

Physical Layout of Data

In relational terms:

- Field = sequence of bytes representing the value of an attribute in a tuple.
- Record = sequence of bytes divided into fields, representing a tuple.
- File = collection of blocks used to hold a relation = set of tuples or records, respectively.

In object-oriented terms:

- Field represents an attribute or relationship.
- Record represents an object.
- File represents extent of a class.

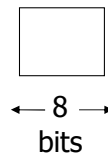
Representing Data Elements

The Mapping Problem

What are the data items we want to store?

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

⇒ What we have available: Bytes



Numbers

Integer: 2/4 bytes

e.g., 35 is

00000000

00100011

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

Characters

→ various coding schemes suggested,
most popular is ascii

Example:

A: 1000001
a: 1100001
5: 0110101
LF: 0001010

Boolean Values

Boolean

e.g., TRUE
FALSE

1111 1111

0000 0000

Application specific Enumerations

e.g., RED → 1 GREEN → 3
BLUE → 2 YELLOW → 4 ...

→ Can we use less than 1 byte/code?

Yes, but only if desperate...

Date and Time

Dates

e.g.: - Integer, # days since Jan 1, 1900
- 8 characters, YYYYMMDD
(not YYMMDD!)
- 7 characters, YYYYDDD
- SQL: YYYY-MM-DD

Time

e.g. - Integer, seconds since midnight
- characters, HHMMSSFF

Strings of Characters

- Null terminated
e.g.,

c	a	t	\0		
---	---	---	----	--	--

- Length given
e.g.,

3	c	a	t		
---	---	---	---	--	--

- Fixed length

Records

Consider fixed-length records first

Record the consists of

- Space for each field of the record.
- Sometimes, it is required to align fields starting at a multiple of 4 or 8.

Example: Employee record

- (1) E#, 2 byte integer
- (2) E_name, 10 char.
- (3) Dept, 2 byte code

Schema

55	s m i t h	02
----	-----------	----

83	j o n e s	01
----	-----------	----

Records

Record Header

Usually the fields of the record are preceded by a header

Header = space for information about the record, e.g.,

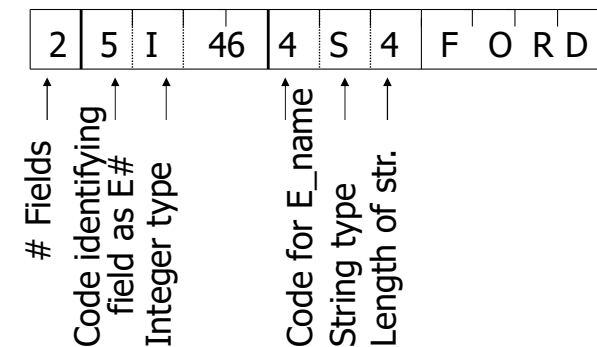
- record format (pointer to schema),
- record length,
- timestamp.

Variable-Length Records

Can occur in case of

- Fields that vary in length
- Repeating fields, e.g., a set of pointers represent a manymany relationship
- Variableformat records: field names are arbitrary
 - Important for selfdescribing data, information integration.

Example: variable format and length



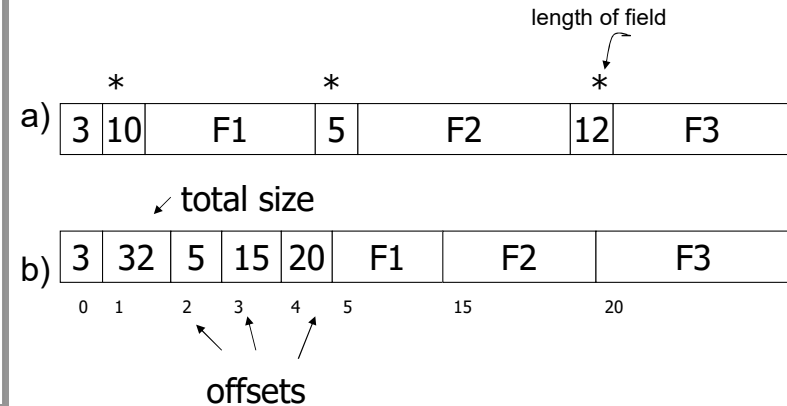
Field name codes could also be strings, i.e. TAGS

Example: Repeating Fields

Employee has one or more children

3	E_name: Fred	Child: Sally	Child: Tom
---	--------------	--------------	------------

Internal Organization of Record



Strategy a) Each field is preceded by a number providing its length

Strategy b) The Record header contains a set of pointers to the variable length fields

Hybrid Format

Hybrid format

- one part is fixed, other variable

E.g.: All employees have E#, name, dept
other fields vary.

25	Smith	Toy	2	Hobby:chess	state:retired
----	-------	-----	---	-------------	---------------

of var
fields

Alternative realization

- Split Records Into Fixed/Variable Parts
- Fixed part has a pointer to space where current value of variable fields can be found.

Placing Records into Blocks

Structure of Blocks:

1. Block header = space for info such as:

- Links to other blocks of a data structure.
- Role info for this block, e.g., for which relation does the block hold tuples?
- Directory of records in the block.
- Block ID.
- Timestamp.

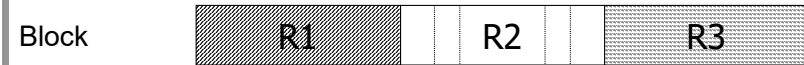
2. Some number of records

Options for storing records in blocks

- (1) separating records
- (2) spanned vs. unspanned
- (3) mixed record types – clustering
- (4) split records
- (5) sequencing
- (6) addressing

Separating records

When does a record ends and the next starts?



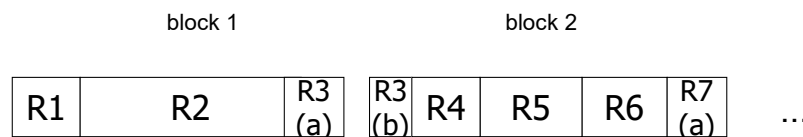
- (a) no need to separate - fixed size recs.
- (b) special marker
- (c) give record lengths (or offsets)
 - within each record
 - in block header

Spanned vs. Unspanned

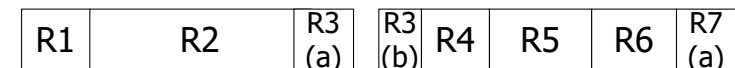
Unspanned: records must be within one block



A spanned record can be divided between two blocks



Spanned records



need indication
of partial record
+ "pointer" to rest

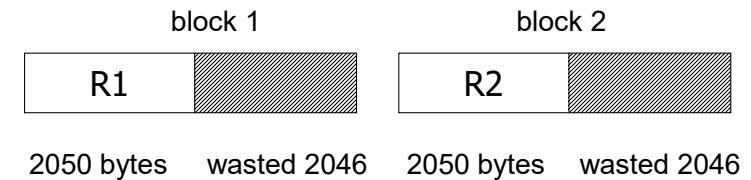
need indication
of continuation

Spanned vs. Unspanned

Unspanned is much simpler, but may waste space...
Spanned essential if
record size > block size

Example

10^6 records
each of size 2,050 bytes (fixed)
block size = 4096 bytes



Total wasted $\approx 2 \times 10^9$ Utilization $\approx 50\%$
Total space $\approx 4 \times 10^9$

Mixed record types

Mixed: records of different types (e.g. EMPLOYEE, DEPT) allowed in same block

e.g., a block

EMP	e1	DEPT	d1	DEPT	d2	
-----	----	------	----	------	----	--

Why do we want to mix?

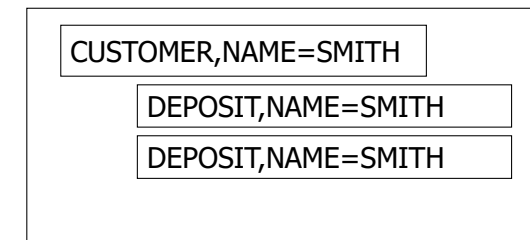
Answer: CLUSTERING

Records that are frequently accessed together should be in the same block

Example

Q1: select A#, C_NAME, C_CITY, ...
from DEPOSIT, CUSTOMER
where DEPOSIT.C_NAME =
CUSTOMER.C.NAME

a block



Options for storing records in blocks

If Q1 frequent, clustering is good

But if Q2 frequent

Q2: `SELECT *`
`FROM CUSTOMER`

CLUSTERING IS COUNTER PRODUCTIVE

Split records

Typically for
hybrid format

Fixed part in
one block

Variable part in
another block

Sequencing

Ordering records in file (and block) by some key value

⇒ sequential file

Why sequencing?

Typically to make it possible to efficiently read records in order

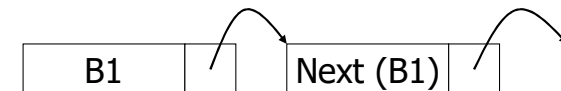
- e.g., to do a merge-join — discussed later

Sequencing Options (1)

(a) Next block physically contiguous



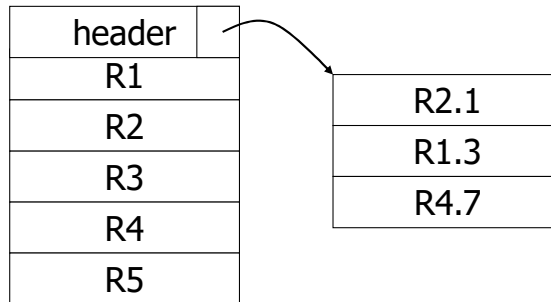
(b) Linked



Sequencing Options (2)

(c) Overflow area

Records
in sequence



Addressing

How does one refer to records?

Two approaches:

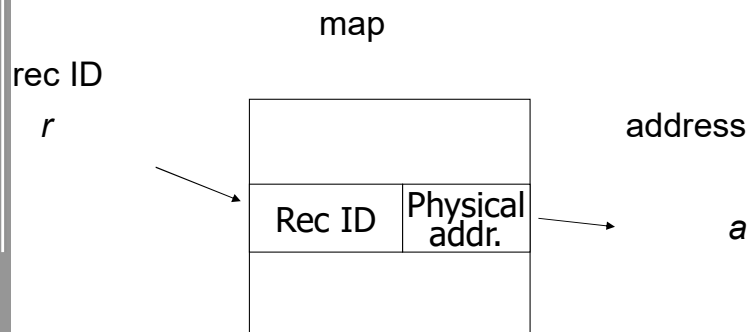
1. Physical: sequence of bytes describing location

For example in case of a HDD: device ID, cylinder #, surface #, block # within track, offset within block (for records).

2. Logical (indirect): a map table associates abstract ID's, perhaps fixed-length character strings, with physical addresses.

Fully Indirect

E.g., Record ID is arbitrary bit string



Addressing (Cont.)

The map table is itself a relation; Fast access is important.

Physical addresses are more efficient.

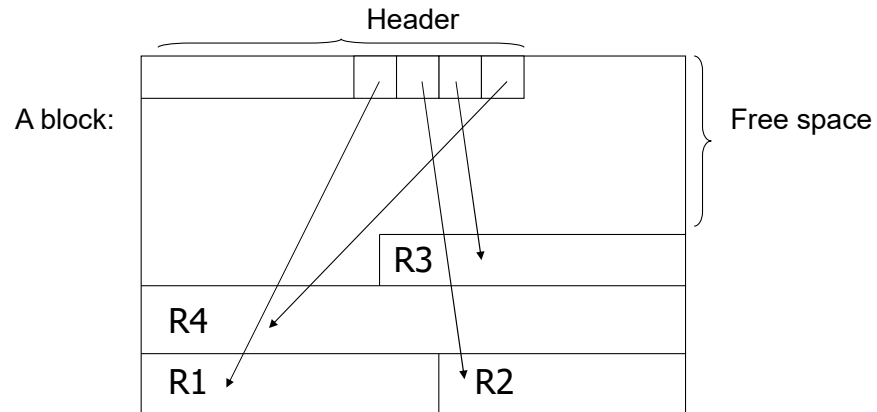
Logical addresses allow flexibility:

- records can move or be deleted without dangling pointers.

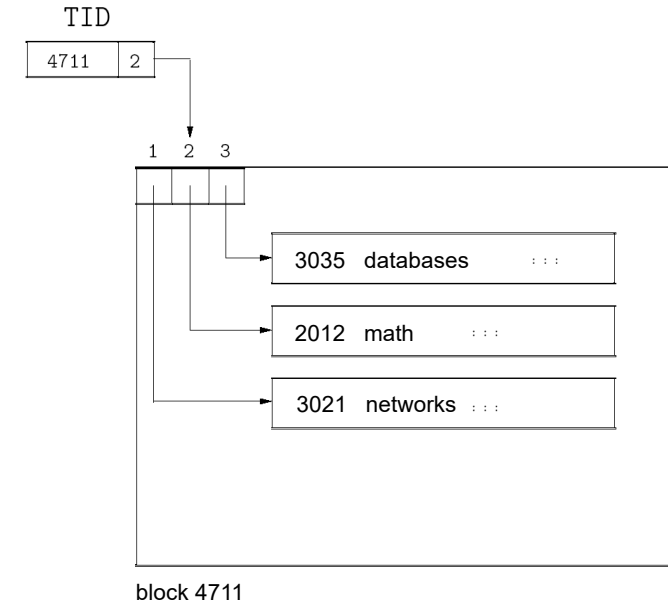
Common compromise: physical to block level, table of record offsets within blocks.

- Tuple-Identifier (TID)
- movement within block: references remain unchanged; only table is modified
- movement to different block: original block contains another TID instead of the record
- regular reorganizations necessary

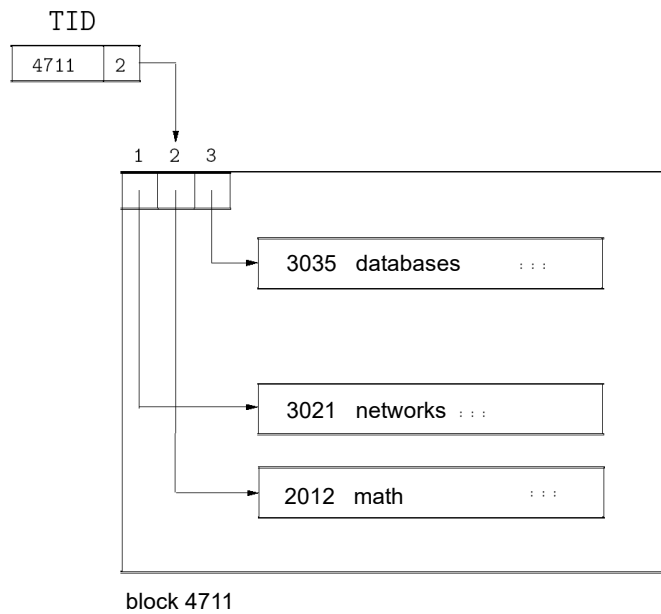
Indirection in Block



TID Example



Movement Within Block



Movement to Different Block

