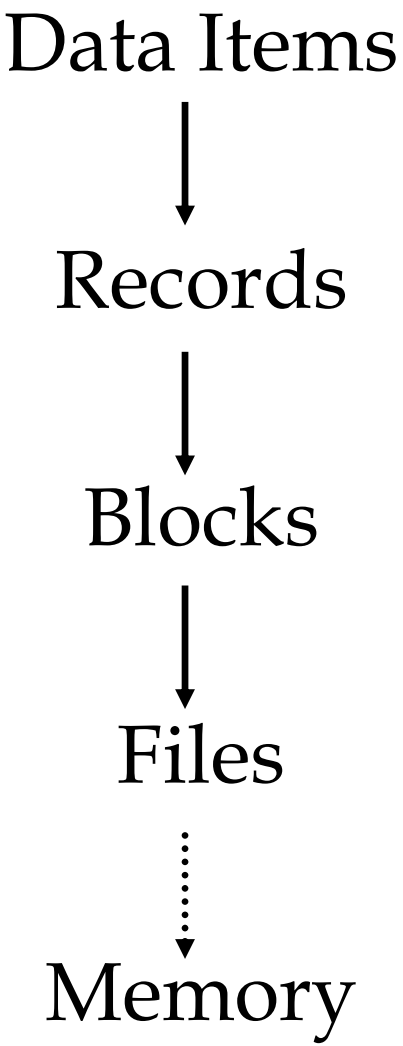
Data Storage and Query Answering

Data Storage and Disk Structure (4)

Introduction

- We have introduced secondary storage devices, in particular disks.
- Disks use blocks as basic units of transfer and storage.
- In a DBS, we have to manage entities, typically represented as records with attributes (relational model).
- Attribute values (data items) can be complex: texts, images, videos.
- How to organize records on disk?

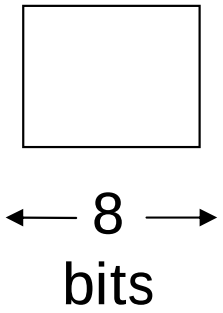
Introduction



What are the data items we want to store?

- a salary
- a name
- a date
- a picture

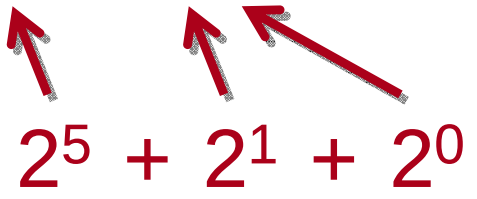
⇨ What we have available: Bytes



Data Items

- Short Integer, unsigned (16 Bits)

  e.g., 35 is 00000000 00100011



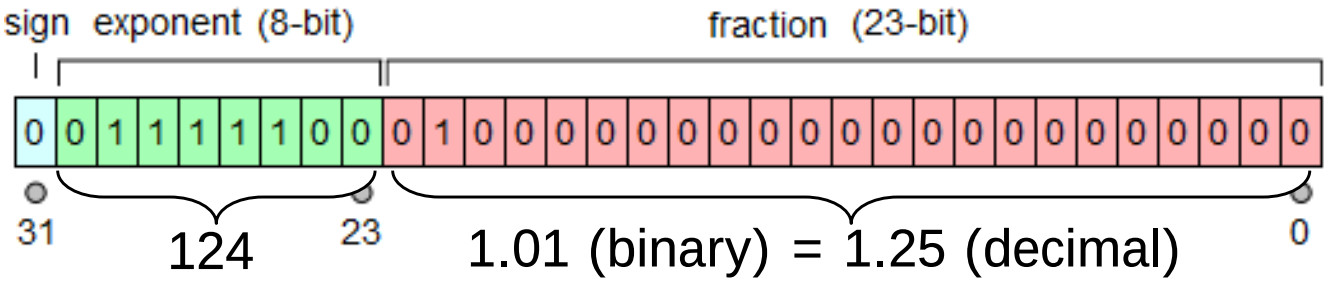
- Short Integer, signed

  e.g., -35 is

  10000000 00100011

Data Items

- Floating point (32 Bits, single precision)
  1 bit for sign, *m* for exponent, *n* bits for mantissa



= 0.15625

$$value = (-1)^{sign} \times 2^{exponent - bias} \times 1.fraction$$

$$bias = 2^{m-1}-1 = 127$$

Data Items

- Characters
  various coding schemes suggested
- most popular is ASCII
  1 Character = 1 Byte = 8 Bits
- examples
  A: 1000001
  a: 1100001
  5: 0110101
  LF: 0001010

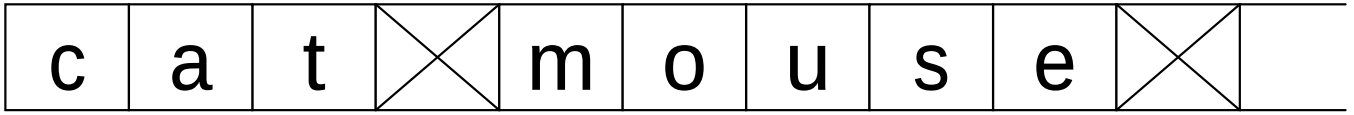
Data Items

- String of characters

VARCHAR

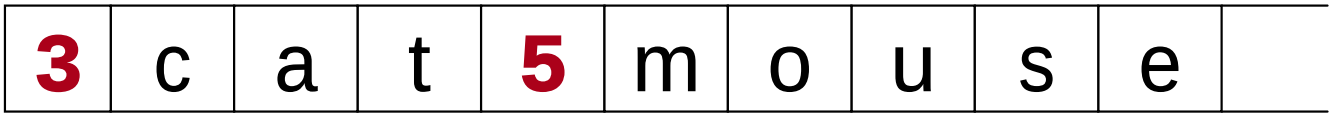
NULL-terminated

e.g.,



length indicator

e.g.,



CHAR

fixed length

do not need terminating NULL

or length indicator

Records

- Fixed format
  - e.g., relational data model
    - record is list of data items
    - number and type of data items fixed

  vs. variable format
  - e.g., XML
    - number of data items variable
- Fixed length

  vs. variable length
  - e.g., VARCHAR, repeating fields

Records

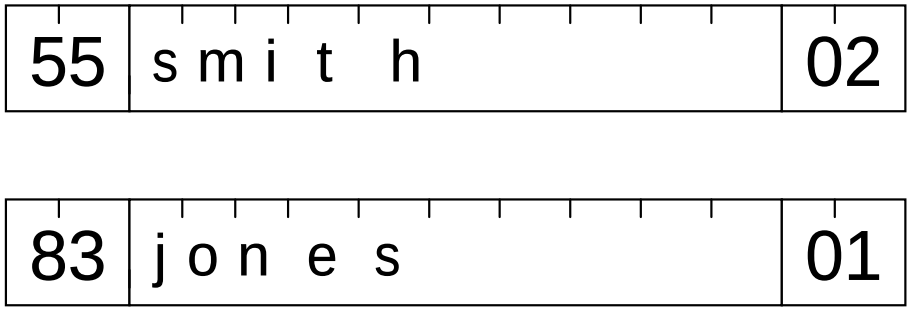
- Record header keeps general information about the record.
- Header contains (some of) the following:
  - pointer to schema,
  - record types,
  - record length
    (to skip record without consulting schema),
  - timestamp of last access,
  - pointers (offsets) to record attributes.

Fixed-Length Records

- Fixed length records are the simplest format.
- Header contains only pointer to schema, record length and timestamp.
- Example

  (1) id, 2 byte integer
  (2) name, 10 char.
  (3) dept, 2 byte code

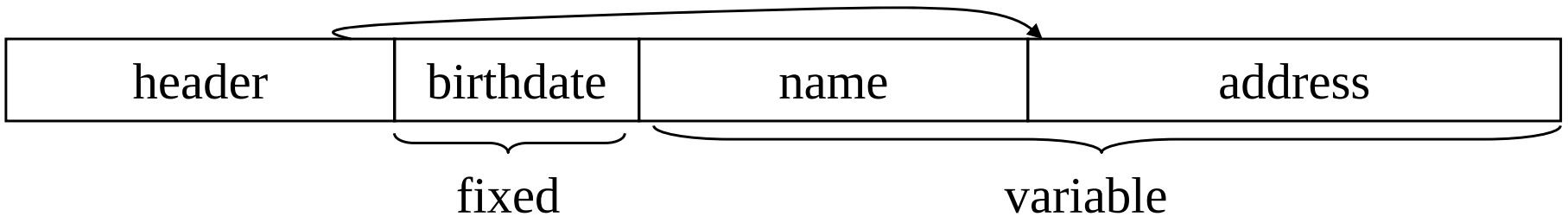
  } Schema



} Records

Variable-Length Records

- Variable length records first store fixed length fields, followed by variable-length fields.
- Header contains also pointers (offsets) to variable-length fields (except first one).
- NULL values can be efficiently represented by null pointers in the header.

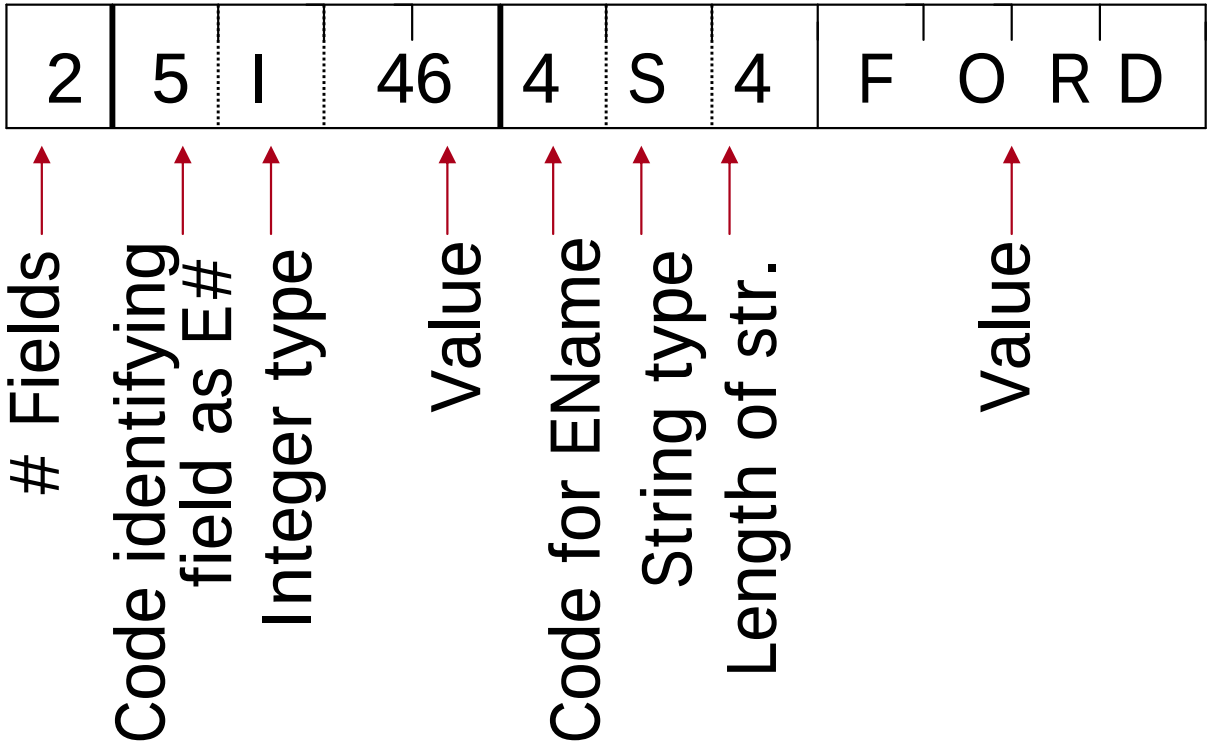


Variable Format Records

- Variable format records are self-describing.
- Simplest representation as sequence of tagged fields.
- Record header contains number of fields.
- Tag contains
  - attribute name,
  - attribute type,
  - field length
    (if not apparent from attribute type).

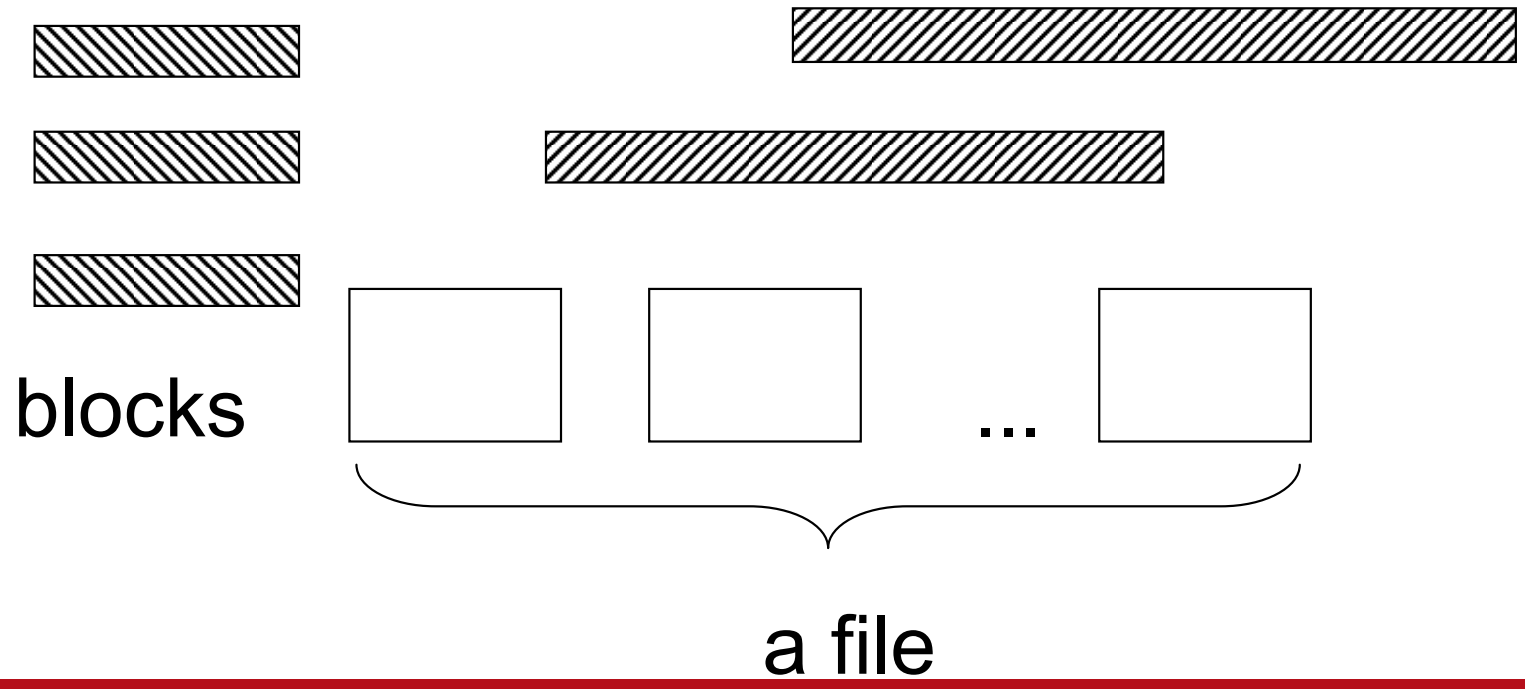
Variable Format Records

Example



Packing Records Into Blocks

- How to pack records into blocks?
- Three important issues to be addressed:
  - Separating records;
  - Spanned vs. unspanned;
  - Ordering.



Packing Records Into Blocks

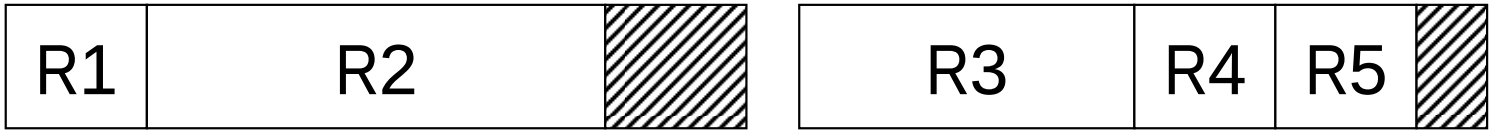
Separating records

- for fixed-length records, no need to separate
- for variable-length records use special marker
- or store record lengths (or offsets)
  - within each record
  - in block header

Packing Records Into Blocks

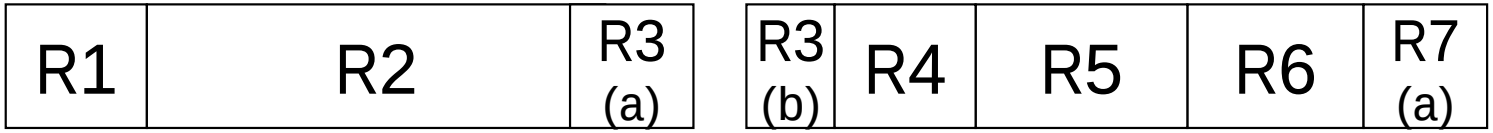
Spanned vs. unspanned

- Unspanned records on a single block



block 1 block 2

- Spanned records split into fragments that are distributed over multiple blocks

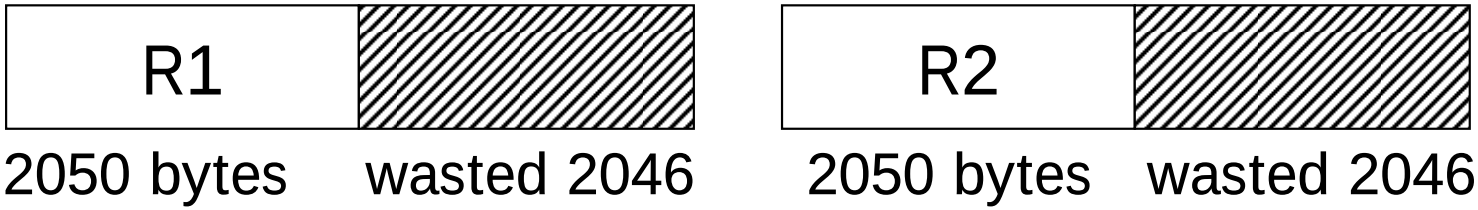


block 1 block 2

Packing Records Into Blocks

Spanned vs. unspanned

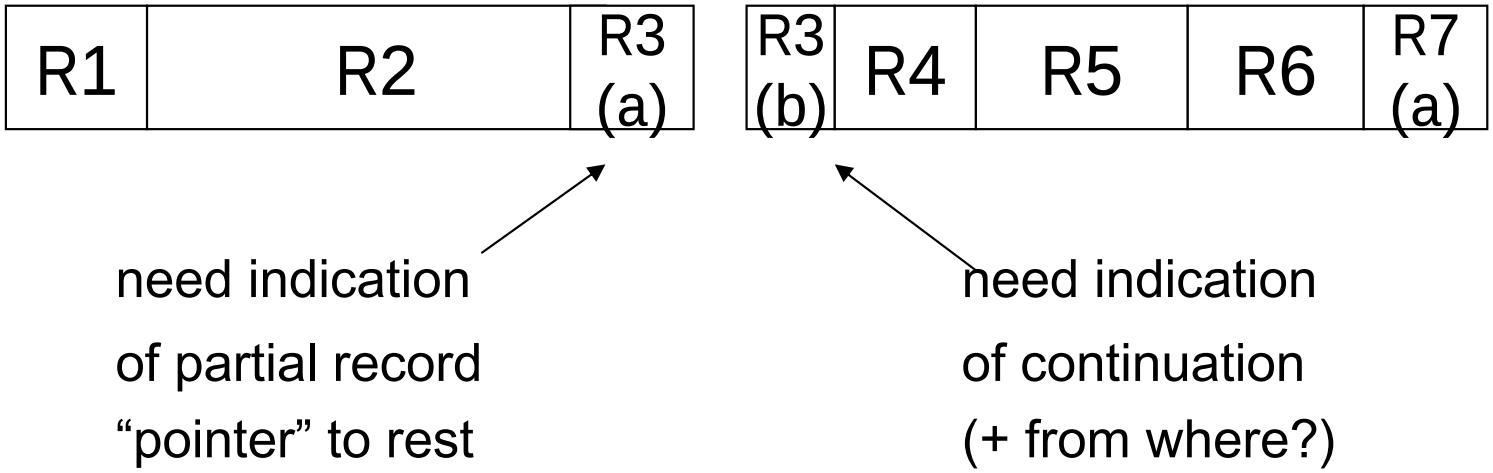
- Spanning necessary if records do not fit in a block, e.g. if they have very long fields.
- Spanning useful for better space utilization, even if records fit in a block.



Packing Records Into Blocks

Spanned vs. unspanned

- Spanned records requires extra header information in records and record fragments:
  - is it fragment? (bit)
  - if fragment, is it first or last? (bit)
  - if applicable, pointers to previous/next fragment.



Packing Records Into Blocks

Ordering records

- Want to efficiently read records in order
- Ordering records in file and block by some key value (sequential file)
- Implementation options
  - next record physically contiguous

- link to next record