Instable!
  If a record inside of a block changes its length, usually all the other records have to be shifted.
- If a record itself gets to long for the block, it has to be placed to another block; then the block number changes as well.

Addressing in Segments

- Logically connected addressing space
- Direct addressing (logical byte-address)
  - -> Instable by displacements
  - -> Therefore indirect addressing

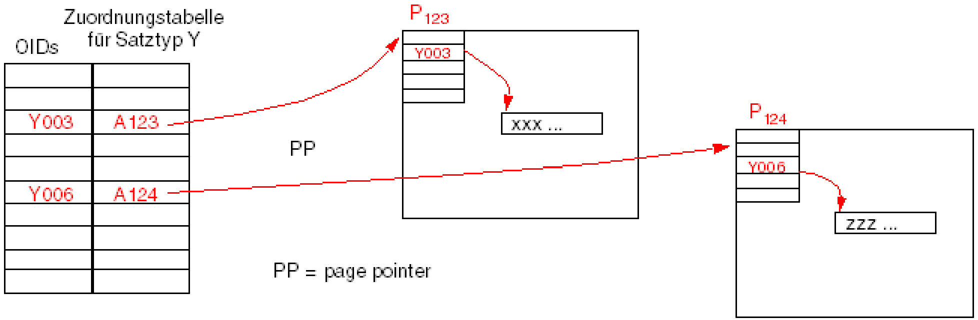
Allocation Table

Dresden Database Systems Group

Record Addressing by Allocation Tables (Complete Indirection)

- Idea:
  administration of array in consecutive pages of the segment that specifies the page number of every index.
  - Inserting of a record
    generally allocation of new index (database key DBK) by the database system
  - Deleting of a record
    the data set entry of the corresponding index is marked invalid
  - Access to a record
    requires 3 record accesses: one for the array, one for the page with with the record itself
  - Relocation of a record to another page
    only the data set entry is changed; the index remains stable – the record is still findable by the index
  - Inside of a page
    the data set entry in the array has to be changed and written again on disk (additional E/A-operation)
  - The allocation table itself takes some storage

Allocation Table (2)



Comment: The DBK Is a „Non-speaking" Address (Telephone Number)

- The database key is built by a record type r and a sequence number f. r and f identify the record during its full lifetime within the DB.
- Table entry eferences to page P_k in segment S_i.

Problem: Where is the Allocation Table Saved?

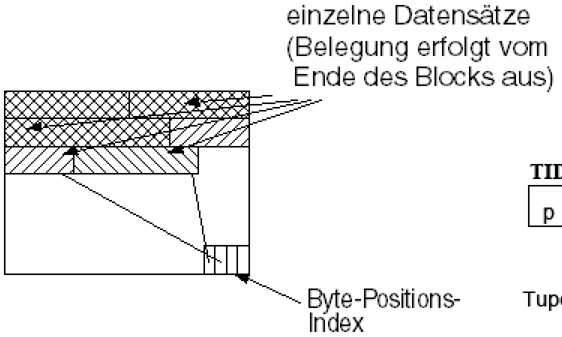
- At the beginning – how to expand?
- At the end – how to expand the data array?
- In an own segment?

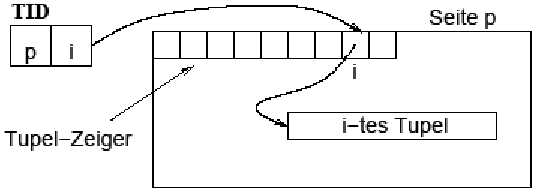
TID-Concept

Record Addressing through Indirection Inside of a page

- Array with byte-positions of records in this page
- Address is a pair consisting of a page number and an index on that array ("TID = Tuple IDentifier")

- Only one (!) page access is needed for the access to a record
- Structure of a page:





TID-Concept (2)

Dresden Database Systems Group

Deleting a Record

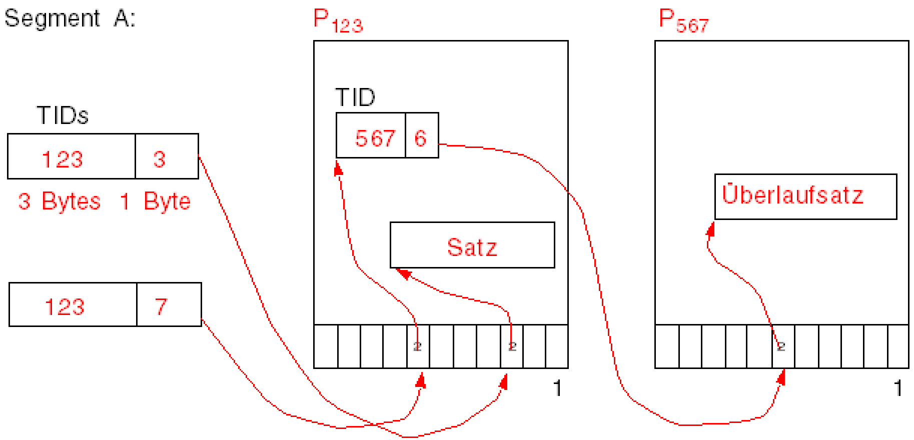
- The corresponding entry of a page array is marked invalid
- All the other records on the same pages can be moved to maximize the free space only the start address change in the page array.
- All page addresses remain stable

Update-Operation of a record

- Can change the length of a data set!
- Data set gets smaller or bigger (without overflow):
  - Every record is moved inside of a page and the byte positions are adjusted accordingly
- Data record gets bigger and there is not enough free space in the block for the storage of the now bigger data record (overflow):
  - Displacement of record to another page!

Overflow Handling TID-Concept

RELOCATION OF A RECORD



Overflow Handling TID-Concept (2)

Dresden Database Systems Group

PROCEDURE

- The new record address referring to the new page remains at the old page at the place of the original record (home address is stable)
  - In that (rare) case, two pages must be read
  - If the record is moved another time, the record address in the first page changes, so there is maximum one indirection

TRICK

- The length of the overflow chain is always smaller or equal 1, i.e. a overflow record must not continuing to overflow, but has to be replaced from its home address.

PROS

- No allocation table
- A record can be moved inside of a page and across pages without changes of the TID

Free Space Administration

Free Space Administration

Situation

- Example: where is enough space to save a new record?

Free Place Administration (FPA)

- In a table F_i to the segment S_i is the information for every page s_k, how many bytes are still free in there
- $F_i(k) = n$ <-> on page s_k of the segment S_i are bytes free.

Problem

- How big is the FPA-table?
- Where (on which pages of a segment) is the PFA-table saved?

Storage Costs for Heap Management

- With
  - L_S = page length
  - L_{SK} = length page header for the describing information on a page
  - L_F = length of the entry (usually 2 byte)
- Result
  - $k = (L_S - L_{SK}) / L_F$ = number of entries per page
  - s = number of pages in the segment
  - $n = s / k$ = taken pages in the segment

Free Space Administration

Storage Location

- Equally spaced allocation of the table pages according to (i*k +1) with i = 0, 1, 2, ..., n – 1 i.e. a table page stands before the k pages for what it saves free storage information.
  - Pro: segment can easily be expanded
  - Con: search for free space „jumps" through the segment
- Therefore the first n pages are usually reserved for the FPA-table when the page is addressed directly
  - con: expanding of the segment
- When the pages are addressed indirectly, the free storage information is kept within the page table

Seite	1	2	3	. . .	s
Block	i	j	k	. . .	r
Freier Platz	F(1)	F(2)	F(3)	. . .	F(s)

Management of Large Objects

Large Unstructured Records

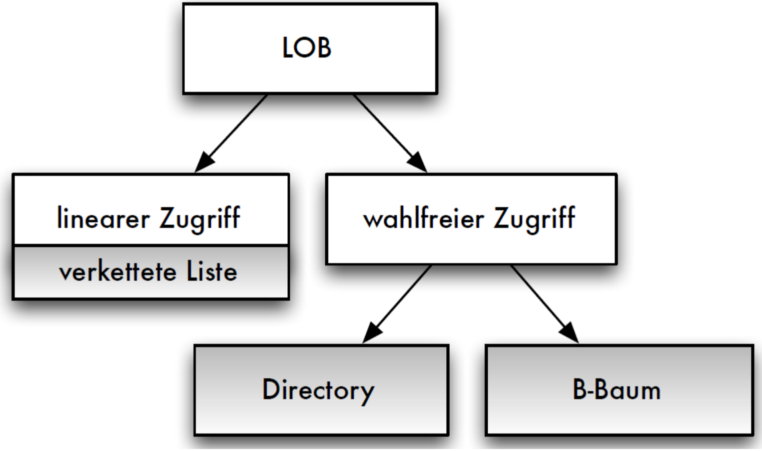
Dresden Database Systems Group

Data Types for very Large, unstructured information

- Binary Large Objects (BLOBs): byte-order as images, audio and video sequences
- Character Large Objects (CLOBs): orders of ASCII-symbol (unstructured ASCII-text)

Observation

- Long arrays don't exceed pages
- NON-BLOB-fields are stored at the original page

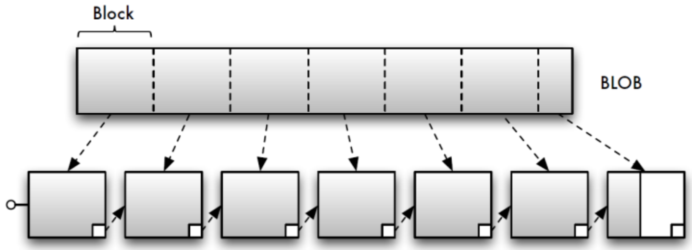


BLOB-Storage

Dresden Database Systems Group

Variation 1

- As attribute value pointer: pointer shows to the beginning of a page or block list that takes BLOB



- Advantage for inserting, deleting and converting
- Disadvantage for random access into the BLOB

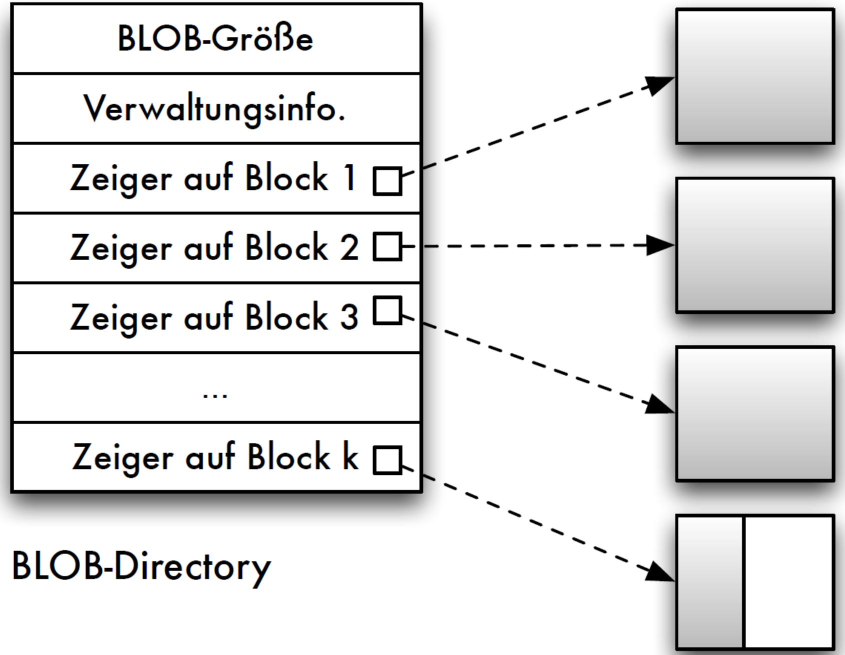
BLOB-Storage

VARIATION 2

- As attribute value BLOB-Directory:
  - BLOB-size
  - Further administration information
  - Several pointers that point to the single pages

- Advantage: fast access to subarea of the BLOB
- Disadvantage: fixed, restricted maximum size of the BLOBs
  (Gigabyte-BLOB: 8-Byte-addressing; page size 1KB→ 8MB for a BLOB-Directory)

- More efficient: B-tree for storage of BLOBs



Management of External Data

Management of External Data

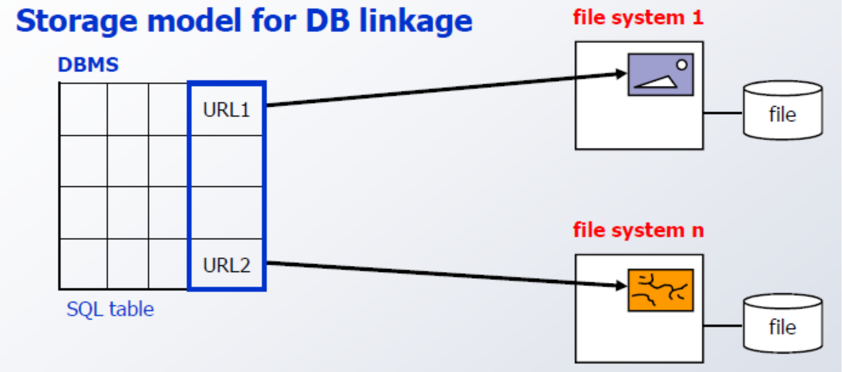
Dresden Database Systems Group

Motivation

- More and more data is (still) saved in files
- Many applications are working in a file-based fashion, native file access has to be supported
  - CAD solutions, Multimedia objects (movies) HTML and XML files

Drawbacks

- File systems do not support classical DB features
  - Referential integrity
  - Fine-grained access control
  - Consistent backup and recovery
  - Transactional consistency/isolation/…
  - Sophisticated support for an efficient search
- DBMS are tuned to work on well-structured (and potentially) large datasets

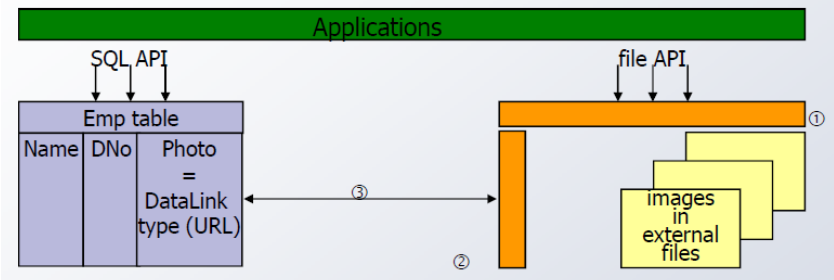


Goal

- Combination of file systems and DBMSs as best-of-breed approach

DB Links

Application Support



DataLinks File System Filter (DLFF)

- Enforces referential integrity when files are renamed or deleted
- Enforced db-centric access control when a file is opened
- File API remains unchanged in the read/write parth for external files
- DLFF does not repage in the read/write parth for external files

DataLinks FileManager (DLFM)

- Executes Link/Unlink operations under transaction protection
- Guarantees referential integrity
- Supports coordinated backup/recovery

DBMS manages/coordinates operations on external files

- Files referenced by URLSs or via DLFM API

DB Links - Options

Dresden Database Systems Group

LINK CONTROL

- NO LINK CONTROL: URL-format of data link; no further control
- FILE LINK CONTROL: existing file must be referenced; type of control determined by further operations

INTEGRITY (INTEGRITY CONTROL OPTION)

- INTEGRITY ALL: referenced files can only be deleted or renamed by SQL
- INTEGRITY SELECTIVE: referenced files can be deleted or renamed by file-manager-operations as long as there is not data linker
- INTEGRITY NONE: referenced files can only be deleted or renamed by file-manager-operations -> not compatible with FILE LINK CONTROL

READ PERMISSION (READ PERMISSION OPTION)

- READ PERMISSION FS: read permission for referenced files is determined by the file-manager
- READ PERMISSION DB: read permission for referenced files is determined by SQL

WRITE PERMISSION (WRITE PERMISSION OPTION)

- WRITE PERMISSION FS: write permission for referenced files is determined by the file-manager
- WRITE PERMISSION BLOCKED: no writing access to referenced files, unless there is an implementation independent mechanism
- WRITE PERMISSION ADMIN [NOT] REQUIRING TOKEN FOR UPDATE: write permission for referenced files is determined by SQL

RECOVERY (RECOVERY OPTION)

- RECOVERY YES: with data base servers coordinated recovery (data linker-mechanism)
- RECOVERY NO: no recovery for referenced files

UNLINKING (UNLINK OPTION)

- ON UNLINK RESTORE: existing permission before setting up the link are restored by the file-manager when unlinking
- ON UNLINK DELETE: Deleting when unlinked
- ON UNLINK NONE: not consequences to the permissions when unlinked

SQL-Functions for DataLinks

Dresden Database Systems Group

New SQL-Functions

- Constructor: DLVALUE, …
- (components of) URLs: DLURLCOMPLETE,

Examples

```
INSERT INTO Movies (Title, Minutes, Movie)
       VALUES('My Life', 126,
               DLVALUE('http://my.server.de/movies/mylife.avi'))
```

```
SELECT Title, DLURLCOMPLETE(Movie)
  FROM Movies
 WHERE Title LIKE '%Life%'
```

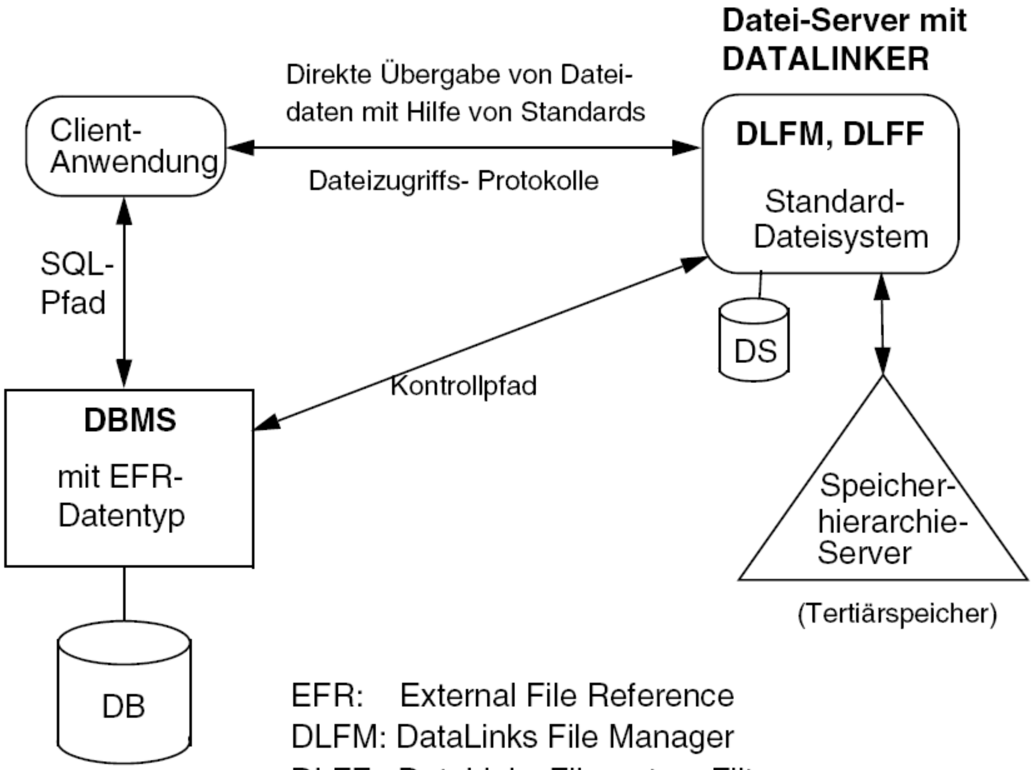
```
UPDATE Movies
  SET Movie =DLVALUE('http://my.newserver.de/mylife.avi')
  WHERE Title = 'My Life'
```

```
SELECT Title, DLURLCOMPLETEWRITE(Movie) INTO :t, :url ...
```

```
UPDATE Movies
  SET Movie = DLNEWCOPY(:url, 1)
  WHERE Title = :t
```

DBLinks Architecture

Dresden Database Systems Group



EFR: External File Reference
DLFM: DataLinks File Manager
DLFF: DataLinks Filesystem Filter

Summary

Dresden Database Systems Group

Representing Records

- Storage of variably long arrays
- Dynamic options of expansion
- Calculation of array addresses

Goals for the external storage based Addressing

- Combination of speed of direct access with the flexibility of an indirection
- Record displacements o a page without consequences

Alternatives

- TID-concept
- Allocation table

Storage of Big Data Sets (BLOBS)

Administration of External Data