Locating records by the DBMS

- Fact: Pointers (addresses) are part (stored) of records.

- This is not typical in tables of a relational DBMS.

- But: this is common for tuples of an Object Oriented DBMS.

- Object Relational DBMS.

- Pointers are also used in index files.

- The DBMS has a management system for pointers:
  - Pointer to a block on disk = a physical address
  - Pointer to a block in memory = a virtual address
Recall that:

A database object (block/record) is identified by:

1. **Database address** of object 
   (= logical/physical address) 
   when object is on disk.

2. A **virtual addr.** in memory 
   when object is read/stored in memory.

(The object is still on disk, but the one on disk is not used until the in-memory copy is written back to disk.)

- The DBMS has/needs a manage need system to convert:

  - Database address \(\rightarrow\) virtual address
Observation:

(1) When a database object (block/record) is on disk, we can ONLY refer to the object using a database address.

(2) When a database object (block/record) is in memory:
(A) we must use the object by virtual memory address.
(B) but: we can refer this object by: <virtual memory address>
So: when the DBMS uses a Database Address to reference an object that is in memory, we must:

- translate the Database Address to a (virtual) memory address!!

Translation table:

<table>
<thead>
<tr>
<th>Database Addr</th>
<th>(Virtual) mem. addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Host ID, DB ID, tr...)</td>
<td>10 10 10...</td>
</tr>
</tbody>
</table>

Table is SPARSE (only objects in memory will have a VALID entry), speed + space savings.
Naive Access to Database Objects

1. Access
2. Hash

Database Addr | Virtual address
---|---
valid | XXX
invalid |

Hash

Record that contains a database address

Go access copy on disk (bring in to memory first)
Advantage:
1. Easy to implement
   (only need a hashing table)
2. Records/Blocks are not "pinned" in memory (later)

Disadvantage: slow...

Faster Access: Trick

[Diagram]
- Reference to another block.
- If this DB object is in memory, replace DB addr with virtual addr.
Pointer swizzling:

When we move a block from disk to main memory, the pointers (\(= \text{Database addresses}\)) in the block may be "swizzled".

map

translated (using the hash table)

from \(\text{Database Address}\)

virtual address.
Implementation:

The data record field for a DB addr. is expanded with 4 bits:

DB Addr. Field

| 0 | (Host ID, Disk ID, - - - - ) |

When switched:

| 1 | 1 1 1 1 1 1 |

Virtual address Unused.
Example:

Disk

When data read in Block B:

Translate table

DB addr

Virt. mem. addr.

WP swizzle the DB addr -> P

Block A

Block B
Swizzling Techniques:

(1) Automatic Swizzling

- Swizzle all DB addr \(\rightarrow\) Mem.addr inside a block when the block is FIRST read in to memory.

(2) On Demand Swizzling

- Leave all DB addr. unswizzled when block read in.
- When DB addr. is used: (first time)

Swizzle (so next accesses will be fast).
Implementing automatic swizzling:

- Information required:
  
  - Knowledge on the location of every DB address type field in a block.

1. Block holds records of one schema

   \( \Rightarrow \) Schema will tell us the location of the DB address fields

2. Block used for indexes (later)

   - Structure is known
   - DB address can be located

3. Structure unknown...

   \( \Rightarrow \) Put a list of offsets in Block header that contain locations of DB addresses.
Implementing on-demand swizzling:

When we read in a block:

Enter translation entries for every record in the block:

Translation table

<table>
<thead>
<tr>
<th>Hash (A)</th>
<th>VMAR (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>VMAR (A)</td>
</tr>
<tr>
<td>B</td>
<td>VMAR (B)</td>
</tr>
<tr>
<td>C</td>
<td>VMAR (B)</td>
</tr>
</tbody>
</table>
Problem created by pointer swizzling

1) Problem: the block in memory is changed:

We must UNSWIZZLE the virtual addr. P \(\Rightarrow\) DB addr. y

BEFORE writing the block
Back to the DISK !!
Naive Solution:

Translation Table:

DB addr. | VM address
---------|------------

Hash

Block

replace

Match

swizzled pointer.

Problem: requires a search to find entry!

Faster possible solution: Add an index (Hashing) based on virtual memory addresses.
Summary: Faster technique to unswizzle a VM address => DB address

1. Use a hash map:

```
VM addr  DB addr
```

VM addr (hash)
In Chapter 14, the book will discuss
an indexing data structure
to implement this search faster.

pp: ???
Problem created by pointer swizzling

1. Problem: a block (that the DBMS wants to remove from memory) cannot be removed from memory.
2. Some records in block are referenced by swizzled pointers in other (in memory) blocks.

Example:

1. DBMS wants to remove this block (replace).
   "pinned"

2. This swizzled pointer will be one UNVALID (dangling?)
   "pinned"

3. Need to replace VM addr. with DB addr (unswizzled)
How to unpin a block:

1. Pinned block
2. Unswizzle these VM addresses
3. Remove (write back) this block
Conclusion:

When the DBMS wants to release/remove a block from memory:

1. DBMS wants to write this block back to disk
2. Make sure block is not pinned

Write back
Implementing unpin efficiently

1. Keep a (linked) list of memory addresses

2. How to use list:

- Each three
- You swizzle an addr, add the VM Addr of DM addr to linked list (get the correct list!)

Hash(P) (HashCP)

0. Find List

<table>
<thead>
<tr>
<th>B</th>
<th>P</th>
</tr>
</thead>
</table>

Unswizzle each VM addr at these VM addrs
More efficient implementation of the linked list:

- Often: the DB address is much larger than a VM address.

![Record diagram]

- We can construct the linked list of swizzled address using the space that store the DB address.
Example:

Reverse Translation Table

DB addr  DB addr  Link

Hash(P)  DB addr (P)

Block that needs to be "released"