

Representing Data Elements

Chapter 3

Outline

- Data elements and fields
- Records
- Representing Block and Record Addresses
- Variable-length Data and Records
- Record modifications

Introduction

- Attributes need to be represented by fixed- or variable-length sequences of bytes, called “fields”
- Fields, in turn, are put together in fixed- or variable-length collections called “records”.
- Records are stored in blocks. They can be reorganized when the database is modified.
- A collection of records that forms a relation is stored in a collection of blocks.

Representing relational data elements

- Consider the following DDL
- **CREATE TABLE MovieSrar(
 name char(30) PRIMARY KEY,
 Address VARCHAR(255),
 Gender CHAR(1)
 Birthdate DATE
);**
- How do we represent SQL data types as fields ?
- How do we represent tuples as records ?
- How do we represent a collection of records in a blocks of memory ?
- How do we represent and store relations as collections of blocks ?
- How do we cope with record sizes that may be different for different tuples or that do not divide the block size evenly or both?
- What happens if the size of the record changes because some field is updated ? How do we find the space in the block when the record grows ?
- How to represent objects and blobs ?

Representing Data Elements

- We have only bytes
 - 8-bit units
- We want to represent
 - a salary
 - a name
 - a date
 - a picture

To represent:

- Integer (short): 2 bytes
e.g., 35 is

00000000	00100011
----------	----------

- Real, floating point
 n bits for mantissa, m for exponent....

To represent:

- Characters

→ various coding schemes suggested,
most popular is ascii

Example:

A: 1000001

a: 1100001

5: 0110101

LF: 0001010

To represent:

- Boolean

e.g., TRUE

1111 1111

0000 0000

FALSE

- Application specific

e.g., RED → 1 GREEN → 3

BLUE → 2 YELLOW → 4 ...

To represent:

- Boolean

e.g., TRUE

1111 1111

0000 0000

FALSE

- Application specific

e.g., RED → 1 GREEN → 3

BLUE → 2 YELLOW → 4 ...

To represent:

- Dates

e.g.: - Integer, # days since Jan 1, 1900

- 8 characters, YYYYMMDD

- 7 characters, YYYYDDD

(not YYMMDD! Why?)

- Time

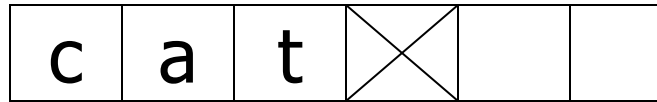
e.g. - Integer, seconds since midnight

- characters, HHMMSSFF

To represent:

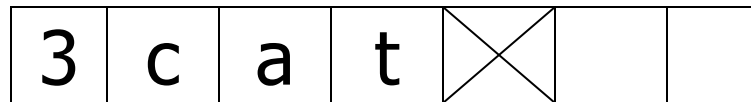
- String of characters
 - Null terminated

e.g.,



- Length given

e.g.,



- Fixed length

To represent:

- Bag of bits

Length	Bits
--------	------

Overview

Data Items



Records



Blocks



Files



Memory

Record - Collection of related data items (called FIELDS)

E.g.: Employee record:

name field,

salary field,

date-of-hire field, ...

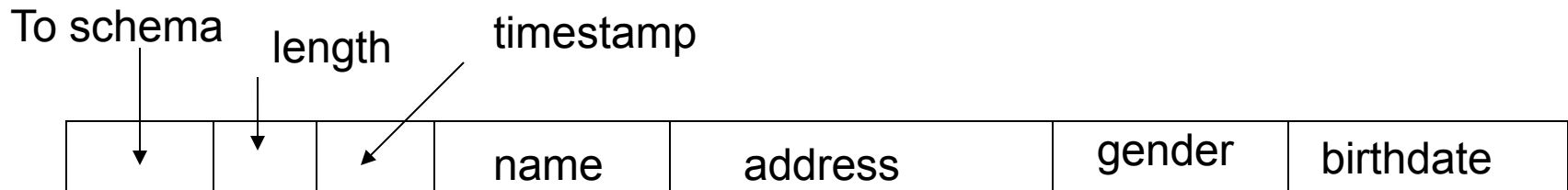
- Each type of record has schema, which is stored by the database.
- The schema includes data types of fields in the record and their offsets within the record
- The schema is consulted when it is necessary to access components of record.

Fixed-Length Records

- Simplest situation
 - All the fields have the same length
- Consider a declaration of movie star relation
 - Name, 30 byte string of characters
 - Address of the type VARCHAR(255)
 - Need 256 bytes
 - Gender: one byte
 - Birth date: SQL2 representation, 10 bytes
- So record of type movie star takes $30+256+1+10=257$ bytes.
 - Field name begins at 0, address begins at offset 30, gender at 286 and birth date at offset 287.
- Some machines read the data at multiples of four.
 - If the tuples of the relation are stored in disk, we have to aware of this issue.

Record headers

- Designing a layout of record.
 - We want to keep more information
 - Pointer to schema
 - Length of the record (to know the beginning of the next record)
 - Timestamps indicating the time last modified
- The database schema maintains schema information
 - The attributes of the relation
 - Their types
 - The order in which attributes appear in the tuple
 - Constraints on the attributes and the relation itself.
 - Primary key
- If we put a pointer to schema then all the information can be obtained when needed.
 - Even though length can be deduced from schema, it is better to keep in the record.



Packing Fixed-Length Records into Blocks

- Records are stored in blocks of the disk and moved into main memory.
- The block header holds the following

header	Record 1	Record2	...		Record n
--------	----------	---------	-----	--	----------

- The block header holds the following information
 - Links to one or more other blocks
 - Information about role played by a block
 - Information about which relation the tuples belongs to
 - A directory giving the offset of each record in the block
 - A block ID
 - Timestamp, indicating the last modification to the block

Representing Block and Record addresses

- At the secondary storage, block is not a part of virtual memory address.
- Sequence of bytes describes the location of the block within the overall system of data accessible to DBMS.
 - Device id for disk, the cylinder number, and so on.
 - A record can be identified by giving its block and the offset of the first byte of the record within the block

Client-server systems

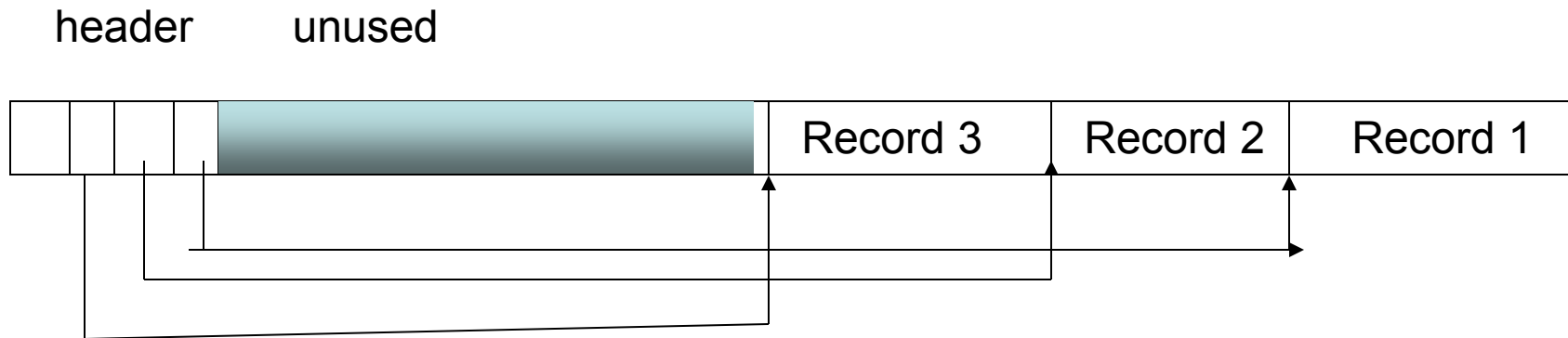
- Database consists of a server process that provides data from secondary storage to one or more client processes.
 - Client application uses a conventional virtual address space, typically 32 bits or about 4 billion different addresses.
 - OS virtual address space to physical addresses.
- The server data lives in database address space
 - Physical addresses
 - Host address, disk id, cylinder of the disk, track number, block number, offset.
 - Logical addresses
 - Arbitrary string of bytes of some fixed length.
- Map table relates to logical addresses to physical addresses.

Regarding logical and physical addresses

- May combinations are possible.
 - Use physical address in the block in the map table.
- We need enough information to locate a record in the block

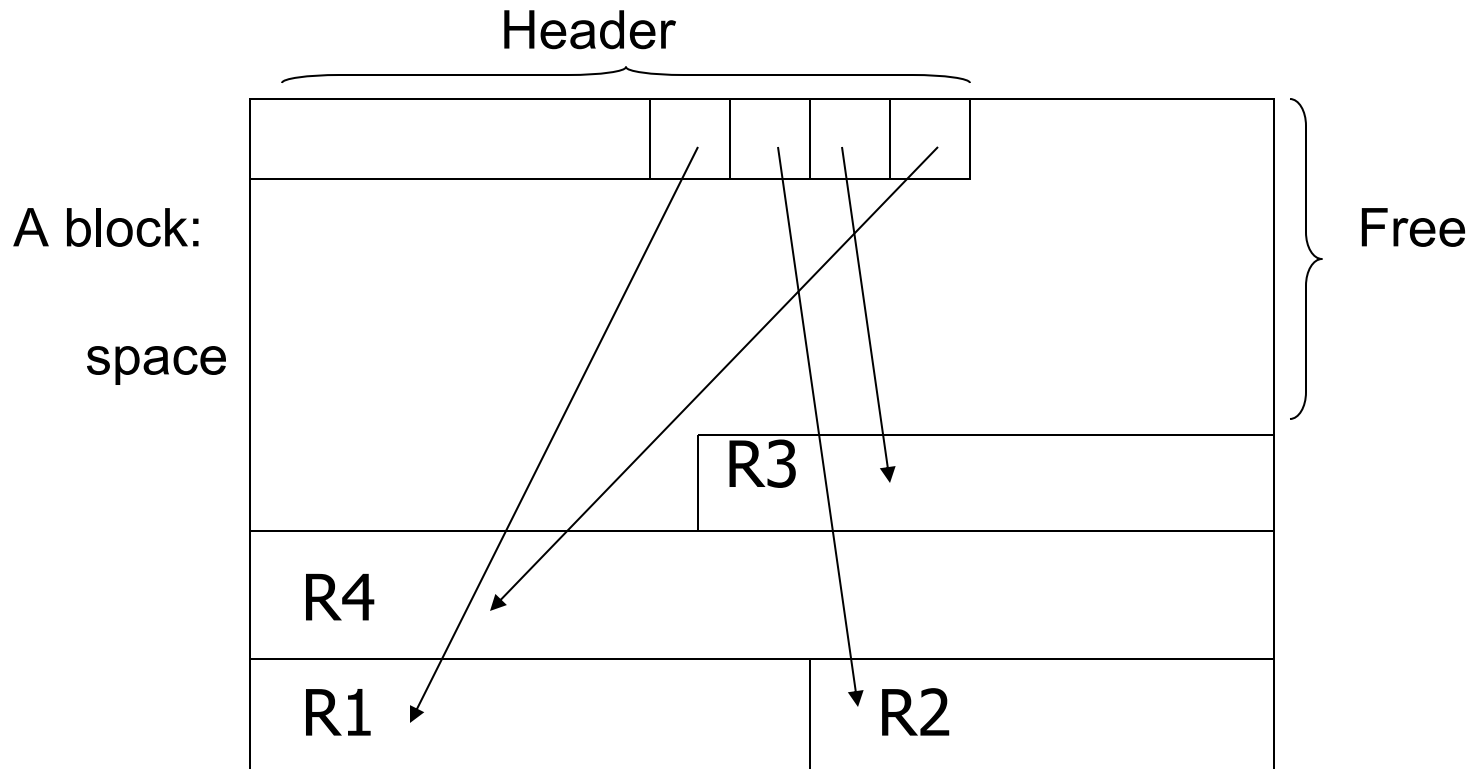
Logical and structured addresses

- Table of offsets can be used to know the position of each record.



- Advantages: modify block without modifying the map table.
 - Modification requires only changing of record entry.
 - Allow record go to other block
 - Change the offset (with forward address)
 - Deletion of record by writing a special character (tombstone).
Pointers to this record will know tombstone. Otherwise, erroneous information can be displayed.

Example: Indirection in block



Two forms of addresses for every block, record, object

- Database address: Address in server address space
 - Sequence of 8 or so bytes which locates the item in the secondary storage.
- Memory address: Address in virtual memory
 - It is four bytes
- Issues
 - When in secondary storage we should use database address
 - When in memory, we can use either database address or memory address.
 - Following a database address is time consuming.
 - We need a translation table

Pointer Swizzling

- Pointers or addresses of records.
- Database address
 - Sequence 8 bytes to locate the item in the secondary storage
- Memory address: Four bytes
- The translation table maps database address to memory address.
- Pointer Swizzling
 - Techniques to avoid repeatedly translating of database addresses to memory addresses.
 - When we move a block from secondary to main memory, pointers within the block may be swizzled, that is translates from database address space to the virtual address space. A pointer actually consists of
 - A bit indicating whether pointer is a database address or a swizzled memory address
 - The database of memory pointer as appropriate
 - Not all the space may not be used

Swizzling strategies

- Automatic swizzling
 - As soon as block comes to main memory we locate all the pointers and addresses and enter into translation table.
- Swizzling on demand
 - Do not swizzle when the block first brought to main memory.
 - Whenever we follow a pointer we swizzle the corresponding address.
- No swizzling
 - Follow pointers in unswizzled form.

Returning blocks to disk

- When a block is moved out to disk, the pointers should be unswizzled
 - Memory addresses must be relaced with corresponding database addresses
 - Appropriate data structures may be employed for searching
 - hash

Pinned Records and Blocks

- Pinned block
 - If we can not safely written back to disk.
 - Known by reading the header of the block
- If a block B1 has within it a swizzling pointer to some data item in block B2, we must be very careful moving block B2 back to disk and reusing main-memory buffer. If we move and reuse the main memory then, the pointer in B1 is dangling pointer.
 - So block B2 is pinned.
- So when we write blocks to disk
 - Unswizzeled
 - Ensure that it is not pinned.
- Use indexes for searching items in translation table

Variable Length Data and Records

- So far, fixed length and fixed schema.
- Life is more complex
 - Data item size varies: Ex. addresses
 - Repeating fields: number of movies acted
 - Variable format records: adding attributes
 - Enormous fields: picture, video file

Records with Variable-Length Fields

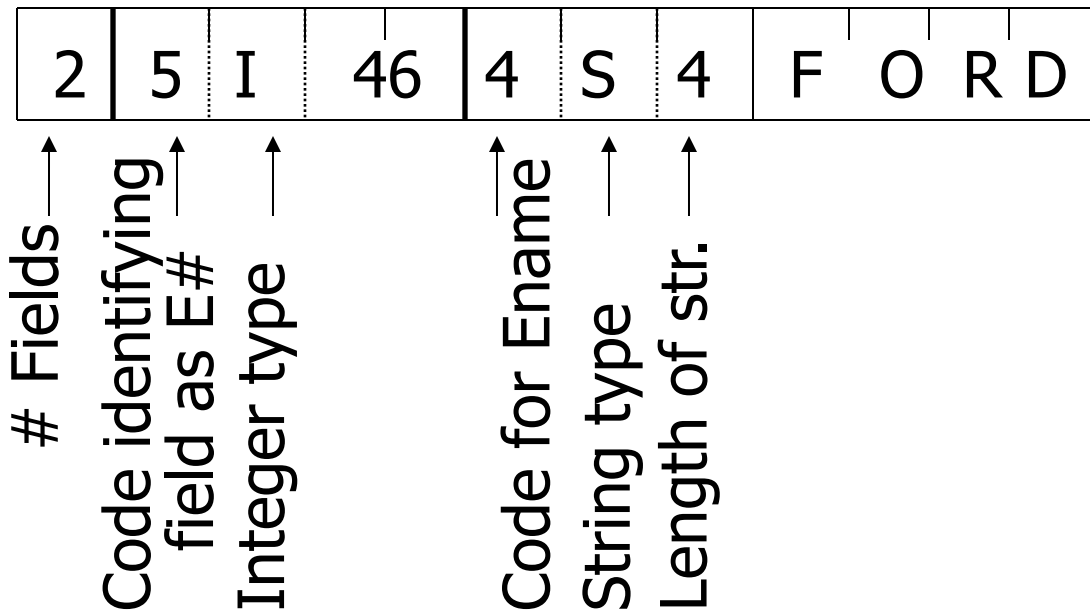
- Record header should contain
 - length of the record and
 - pointers (offsets) to variable length fields.

Records with Repeating Fields

- Field itself is fixed length
- Group all the occurrences and put a pointers to the first
 - Locate by adding offset
- Alternative representation
 - Fixed portion in one block with pointers
 - Variable portion in another block
 - Pointers to the place where each repeating filed begins
- Searching is difficult in variable length records.

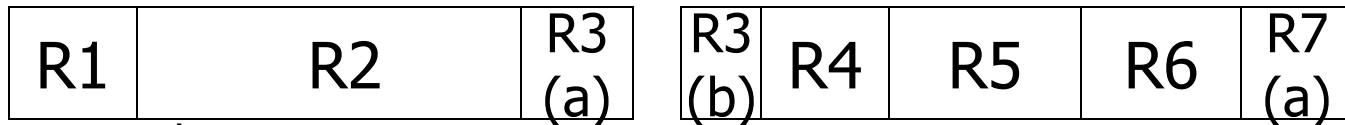
Variable Format Records

- If schema varies ?
- Order is not determined by schema ?
- Simplest representation
 - Sequence of tagged fields

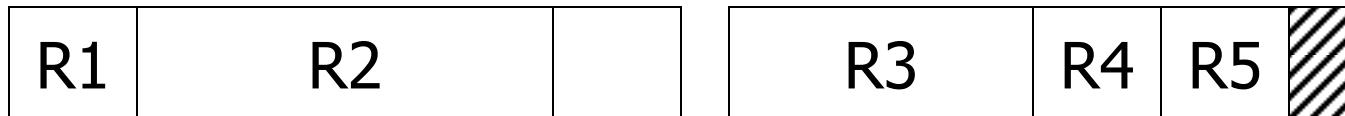


Records that do not fit in a block

- Video clips
- Allow records split among blocks
- Spanned record
 - A record with two or more fragments are called spanned.



- Unspanned
 - Records that do not cross block boundary



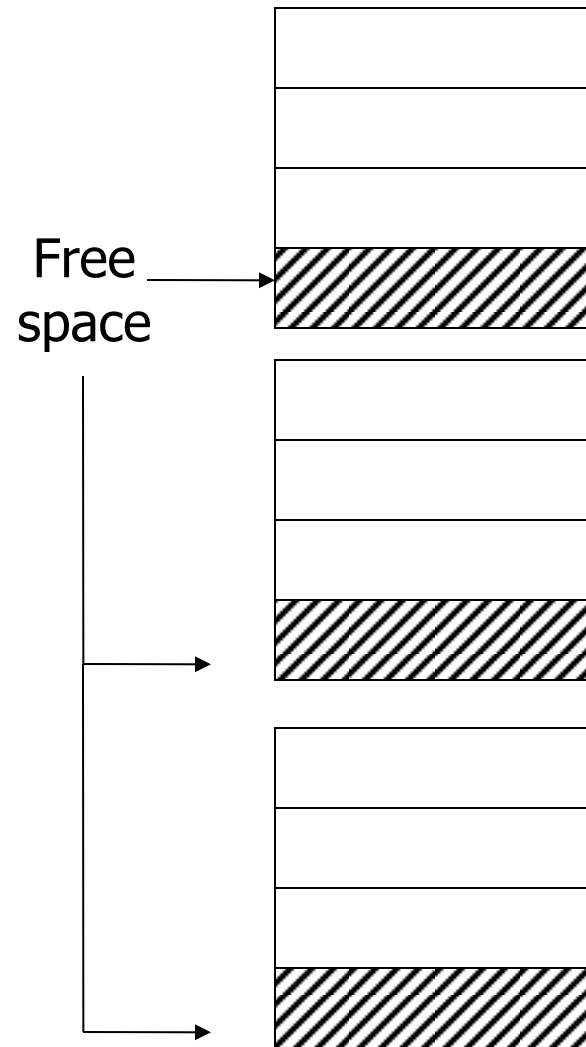
- If the records are spanned, the header should contain more information
 - A bit telling whether it is a fragment or not
 - Another bit telling whether it is a first or last fragment of the record
 - Pointers to the next fragment.

BLOBS

- Movies, JPEG
- Stored in a sequence of blocks
 - Allocate consecutive cylinders
 - Stripe the BLOBS over several disks

Record Modifications

- Insertion
 - Find the space on a nearby block B1. If there is no space find the space in the nearby block B2.
 - Slide the records to B2. Maintain the sorted order
 - Create an overflow block
 - Each block B has a overflow pointer which points to overflow blocks



Deletion

- Delete the record and reclaim the space
- IF we can not slide the records, keep the maintain the available space list in the header.
 - We have to take care of pointers to the deleted records.
 - Place a thumbstone

Update

- Update has no effect on the storage system.
- If the variable length record is updated
 - Follow insertion method

Row vs Column Store

- So far we assumed that fields of a record are stored contiguously (row store)...
- Another option is to store like fields together (column store)

Row Store

- Example: Order consists of
 - id, cust, prod, store, price, date, qty

id1	cust1	prod1	store1	price1	date1	qty1
-----	-------	-------	--------	--------	-------	------

id2	cust2	prod2	store2	price2	date2	qty2
-----	-------	-------	--------	--------	-------	------

id3	cust3	prod3	store3	price3	date3	qty3
-----	-------	-------	--------	--------	-------	------

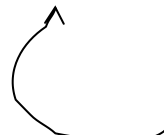
Column Store

- Example: Order consists of
 - id, cust, prod, store, price, date, qty

id1	cust1
id2	cust2
id3	cust3
id4	cust4
...	...

id1	prod1
id2	prod2
id3	prod3
id4	prod4
...	...

id1	price1	qty1
id2	price2	qty2
id3	price3	qty3
id4	price4	qty4
...

 ids may or may not be stored explicitly

Row vs Column Store

- Advantages of Column Store
 - more compact storage (fields need not start at byte boundaries)
 - efficient reads on data mining operations
- Advantages of Row Store
 - writes (multiple fields of one record) more efficient
 - efficient reads for record access (OLTP)

Interesting paper to read:

- Mike Stonebreaker, Elizabeth (Betty) O'Neil, Pat O'Neil, Xuedong Chen, et al.
" C-Store: A Column-oriented DBMS,"
Presented at the 31st VLDB
Conference, September 2005.
- http://www.cs.umb.edu/%7Eponeil/vldb05_cstore.pdf

Comparison

- There are 10,000,000 ways to organize my data on disk...

Which is right for me?

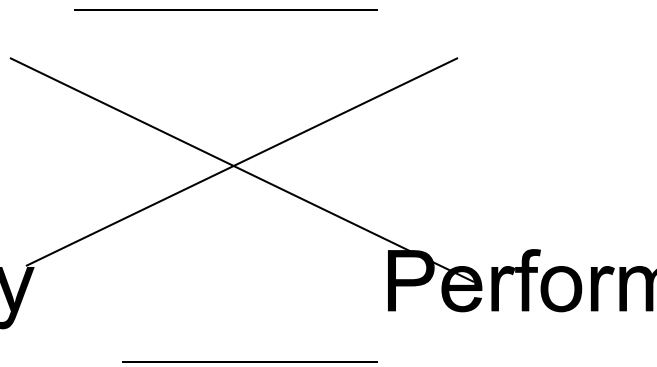
Issues:

Flexibility

Space Utilization

Complexity

Performance



☆ To evaluate a given strategy, compute following parameters:

-> space used for expected data

-> expected time to

- fetch record given key
- fetch record with next key
- insert record
- append record
- delete record
- update record
- read all file
- reorganize file

Example

How would you design Megatron 3000 storage system? (for a relational DB, low end)

- Variable length records?
- Spanned?
- What data types?
- Fixed format?
- Record IDs ?
- Sequencing?
- How to handle deletions?

Summary

- How to lay out data on disk

