R

A	B	C
a1	b1	5
a1	b2	6
a2	b3	8
a2	b4	12

S

B	E
b1	3
b2	7
b3	10
b3	2
b5	2

(1) R和S的连接, 连接条件 C<E
(2) R和S的等值连接 提示：R.B=S.B
(3) R和S的自然连接

一般连接 $R \underset{C<E}{\bowtie} S$的结果如下：

A	R.B	C	S.B	E
a1	b1	5	b2	7
a1	b1	5	b3	10
a1	b2	6	b2	7
a1	b2	6	b3	10
a2	b3	8	b3	10

等值连接 $R \underset{R.B=S.B}{\bowtie} S$ 的结果如下：

A	R.B	C	S.B	E
a1	b1	5	b1	3
a1	b2	6	b2	7
a2	b3	8	b3	10
a2	b3	8	b3	2

自然连接 $R \bowtie S$的结果如下：

A	B	C	E
a1	b1	5	3
a1	b2	6	7
a2	b3	8	10
a2	b3	8	2

employee(employee_name, street, city)
works(employee_name, company_name, salary)
company(company_name, city)
manager(employee_name, manager_name)

考虑3-20中的雇员数据库，其中加下划线的是主码。给出下面每个查询对应的SQL表达式。

3.16

a. 找出所有为 “First Bank Corporation” 工作的雇员名字
c. 找出所有数据库中所有居住的街道和城市与其经理相同的雇员

3.17 考虑图3-20中的雇员数据库。给出下面每个查询对应的SQL表达式：
a）为 “First Bank Corporation” 的所有雇员增长10%的工资
c）删除 “Small Bank Corporation” 的雇员在works关系中的所有元组

考虑3-20中的雇员数据库，其中加下划线的是主码。给出下面每个查询对应的SQL表达式。

3.16 a. 找出所有为“First Bank Corporation”工作的雇员名字

c）找出所有数据库中所有居住的街道和城市与其经理相同的雇员

employee(employee_name, street, city)
works(employee_name, company_name, salary)
company(company_name, city)
manager(employee_name, manager_name)

```
select employee_name
from works
where company_name=   'First Bank Corporation'

select P.employee name
from employee P, employee R, manages M
where P.employee name = M.employee name and
M.manager name = R.employee name and
P.street = R.street and P.city = R.city
```

- select employee_name
- from employee as A,employee as B
- where A.street = B.street and A.city = B.city
- union
- select employee_name
- from managers as C,managers as D
- where C.manager_name = D.manager_name

```
Select employee_name

From employee natural join

    (Select street,city From employee,managers

   Where employee.employee_name= managers. manager_name )
```

3.17 考虑图3-20中的雇员数据库。给出下面每个查询对应的SQL表达式：
a）为“First Bank Corporation”的所有雇员增长10%的工资
c）删除“Small Bank Corporation”的雇员在works关系中的所有元组

employee(employee_name, street, city)
works(employee_name, company_name, salary)
company(company_name, city)
manager(employee_name, manager_name)

```
update works
set salary = salary * 1.1
where company name = ' First Bank Corporation'

delete from works
where company name = ' Small Bank Corporation'
```

employee(employee_name, street, city)
works(employee_name, company_name, salary)
company(company_name, city)
manager(employee_name, manager_name)

4.12 对于图4-11中的数据库，写出一个查询来找到那些没有经理的雇员。注意一个雇员可能只是没有列出其经理，或者可能有null经理。使用外连接书写查询，然后不用外连接再写查询。

4.14 给定学生每年修到的学分总数

如何定义视图tot_credits(id, year, num_credits)

course (course_id, title, dept_name, credits)
teaches (id, course_id, sec_id, semester, year)
student (id, name, dept_name, tot_cred)

4.12 对于图4-11中的数据库，写出一个查询来找到那些没有经理的雇员。注意一个雇员可能只是没有列出其经理，或者可能有null经理。使用外连接书写查询，然后不用外连接再写查询。

employee(employee_name, street, city)
works(employee_name, company_name, salary)
company(company_name, city)
manager(employee_name, manager_name)

外连接

```
select employee_name
from works narural left outer join manages
where manager_name is null
```

employee(employee_name, street, city)
works(employee_name, company_name, salary)
company(company_name, city)
manager(employee_name, manager_name)

非外连接

```
select employee_name
from employee
where employee_name not in (
            select employee_name
            from managers
             where manager_name is not null)
```

```
select employee_name
from works
where not exits
    (select * from manages
     where manages.employee_name = works.employee_name
                and manager_name is not null)
```

4.14 给定学生每年修到的学分总数

如何定义视图tot_credits(id, year, num_credits)

course (course_id, title, dept_name, credits)
teaches (id, course_id, sec_id, semester, year)
student (id, name, dept_name, tot_cred)

```
create view tot_credit(id, year, num_credits) as
select id, year, sum(credits)
from student, teaches, course
where student.id=teaches.id and teaches.course_id=course.course_id
group by id, year
```

5.8 考虑图5-25中的银行数据库。写一个SQL触发器来执行下列动作：在对账户执行delete操作时，对账户的每一个拥有者，检查他是否有其他账户，如果没有，把他从depositor关系中删除。

branch(branch_name, branch_city, asset)
customer(customer_name, customer_street, customer_city)
loan(loan_number, branch_name,amount)
borrower(customer_name, loan_number)
account(account_number, branch_name, balance)
depositor(customer_name, account_number)

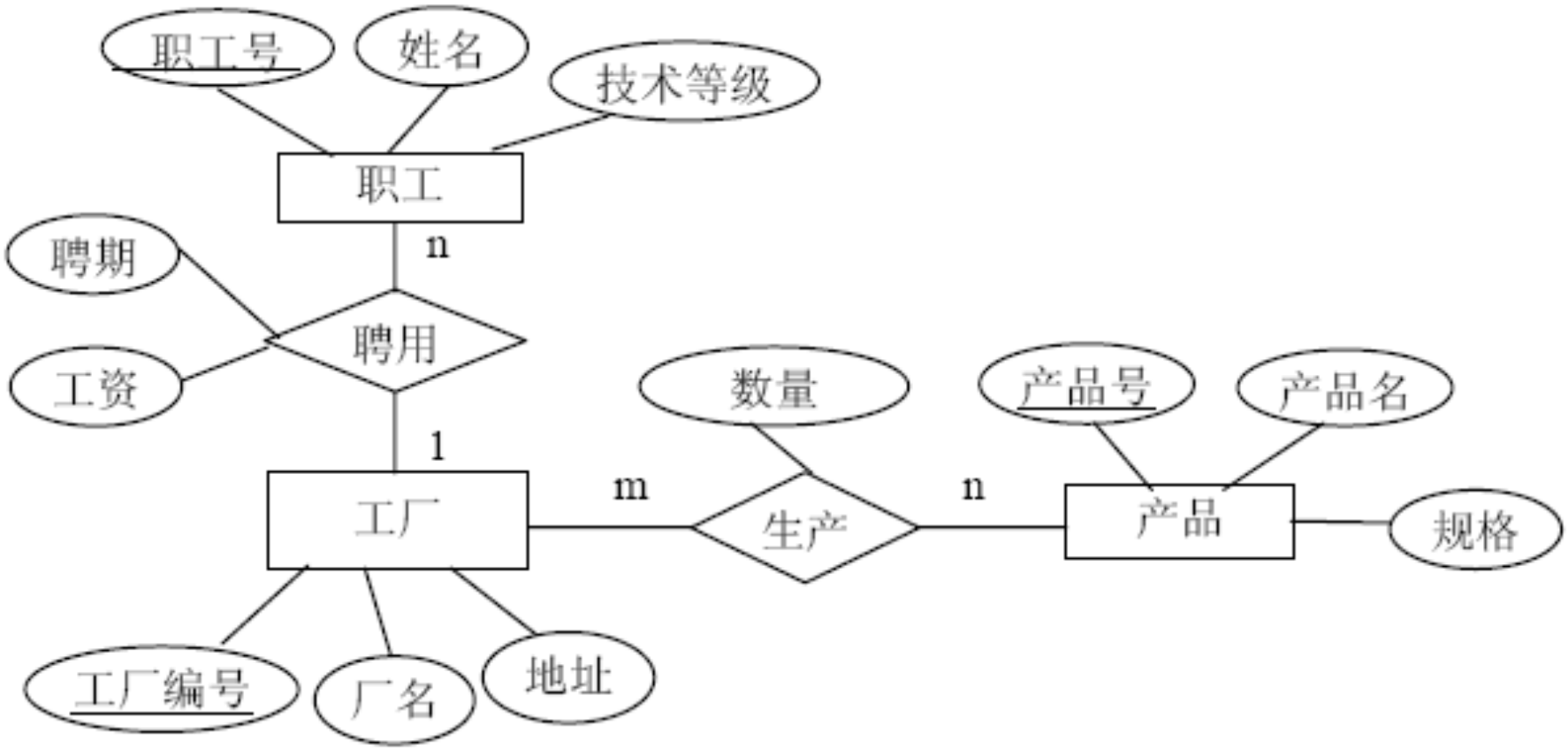
5.8 考虑图5-25中的银行数据库。写一个SQL触发器来执行下列动作：在对账户执行delete操作时，对账户的每一个拥有者，检查他是否有其他账户，如果没有，把他从depositor关系中删除。

branch(branch_name, branch_city, asset)
customer(customer_name, customer_street, customer_city)
loan(loan_number, branch_name,amount)
borrower(customer_name, loan_number)
account(account_number, branch_name, balance)
depositor(customer_name, account_number)

```
create trigger check-delete-trigger after delete on account
referencing old row as orow
for each row
begin
    delete from depositor
    where depositor.customer_name not in
            ( select customer_name from depositor
              where account_number <> orow.account_number )
end
```

练习

某企业集团有若干工厂，每个工厂生产多种产品，且每一种产品可以在多个工厂生产，每个工厂按照固定的计划数量生产产品，计划数量不低于300；每个工厂聘用多名职工，且每名职工只能在一个工厂工作，工厂聘用职工有聘期和工资。工厂的属性有工厂编号、厂名、地址，产品的属性有产品编号、产品名、规格，职工的属性有职工号、姓名、技术等级。请为该集团进行概念设计，画出E-R图。

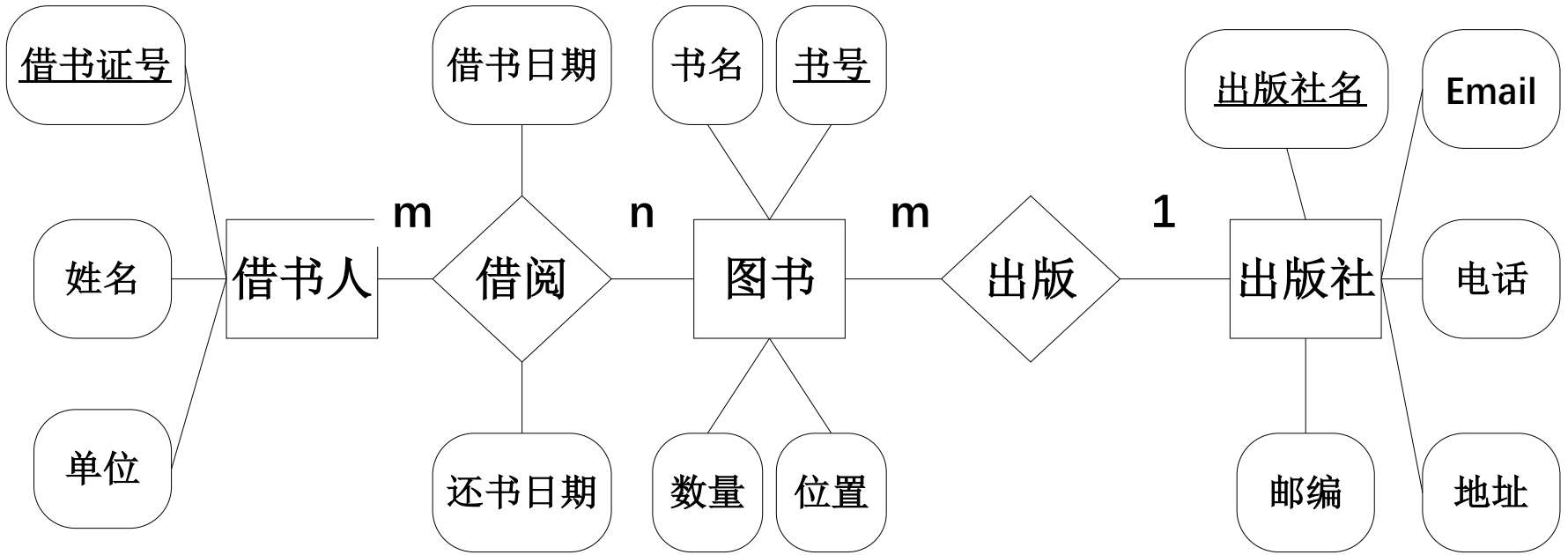

职工号
姓名
技术等级
职工
n
聘期
聘用
工资
1
工厂
m
数量
生产
n
产品号
产品名
产品
规格
工厂编号
厂名
地址

练习

图书借阅管理系统具有以下功能:

1. 可随时查询书库中现有书籍的品种、数量与存放位置。
   - 所有各类书籍均可由书号唯一标识。
2. 可随时查询书籍借还情况，包括借书人单位、姓名、借书证号、借书日期和还书日期。
   - 任何人可借多种书，任何一种书可为多个人所借；
   - 借书证号具有唯一性。
3. 可通过数据库中保存的出版社的Email、电话、邮编及地址等信息向相应出版社增购有关书籍。
   - 一个出版社可出版多种书籍，同一本书仅为一个出版社出版；
   - 出版社名具有唯一性。

请为该系统作概念模型设计,画出ER图.


借书证号
借书日期
书名
书号
出版社名
Email
m
n
m
1
姓名
借书人
借阅
图书
出版
出版社
电话
单位
还书日期
数量
位置
邮编
地址

作业

10.4 考虑从图10-6的文件中删除记录5。比较下列实现删除的技术的相对优点：

a. 移动记录6到记录5所占用的空间，然后移动记录7到记录6所占用的空间。

b. 移动记录7到记录5所占用的空间。

c. 标记记录5被删除，不移动任何记录。

记录0	10101	Srinivasan	Comp. Sci.	65000
记录1	12121	Wu	Finance	90000
记录2	15151	Mozart	Music	40000
记录11	98345	Kim	Elec. Eng.	80000
记录4	32343	El Said	History	60000
记录5	33456	Gold	Physics	87000
记录6	45565	Katz	Comp. Sci.	75000
记录7	58583	Califieri	History	62000
记录8	76543	Singh	Finance	80000
记录9	76766	Crick	Biology	72000
记录10	83821	Brandt	Comp. Sci.	92000

作业

10.15

解释为什么在磁盘块上分配记录的策略会显著影响数据库系统的性能？

10.18

在顺序文件组织中，为什么即使当前只有一条溢出记录，也要使用一个溢出块。

作业

11.3 用下面的关键码值集合建立一个B+树

（2 , 3 , 5 , 7 , 11 , 17 , 19 , 23 , 29 , 31）

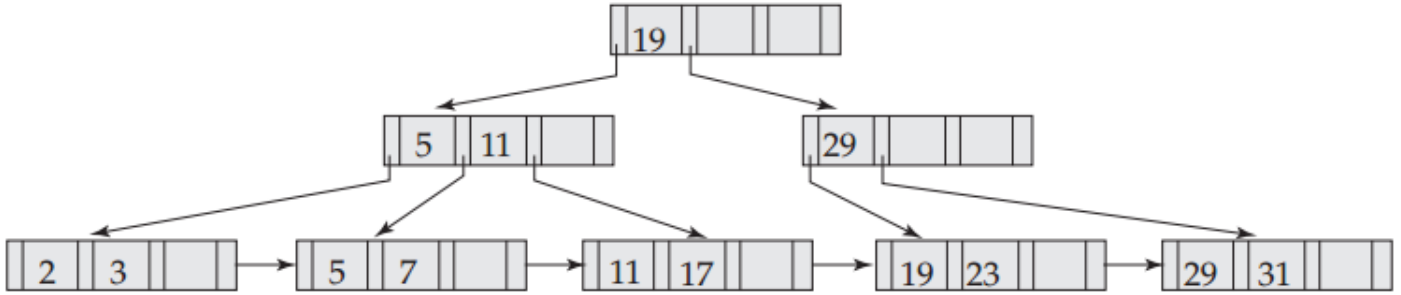
假设树初始为空，值按上升顺序加入。根据一个结点所能容纳指针数的下列情况分别构造B+树。

a.4

b.6

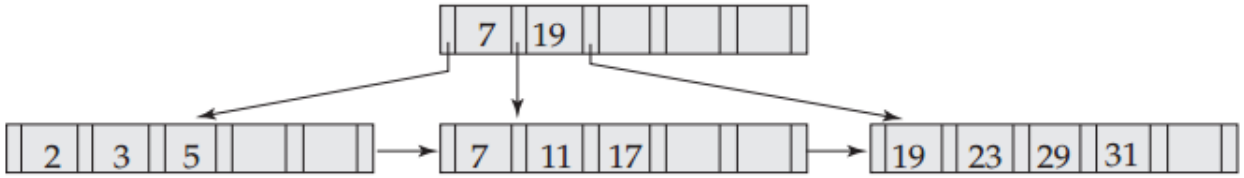
c.8

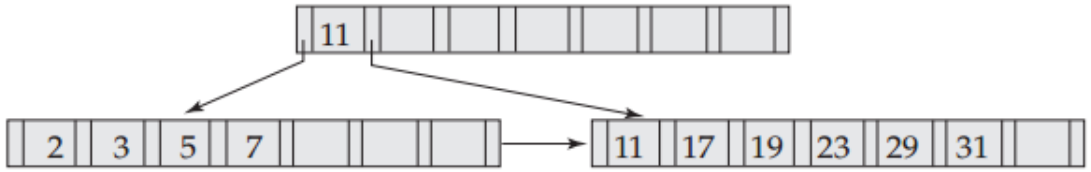
11.4 插入9　插入10　插入8　删除23　删除19



11.4 插入9　　插入10　　插入8　删除23　删除19

（2，3，5，7，11，17，19，23，29，31）





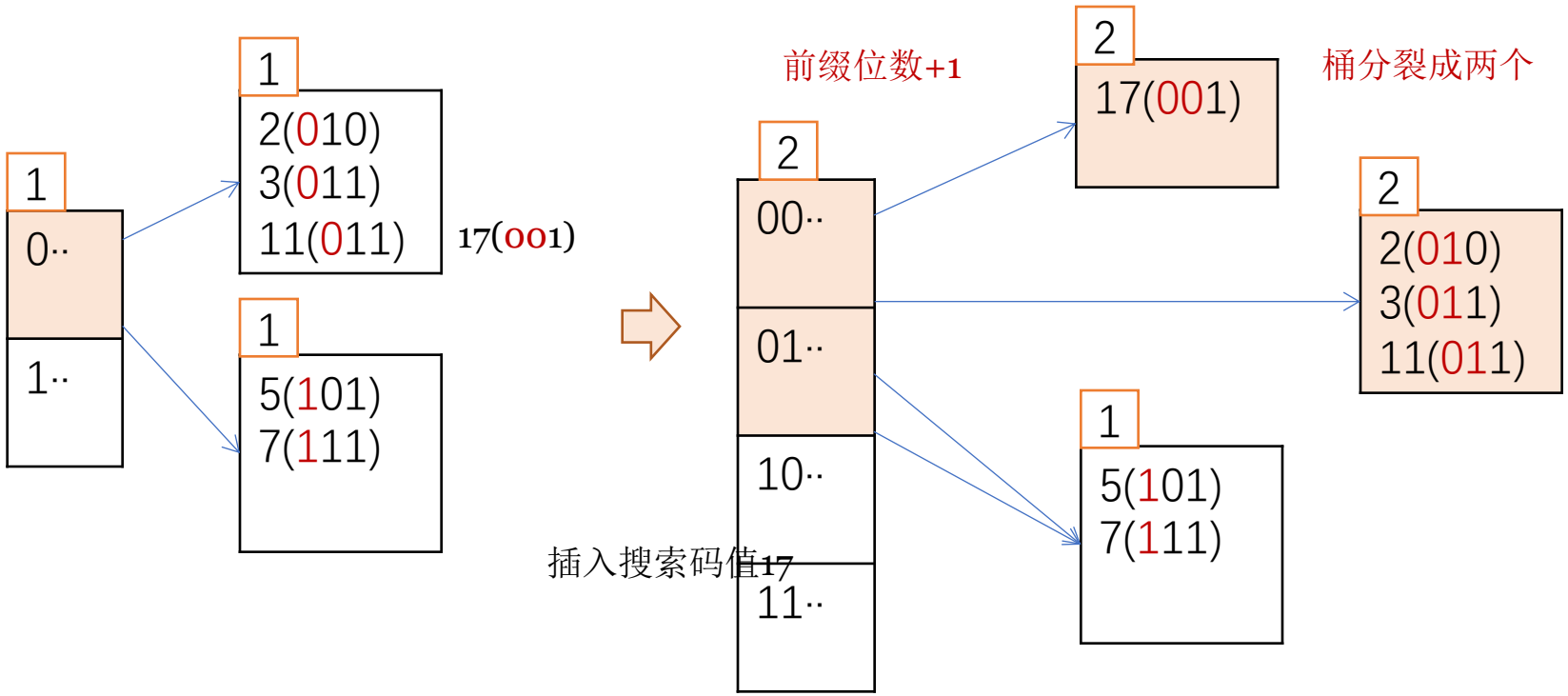
11.6 假设我们在一个文件上使用可扩充散列，该文件所含记录的搜索码值如下：

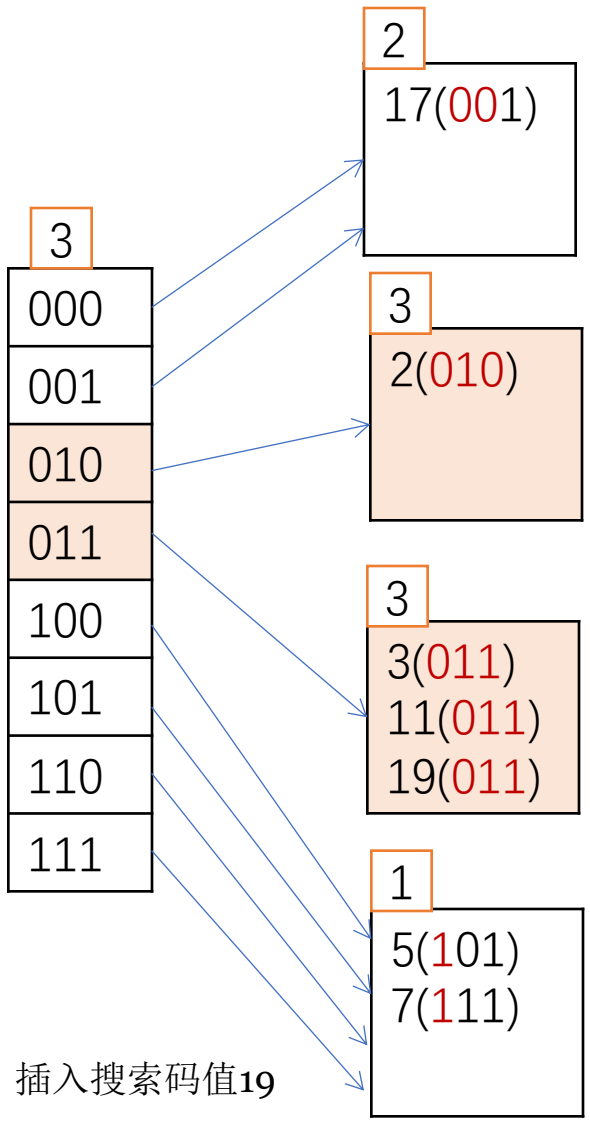
2、3、5、7、11、17、19、23、29、31

如果散列函数为h(x)=x mod 8，且每个桶可以容纳3条记录。给出此文件的可扩充散列结构。

文件所含记录的搜索码值：2、3、5、7、11、17、19、23、29、31
散列函数为h(x)=x mod 8，且每个桶可以容纳3条记录

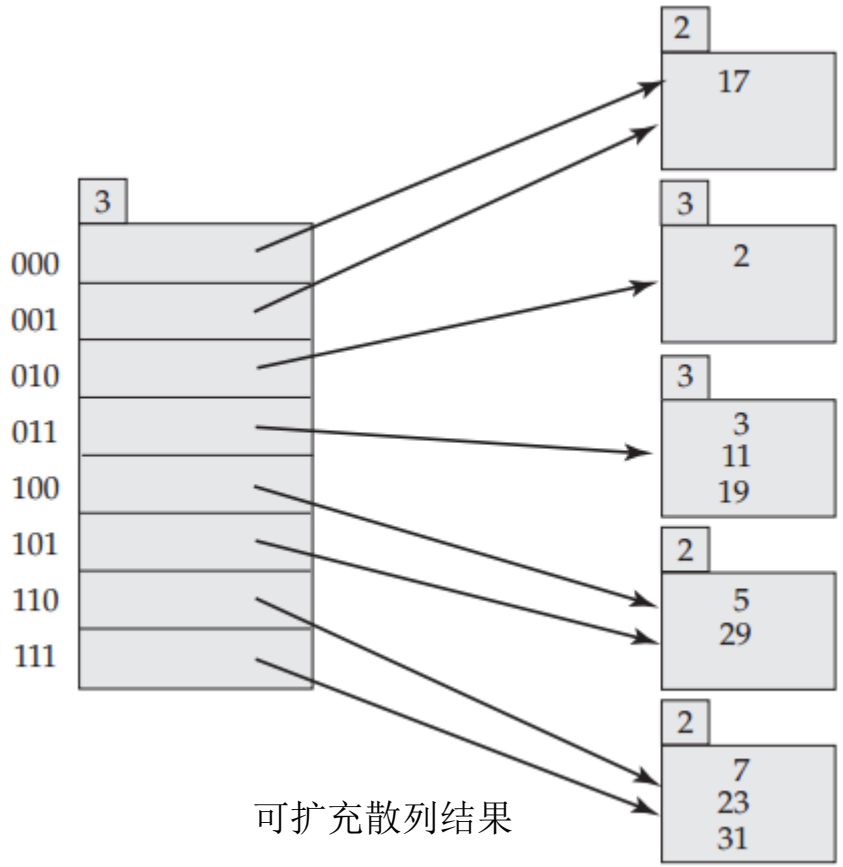
搜索码值	2	3	5	7	11	17	19	23	29	31
h(x)	2	3	5	7	3	1	3	7	5	7
h(x)二进制	010	011	101	111	011	001	011	111	101	111





插入搜索码值19

搜索码值	17	19	23	29	31
h(x)	1	3	7	5	7
h(x)二进制	001	011	111	101	111



可扩充散列结果

作业

12.2 考虑图12-13中银行数据库的例子，其中主码以下划线标出。

考虑下面的SQL语句：

select T.branch_name

from branch T, branch S

where T.assets > S.assets and S.branch_city = “Brooklyn”

写出一个与此等价的、高效的关系代数表达式，并论证你的选择。

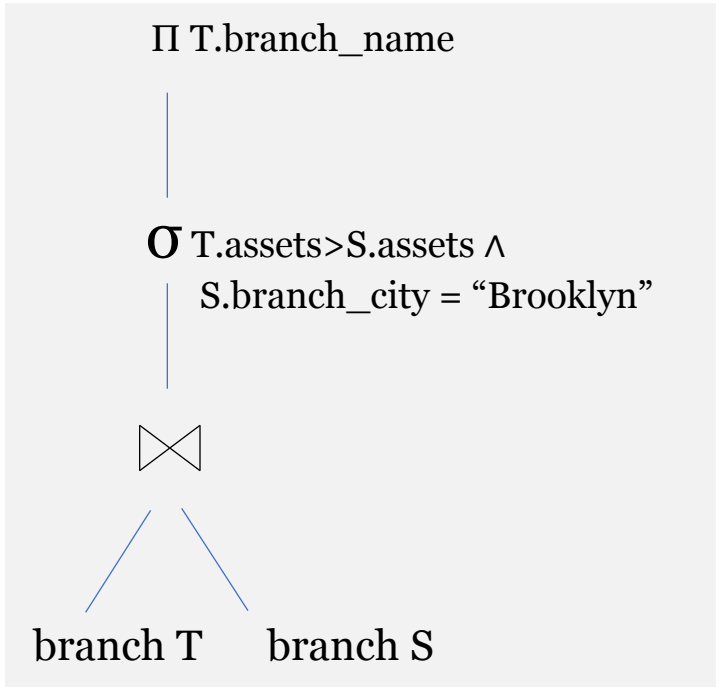
branch(T.branch_name, branch_city, assets)

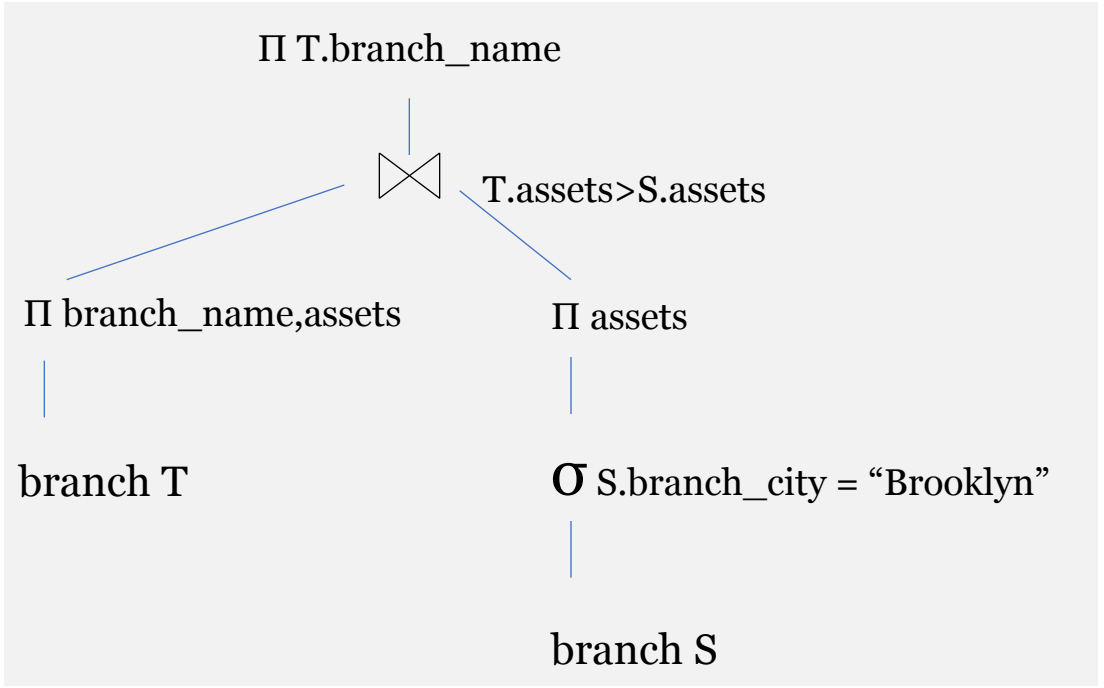
查询优化一般规则：

- 尽量提前做选择操作；在每个操作后，应做个投影操作，去掉不用的属性值

```
select T.branch_name
from branch T, branch S
where T.assets > S.assets and S.branch_city = “Brooklyn”
```

$$\Pi_{T.branch\_name}((\Pi_{branch\_name,\ assets}(\rho_T(branch))) \bowtie_{T.assets\ >\ S.assets} (\Pi_{assets}\ (\sigma_{(branch\_city\ =\ 'Brooklyn')}(\rho_S(branch)))))$$





讨论

13.4

考虑关系$r_1(A,B,C)$，$r_2(C,D,E)$和$r_3(E,F)$，它们的主码分别为A、C、E。假设r_1有1000个元组，r_2有1500个元组，r_3有750个元组。估计$r_1 \bowtie r_2 \bowtie r_3$的大小，给出一个有效的计算这个连接的策略。

（1）由于连接运算满足交换律，r1、r2、r3不论以何种顺序连接，结果都是一样的。

因此考虑((r1 ⋈r2) ⋈r3：

r1与r2连接得到一个1000元组的关系r12（C是r2的主码）

r12与r3连接得到一个1000元组的关系r123（E是r3的主码）

因此最终的连接结果有1000个元组。

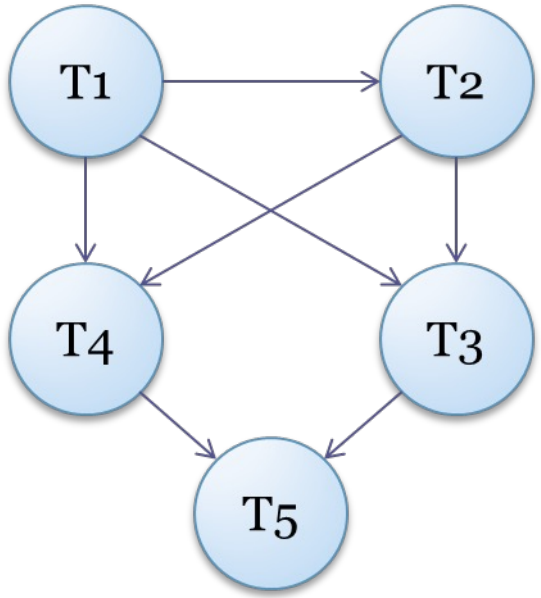
（2）在r2上对C属性建立索引，在r3上对E属性建立索引。

然后对r1的每一个元组：

以r2.C索引，在r2钟找到一个（最多找到一个）与r1中C匹配的元组；

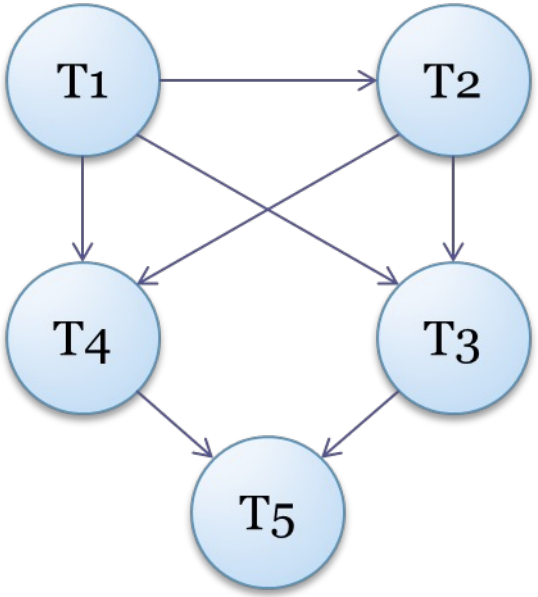
以r3.E索引，在r3中找到一个（最多找到一个）与r2中E匹配的元组。

14.6 考虑图14-16所示的优先图，相应的调度是冲突可串行化的吗？解释你的回答。



若一个调度S与串行调度冲突等价，称调度S是冲突可串行化的。

14.6 考虑图14-16所示的优先图，相应的调度是冲突可串行化的吗？解释你的回答。



依次去掉“仅有出弧”的事务节点

若一个调度S与串行调度冲突等价，称调度S是冲突可串行化的。

题中的调度是冲突可串行化的，因为该调度优先图是无环的。
如：T1,T2,T3,T4,T5
或者：T1,T2,T4,T3,T5

判断标准：

冲突(conflict)：即在一个 schedule 里面，对同一样事物，一个在写，同时另一个在读或者写，会造成冲突。

冲突可串行化(conflict serializable)：一个 schedule能将冲突的操作调节成类似串行化的操作。

优先图判断是否是冲突可串行化：

先画出每个 transaction(T1 T2 T3...)

找到所有的 读写操作，按照先后顺序，将 read—>write, write—>read, write—>write 连起来，都是前指向后，这些读写都是不同 transaction之间，相同的 transaction 内部不管，如果是相同路径的线段重复多条，只画一条。

如果画完了有环，则是非冲突可串行化，就是有冲突，但是不能串行化的。

15.2 考虑下面两个事务：

```
T34:    read(A);
        read(B);
        if A=0 then B:=B+1;
        write(B);

T35:    read(B);
        read(A);
        if B=0 then A:=A+1;
        write(A);
```

给事务T34与T35增加加锁、解锁指令，使它们遵从两阶段封锁协议。这两个事务会引起死锁吗？

15.3 强两阶段封锁协议带来什么好处？它与其他形式的两阶段封锁协议相比有何异同？

两阶段锁协议：

整个事务分为两个阶段，前一个阶段为加锁，后一个阶段为解锁。

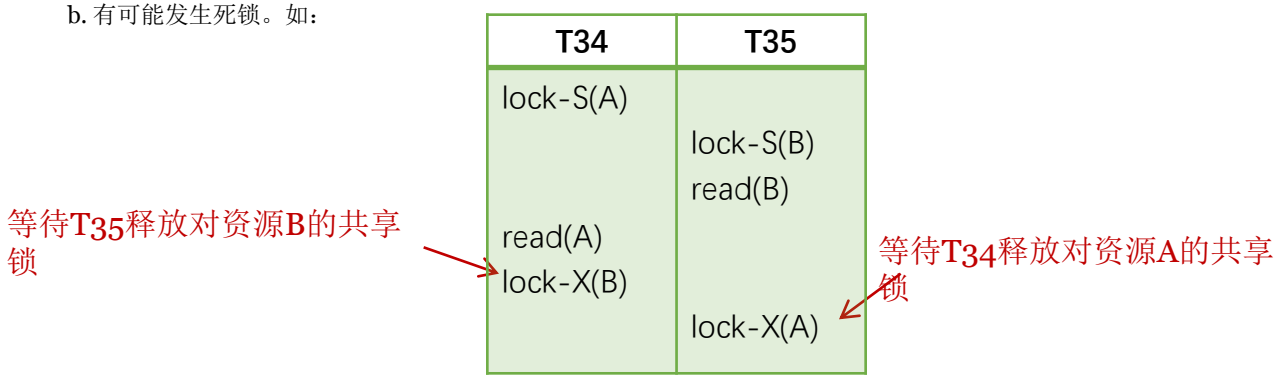
在加锁阶段，事务只能加锁，也可以操作数据，但不能解锁，直到事务释放第一个锁，就进入解锁阶段，此过程中事务只能解锁，也可以操作数据，不能再加锁。两阶段锁协议使得事务具有较高的并发度，因为解锁不必发生在事务结尾。它的不足是没有解决死锁的问题，因为它在加锁阶段没有顺序要求。如两个事务分别申请了A, B锁，接着又申请对方的锁，此时进入死锁状态。

a. Lock and unlock instructions:

T 34:
```
lock-S(A)
read(A)
lock-X(B)
read(B)
if A = 0
then B := B + 1
write(B)
unlock(A)
unlock(B)
```

T 35:
```
lock-S(B)
read(B)
lock-X(A)
read(A)
if B = 0
then A := A + 1
write(A)
unlock(B)
unlock(A)
```

b. 有可能发生死锁。如：



15.3 强两阶段封锁协议带来什么好处？
它与其他形式的两阶段封锁协议相比
有何异同？

两阶段封锁协议的两种变体

- 严格两阶段封锁协议：除了要求封锁是两阶段之外，还要求事务持有的所有排他锁必须在事务提交之后方可释放。这个要求保证未提交事务所写的任何数据在该事务提交之前均以排他方式加锁，防止其他事务读取这些数据；

- 强两阶段封锁协议：它要求事务提交之前不得释放任何锁。它旨在让冲突的事务尽可能地串行执行，这样的话，调度中的事务可以按其提交的顺序串行化。

进行时间戳排序协议的分析

T_1	T_2	T_3	T_4	T_5
				read (X)
read (Y)				
	read (Y)			
		write (Y)		
	read (Z)			
				read(W)
		write (Z)		
read (X)				
			read (W)	
		write (W)		
				write (Y)
				write (Z)

- 15.28

当一个事务在时间戳排序协议下回滚，它被赋予新时间戳。为什么它不能简单地保持原有时间戳？

课堂小测试

<T0,start>
<T0,A,1000,800>
<T1,start>
<T0,B,2000,1600>
<T1,C,700,600>
<T0,commit>

如何进行恢复？