Storing Data on Disks (Storage Structure)

Overview: Relation

One tuple stored as one record

Disk:

Block: a number of sectors

Block Structure:

Block header

Contains information about the Data records stored in the block

Normally: size (one record) < size (block)
Discussion:

(1) Store a fixed length record.
   (record header).

(2) Pack a fixed length record
    into 1 block.
    (block header)

(3) Addresses to identify
    a block/record

   - Physical Address (Host ID,
     Disk ID
     Ctrl # Trk # Blk #)

   - Virtual Address (when block in memory)

(4) Logical Address + Map Table
    stored on Disk!

   (Modification: more records!!!)
String fixed-length data records.

- Common Practice:

  Fixed length

  \[ 4 \quad 4 \quad 4 \quad 4 \]

  Each field begins at an address that is a multiple of 4 or 8.

  (That's because of the "alignment requirement" of int or float (4)
  or double (8) variables.)

- Format:

  Header information for fixed-length records:

  Record

  Record block header

  Pointer to the record length. Time stamp (last updated).

  Schema (structure)
Example:

Movie Star Record:

<table>
<thead>
<tr>
<th>Name</th>
<th>address</th>
<th>gender</th>
<th>Birthdate</th>
</tr>
</thead>
</table>

30 char: 256 char:

4 char: Date (10 bytes)

1. Records stored as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>address</th>
<th>G</th>
<th>BD</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>288</td>
<td>292</td>
<td>304</td>
</tr>
</tbody>
</table>

(32+256).

2. Record + Record Header:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>G</th>
<th>Birthdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>44</td>
<td>300</td>
<td>304</td>
</tr>
</tbody>
</table>

316

316

4. Pointer to Record Schema

*Assume each field is 4 bytes long.*
Packing fixed-length Records into Blocks:

- Records (of the format given previously) as stored (packed) into 1 block.

  as follows:

  "block Header" 

```
| record1 | record2 | record3 | ... | recordn |
```

In general:

- A block can store records from DIFFERENT relations.

More common:

- A block stores records from SAME relation.

Assume this is the case.
What info. stored in Block Header:

- Block header
- Offsets of each record in this block
- Record 1
- Record 2

- Links to other blocks
- More in "Indexing"
- Next Chapter

- Info. about the role of this block
- Time stamp of block's last modification/access

- Relation that records belong to

- All records belong to this relation.
Block and Record Addresses

* Fact:
  
  A block and a record is identified by:
  
  block \rightarrow \text{block address}
  
  record \rightarrow \text{record address}

* A block/record can be located in 2 places:
  
  (1) on the disk
   
  (that's the permanent storage location.
  
  (2) in main memory,
   
  (while the block/record is being accessed by the DBMS).

* A block located on a disk is identified by: a physical address.

A block/record located in main memory is identified by: a \underline{virtual address}

(64 bit binary number)
Physical Address

How to locate a block/record on DISK.

Block address on Disk:

- Platter
- Cylinder
- Block

1. The "physical" address of a block consists of:
   1. Host ID (that contains the disk)
   2. Disk ID (that identifies the disk)
   3. Cylinder number (of the disk)
   4. Track number (which platter)
   5. Block number (inside the track)

2. The "physical" address of a record:
   = Physical addr. of the block (that contains the record)
   + Offset in the block.
Alternative way to identify block:

Logical Addresses of blocks/records.

Each block/record has a logical address.

Logical address = an arbitrary string of bits of some fixed length (large enough to identify every block/record in the whole database).

Map table = table that maps

Access: Logical Addr

Map table

n bits \( \Rightarrow 2^n \) "database objects"
Fact: The map table is stored on (desk) (in a known location)
Motivation: Why use "logical" block/record address?

Reason: **flexibility.**

Example:

Some techniques store "pointers" to a record (block) inside a record. (e.g.: foreign key).

- If we store a physical address:

  ![Diagram of record structure]

  then it is **HARD** to relocate (move) this record.

  (Need to update all these)
If we store a logical address, we might end up storing it in another record. Let's say the record has a logical address of 1010 10...1. Then we store a block of data, and we need to update the map table accordingly.

We can move this block by updating the map table. For example, if we have a new partition ID, we can update the map table to point to the new block.
Definition:

Database address of an object (block/record) = logical/physical address of object

If you use a logical address, you can use the MAP table to obtain the physical address.

So: without loss of generality, by a "Database addr", we will always assume:

```
[Host ID, Disk ID, Cyl, Ht, sect #, Blk #]
```

= physical address.
Virtual address is used to identify a block while it's in memory!

Virtual Address: paging

DBMS program

Main Memory

Paging table

Virtual memory

Very large memory space

If a data block is read (from disk) into a buffer inside the DBMS program, the block in memory will be used instead of the one stored on disk!!!
Different combinations of logical/physical addresses to identify a record are possible.

**Example 1:**

Address to identify a record:

```
Host ID, Disk ID, Cy #: Tr #: Blk #: Offset
```

Physical address of block

**Example 2:**

Address to identify a record:

```
Host ID, Disk ID, Cy #: Tr #: Blk #: Key
```

After reading the block, performs linear search to find record using key.
Example 3:

Recall that **Block Header** store record offsets:

```
0, 1, 2, 3
```

Links to other blocks:

Relation that records in this block belongs to

Address to identify a record:

```
HID, DID, CID, TID, PID
```

In the index:

```
0, 1, 2, 3
```

Eg: 4
Advantages of this record addr. structure:

Addr. to identify a record:

```
HostID, Disk ID, Page, Table, Block, Index
```

addr. of a block

Block & record offset table

1. We can move the record around within the block:

```
Block: 202
```

2. Record can be moved between blocks using larger "forwarding addresses".

```
Block: new block # + offset
```
② We can handle

**Record deletion**

by leaving a

*tombstone* (\(\approx\) special value like "null")

to indicate the record has been deleted.

Example:

Valid record:

Address to identify a record:

Deleted record:

Block: illegal offset
A note on deleting records.

Recall: Some storage techniques use record addresses.

Record 1:

Map Table: (on disk).

Logical record address

Physical address

Block A

Host ID, Disk ID ...

Referenced record
When we **DELETE** a record:

- **Block A:**
  - Deleted.

The space for this record can be **REUSED** later.

Resulting in:

- **Block A**
  - 104

Now, the "old reference" to the record:

- **Record 1:**
  - 10101...1

Map Table: (on Disk)

- Host ID...1
  - 10101...1
Will now reference to
another record !!!

* One way to handle "record deletion"
  is: use a "tombstone" marker

Record 1:

Map Table:

Deleted record