



Chapter 11.a File Organization

▼ 1. 文件组织file organization

一个数据库 → 多个文件

一个文件 → a sequence of records

a record → a sequence of fields

一个文件 → 一些固定长度的存储单元block

▼ 2. 定长Fixed-Length record

将instructor record记录构成一个文件

```
type instructor = record
    ID varchar(5);
    name varchar(20);
    dept_name varchar(20);
    salary numeric(8,2);
end
```

一个record → 一个tuple

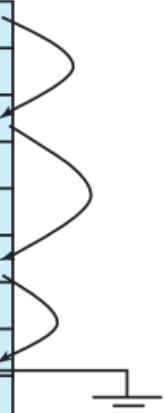
以上的一个instructor record是(5+20+20+8 = 53)个字节

我们在一个block 中只分配它能容纳的最多记录的数目

由于插入比删除更加频繁,因此让被删除的记录所占据的空间先空着,一直等到后面插入操作重新使用这个空间

在文件的开头,分配特定的字节作为头文件 (file header), 并构成**自由链表free list**

header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000



当插入的时候 `list.pop()`,

当删除的时候 `list.add()`,

当 `list.is_empty == true` 的时候插入在文件末尾

▼ 3.变长记录Variable-Length Records

需要解决的两个问题:

1. 如何表示单条记录,使得此记录的单个属性能够被轻松地提取,即使这些属性是变长的
2. 如何在一个块中存储变长的record,使得一个block中的记录能够被轻松地提取

有变长属性的记录包括两个部分:

1. 带有定长信息的初始部分
2. 紧接着是变长属性的内容 → 在初始部分被表示成一个(offset, length)数组

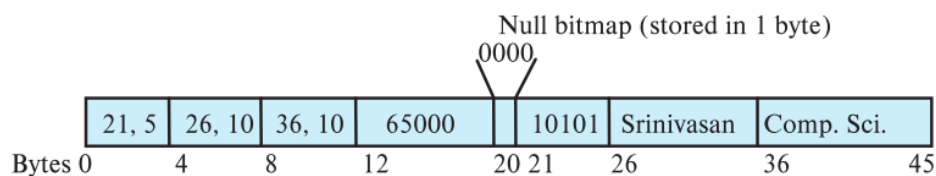
假设一个instructor 的记录前三个属性ID, name和dept_name是变长字符串,

第四个salary 是一个大小固定的数值

offset 和length分别储存在2个字节中,集每个可变长属性的初始部分都占用4个字节

0-3, 4-7, 8-11, 12-19

salary属性占用8个字节



四个bit的NULL bitmap用来标记哪些位置是空位, 有4个属性于是有4个null bit.

▼ 4. 文件中记录的组织 Organization of Records in Files

在完成如何在文件中表示记录后

下一个问题是如何在文件中组织他们

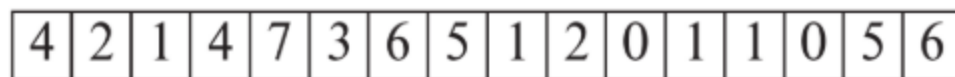
▼ Heap file堆文件组织

一个记录可以被存放在一个关系文件的任何一个位置,但是记录一旦被放置,通常不被移动

自由空间图free-space map: 一种用来节省空间的数据结构(数组)

对于关系文件中的**每一个块**对应到了数组中的一个元素

每个元素的值是自由空间的比例



2的意思就是2/8的比例的空间是自由的

储存一个新的记录,就去这个数组中找到一个合适的空间

如果不存在,就给他分配一个新块

这种方法比直接遍历查找要快,但仍然不够快

建立二级自由图:比如它的每一个元素代表了firstlevel free-space map的100项之内的自由空间最大值

4	7	2	6
---	---	---	---

在非常大的关系图中,第一次搜索的时间变成了 $1/100$, 第二次搜索的范围是100

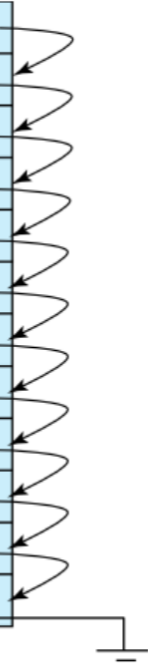
减少搜索成本的代价是怎加建立**建立二级自由图的成本**,所以建立二级自由图被周期性的写入(而不是及时更改)于是会发生建立二级自由图过时错误现象

▼ Sequential file顺序文件组织

高效处理**搜索码(search key)**:可以是任意的属性或者属性的集合)

用**链表**将搜索码**按顺序连接**(为了最大程度减少顺序文件处理中的block访问数量)

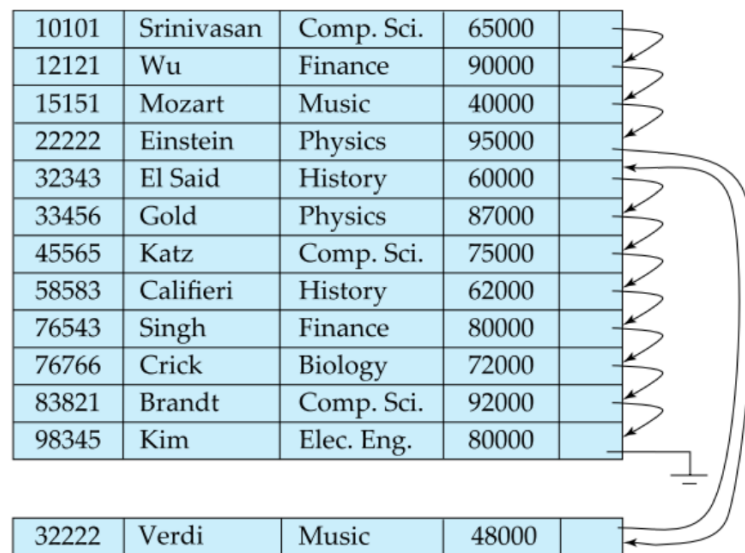
10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	



顺序排列的好处是可以**binary search**($O(\log N)$)

插入操作

1. 定位插入记录之前的那条记录
2. 安置
 - a. 如果这条记录所在的块中含有存放至少一条**自由记录**的空间, 则插入这条新记录
 - b. 否则,新纪录插入一个溢出块(**overflow block**)



3. 调整指针的连接

缺点:当存储之初是很快的因为在物理上block相邻而且record顺序排列,查找速度快
但当溢出块数量增多的时候搜索码和物理block之间的顺序的相似性变得很差,顺序处理的效率降低

文件需要根据现有的顺序**重组**,但是代价高

▼ Multitable Clustering File Organization多聚文件组织

在单个块中不止存储一个关系的记录

计算department和instructor关系的联系

```
select dept_name, building, budget, ID, name, salary
from department natural join instructor
```

对department的每一个元组,系统必须找到具有相同dept_name 值的instructor元组
在最坏的情况下, 每条instructor记录存储在不同的block上 → 每次查到所需要的记录就需要执行一次读块操作。

<i>department</i>	<i>dept_name</i>	<i>building</i>	<i>budget</i>
	Comp. Sci.	Taylor	100000
	Physics	Watson	70000

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

multitable clustering
of *department* and
instructor

Comp. Sci.	Taylor	100000	
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
Physics	Watson	70000	
33456	Gold	Physics	87000

Multitable Clustering定义：

正对特定的A_B聚簇的所有A关系元组 a_i 被输出在A_B所对应的 b_1 附近, 我们就说这两个关系A, B在A_B码上是聚簇的。

Cluster key聚簇码：

一个属性, 他定义了哪些记录被储存在一起, 比如例子中的**dept_name**

优点：加速了特定的连接查询

缺点：其他处理变慢

▼ Partitioning划分

在一个表中（关系中），记录（元组）可以被划分成更小的关系。这些关系分别进行存储这种**表划分partitioning**通常基于一个属性值来完成

例如把transaction交易记录这个关系按照年份划分

```
select *  
from transaction  
where year = 2019
```

经过划分后，将只访问**transaction_2019**关系

优点：

1. 减小表（关系）的规模可以减少操作的时间代价，如寻找自由空间
2. 可以将不同的partitioned relation 存储在不同的磁盘上，根据访问频率来设置放置区域

▼ 5. 数据字典存储 Data Dictionary Storage

metadata元数据：关于数据的数据，（如：关于关系（表）的数据：关系的schemas）

元数据储存在**Data Dictionary (system catalog)**中

需要储存的信息有：

关系相关

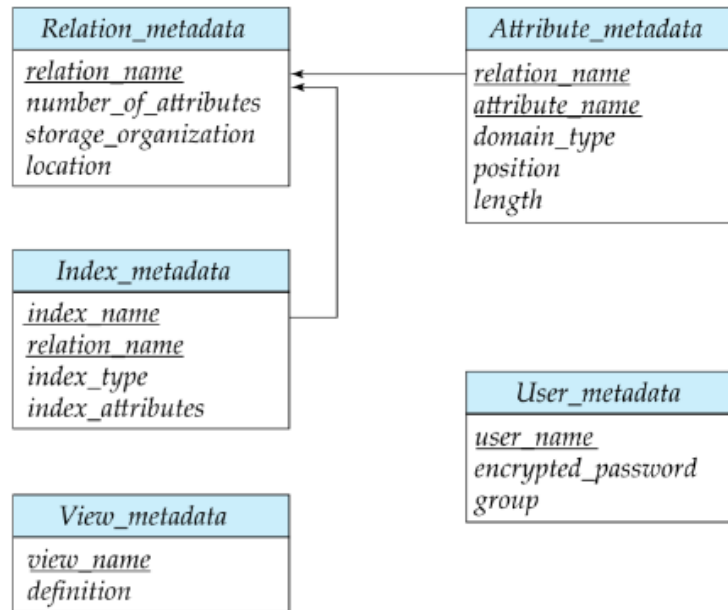
- 关系的名称
- 每个关系的属性的名称
- 属性的域和长度
- 视图的名称和定义
- 完整性约束

用户相关

- 用户名称、密码、权限

物理相关

- 存储组织（堆heap、顺序sequential、hash...）
- 关系的物理位置



元数据是用来维护关系数据的数据。

▼ 6. 数据缓冲区

▼ Buffer概念

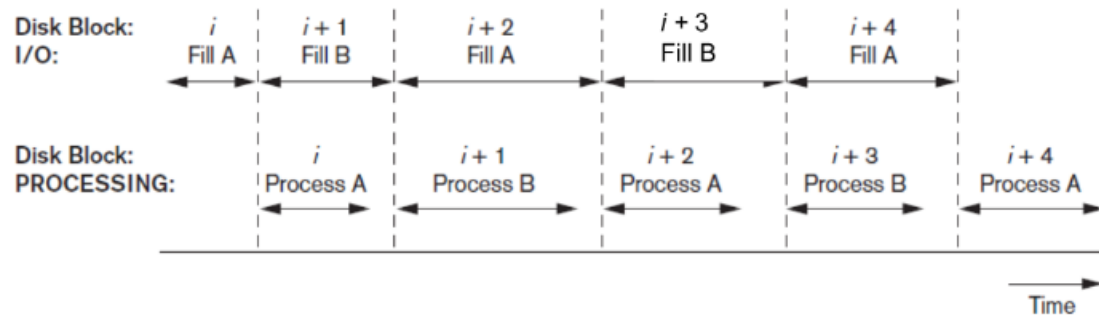
数据库的数据主要存在磁盘上

- 磁盘访问数据比内存访问数据慢
- 尽量减少磁盘和内存之间传输的block数量
- 减少磁盘访问次数
- 在主存main memory中保留尽可能多的block
- 最大化所需要的块已经在主存中的概率 ← 主存中保留所有块是不可能的

Buffer缓冲区：主存中用来存储磁盘block拷贝的区域。**Buffer Manager** 是分配buffer空间的系统

▼ Double buffering

使用两个缓冲区，A和B，从磁盘读取



▼ Buffer Manager

需要调用磁盘的块时（伪代码）：

```
if (block in buffer){
    return block address to caller
}
else{
    挑选一个buffer中被更改写过的block写回磁盘中
    之后将这个block的位置替换成目标Block
    return block address to caller
}
```

buffer 替换策略 (evicted)

1. Least Recently Used (LRU) 最近最少使用
2. First in First out (FIFO)
3. Most recently used (MRU)最近最经常使用

▼ Pinned Block被钉住的块

buffer中读取块的时候被另一个进程移出，以及Buffer中移出块的时候正在被写入 都会出现问题

pin:一个被pin的块据对不会被移出

unpin : 在读取结束后使用解除钉住

▼ Shared and exclusive locks on buffer 共享排他锁

在增加或减少元组的时候， 任何的其他进程都不应该做读取操作。

- 一次只能有一个进程获得排他锁。
 - 当一个进程持有排他锁的时候，其他进程不能执行此块上的共享锁
 - 只能一次一次的授予进程排他锁
- 一个块上共享锁不能与排他锁同时持有。
 - 在持有共享锁的时候被申请排他锁，不会立即得到申请批准，而是等待状态
- 多个进程可以被并发地给予共享锁
 - 如果一个进程申请一个块上的共享锁，而且此块不被其他进程持有排他锁，那么共享锁可以被授予
 - 否则共享锁之后再此排他锁被释放后才能授予

▼ Buffer-Replacement Policies 替换策略

数据库系统能够比OS更准确的预测未来的访问模式

通过查看执行用户的请求操作所需的每一步，数据库系统可以预先知道哪些块时需要的

▼ Toss-immediate strategy 立即丢弃策略

一旦一个关系文件中的一个块中的最后一个元组被处理完，就应该命令buffer manager数仓这个块占用的空间

做natural join操作并输出所有属性

```
select * from instructor natural join department;
```

伪代码如下：

```

for each tuple i of instructor do
  for each tuple d of department do
    if i[dept name]= d[dept name]
    then begin
      let x be a tuple defined as follows:
      x[ID]:= i[ID]
      x[dept name]:= i[dept name]
      x[name]:= i[name]
      x[salary]:= i[salary]
      x[building]:= d[building]
      x[budget]:= d[budget]
      include tuple x as part of result of instructor ⋈ department
    end
  end
end
end

```

一点处理问instructor中几个元素构成的一个完整的块，这个块就不在需要再存储在主存中了。

▼ Most recently used (MRU) 最近最经常使用策略

考虑上面一个例子中的department。当for循环正在处理一个 $department_i$ 的时候，他被pin。

当for循环处理完一个 $department_i$ 后这个元组在接下来的for循环中都不会出现，因此他被标记成最近最常用(MRU)。在unpined之后他可以被移除。

▼ Column-Oriented Storage列式存储

分别存储关系的每个属性

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

好处:

- 如果只有一些属性被访问，减少I/O
- 提高CPU缓存性能
- 提高压缩性能
- 现代CPU架构上的向量处理，允许CPU操作并行应用于一个数组的多个元素(例如，比较一个属性和一个常量)

缺点

- 从柱状表示重构元组的代价
- 元组删除和更新的代价
- 解压的代价

柱状表示在决策支持方面通常比面向决策的表示更有效
一些数据库支持这两种表示方式

- 称为行/列混合存储