第二部分：存储管理器

韩丽萍
计算机学院
联系邮件：liping@njupt.edu.com

课程回顾

1、Megatron2000原型概况

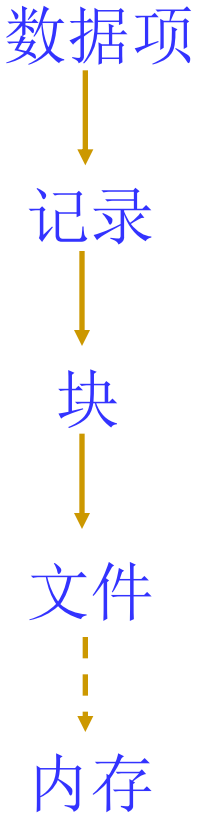
2、 Megatron2000原型的不足

3、DBMS提供的能力

4、DBMS结构

课程要点（对应教材第二和第三章）

关系数据的一张表如何存储？

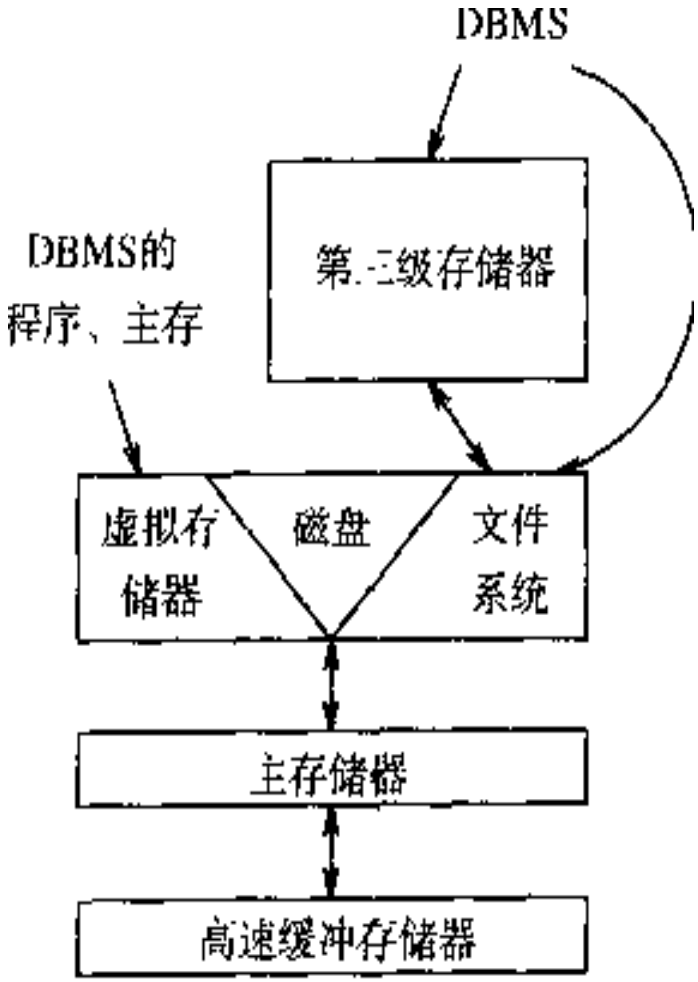


目录

存储器层次

1、存储器层次

存储器层次



高速缓冲存储器

1、高速缓存（cache）分成一级缓存和二级缓存，一般一级缓存和处理器同一个芯片，二级缓存位于另外一个芯片。

2、高速缓存和处理器间的数据读写以处理器指令的速度执行，通常为几纳秒左右（1ns=10^{-9}s，2010年后）。

主存储器

1、主存是随机访问的，即同一时间内可获得任何一个字节

2、主存上的一次数据访问时间通常是10纳秒（$1ns=10^{-9}s$）。

（自学pp.17页摩尔定律部分）

虚拟存储器（二级存储器的一部分）

1、由于虚拟存储器的地址空间比内存大，虚拟存储器的大部分内容是保存在硬盘上。

2、硬盘逻辑块（block）的大小一般是4~64KB。

3、虚拟存储器以块为单位在硬盘和主存间移动，在主存中块常被称为页（page）

二级存储器

1、从磁盘上读一个块到内存，是一次磁盘读

2、内存中的一页写到磁盘，是一次磁盘写。

3、一次磁盘读或者一次磁盘写都称为一次磁盘I/O

二级存储器

4、磁盘上一个读写一般是10毫秒左右（2010年）

5、结论：二级存储器的访问速度比内存访问速度慢10^5～10^6倍。

6、硬盘不仅支持文件系统，也支持虚拟存储器，即有些磁盘块被用于保存一个应用程序的虚拟内存页面，有些磁盘块用于保存文件（磁盘=文件系统+虚拟存储）。

。

三级存储器

1、与二级存储器相比，读/写时间更长，容量更大

2、访问时间范围较宽，取决于数据与读/写点靠的有多近

3、主要的三级存储器设备：磁带存储器、自动光盘机、磁带仓

思考题

试题内容

在计算机系统的存储层次结构中，能被CPU中的计算单元和控制 单元以最快速度来使用的是()。

A.高速缓存(Cache)

B.主存储器(DRAM)

C.闪存(FLASHMemory)

D.寄存器(Registers)

计算模型（算法研究）

1、RAM计算模型（随机存取机模型、冯.诺依曼模型）

2、I/O计算模型

RAM计算模型

1、假定数据放在主存储器中，对任何单位数据访问所花费的时间一样多（均匀耗费标准）。

I/O计算模型

1、DBMS中，数据不能装在主存中，必须考虑二级甚至三级存储器。

2、二级、三级存储器中处理数据的最佳算法，不同于在内存中处理的算法。

3、重点考虑降低磁盘访问次数。

I/O计算模型举例子

例子：假设数据库有关系R，有一个对元组的查询请求，该元组有一个确定的键值K。为了快速响应查询，要在R上创建一个索引，用来标志带有键值K的元组出现的磁盘块，而索引是否告诉我们元组在磁盘块的什么位置并不重要。

分析：读一个4～64KB的块大约要花10毫秒（1毫秒=10^{-3}秒），一个处理器在这10毫秒内可以处理上百万条指令。然而，一旦块在主存中，即使采用线性搜索，搜索键值K仅执行成千条指令，因此在主存中搜索时间将小于块访问时间的1%，因此可以忽略不计。

思考题

判断题：对于I/O模型，索引可以有效地提高查询效率，但即使有索引，查询的时间仍然主要由磁盘块的访问时间决定。

答案：正确

思考题

分析题：假设数据库关系R已经创建了索引来加速查询，且每次查询的查询时间主要由磁盘块访问时间决定。分析并解释为什么即使有索引，数据库查询在主存中进行时，时间消耗主要集中在磁盘块读取而非数据搜索上。

- 由于磁盘访问时间较长，相比之下，主存中的数据搜索时间非常短。
- 索引优化查询时，主要是减少需要访问的磁盘块数量。即使有索引，查询过程依然要依赖磁盘块的读取时间。
- 主存中的搜索时间几乎可以忽略不计，尤其是当索引指向的磁盘块已经加载到内存中时，搜索仅需执行几千条指令。

目录

数据元素表示

基本SQL数据类型？

（1）定长字符串-char(n)
（2）变长字符串-varchar(n)
（3）整数类型-int or integer
（4）浮点类型-float or real
（5）位串类型-定长位串bit(n)和变长位串bitvaring(n)
（6）日期时间类型-date or time

数据元素表示

基本SQL数据类型如何表示成记录的字段？

总则：

（1）不同的数据类型用不同的字节序列表示

（2）一个记录的所有的字段最终被表示成一个字节序列

数据元素表示

定长字符串char(n)：表示为长度是n的字符数组，未占用的用填充字符补齐

例如：如果元组中某个属性值是'cat'，则实际字符数组是

cat⊥⊥

这里⊥是填充字符，占据这个数组的第四和第五个字节。

注意：SQL程序中指明字符串所用的单引号，并不与字符串的值一起存储

数据元素表示

变长字符串varchar(n):用长度为**n+1**的字符数组表示

方法1——长度加内容：第一个字符存储长度，后面存储内容

方法2——空值－终止字符串：在内容后加一个特殊的终止字符。

例如：属性A声明为varchar(10)，实际存储占用11个字节

方法1：3cat，这里3是8位二进制整数，即00000011，而不是字符‘3’，余下的7个位置无关紧要

方法2：cat⊥，余下的7个位置无关紧要

数据元素表示

日期和时间：表示为符合某种格式的定长字符串

例如：SQL2中
年月日：‘YYYY-MM-DD’
小时分秒：‘HH:MM:SS’

数据元素表示

二进制位串bit(n)：（最好是用C实现）

(1)按每8位构成一个字节的方式组装

(2)如果n不能被8整除，则忽略最后一个字节中未用的二进制位，一般用0填充

例如：二进制位序列010111110011，用两个字节表示，第一个字节是01011111，第二字节是00110000，其中的后四位不是任何字段的某一部分。

例如：布尔值可以用8位表示，11111111是真，00000000是假

数据元素表示

枚举类型：

用整数编码表示

课程回顾

- 存储器层次
  - 高速缓存
  - 主存储器
  - 二级存储器
  - 三级存储器
- 计算模型
  - RAM计算模型
  - I/O计算模型
- 数据元素的表示（定长/变长字符串、整数/浮点类型、日期、枚举）

目录

定长记录

1、讨论字段如何组合成记录

2、数据库中每一种类型的记录必须有一个模式（所有模式存放在一个文件或者和每个表的模式和实例存放在一个文件），模式也由数据库存储（*记住！*），包括

- 每个字段名称
- 每个字段数据类型
- 每个字段在记录内的偏移量

定长记录——记录的构造

记录的所有字段是**定长**，将字段连接成**记录**

例如：

```
Create table moviestar(
        name char(30),
        address varchar(255),
        gender char(1),
        birthday date
);
```

定长记录——记录的构造

理论上占用的空间大小？

moviestar类型的记录占用

30+256+1+10＝297字节

其中：

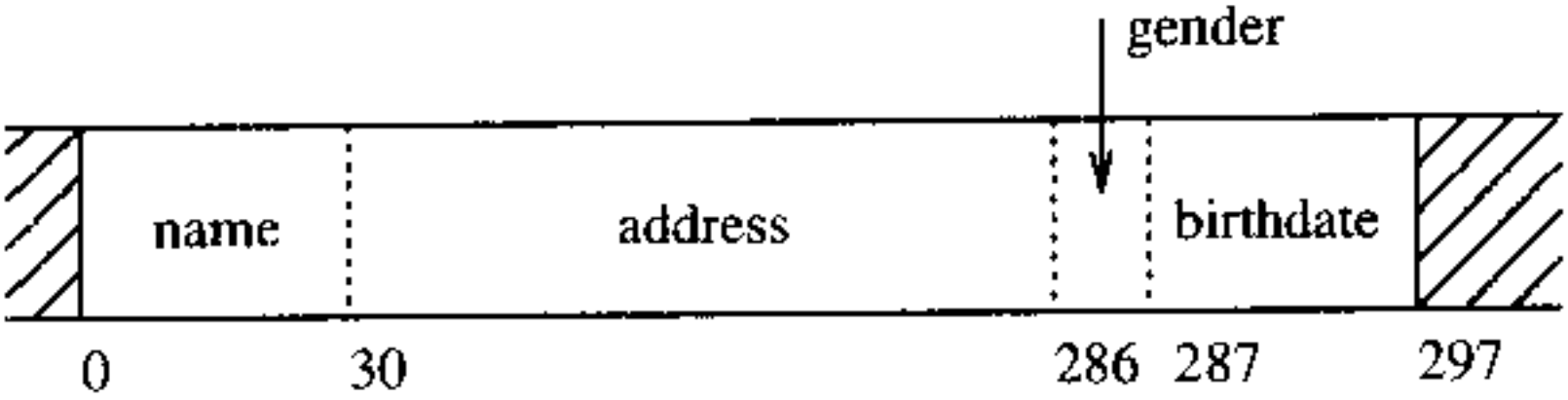
name的偏移量(记录内起始字节的位置)是0

address偏移量是30

gender偏移量是286（实际占用256个字节）

birthday偏移量是287

定长记录——记录的构造



定长记录——记录的构造

存在问题：

1、内存对齐

2、将一个块从磁盘读入主存时，块的第一个字节肯定放到主存的2的高次幂的某个位置。例如：如果块的大小是4096，那么将放在2^{12}的倍数处的位置。

3、因此某些字段要放在2的高次幂的主存位置这一需求，转化为字段在磁盘块内的偏移必须具有相同的因子。

定长记录——记录的构造

简化规则：

1、每一条记录从块内4的倍数的字节处开始，并且

2、记录中所有字段相对于记录起始位置的偏移量都是4的倍数

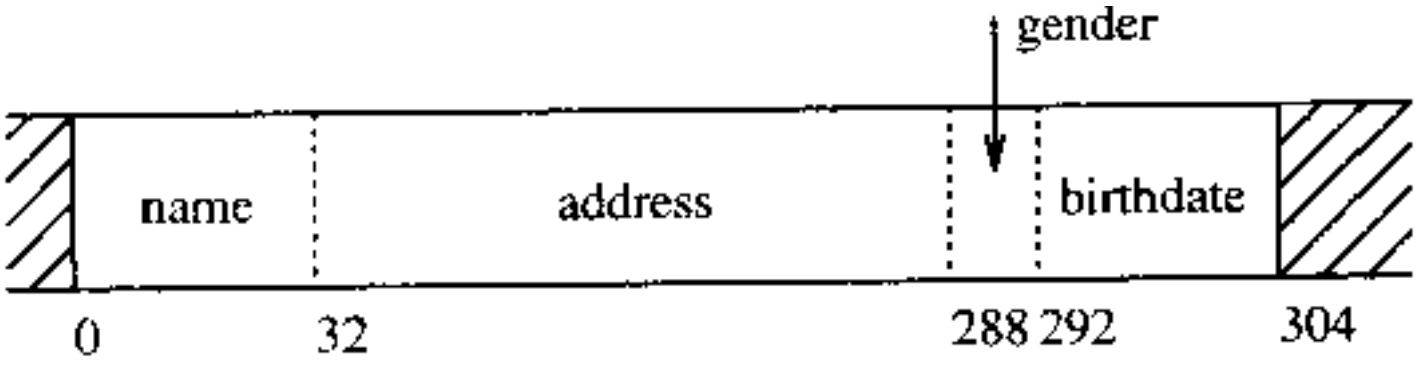
Moviestar各个字段的偏移量修正为：

- name:0
- address:32
- gender:288
- birthday:292

实际占用304个字节

定长记录——记录的构造

```
Create table moviestar(
        name char(30),
        address varchar(255),
        gender char(1),
        birthday date
);
```



定长记录——记录的首部

除了用户数据外，记录须以首部（header）开始，首部是关于记录自身信息的一个定长区域，一般包括

###

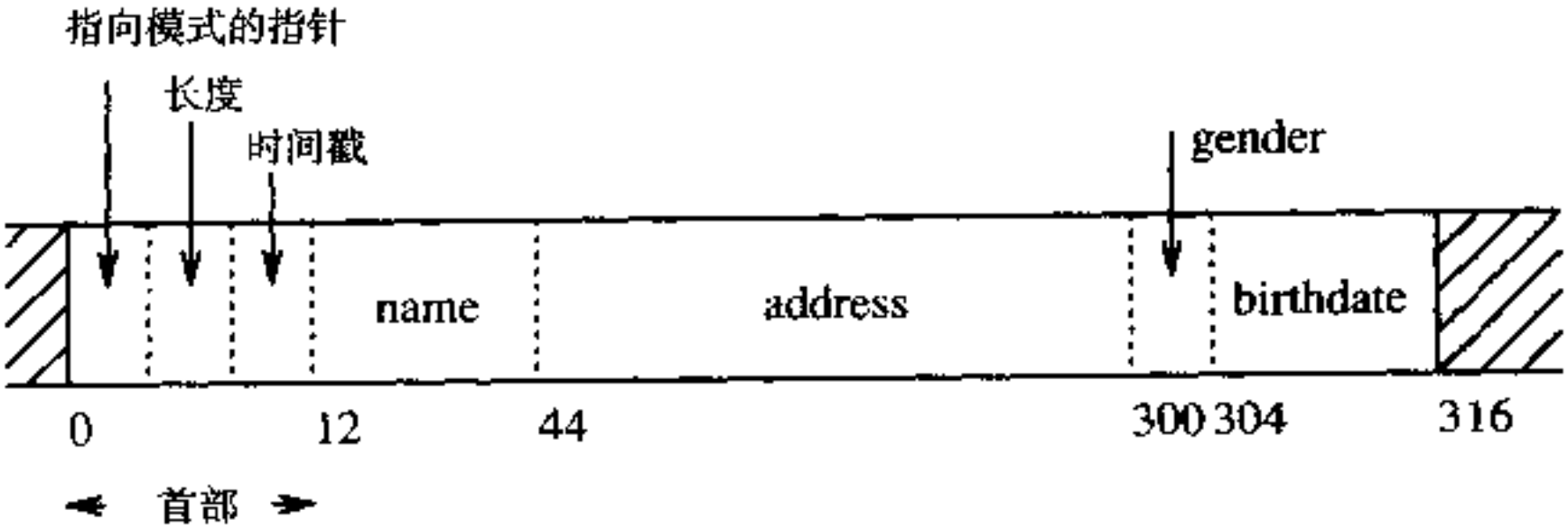
1）指向记录模式的指针——*帮助找到记录的各个字段*

2）记录长度——*不用查看模式的情况下略过某些记录*

3）时间戳——*最后一次被修改或读的时间*

###

定长记录——记录的首部


指向模式的指针
长度
时间戳
gender
name
address
birthdate
0
12
44
300
304
316
首部

定长记录——记录的首部

记录模式信息（一般单独存放，并且缓存）

###
属性1|属性1的类型|偏移|主键约束、域约束等
…
属性n|属性n的类型|偏移|主键约束、域约束等
###

定长记录——块的格式

记录存储在块中，记录存取和修改时，要整块读取到内存，一个块4K或8K,块的构造如下

||块首部(可选)|记录1|记录2|…|记录n|剩余空间||

可选的块首部结构：

||块ID|块和其他块的链接信息|记录所属的关系|块最后一次修改和存取的时间戳|每条记录在块内偏移量目录||

注意：只是示意，实际实现有差异

定长记录——块的格式

举例：

假设我们要存储前例子的记录，一个块是4096个字节，块首部占用12个字节，一个记录占用316个字节，则可以存储12条记录，还剩余292个字节。

思考题

假设我们要存储前例子的记录，一个块是8192个字节，块首部占用12个字节，一个记录占用400个字节，则可以存储__条记录，还剩余__个字节？

目录

块和记录地址

内存：

块地址→块的第一个字节的虚拟内存地址

记录地址→记录的第一个字节的虚拟内存地址

外存：

块地址->设备ID+柱面号+扇区号

记录地址->块地址+块内偏移量

注意：实际开发时多用（文件+块号+偏移量）

块和记录地址

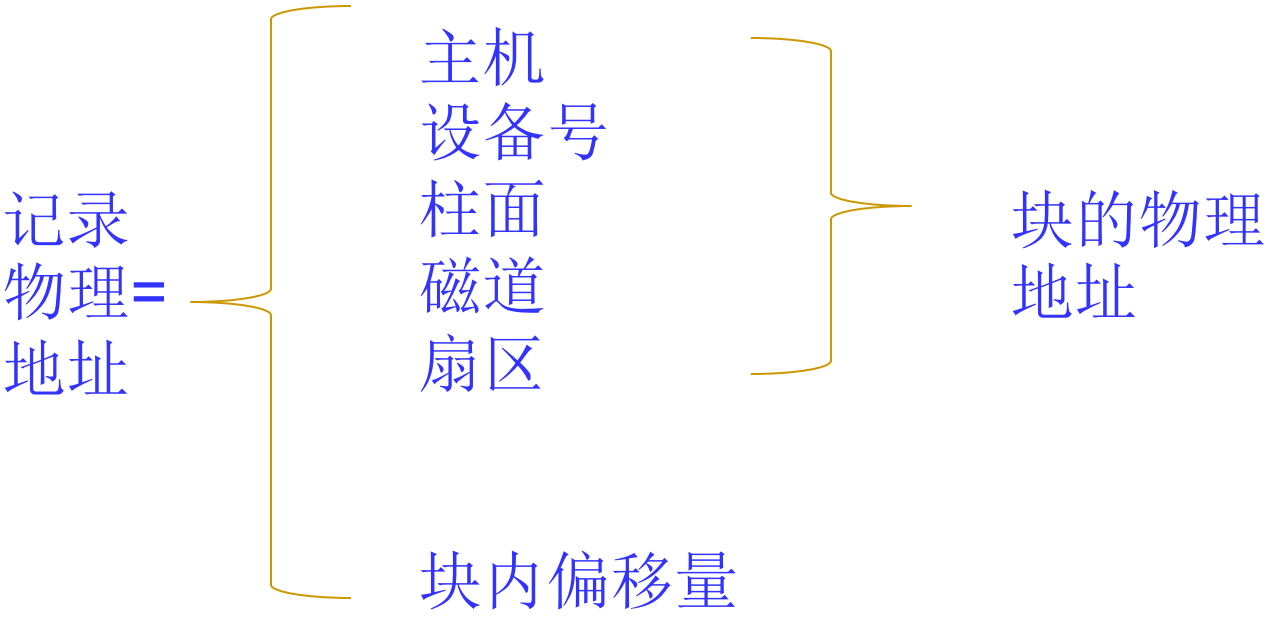
物理地址：指明块或记录的物理位置，如下

主机+设备ID+柱面号+扇区号+块号（块地址）

主机+设备ID+柱面号+扇区号+块号+块内偏移量（记录地址）

逻辑地址：每个记录有一个“逻辑地址”，是一个固定长度的字节串，通常是"表名+记录ID"。

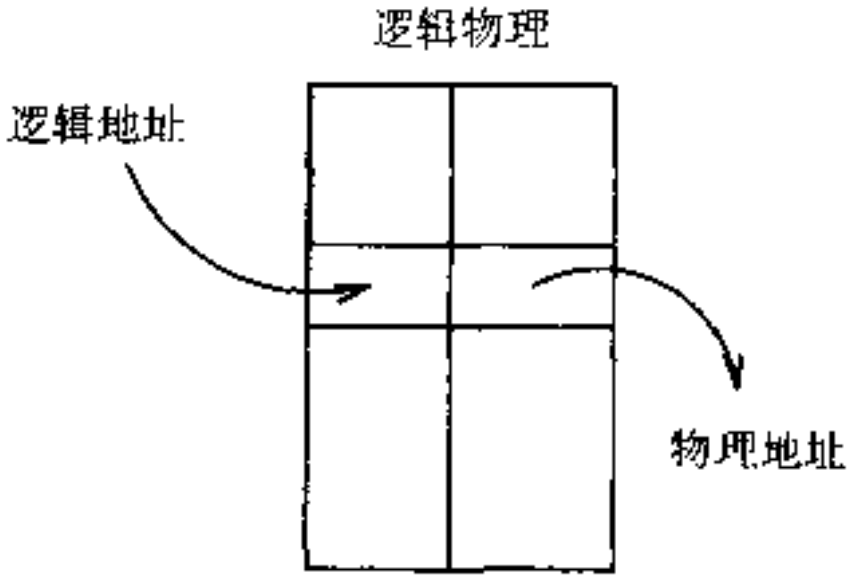
块和记录地址-物理地址



块和记录地址——逻辑地址

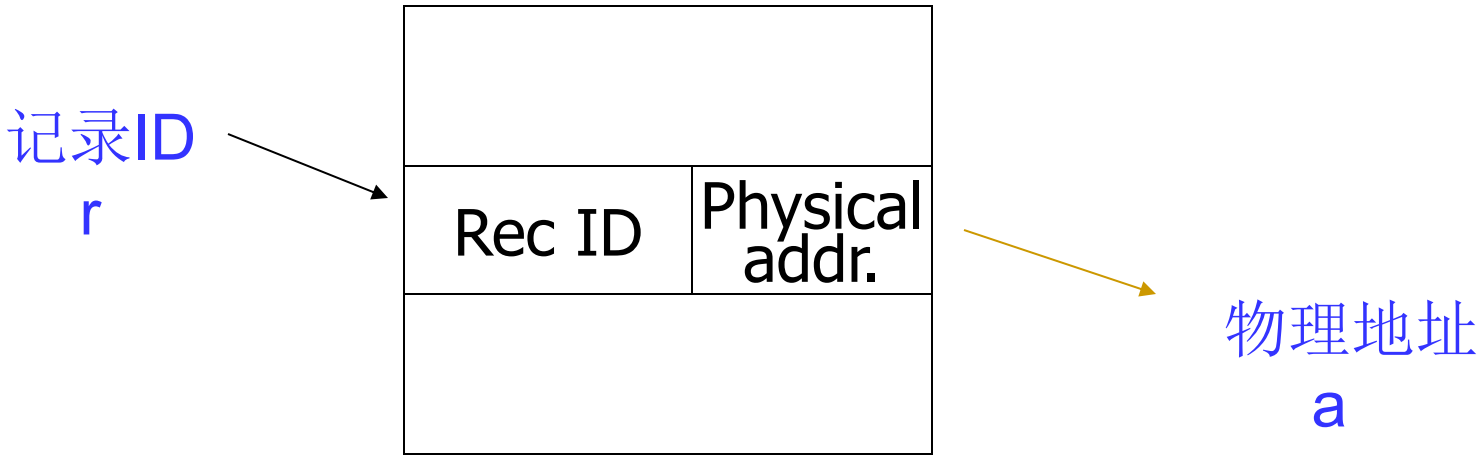
（1）逻辑地址便于从用户角度标识一条记录。

（2）存储在磁盘上的一个映射表将逻辑地址和物理地址联系起来。



块和记录地址——逻辑地址

记录ID是一个固定长度的字符串。



块和记录地址

结构地址：物理地址和逻辑地址的不同组合，构成结构地址，它提供更好的灵活性。

例如：一个结构地址可以包括块的物理地址和块内记录的键值。查找记录时，首先根据块的物理地址定位到块，然后在块内根据键值找到记录。

块和记录地址——结构地址

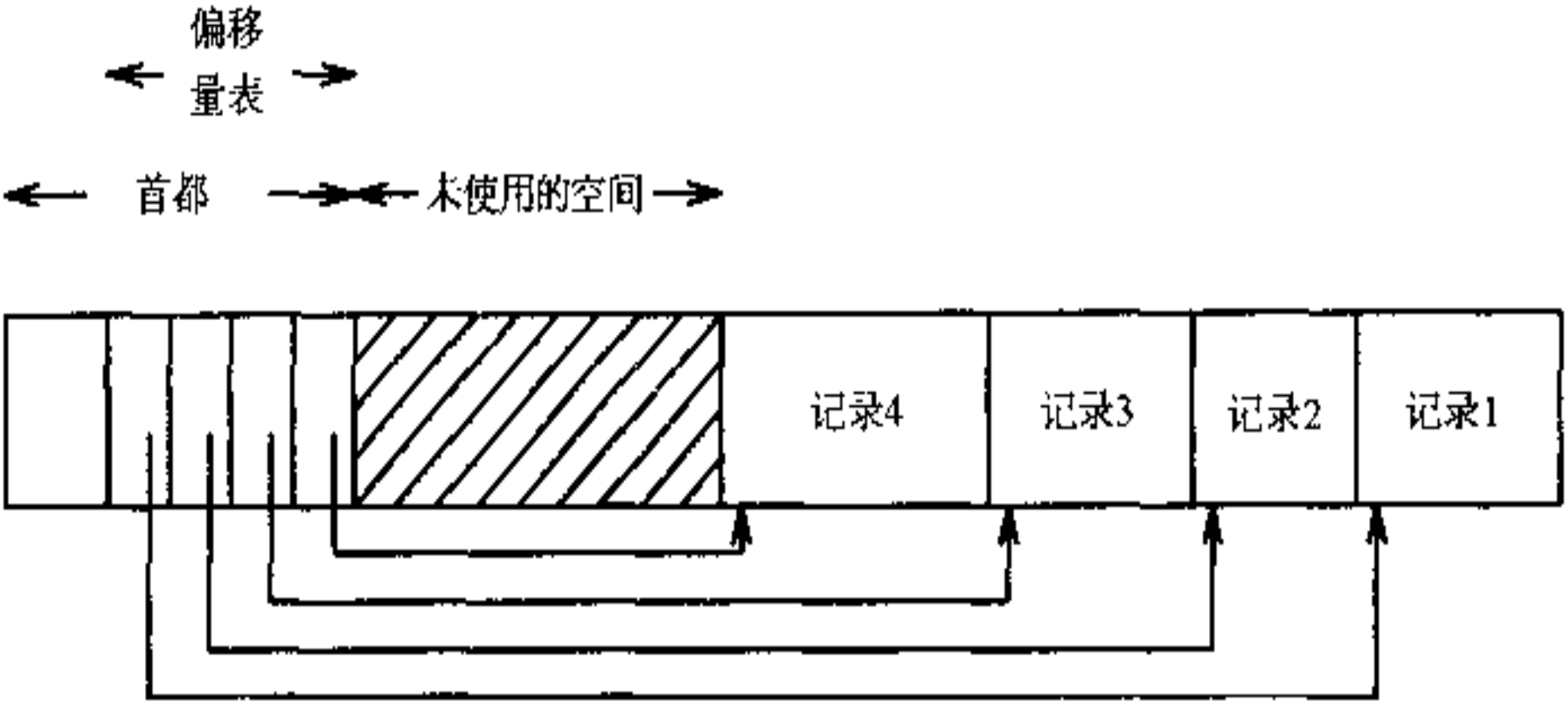
结构地址实现技术：

常用的物理地址和逻辑地址组合技巧是每一个块内存储一个偏移量表，它保存了记录的块内偏移量。

注意：在块内，偏移量表从前往后存储，记录从后往前存储。

记录的结构地址：块地址+对应偏移量表项的偏移量（参见图3-8）

块和记录地址——结构地址



块和记录地址——服务器数据地址空间

结构地址优点：

（1）可以在块内移动记录，我们只要修改对应的偏移量表中的表项，外部指向本块的指针不变。

（2）如果偏移量表项足够大，可以存储这条记录的“转向地址”，甚至可以将记录在块间迁移。

（3）记录删除后，对应表项修正为空指针。

块和记录地址——指针混写

针对问题？块在主存和二级存储器间移动时的指针管理。

每一个块、记录、对象或其他数据项都有两种地址形式：

（1）数据库地址，通常是8个字节，指明它在第二级存储器中的地址。

（2）内存地址，通常是4个字节（32位机）。

注意：一般地址指二级存储的地址，指针指内存中的地址。

块和记录地址——指针混写

地址（指针）维护的基本原则：

（1）当数据项位于二级存储器时，必然使用数据库地址（包括物理地址、逻辑地址或结构地址）。

（2）当数据项进入缓存时，既能通过数据库地址引用它，也可以通过主存指针引用它（如果该块在主存中则用内存指针，否则用数据库地址）。

指针维护的方法：

在内存中维护一个转换表（pp.72图3-9），该表记录了虚拟存储中所有的数据库地址和主存地址的对应关系。

块和记录地址——指针混写

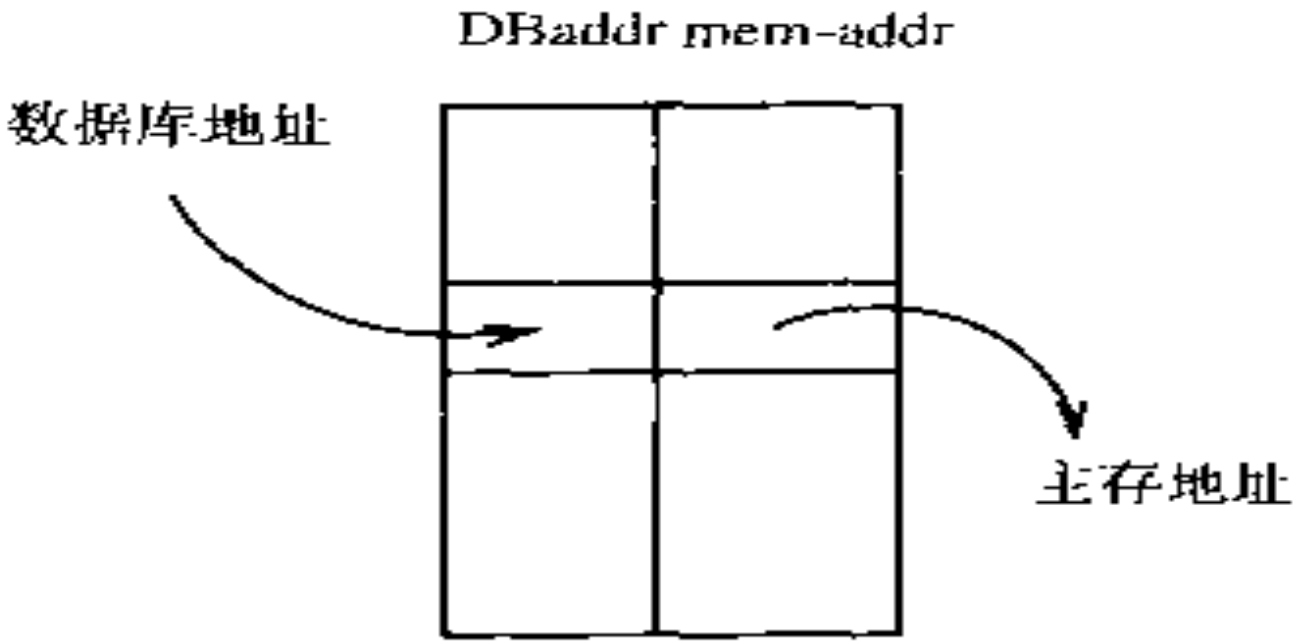


图3-9　转换表将数据库地址转换成内存中的相应地址

块和记录地址——指针混写

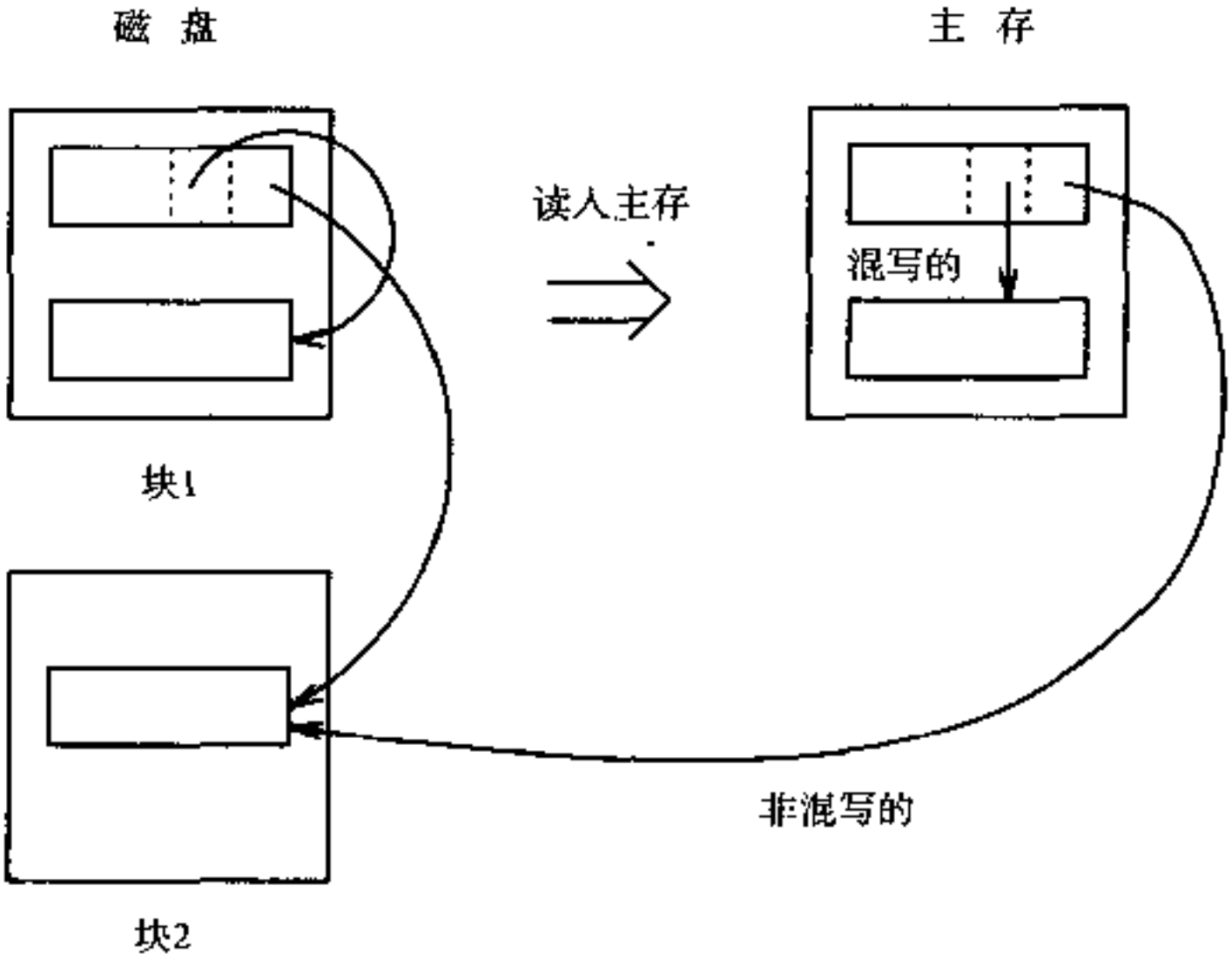
为了方便维护，一个地址的表达包含两个部分（类似C的struct结构）

（1）一个二进制位，指明当前的指针是数据库地址还是内存指针。

（2）一个内存指针或数据库地址，注意要保留足够长的空间。（类似C的union结构）

指针混写：把块从二级存储器移动到主存时，涉及的地址要进行指针混写，即从数据库地址转换为内存指针（见下图）。

块和记录地址——指针混写



块和记录地址——指针混写

前图的分析（pp.72例3.9）

（1）块1中有一条记录，这个记录中有两个指针，第一个指针指向块1中的另外一条记录，第二个指针指向块2中的一条记录。

（2）当块1被拷贝到内存中时，第一个指针被混写，指向内存中的目标记录。由于块2不在内存中，因此块1中的第二个指针不被混写。

块和记录地址——指针混写

一个块放入主存，有两类地址要考虑进行混写

（1）块和块内记录的地址

（2）块中的地址（指针）项

块和记录地址——指针混写

混写策略

（1）自动混写：块一旦放入内存，为它所有的地址定位，如果对应地址也已内存，则改写为内存地址；并且如果地址不在转换表中，则将它们放入转换表中。

（2）按需混写：块第一次进入内存时，只在转换表中记录

块和记录的地址->内存位置，

块中包含的地址->内存位置

块中包含的地址当被追踪时再进行混写。

（3）不混写：除块地址的对应关系外，不进行其他对应关系的混写。

块和记录地址——指针混写

自动混写

把块放入主存时，对块的所有的指针混写，并且写入转换表，指针包括

（1）块的地址
（2）块内记录的地址
（3）指向其他块的指针？

块和记录地址——指针混写

块内各个地址定位机制包括

（1）块首部存储记录位置的偏移量列表

（2）根据存储记录的模式找到记录内的指针

块和记录地址——指针混写

指针自动混写过程

（1）块地址和块内记录地址直接添加到内存中的转换表

（2）对于记录内的指针，向转换表添加数据库地址A时

（a）如果A在转换表中，用对应项的内存地址替换A，同时将混写位置为“真”。

（b）如果A不在转换表中，表示对应块还未拷贝入内存，此时不用混写指针（既不改写，也不加项），将其保留为数据库地址。

块和记录地址——指针混写

指针按需混写过程和自动混写类似，只不过时机选择策略不一样。

（1）开始时，块地址、块内每个记录地址和对应的内存地址写入转换表。

（2）记录内的指针，只在转换表写入指针项和对应内存位置，但对应内存位置的内容不修改。

（3）记录内指针被追踪时，再改写对应位置的内容。