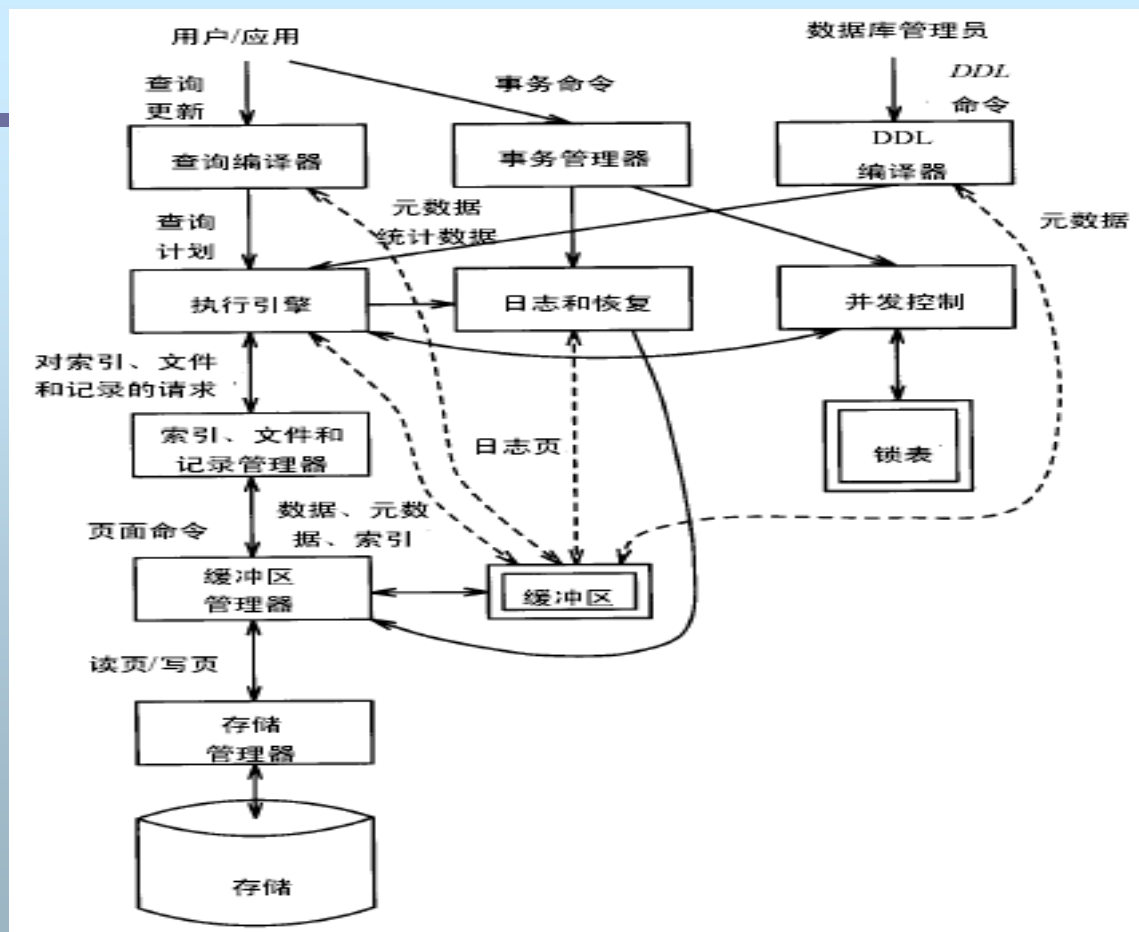


Data Representation





主要内容

- 数据项的表示 (**Data Items**)
- 记录的表示 (**Records**)
- 记录在块中的组织 (**Block**)
- 记录的修改
- 块在文件中的组织

数据元素的表示层次

数据项

属性值的物理组织



记录

元组的物理组织



块

记录的物理存放



文件

文件由磁盘块构成

一、数据项的表示

■ 数据项

- 字节序列
- 表示关系数据库中元组的属性值

1、数据项表示的内容

■ 表示什么？

- 姓名
- 年龄
- 出生日期
- 照片
-

■ 用什么表示？

- **Bytes**

2、数据项表示方法：SQL数据类型

■ Integer (short)

- 2 bytes

- 例如, 35 表示为

00000000

00100011

■ Real, Float

- 4 bytes (32 bits)

- N bits表示小数, M bits表示指数

2、数据项表示方法：SQL数据类型

■ Char(n) 或 Character(n) 定长字符串

- 小于n时使用特殊填充符

- 例如，若属性类型为**Char(5)**，则属性值'cat' 表示为

c	a	t	⊥	⊥
---	---	---	---	---

由DBMS决定填充什么

■ Varchar(n) 变长字符串

定长数据 NULL终止符，例 **Varchar(5)**

c	a	t	⊠
---	---	---	---

常用方式 ● 带长度

3	c	a	t	⊠
---	---	---	---	---

提高读的能力

- 定长表示，**n+1 bytes**

Varchar(4):

c	a	t	⊥	⊥
---	---	---	---	---

空间利用率低，时间上高效

2、数据项表示方法：SQL数据类型

■ Boolean

- **TRUE**

1111 1111

- **FALSE**

0000 0000

■ 枚举类型

- **{RED, GREEN, YELLOW}**

- **整数表示**

- ◆ RED \leftrightarrow 1, GREEN \leftrightarrow 2, YELLOW \leftrightarrow 3

- ◆ 若用两个字节的短整型来表示, 则可以表示 2^{16} 个不同值

2、数据项表示方法：SQL数据类型

■ Date

Oracle 日期转换函数 { 输入 to_date ()
输出 to_char ()

- 10字符(SQL92): 'YYYY-MM-DD'字符串表示
- 8字符: 'YYYYMMDD'
- 7字符: 'YYYYDDD', NOT 'YYMMDD'! → 天数
- Integer, 自1900-01-01以来的天数

■ Time

- 8字符(SQL92): 'HH:NN:SS' ——整数秒
- Varchar(n): 'HH:NN:SS.FF' ——带小数秒
- Integer, 自00:00:00以来的秒数

2、数据项表示方法：SQL数据类型

■ Bit

- 带长度的二进制位串

Length	Bits
--------	------

- 按字节表示，例如 **010111110011**

01011111	00110000
----------	----------

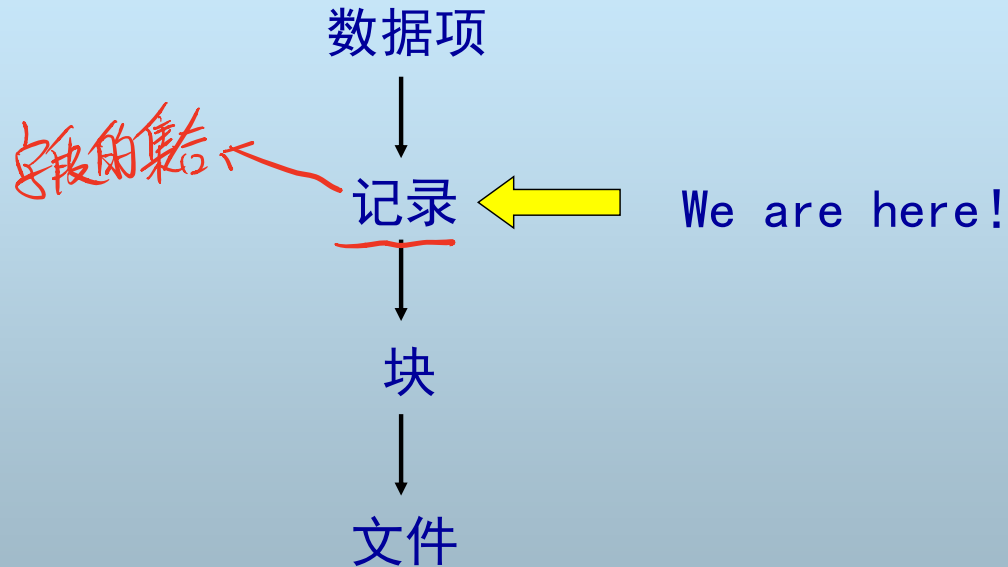
3、两种不同的数据项表示

- 定长数据项
- 变长数据项
 - 带长度（常用!）
 - **Null Terminated**

数据项表示总结

类 型	表 示 方 法		
整数和实数	字节串		
定长字符串	<i>n</i> 字节的数组		
变长字符串	用 <i>n</i> + 1字节		
VARCHAR(<i>n</i>)	长度加内容	空值-终止字符串	
日期和时间	某种格式的定长字符串	变长值	整数
二进制位序列	长度加内容	字节表示	
枚举类型	使用整数编码表示一个枚举类型的值		

Where are we?



二、记录的组织

■ 记录

- 数据项 [字段, **Fields**] 的集合

E.g.: Employee record:

name field,

salary field,

date-of-hire field, ...

1、记录的类型

- 固定格式 vs. 可变格式
Fixed Format vs. Variable Format
- 定长 vs. 变长
Fixed Length vs. Variable Length

2、固定格式定长记录

- 所有记录具有相同的逻辑结构（模式）
- 记录的模式（**Schema**）
 - # fields
 - Name of each field
 - Type of each field
 - Order in record
 - Offset of each field in the record

E.g. 固定格式定长记录

Employee record

(1) E#, 2 byte integer

(2) Ename, 10 char.

(3) Dept, 2 byte code

Schema

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

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

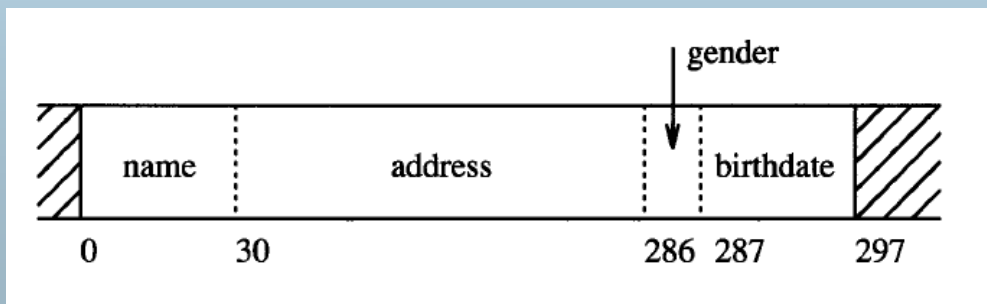
Records

2、固定格式定长记录

■ 构造

```
CREATE TABLE MovieStar(  
    name CHAR(30) PRIMARY KEY,  
    address VARCHAR(255), 定长  
    gender CHAR(1),  
    birthdate DATE  
);
```

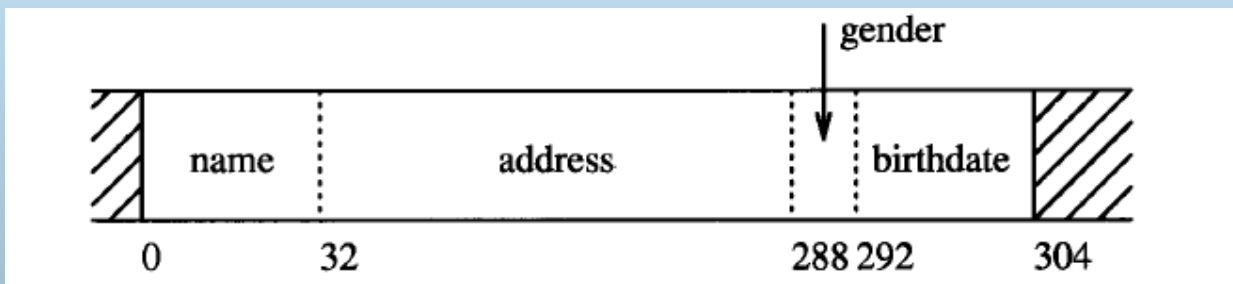
■ 不考虑寻址特点



2、固定格式定长记录

■ 考虑寻址特点

- 假设记录和字段的开始地址必须是4的倍数



3、记录首部

■ 在记录首部（Head）的描述记录的信息

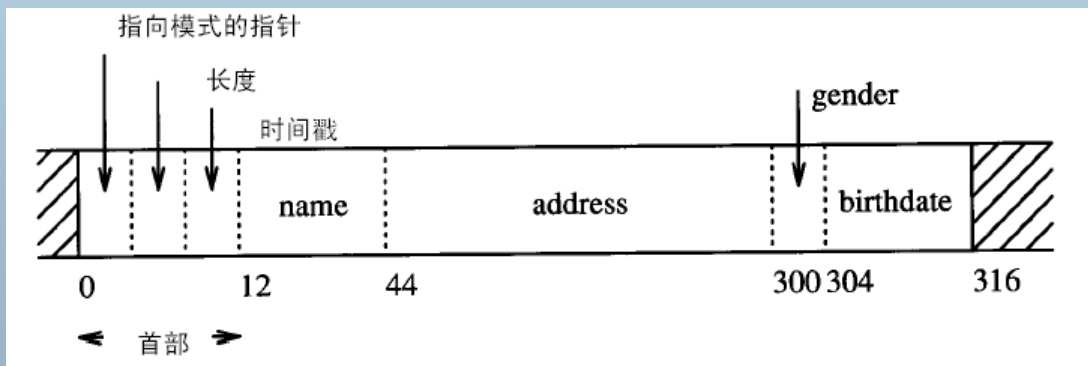
- 记录类型（模式信息）

首部 → 96 byte.

- 记录长度

- 时间戳 控制数据的写冲突

- 其它信息

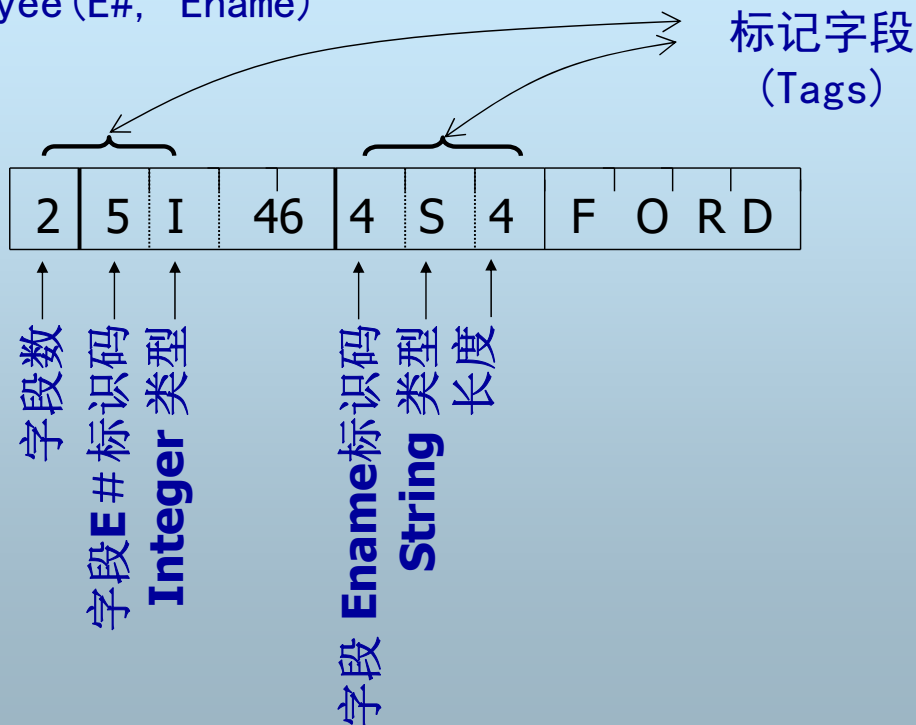


4、可变格式记录

- 每个记录的格式不同
- 记录的格式存储于记录中

E.g. 可变格式变长记录表示

Employee (E#, Ename)



E.g. 可变格式变长记录表示

- **Key-Value**
- 记录都以“**KEY+VALUE**”方式表示
- **KEY与VALUE**都以字节流（**byte string**）存储，如下：
↳ 变长的字符串

```
typedef struct {  
    void *data; //字节流指针  
    int size; //字节流长度  
} DBT;
```

- BerkeleyDB
- Memcached
- Redis
- LevelDB

- 数据类型没有限制
- 应用与数据库之间不需转换数据格式
- 不提供KEY和VALUE的内容和结构信息
- 应用必须知道所用的VALUE的含义

4、可变格式记录

■ 好处

- 灵活的记录格式，适合“松散”记录
 - ◆ 尽管一个记录可能有大量字段，但某个记录通常只有有限的几个字段
 - ◆ 例如，病人的检验结果
- 适合处理重复字段
- 适合记录格式演变

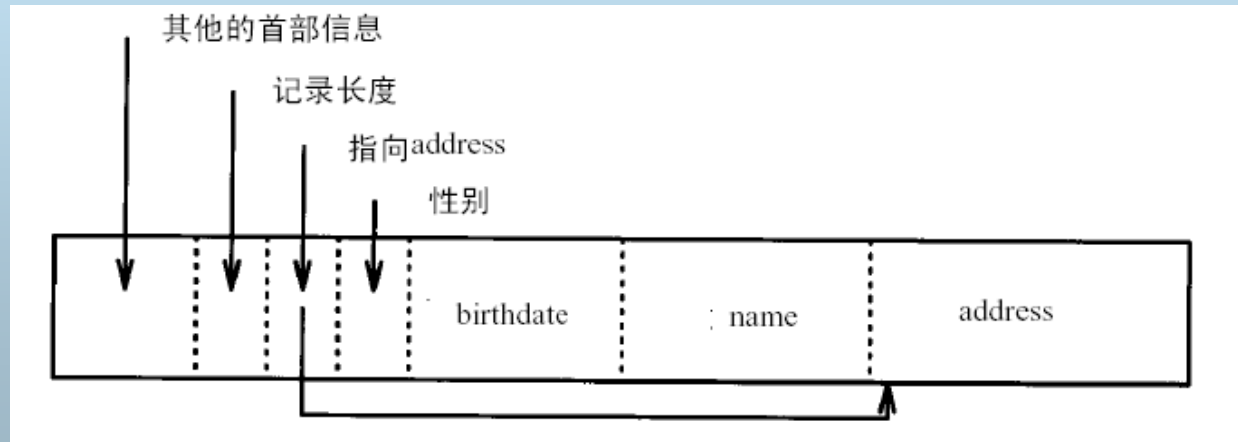
■ 缺点

- 浪费存储空间

6、变长记录表示

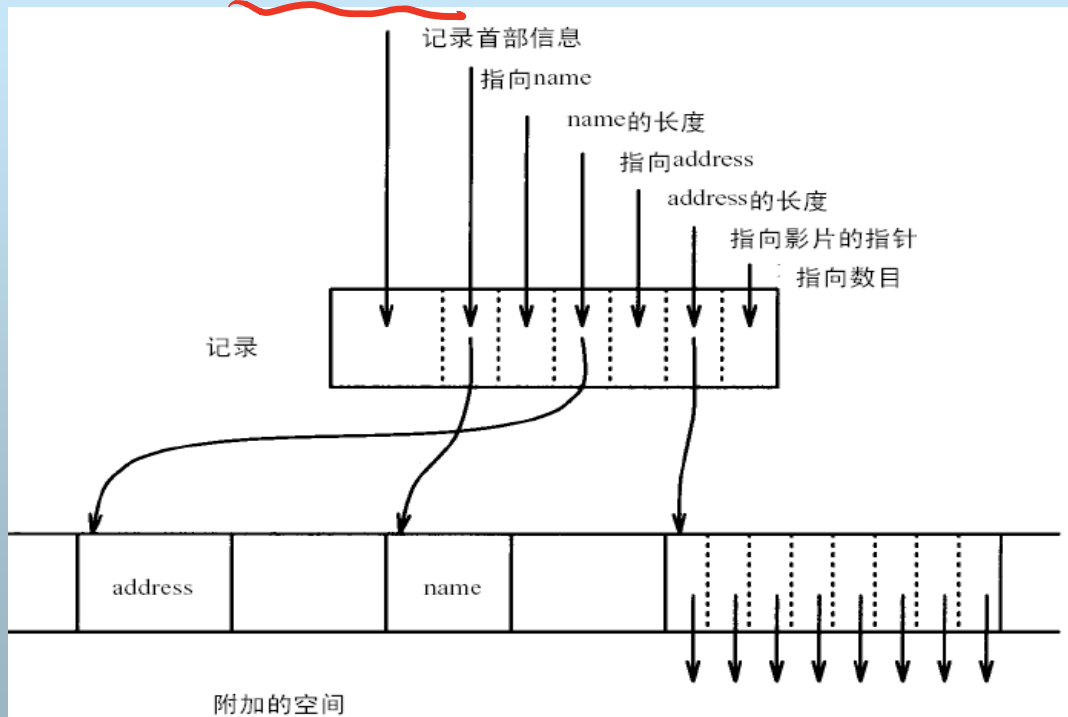
■ 首部指针法

- 定长字段在前，变长字段在后
name、address变长



6、变长记录表示

- 混合格式：定长记录 + 变长记录



Where are we?

数据项



记录



块



We are here!

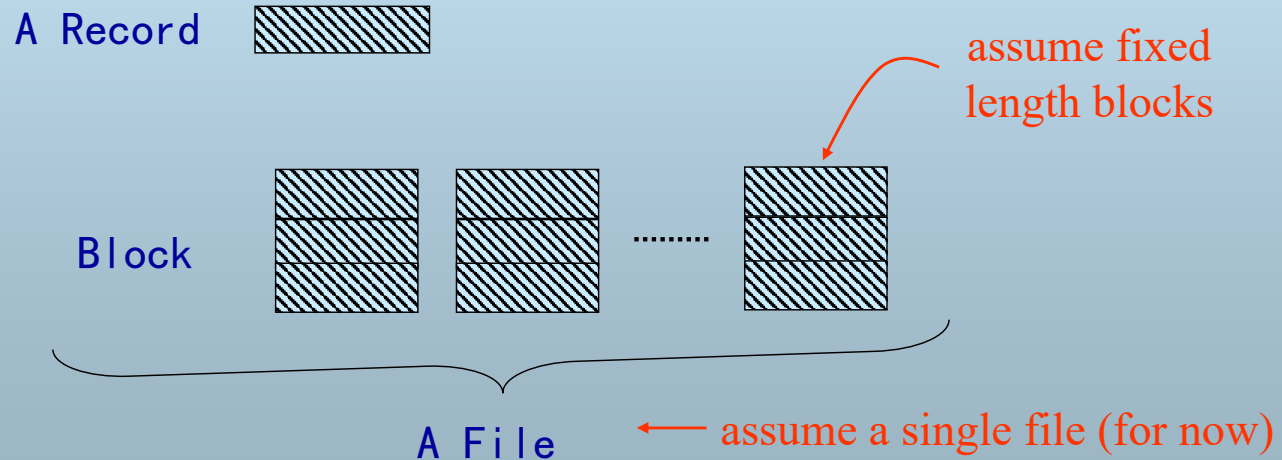


文件

三、记录在块中的组织

■ 假设

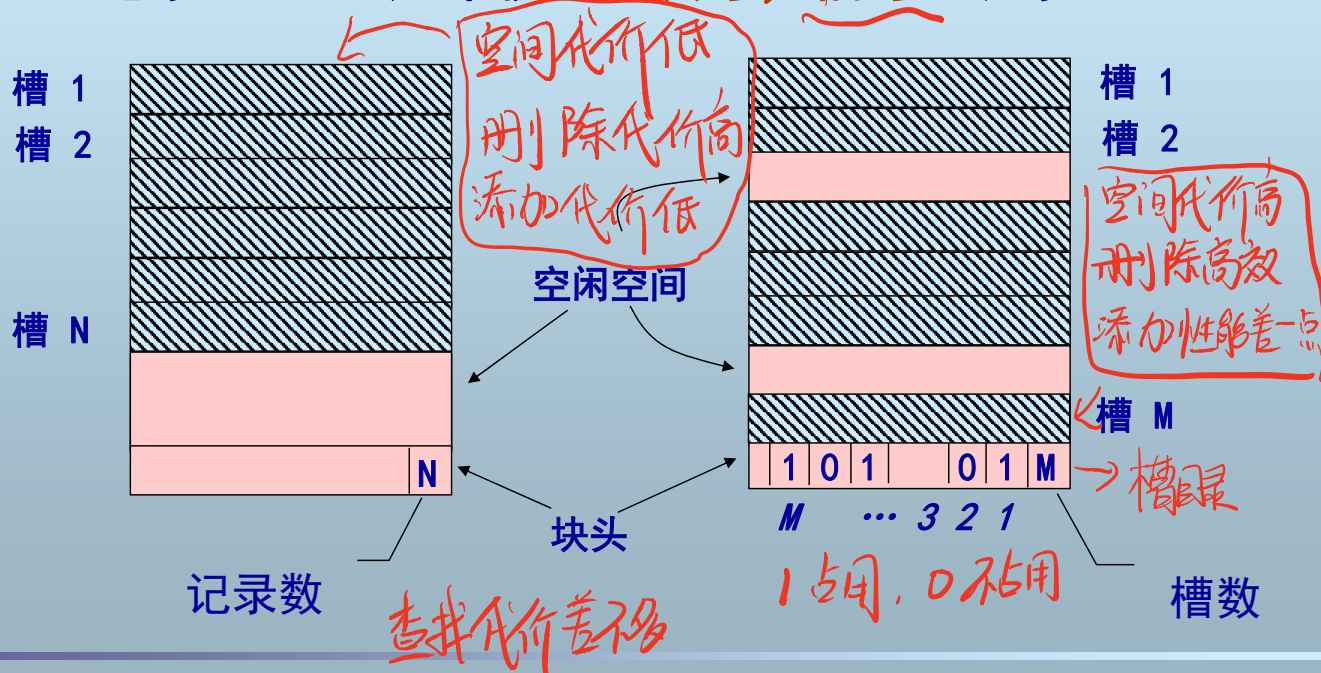
- 块的大小固定
- 记录组织成单个文件



三、记录在块中的组织

■ 定长记录的两种块内组织

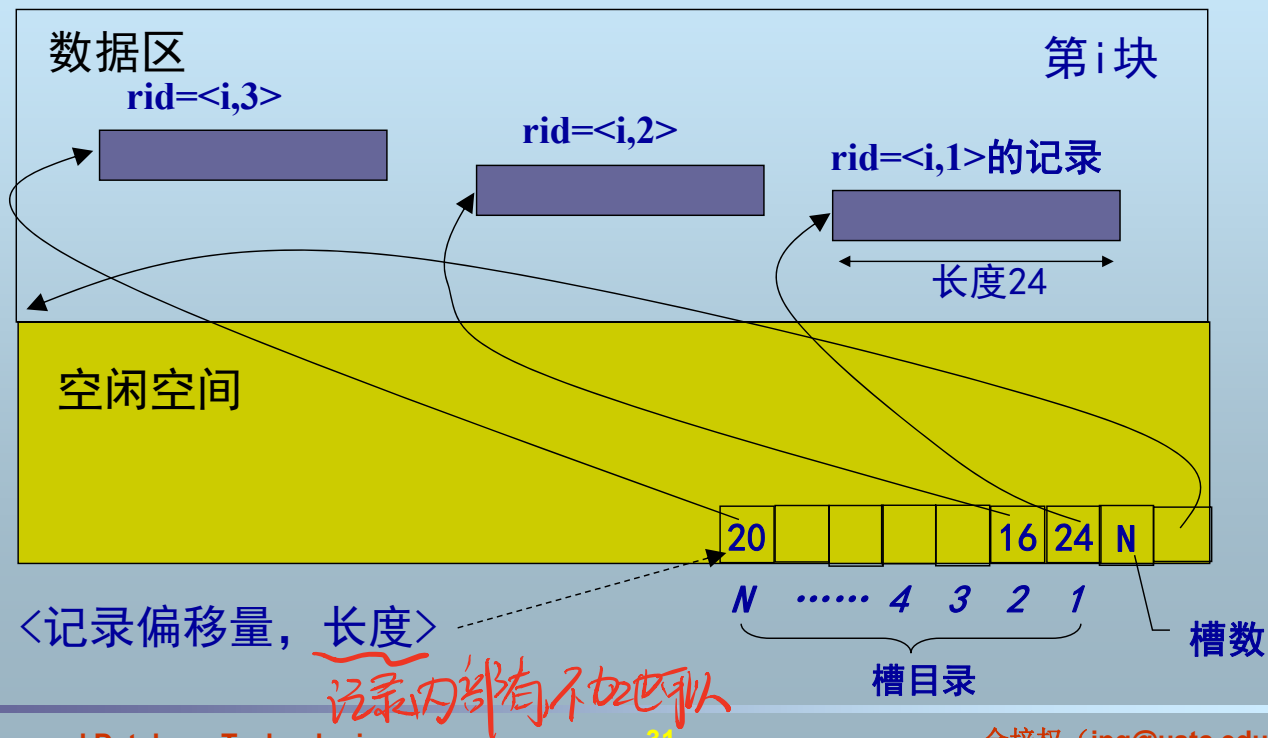
- 记录地址rid通常使用 <块号, 槽号> 表示



三、记录在块中的组织

需要寻址

■ 变长记录在块内的组织

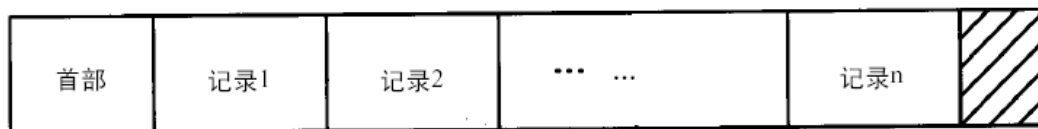


三、记录在块中的组织

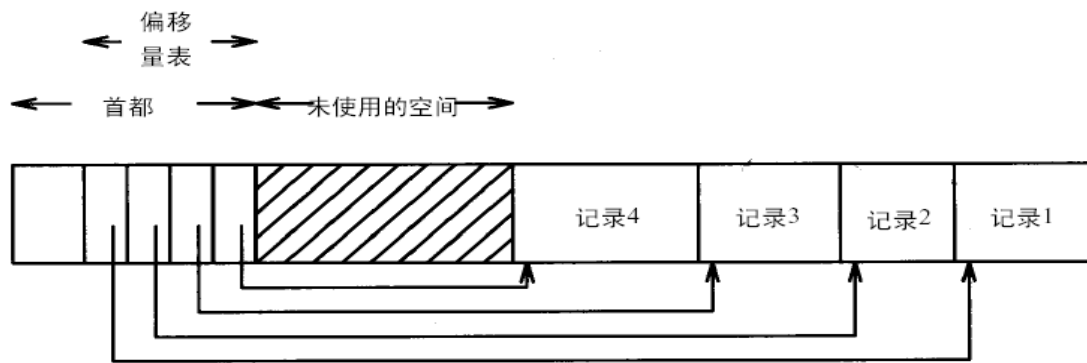
■ 其他问题

- 记录在块中的分隔 (**separating records**)
- 记录跨块 **vs.** 记录不跨块 (**spanned vs. unspanned**)
- 不同类型的记录聚簇 (**mixed record types – clustering**)
- 按序组织 (**sequencing**)
- 记录的分裂 (**split records**)
- 记录地址 (**record address**)
- 记录的修改

1、记录在块内的分隔

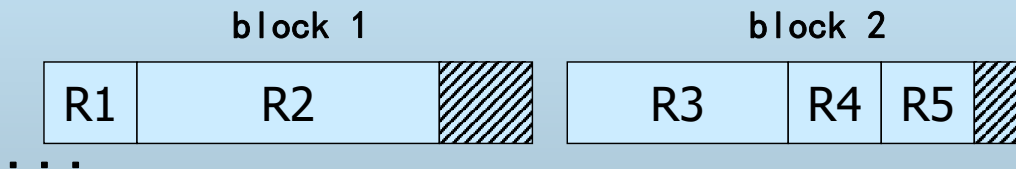


- 定长记录：不需分隔
- 使用特殊标记
- 通过块内偏移量

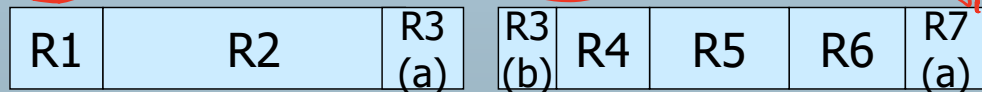


2、跨块 vs. 不跨块

- Unspanned: 记录必须在一个块中存储



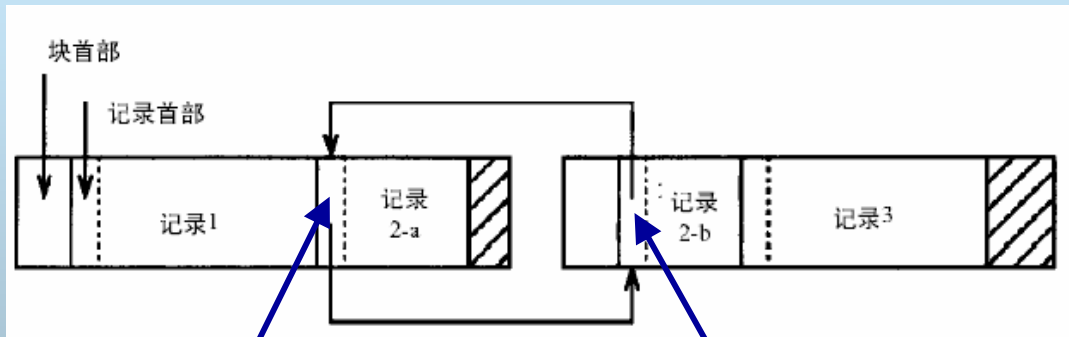
- Spanned: 记录可跨块存储



空间利用率高, 但维护代价高

2、跨块 vs. 不跨块

■ 跨块



What' s the rest?

From where?

2、跨块 vs. 不可跨块

■ 比较

- **unspanned**: 实现简单, 但空间浪费
- **spanned**: 有效利用空间, 实现更复杂

■ But

- **If record size > block size, MUST be spanned**

3、不同类型的记录聚簇

物理设计一瞥

- 一个块中存储不同类型的记录
(对于RDB: 多关系上的聚簇)



A Block

- 好处——聚簇 (clustering)

物理设计

- 经常一起访问的记录存储在同一块或连续块中

3、不同类型的记录聚簇

Block



学生表与课程表通过簇键“学号”聚簇

3、不同类型的记录聚簇

STUDENT (s#, sname, age)

SC (s#, cname, score)

Q1: select student.s#,sc.cname from student s,sc where s.s# = sc.s#

Q2: select * from student


- 如果Q1经常被查询，则聚簇非常有效
- 若Q2经常被查询，则聚簇反而降低了效率

4、在块中按序存储记录

- 另一种聚簇（对于RDB：单关系上的聚簇）
 - 将记录按某个字段顺序排列在块中
- 好处
 - 加快按排序字段查询记录时的效率
 - 利于归并联接（**will be discussed later**）

4、在块中按序存储记录

按Dept顺序组织的Student记录



化学系
化学系
化学系
化学系
物理系
物理系
物理系
中文系

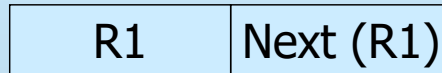
无序组织的Student记录

化学系
物理系
物理系
化学系
中文系
化学系
物理系
化学系

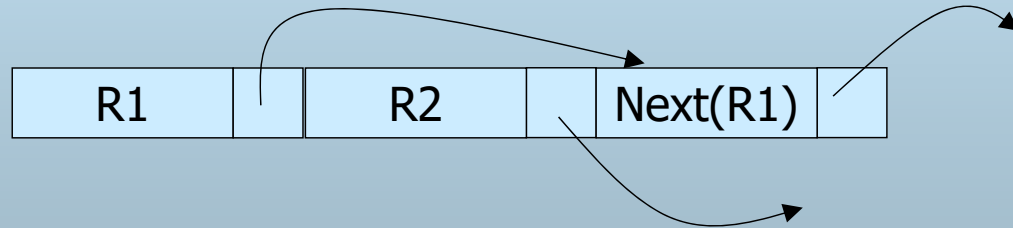
假设一个磁盘块2条定长记录

4、在块中按序存储记录

■ 物理连续



■ 指针连接



5、记录的分裂

■ 适合于变长记录的混合格式表示

- 定长部分存储于某个块中
- 变长部分存储于另一个块中
- 与spanned存储类似

跨块

6、记录地址

- 物理地址
- 逻辑地址（间接地址）

6、记录地址

■ 记录的纯物理地址

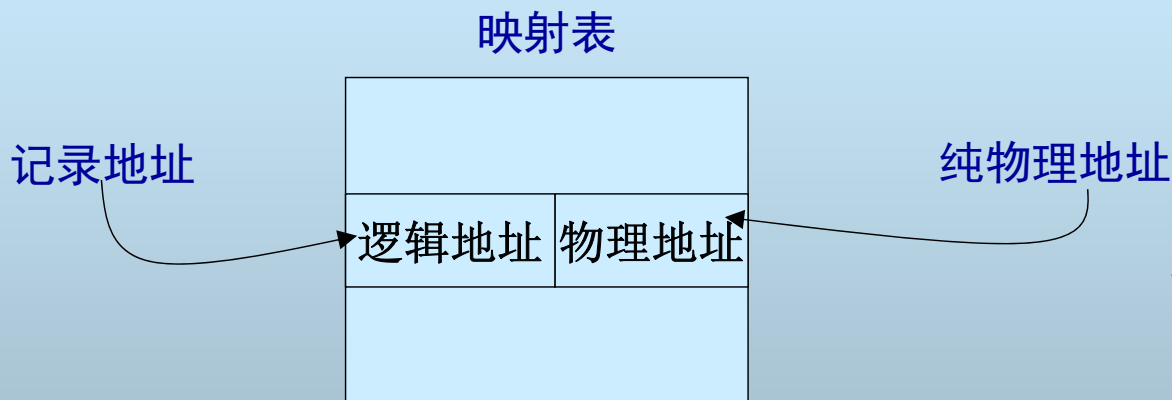
- 主机标识
- 磁盘或其他设备标识
- 柱面号
- 磁头号（盘面号）
- 块号
- 块内的偏移量



块地址

6、记录地址

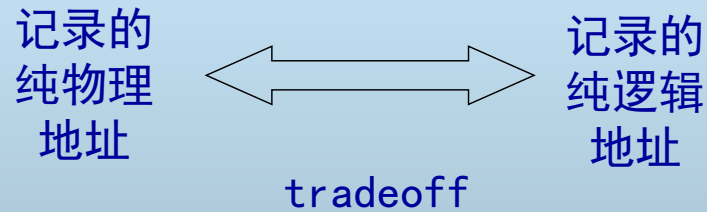
■ 记录的纯逻辑地址



缺点——访问代价增加：映射表占存储空间；需要地址转换

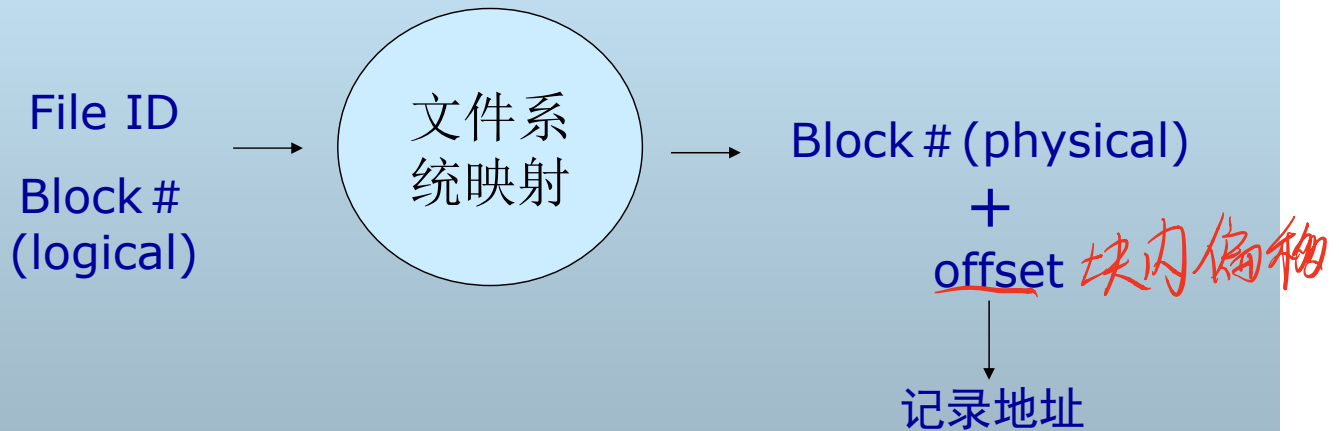
好处——灵活性：删除或移动记录时只要改变映射表项

6、记录地址



6、记录地址

- 借助文件系统的逻辑块地址
 - 文件号 + 逻辑块地址 + 块内偏移



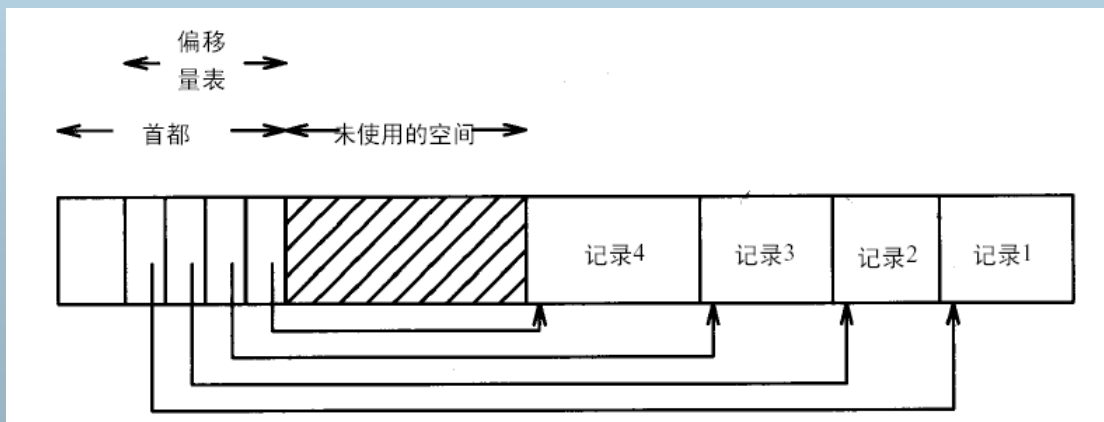
四、记录的修改

- 插入
- 删除

1、插入

■ 记录无序

- 插入到任意块的空闲空间中
- 或申请一个新块（当所有块都已满时）
- 记录变长时，可使用偏移量表



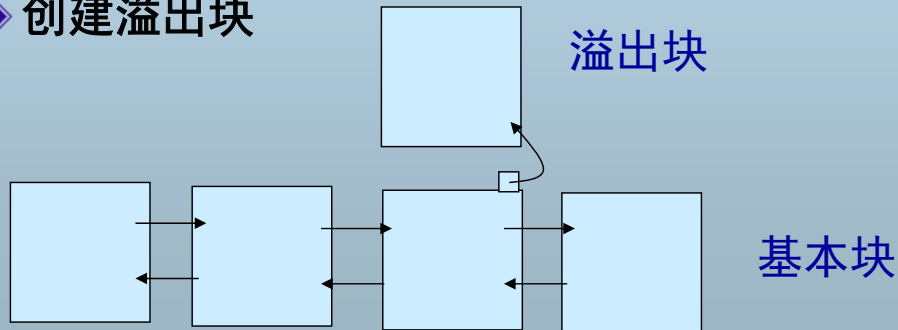
1、插入

■ 记录有序

- 找到记录应该放置的块
- 如果有空间，放入并调节记录顺序即可，否则有两种方法：

- ◆ 在“邻近块”中找空间

- ◆ 创建溢出块



2、删除

■ 立即回收空间

- 例如，加到可用空间列表中

■ 删除记录时处理溢出块

- 若删除的记录位于溢出块链上，则删除记录后可对整个链进行重新组织以去除溢出块

2、删除

■ 使用删除标记

- 若使用偏移表，则可以修改偏移表项指针，将其置空
- 若使用逻辑—物理地址映射表，则可以将物理地址置空
- 可以在记录首部预留一开始位：**0**—未删除，**1**—已删除

	1	记录1	0	记录2
--	---	-----	---	-----

Where are we?

数据项



记录



块



文件

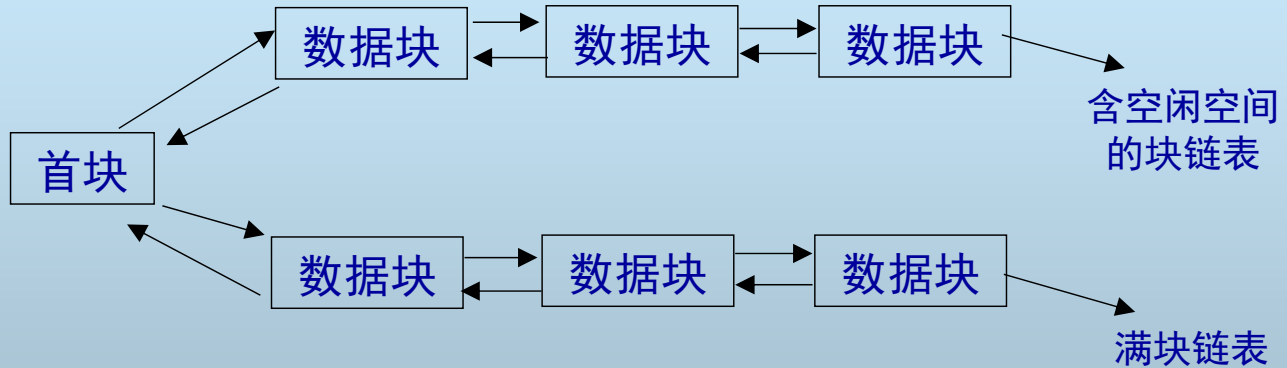


We are here!

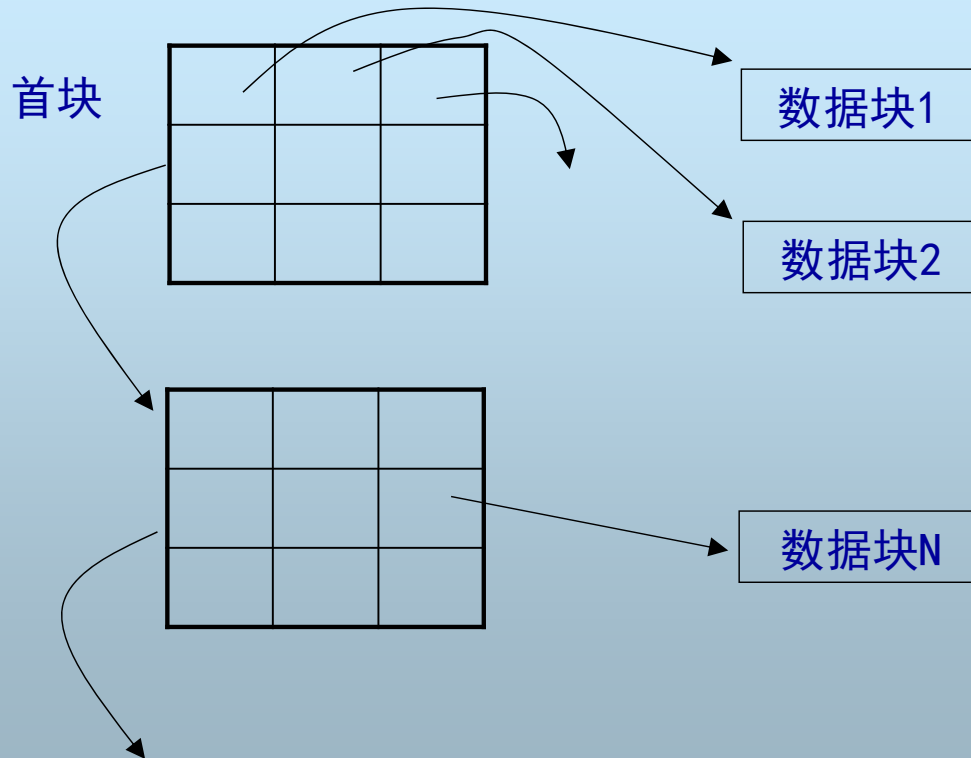
五、块在文件中的组织

- 堆文件（**Heap File**）
 - 最基本、最简单的文件结构
 - 记录不以任何顺序排序
 - 记录可能存放在物理不邻接的块上
- 插入容易，但查找和删除代价高

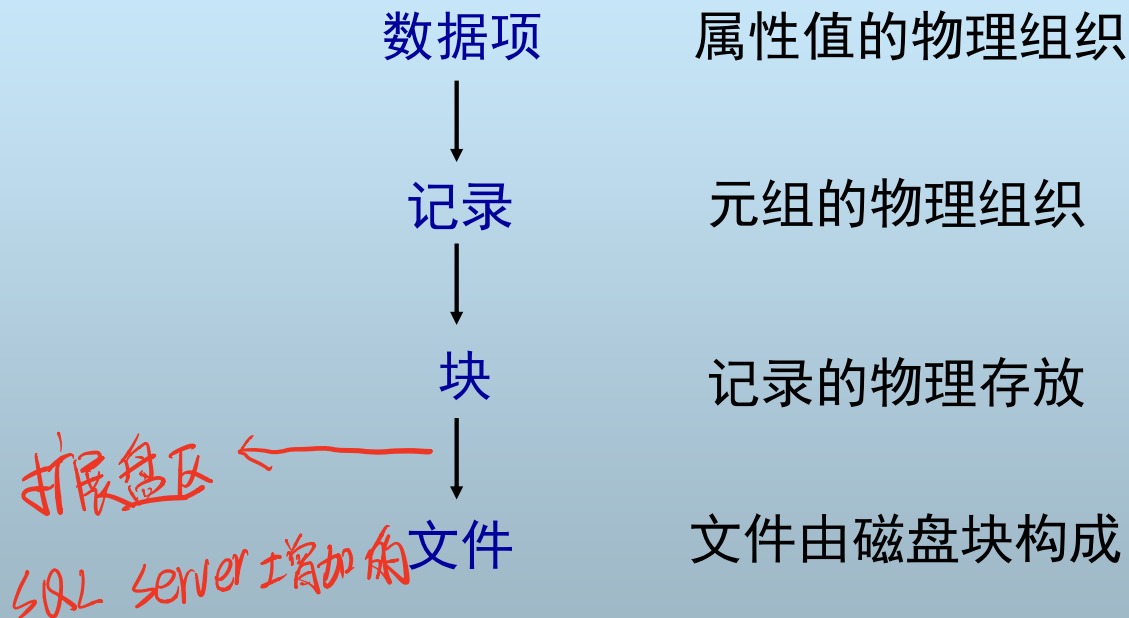
1、链表式堆文件组织



2、目录式堆文件组织



回顾：数据元素的表示层次



5层

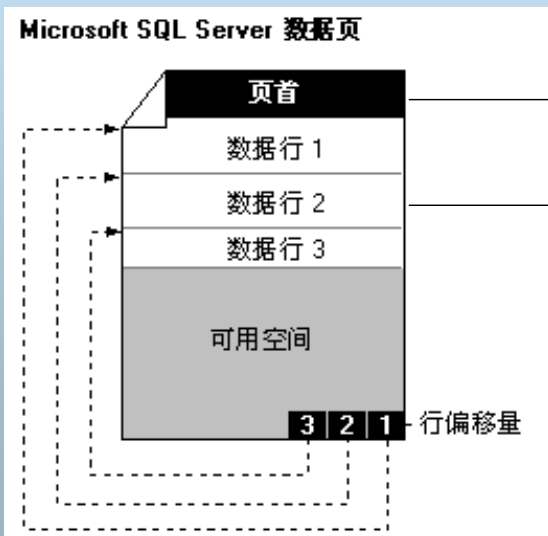
六、SQL Server的数据存储结构

- SQL Server的数据库文件是多个对象的集合，包括多个表、索引等

1、页 相当于块

- 在SQL Server中，数据存储的基本单位是页。在 SQL Server 2000 中，页的大小是 8 KB。

DB2等页的大小是可变的



96字节

单个数据行
最大8060字节

页地址：〈文件号，页号〉
数据行地址：〈页地址，槽号〉

2、扩展盘区

预先分配,降低存储管理代价

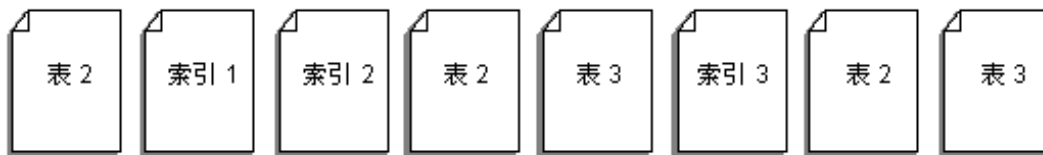
- 扩展盘区是一种基本单元,可将其中的空间分配给表和索引。一个扩展盘区是 8 个邻接的页 (或 64 KB)。
- 为了使空间分配更有效,SQL Server 2000 对只含少量数据的表不分配完整的扩展盘区。SQL Server 2000 有两种类型的扩展盘区:
 - 统一扩展盘区:由单个对象所有,扩展盘区中的所有八页只能由拥有该盘区的对象使用。
 - 混合扩展盘区:最多可由 8 个对象共享。
- 通常从混合扩展盘区中向新表或新索引分配页。当表或索引增长到 8 页时,就变成统一扩展盘区。

2、扩展盘区

■ 混合扩展盘区和统一扩展盘区

多个对象共享

混合扩展盘区



统一扩展盘区



由单个对象所有

3、SQL Server文件组织

■ SQL Server 2000 数据库有三种类型的文件：

● 主要数据文件

- ◆ 主要数据文件是数据库的起点，指向数据库中文件的其它部分。每个数据库都有一个主要数据文件。主要数据文件的推荐文件扩展名是 **.mdf**。

● 次要数据文件

- ◆ 次要数据文件包含除主要数据文件外的所有数据文件。有些数据库可能没有次要数据文件，而有些数据库则有多个次要数据文件。次要数据文件的推荐文件扩展名是 **.ndf**。

● 日志文件

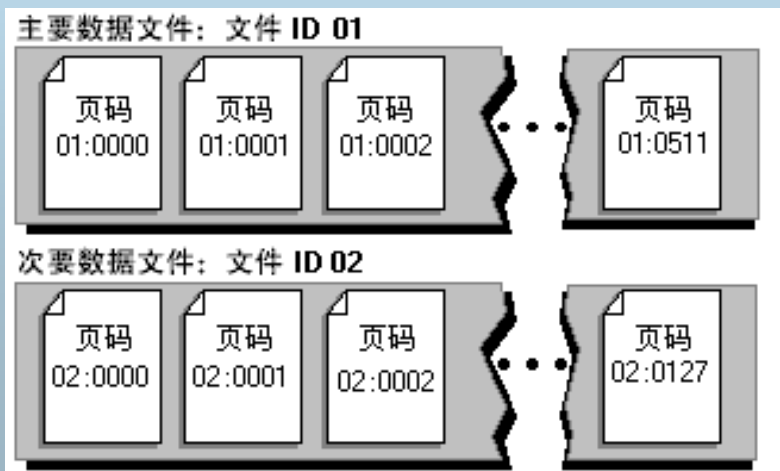
- ◆ 日志文件包含恢复数据库所需的所有日志信息。每个数据库必须至少有一个日志文件，但可以不止一个。日志文件的推荐文件扩展名是 **.ldf**。

3、SQL Server文件组织



3、SQL Server文件组织

- 数据文件的页按顺序编号，文件首页的页码是 0。每个文件都有一个文件 ID 号。在数据库中唯一标识一页需要同时使用文件 ID 和页码。



3、SQL Server文件组织

■ 数据文件的起始结构



3、SQL Server文件组织

■ 数据文件的起始结构

- **PFS**页：给对象分配了扩展盘区后，SQL Server 使用**页可用空间 (PFS)** 页记录扩展盘区的哪些页已分配或可用，以及有多少可用的剩余空间。每个 **PFS** 页包含大约 **8,000** 页。**PFS** 对每一页都有一个相应的位图，该位图记录这一页是空的、**1-50%** 已满、**51-80%** 已满、**81-95%** 已满还是 **96-100%** 已满。

3、SQL Server文件组织

■ 数据文件的起始结构

- **GAM页**：全局分配映射表 (GAM) 页记录已分配的扩展盘区。每个 **GAM** 包含 **64,000** 个扩展盘区，将近 **4 GB** 的数据。**GAM** 对所涵盖区间内的每个扩展盘区都有一位。如果这个位是 **1**，则扩展盘区可用；如果这个位是 **0**，则扩展盘区已分配。

3、SQL Server文件组织

■ 数据文件的起始结构

- **SGAM 页**：**共享全局分配映射表 (SGAM) 页记录特定的扩展盘区，这些盘区当前用作混合扩展盘区而且至少有一个未使用的页。**每个 **SGAM** 包含 **64,000** 个扩展盘区。**SGAM** 对所涵盖区间内的每个扩展盘区都有一位。如果这个位是 **1**，则该扩展盘区就用作混合扩展盘区且有可用的页；如果这个位是 **0**，则该扩展盘区不用作混合扩展盘区，或者虽然用作混合扩展盘区但其所有页都正在使用中。

3、SQL Server文件组织

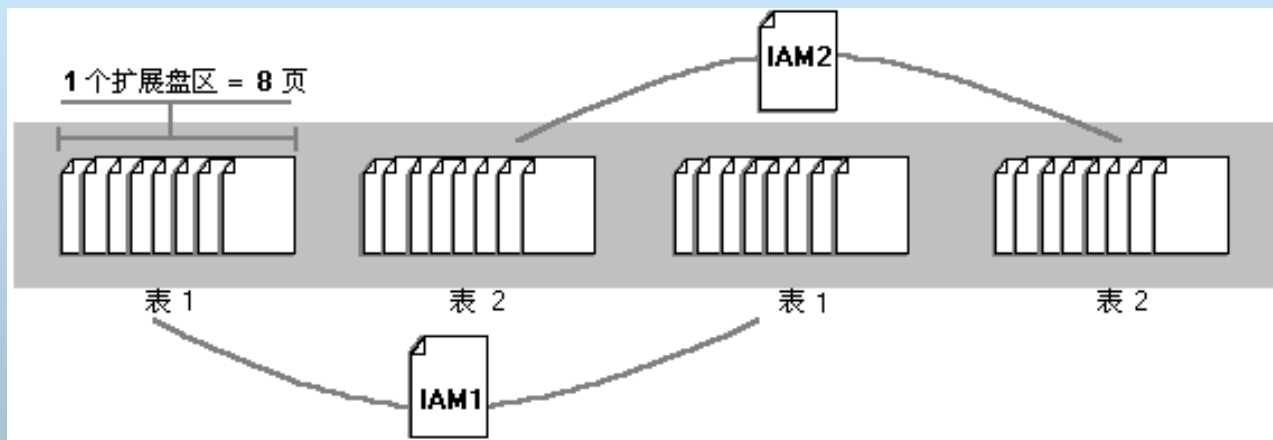
■ 数据文件的起始结构

扩展盘区的当前使用情况	GAM 位设置	SGAM 位设置
可用，未使用	1	0
统一扩展盘区或已满的混合扩展盘区	0	0
有可用页的混合扩展盘区	0	1

1. 若要分配统一扩展盘区，SQL Server 在 GAM 中搜索是 1 的位，然后将它设成 0。
2. 若要查找有可用页的混合扩展盘区，SQL Server 在 SGAM 中搜索是 1 的位。
3. 若要分配混合扩展盘区，SQL Server 在 GAM 中搜索是 1 的位，并将它设置为 0，然后将 SGAM 中相应的位也设置为 1。
4. 若要释放扩展盘区，SQL Server 应确保 GAM 位设置为 1 而且 SGAM 位设置为 0。

3、SQL Server文件组织

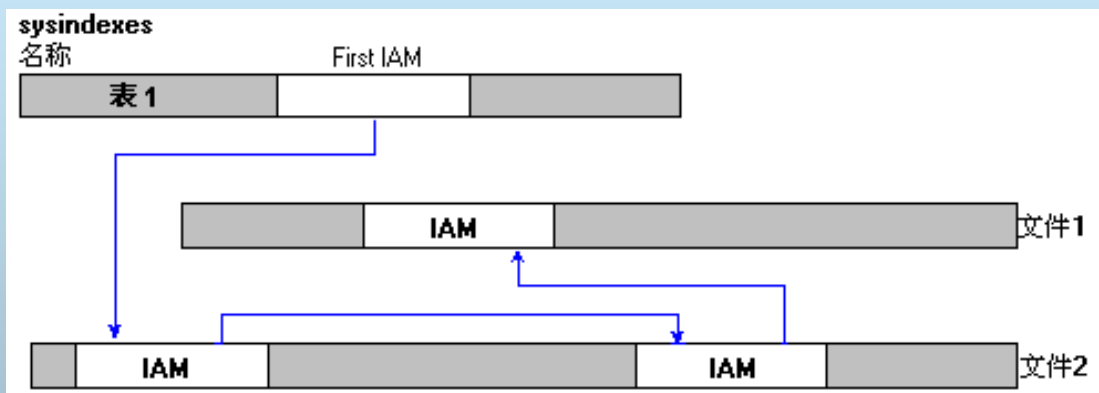
■ 表（Table）的组织



索引分配映射表（IAM）页记录了分配给对象的扩展盘区。

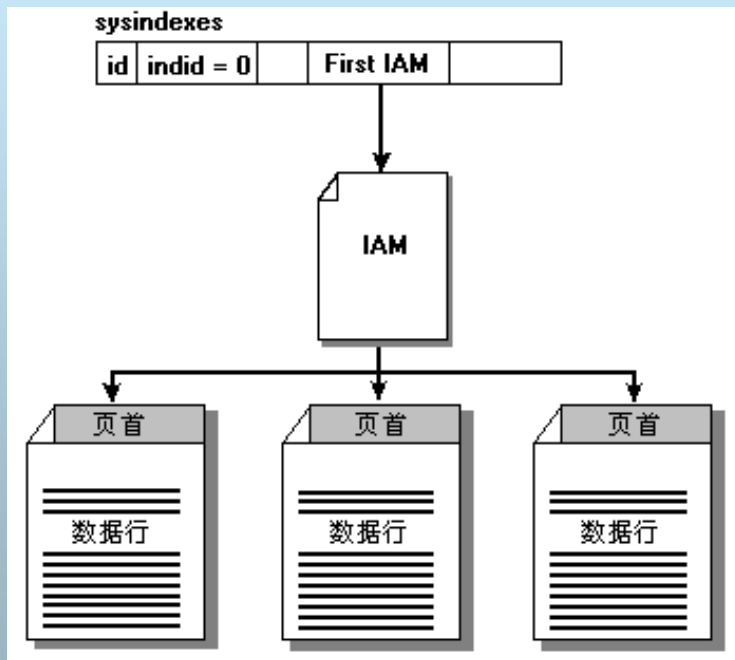
3、SQL Server文件组织

■ 表（Table）的组织



3、SQL Server文件组织

■ 表 (Table) 的组织



数据页没有任何特定的顺序，也不链接在一起。数据页之间唯一的逻辑连接是记录在 IAM 页内的连接。

服务器使用 IAM 页查找数据页集合内的页，进行表扫描或串行读。

3、SQL Server文件组织

■ 表（Table）的组织

- 当需要插入新行而当前页没有可用空间时，SQL Server 使用 IAM 页查找分配给对象的扩展盘区。对于每个扩展盘区，SQL Server 搜索 PFS 页以查看是否有一页具有足够的空间容纳这一行。

3、SQL Server文件组织

```
Create Table student  
( no char(3),  
  name varchar(15),  
  class varchar(5),  
  address varchar(10),  
  age smallint )
```

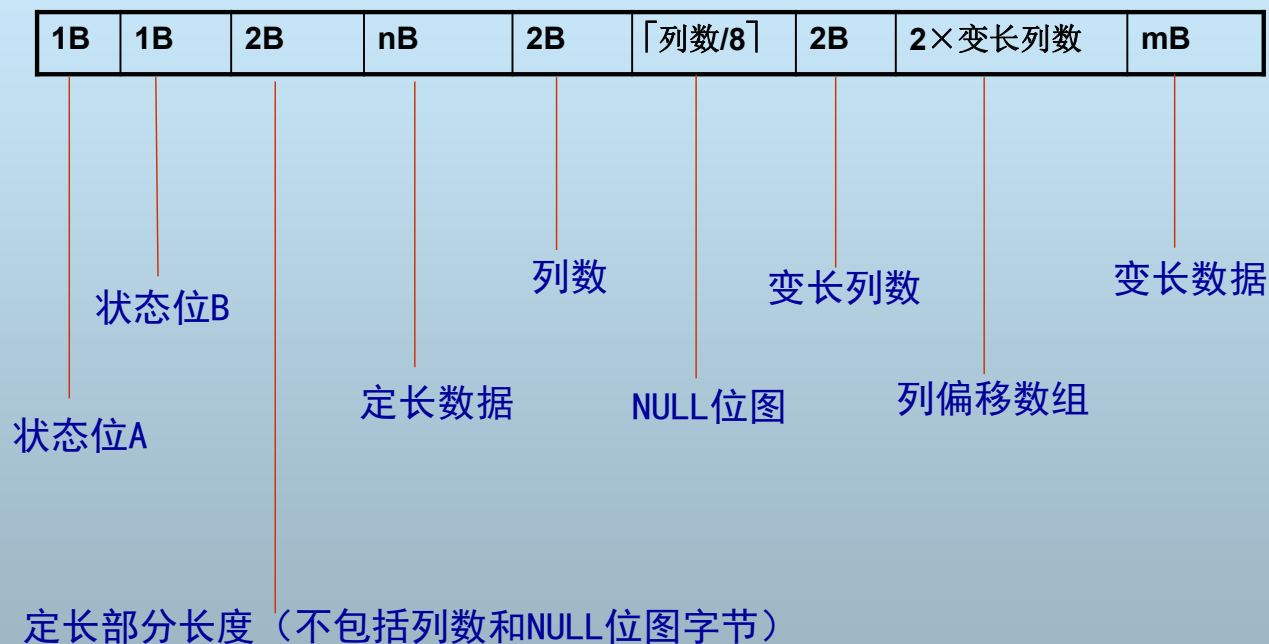
0表示无索引，
是堆文件

IAM首页

sysindexes表

	id	name	rows	first	indid	root	firstIAM
1	1977058079	student	0	0x0000000000000000	0	0x0000000000000000	0x0000000000000000

4、SQL Server记录结构



4、SQL Server记录结构

```
Create Table Fixed  
( col1 int,  
  col2 char(5),  
  col3 char(3),  
  col4 float )
```

不包括列数和NULL位图

sysindexes

	id	name	indid	first	minlen
1	1993058136	Fixed	0	0x00000000000000	24

```
Insert Into Fixed Values(123, 'ABCD', NULL, 45.6)
```

	id	name	indid	first	minlen
1	1993058136	Fixed	0	0x1E00000000100	24

文件号：1
页号：30

已分配1个数据页
00010000001E

4、SQL Server记录结构

Insert Into Fixed Values(123, 'ABCD', NULL, 45.6)

执行DBCC PAGE (test, 1, 30, 1), 4个字节一组显示

```
DATA:
-----
Slot 0, Offset 0x60
-----
Record Type = PRIMARY_RECORD
Record Attributes = NULL_BITMAP
199d6060: 00180010 0000007b 44434241 00000020 .... {...ABCD ...
199d6070: cccccccd 4046cccc 0400004 .....F@...

OFFSET TABLE:
-----
Row - Offset
0 (0x0) - 96 (0x60)
```

00000100

10	00	1800	7b000000	4142434420	000000	cdcccccccccc4640	0400	04
----	----	------	----------	------------	--------	------------------	------	----

定长部分长度24

定长字段

列数

NULL
位图

考虑

- 如何衡量数据组织方法的好坏？
 - 性能 (**performance**)
 - 灵活性 (**flexibility**)
 - 复杂程度 (**complexity**)
 - 空间利用率 (**space utilization**)

本章小结

- 数据项的表示
- 记录的表示
- 记录在块中的组织
- 记录的修改
- 块在文件中的组织