

第六章

实践习题

6.2

请考虑图 6-11 的职员数据库。请给出关系代数表达式来表示下面的每个查询：

- 请找出居住在城市“Miami”的每位职员的姓名
- 请找出薪水超过\$100 000 的每位职员的姓名
- 请找出居住在“Miami”并且薪水超过 100000 美元的每位职员的姓名

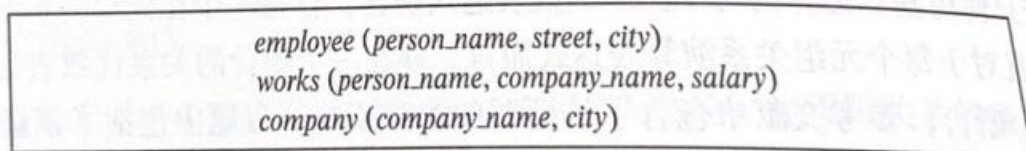


图 6-11 职员数据库

- $\Pi_{person_name} (\sigma_{city = \text{"Miami"}} (employee))$
- $\Pi_{person_name} (\sigma_{salary > 100000} (employee \bowtie works))$
- $\Pi_{person_name} (\sigma_{city = \text{"Miami"} \wedge salary > 100000} (employee \bowtie works))$

6.3

请考虑图 6-12 的银行数据库。请给出关系代数表达式来表示下面的每个查询：

- 请找出位于“Chicago”的每家支行的名称
- 请找出在“Downtown”支行有贷款的每位贷款人的 ID

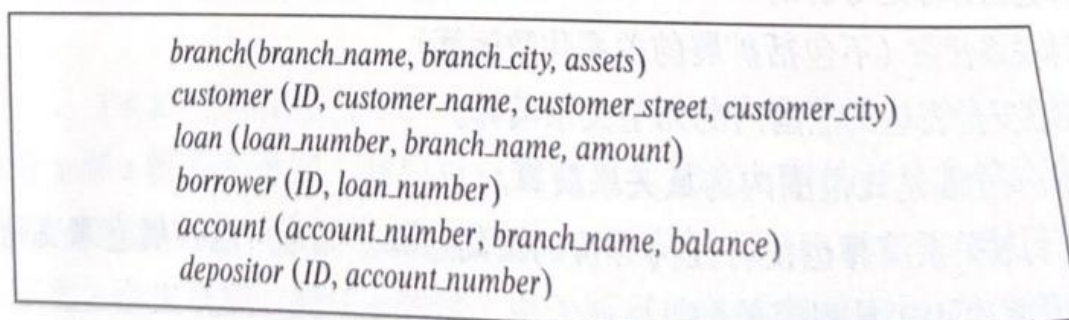


图 6-12 银行数据库

- $\Pi_{branch_name} (\sigma_{branch_city = \text{"Chicago"}} (branch))$
- $\Pi_{ID} (\sigma_{branch_name = \text{"Downtown"}} (borrower \bowtie_{borrower.loan_number = loan.loan_number} loan))$

6.9

请考虑图 6-13 中的关系数据库，其中主码用下划线标示。请为下面的每个查询给出元组关系演算表达式：

- 请找出直接为“Jones”工作的所有员工
- 请找出直接为“Jones”工作的所有员工所居住的所有城市

- c. 请找出“Jones”的经理的经理的姓名
d. 请找出比居住在城市“Mumbai”的所有员工的收入都高的那些员工

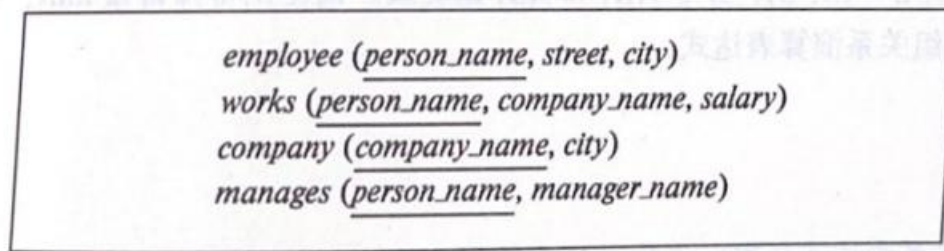


图 6-13 员工数据库

- a.
 $\{t \mid \exists m \in \text{manages} (t[\text{person_name}] = m[\text{person_name}] \wedge m[\text{manager_name}] = \text{'Jones'})\}$
- b.
 $\{t \mid \exists m \in \text{manages} \exists e \in \text{employee} (e[\text{person_name}] = m[\text{person_name}] \wedge m[\text{manager_name}] = \text{'Jones'} \wedge t[\text{city}] = e[\text{city}])\}$
- c.
 $\{t \mid \exists m1 \in \text{manages} \exists m2 \in \text{manages} (m1[\text{manager_name}] = m2[\text{person_name}] \wedge m1[\text{person_name}] = \text{'Jones'} \wedge t[\text{manager_name}] = m2[\text{manager_name}])\}$
- d.
 $\{t \mid \exists w1 \in \text{works} (t[\text{person_name}] = w1[\text{person_name}] \wedge \neg \exists w2 \in \text{works} (w1[\text{salary}] \leq w2[\text{salary}] \wedge \exists e2 \in \text{employee} (w2[\text{person_name}] = e2[\text{person_name}] \wedge e2[\text{city}] = \text{'Mumbai'})))\}$

习题

6.10

请考虑图 6-11 的职员数据库。请给出关系代数表达式来表示下面的每个查询：

- a. 请找出为“BigBank”工作的每位职员的 ID 和姓名。
b. 请找出为“BigBank”工作的每位职员的 ID、姓名和所居住的城市。
c. 请找出为“BigBank”工作且薪水超过 10 000 美元的每位职员的 ID、姓名、街道地址和所居住的城市。
d. 请找出在这个数据库中居住地与工作的公司所在地为同一城市的每位职员的 ID 和姓名。

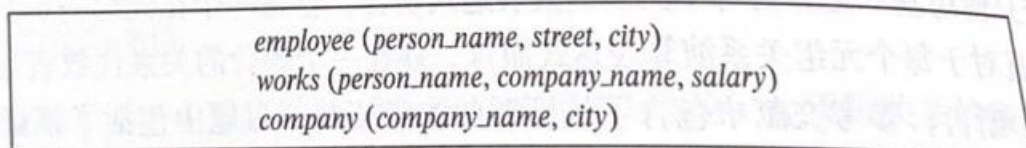


图 6-11 职员数据库

- a.

$\Pi_{\text{person_name}} (\sigma_{\text{company_name}=\text{"BigBank"}}(\text{works}))$

b.

$\Pi_{\text{person_name}, \text{city}} (\sigma_{\text{company_name}=\text{"BigBank"}}(\text{works} \bowtie \text{employee}))$

c.

$\Pi_{\text{person_name}, \text{street}, \text{city}} (\sigma_{\text{company_name}=\text{"BigBank"} \wedge \text{salary} > 10000}(\text{works} \bowtie \text{employee}))$

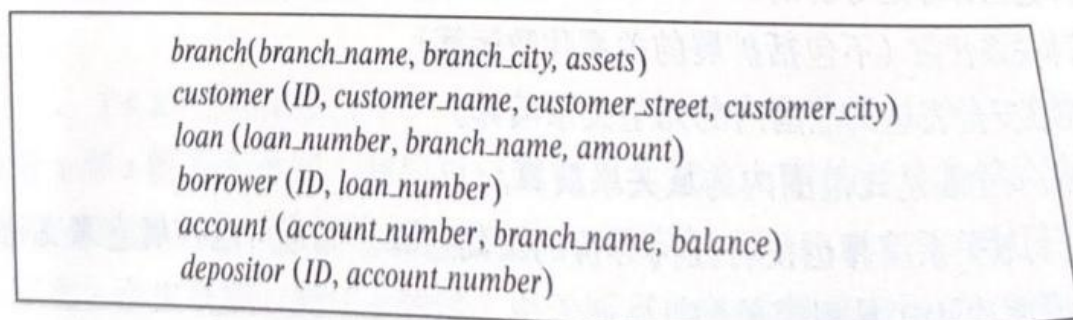
d.

$\Pi_{\text{person_name}} (\text{company} \bowtie \text{works} \bowtie \text{employee})$

6.11

请考虑图 6-12 的银行数据库。请给出关系代数表达式来表示下面的每个查询：

- 请找出贷款额度超过 10 000 美元的每个贷款号。
- 请找出每个这样的存款人的 ID，他拥有一个存款余额大于 6000 美元的账户。
- 请找出每个这样的存款人的 ID，他在“Uptown”支行拥有一个存款余额大于 6000 美元的账户。



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

图 6-12 银行数据库

a. $\Pi_{\text{loan_number}} (\sigma_{\text{amount} > 10000}(\text{loan}))$

b. $\Pi_{\text{id}} (\sigma_{\text{balance} > 6000}(\text{depositor} \bowtie \text{account}))$

c. $\Pi_{\text{id}} (\sigma_{\text{branch_name}=\text{"Uptown"} \wedge \text{balance} > 6000}(\text{depositor} \bowtie \text{account}))$

6.12

对于大学模式，请用关系代数编写如下查询：

- 请找出物理系中每位教师的 ID 和姓名。
- 请找出位于“Watson”教学楼的系里的每位教师的 ID 和姓名。
- 请找出至少选修过“Comp. Sci.”系的一门课程的每名学生的 ID 和姓名。
- 请找出在 2018 年至少上过一个课程段的每名学生的 ID 和姓名。
- 请找出在 2018 年未上过任何一个课程段的每名学生的 ID 和姓名。

```

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

```

图 2-8 大学数据库的模式

- a. $\Pi_{id,name} (\sigma_{dept_name='physical'}(instructor))$
- b. $\Pi_{id,name} (\sigma_{building='Waston'}(instructor \bowtie department))$
- c. $\Pi_{id,name} (student \bowtie takes \bowtie \Pi_{course_id} (\sigma_{dept_name='Comp.Sci. '}(course)))$
- d. $\Pi_{id,name} (\sigma_{year=2018}(student \bowtie takes))$
- e. $\Pi_{id,name} (student) - \Pi_{id,name} (\sigma_{year=2018}(student \bowtie takes))$

6.13 请考虑图 6-13 的员工数据库。请为下面的每个查询给出元组关系演算表达式:

- a. 请找出为“FBC”工作的所有员工的姓名。
- b. 请找出为“FBC”工作的所有员工的姓名和居住城市。
- c. 请找出为“FBC”工作且薪水超过 10 000 美元的所有员工的姓名、街道地址和居住城市。
- d. 请找出居住在与其工作的公司相同城市的所有员工。
- e. 请找出与其经理居住在相同城市和相同街道的所有员工。
- f. 请找出在数据库中不为“FBC”工作的所有员工。
- g. 请找出挣得比“SBC”的每位员工多的所有员工。
- h. 假设公司可以位于好几个城市中。请找出所有这样的公司，它位于“SBC”所位于的每一座城市。

- a. $\{t \mid \exists w \in works(w[person_name] = t[person_name] \wedge w[company_name] = 'FBC')\}$
- b. $\{t \mid \exists e \in employee \exists w \in works(e[person_name] = w[person_name] \wedge e[person_name] = t[person_name] \wedge e[city] = t[city] \wedge w[company_name] = 'FBC')\}$
- c. $\{t \mid t \in employee \wedge (\exists w \in works(w[person_name] = t[person_name]$

$\wedge w[\text{company_name}] = \text{'FBC'} \wedge w[\text{salary}] > 10000)) \}$

d. $\{t \mid \exists e \in \text{employee} \exists c \in \text{company} \exists w \in \text{works}(t[\text{person_name}] =$
 $e[\text{person_name}] \wedge e[\text{person_name}] = w[\text{person_name}] \wedge$
 $w[\text{company_name}] = c[\text{company_name}] \wedge c[\text{city}] = e[\text{city}])\}$

e. $\{t \mid \exists e \in \text{employee} \exists e2 \in \text{employee} \exists m \in \text{manages}$
 $(t[\text{person_name}] = e[\text{person_name}] \wedge e[\text{person_name}] = m[\text{person_name}]$
 $\wedge m[\text{manager_name}] = e2[\text{person_name}] \wedge e[\text{city}] = e2[\text{city}] \wedge$
 $e[\text{street}] = e2[\text{street}])\}$

f. $\{t \mid \exists e \in \text{employee} (t[\text{person_name}] = e[\text{person_name}] \wedge (\neg \exists w \in$
 $\text{works}(e[\text{person_name}] = w[\text{person_name}]$
 $\wedge w[\text{company_name}] = \text{'FBC'}))\}$

g. $\{t \mid \exists w \in \text{works}(w[\text{person_name}] = t[\text{person_name}] \wedge$
 $\forall w2 \in \text{works}(w2[\text{company_name}] = \text{'SBC'} \Rightarrow w[\text{salary}] > w2[\text{salary}]))\}$

h. $\{t \mid \exists c \in \text{company}(t[\text{company_name}] = c[\text{company_name}]$
 $\wedge \forall c2 \in \text{company}(c2[\text{company_name}] = \text{'SBC'}$
 $\Rightarrow \exists c3 \in \text{company}(c3[\text{company_name}] = t[\text{company_name}] \wedge$
 $c3[\text{city}] = c2[\text{city}]))\}$

第十章

实践习题

10.1

SSD 可以用作内存和磁盘之间的存储层，数据库的某些部分(例如某些关系)存储在 SSD 上，而其余部分可以存储在磁盘上。另一种可选方案是，SSD 可以用作缓冲区或缓存；经常使用的块将驻留在 SSD 层，而不经常使用的块将驻留在磁盘上。

- 如果你需要支持实时查询，这些查询必须在有保证的短时间内得到答案，你会选择两种备选方案的哪一种？请解释为什么。
- 如果你有一个非常大型的客户（customer）关系，该关系中仅有一些磁盘块被经常访问，而其他块很少被访问，你将选择两种备选方案中的哪一种？

在第一种情况下，SSD 作为存储层更好，因为性能得到了保证。使用 SSD 作为缓存，一些请求可能需要从磁盘读取，从而导致延迟。

在第二种情况下，由于我们不清楚哪些块在更高级别上频繁访问，因此不可能将部分关系分配给 SSD。因为关系非常大，所以不可能将所有关系分配给 SSD，SSD 作为缓存选项在这种情况下会更好工作。

10.2

一些数据库以这样的方式使用磁盘：仅使用外侧磁道中的扇区，而不使用内侧磁道中的扇区。这样做可能的优势是什么？

磁盘的数据传输速率在外侧磁道上要比在内侧磁道上大。这是因为磁盘以恒定的速率旋转，所以在给定的时间内，当磁盘臂位于外侧磁道上时，比位于内侧磁道上时通过磁盘臂下方的扇区更多。更重要的是，通过只使用外侧磁道，磁盘臂的移动被最小化，减少了磁盘访问延迟。

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：

- 将记录 6 移到记录 5 所占用的空间，并将记录 7 移到记录 6 所占用的空间。
- 将记录 7 移到记录 5 所占用的空间。
- 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

虽然将记录 6 移动到 5 的空间，将记录 7 移动到 6 的空间是最简单的方法，但这需要移动最多的记录，并且涉及最多的访问。

将记录 7 移动到 5 的空间会移动更少的记录，但会破坏文件中的顺序。

将 5 的空间标记为已删除将保留顺序，不移动任何记录，但需要额外的开销来跟踪文件中的所有可用空间。这种方法可能会导致文件中有太多的“漏洞”，如果不经常压缩这些漏洞，

就会影响性能，因为连续可用记录的可用性会降低。

10.5

给出经过下面每一步后图 10-7 中文件的结构：

- a. 插入 (24556, Turnamian, Finance, 9800)
- b. 删除记录 2
- c. 插入 (34556, Thompson, Music, 67000)

文件头				
记录 0	10101	Srinivasan	Comp. Sci.	65000
记录 1				
记录 2	15151	Mozart	Music	40000
记录 3	22222	Einstein	Physics	95000
记录 4				
记录 5	33456	Gold	Physics	87000
记录 6				
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
记录 11	98345	Kim	Elec. Eng.	80000

插入 (24556, Turnamian, Finance, 9800)

删除记录 2

header				↑ 2
record 0	10101	Sriniva san	Comp. Sci.	65000
record 1	24556	Turna mian	Finan ce	98000
record 2				↑ 4
record 3	22222	Einstein	Physic s	95000
record 4				↑ 6
record 5	33456	Gold	Physic s	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finan ce	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

插入 (34556, Thompson, Music, 67000)

header				↑ 4
record 0	10101	Sriniva san	Comp. Sci.	65000
record 1	24556	Turna mian	Finan ce	98000
record 2	34556	Thomp son	Mus ic	67000
record 3	22222	Einstein	Physic s	95000
record 4				↑ 6
record 5	33456	Gold	Physic s	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finan ce	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

10.6

考虑 section 和 takes 关系。给出这两个关系的一个实例，包括 3 个课程段，每个课程段有 5 名学生选课。给出对这些关系使用多表聚簇的一种文件结构。

关系 section:

course_id	sec_id	semester	year	building	room_number	time_slot_id
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-347	1	Fall	2009	Taylor	3128	C

每个课程段有 5 名学生选课

ID	course_id	sec_id	semester	year	grade
00128	CS-101	1	Fall	2009	A
00128	CS-347	1	Fall	2009	A-
12345	CS-347	1	Fall	2009	A
12345	CS-101	1	Fall	2009	C
17968	BIO-301	1	Summer	2010	null
23856	CS-347	1	Fall	2009	A
45678	CS-101	1	Fall	2009	F
54321	CS-101	1	Fall	2009	A-
54321	CS-347	1	Fall	2009	A
59762	BIO-301	1	Summer	2010	null
76543	CS-101	1	Fall	2009	A
76543	CS-347	1	Fall	2009	A
78546	BIO-301	1	Summer	2010	null
89729	BIO-301	1	Summer	2010	null
98988	BIO-301	1	Summer	2010	null

使用多表聚簇:

BIO-301	1	Summer	2010	Painter	514	A
17968	BIO-301	1	Summer	2010	null	
59762	BIO-301	1	Summer	2010	null	
78546	BIO-301	1	Summer	2010	null	
89729	BIO-301	1	Summer	2010	null	
98988	BIO-301	1	Summer	2010	null	
CS-101	1	Fall	2009	Packard	101	H
00128	CS-101	1	Fall	2009	A	
12345	CS-101	1	Fall	2009	C	
45678	CS-101	1	Fall	2009	F	
54321	CS-101	1	Fall	2009	A-	
76543	CS-101	1	Fall	2009	A	
CS-347	1	Fall	2009	Taylor	3128	C
00128	CS-347	1	Fall	2009	A-	
12345	CS-347	1	Fall	2009	A	
23856	CS-347	1	Fall	2009	A	
54321	CS-347	1	Fall	2009	A	
76543	CS-347	1	Fall	2009	A	

10.10

对于下面的每种情况，给出一个关系代数表达式和一个查询处理策略的示例：

a. MRU 优于 LRU

b. LRU 优于 MRU

MRU 优于 LRU，其中 $R1 \bowtie R2$ 是通过使用嵌套循环处理策略计算的，其中 $R2$ 中的每个元组必须与 $R1$ 中的每个块进行比较。在处理 $R2$ 的第一个元组之后，下一个所需块是 $R1$ 中第一个块。但是，由于它是最近使用最少的，如果系统需要新块，LRU 缓冲区管理策略将替换该块。

LRU 优于 MRU，绝大多数数据库系统都使用 LRU 操作，例如数据字典是数据库最常被访问的部分之一，每个查询的处理都需要访问数据字典，LRU 是将最近最少使用的块移出，缓冲区管理器尽量不把数据字典从主存移出，因此使用 LRU 较好。

在 MRU 下，一些未使用的块可能永远保留在内存中。在实践中，MRU 只能在特殊情况下使用。

习题

10.14

在变长记录表示中，用空位图来表示属性是否为空值。

- a. 对于变长字段，如果值为空，那么偏移量字段和长度字段中应该存储什么？
- b. 在一些应用中，元组拥有非常大量的属性，其中大多数属性为空。你能否更改记录表示，使得一个空值属性的开销仅为空位图中的一个位？
 - a. 将偏移量字段和长度字段中存储 0。
 - b. 将空位图存储在记录的开头，并且对于取空值的属性，根本不存储数据（值或偏移量/长度）。

10.15

请解释为什么在磁盘块上分配记录会显著影响数据库系统的性能。

如果我们将相关记录分配给块，我们通常可以通过一次磁盘访问的查询检索大部分或全部请求的记录。磁盘访问往往是数据库中的瓶颈；由于此分配策略减少了给定操作的磁盘访问次数，因此显著提高了性能。

10.19

标准的缓冲区管理器假定每个块的大小和读取代价都是相同的。设想一个缓冲区管理器使用对象引用率而不是 LRU，对象引用率是指一个对象在此前的 n 秒内被访问的频率。假设我们要在缓冲区中存储变长和读取代价可变（例如网页，它的读取代价取决于它从哪个站点被获取）的对象。试建议缓冲区管理器可以如何选择要从缓冲区中移出哪个块。

一个好的解决方案是使用优先级队列来移出页面，其中优先级(p)是根据在过去 n 秒内的访问频率(f)，重新读取页面的预期成本(c)，以及页面的大小 s 排序的： $p = f * c / s$ 。缓冲区管理器应该选择移出 p 值最低的页面，直到有足够的空闲空间来读入新引用的对象。

第十一章

11.1

索引加快了查询处理，但是在作为潜在搜索码的每个属性上或者每个属性组合上创建索引，通常是个坏主意，请解释为什么

- ① 每个索引在插入和删除过程中都需要额外的 CPU 时间和磁盘 I/O 开销。
 - ② 非主键上的索引可能必须在更新时更改，尽管主键上的索引可能不会更改(这是因为更新时不修改主键属性)。
 - ③ 每个额外的索引都需要额外的存储空间。
 - ④ 对于涉及多个搜索键上的条件的查询，即使只有一些键上有索引，效率可能也不会很差。
- 因此，当已经存在许多索引时，添加索引对数据库性能的改善较小。

11.2

在同一个关系的不同搜索码上建立两个聚集索引一般来说是否可行？请解释你的答案

通常，对于不同的键，不可能在相同的系统上有两个主索引，因为关系中的元组必须以不同的顺序存储，才能将相同的值存储在一起。我们可以通过两次存储关系并复制所有值来实现这一点，但对于集中式系统来说，这是低效的。

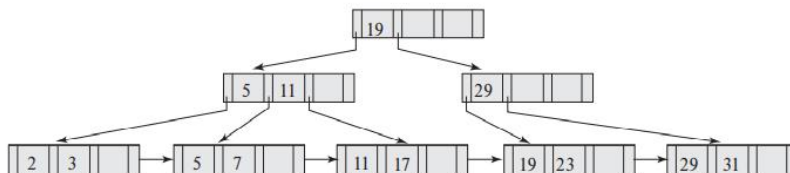
11.3

用下面的码值集合建立一棵 B+树：(2, 3, 5, 7, 11, 17, 19, 23, 29, 31)

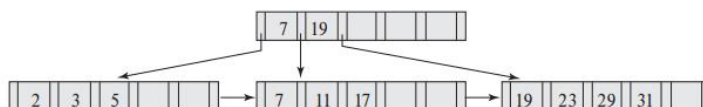
假设树初始为空，按升序添加这些值。当一个节点所能容纳的指针数是下列情况时，请分别构造 B+树：

a.4 b.6 c.8

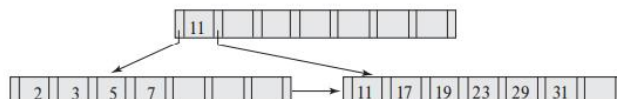
a.



b.



c.



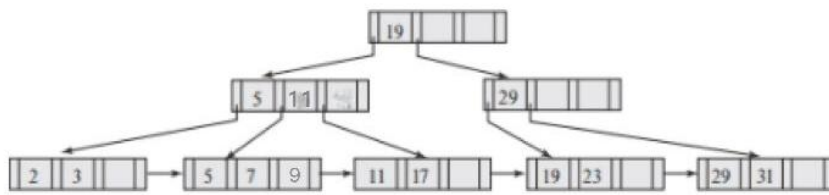
11.4

对于实践习题 11.3 中的每一棵 B+树，请给出下列各操作后树的形状：

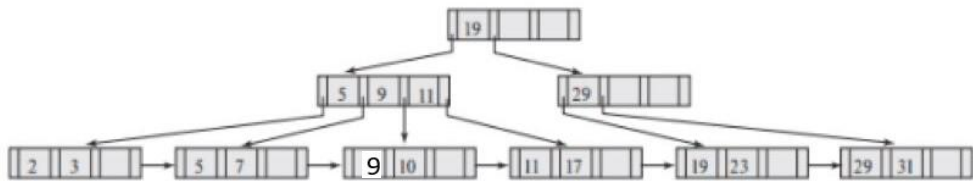
a.插入 9 b.插入 10 c.插入 8 d.删除 23 e.删除 19

对 a:

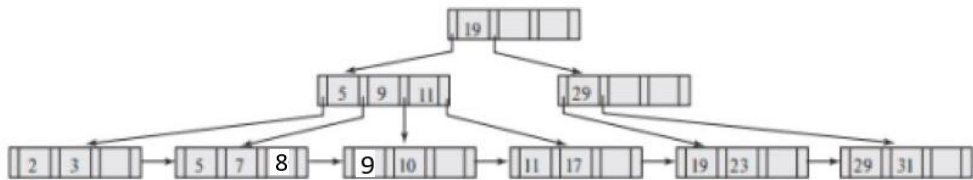
Insert 9:



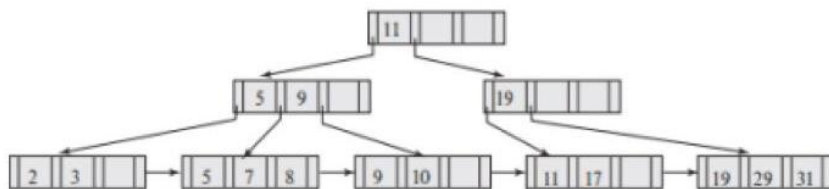
Insert 10:



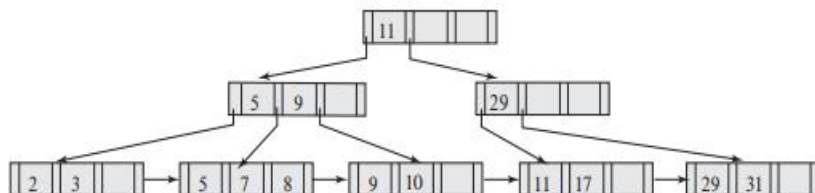
Insert 8:



Delete 23:

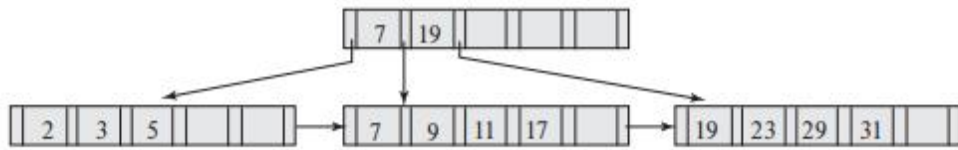


Delete 19:

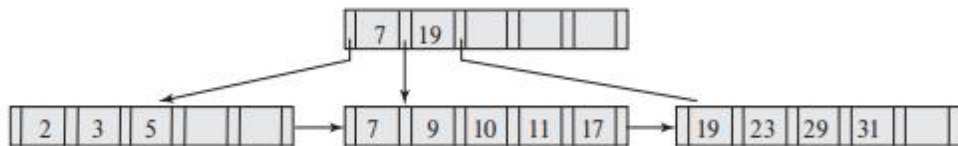


对 b:

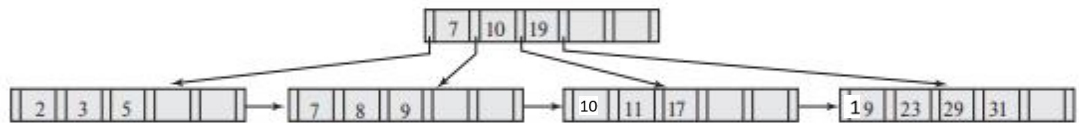
Insert 9:



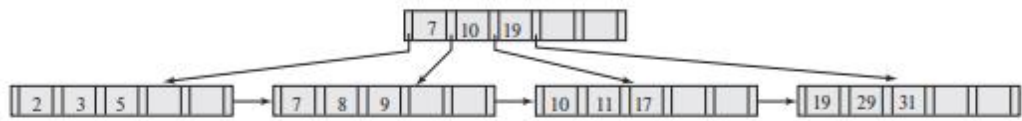
Insert 10:



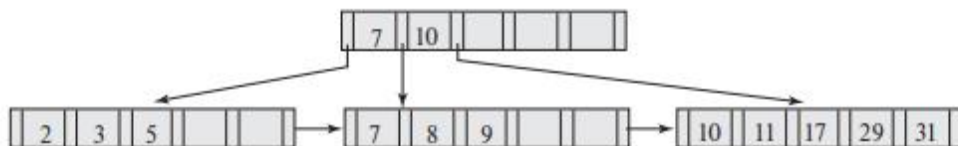
Insert 8:



Delete 23:

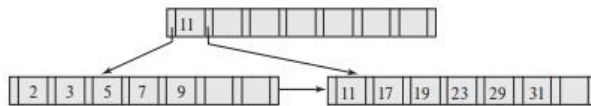


Delete 19:

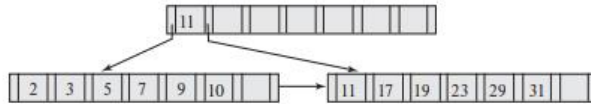


对 c:

Insert 9:



Insert 10:



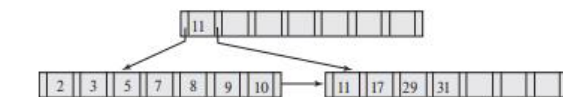
Insert 8:



Delete 23:



Delete 19:



11.9

假设给你一个数据库模式和一些经常执行的查询。你将如何利用这些信息来决定要创建什么样的索引？

在有选择条件的任何属性上创建索引;如果该属性只有几个不同的值，则可以创建位图索引，否则创建普通的 B+ 树索引。

在主键和外键属性上创建 B+ 树索引。

对查询中连接条件所涉及的属性创建索引。

11.13

对于实践习题 11.3 中的每棵 B+ 树，请给出下列查询中涉及的步骤：

a. 找出搜索码值为 11 的记录

b. 找出搜索码值在 7 和 17（包括 7 和 17）的记录

11.3a

A. 查找搜索码值为 11 的记录

搜索第一级索引；找到第一个指针。

搜索下一层；找到第三个指针。

搜索叶节点；找到第一个指向键值为 11 的记录的指针。

B. 查找搜索码值在 7 至 17 之间的记录(包括)

搜索顶部索引；找到第一个指针。

搜索下一个级别：找到第二个指针。

搜索第三级：找到第二个指向键值为 7 的记录的指针，访问它们之后，返回到叶节点。

按照第四个指针指向链中的下一个叶块。

首先找到指向键值为 11 的记录的指针，然后返回。

找到第二个指针到键值为 17 的记录。

11.3b:

A.查找搜索码值为 11 的记录

搜索顶层：找到第二个指针。

搜索下一个级别：找到第二个指向键值为 11 的记录的指针。

B.查找搜索码值在 7 至 17 之间的记录(包括)

搜索顶层：找到第二个指针。

搜索下一个级别：首先找到指向键值为 7 的记录的指针，然后返回。

按照第二个指针指向键值为 11 的记录，然后返回。

按照第三个指针指向键值为 17 的记录。

11.3c:

A.查找搜索码值为 11 的记录

搜索顶层：找到第二个指针。

搜索下一个级别：找到第一个指针找到键值为 11 的记录。

B.查找搜索码值在 7 至 17 之间的记录(包括)

搜索顶层：找到第一个指针。

搜索下一个级别：找到第四个指针找到键值为 7 的记录，然后返回。

按照第八个指针指向下一个叶块。

按照第一个指针指向搜索码值为 11 的记录，返回。

按照第二个指针，指向键值为 17 的记录。

11.15

假设有一个关系 $r(A,B,C)$ 带有一个搜索码 (A,B) 上的 B+树索引。

a.用这个索引来查找满足 $10 < A < 50$ 的记录，最坏情况下的代价是多少?请用获取的记录数目 n_1 和树的高度 h 来度量。

b.用这个索引来查找满足 $10 < A < 50 \wedge 5 < B < 10$ 的记录，最坏情况下的代价是多少?请用满足此选择的记录数目 n_2 以及上面定义的 n_1 和 h 来度量。

c.当 n_1 和 n_2 满足什么条件的时候，此索引是查找满足 $10 < A < 50 \wedge 5 < B < 10$ 的记录的一种高效方式。

a:它查找 A 值在 10 到 50 之间的记录。但是，由于文件中记录的顺序不同，每个记录可能位于不同的块中，这导致了許多 I/O 操作。在最坏的情况下，对于每条记录，它需要遍历整棵树(成本是 h)，所以总成本是 $n_1 * h$ 。

b:对于 10 到 50 之间的每个 A 值，系统定位的记录 B 值在 5 到 10 之间。但是，每个记录很可能位于不同的磁盘块中。这相当于根据 A 上的条件执行查询，花费 $n_1 * h$ 。然后检查这些记录，以查看 B 上的条件是否满足。最坏情况下的总成本是 $n_1 * h$ 。

c: n_1 条记录满足第一个条件， n_2 条记录满足第二个条件。当查询这两个条件时，在第一阶段输出 n_1 条记录。因此，在 $n_1 = n_2$ 的情况下，第一级不输出额外的记录。