TONGJI UNIVERSITY

ADMIS Lab

先进数据与机器智能系统实验室

作业习题评讲课

数据库系统原理课程

2024

第一次作业 2.1

2.1 Consider the employee database of Figure 2.17. What are the appropriate primary keys?

employee (*person_name*, *street*, *city*)
works (*person_name*, *company_name*, *salary*)
company (*company_name*, *city*)

Figure 2.17 Employee database.

$$employee(\underline{person\_name}, \underline{street}, \underline{city})$$

$$works(\underline{person\_name}, \underline{company\_name}, salary)$$

$$company(\underline{company\_name}, city)$$

第一次作业 2.6

employee (*person_name*, *street*, *city*)
works (*person_name*, *company_name*, *salary*)
company (*company_name*, *city*)

Figure 2.17 Employee database.

2.6 Consider the employee database of Figure 2.17. Give an expression in the relational algebra to express each of the following queries:

a. Find the name of each employee who lives in city “Miami”.

b. Find the name of each employee whose salary is greater than \$100000.

c. Find the name of each employee who lives in “Miami” and whose salary is greater than \$100000.

答案：

a. $\Pi_{person\_name}(\sigma_{city="Miami"}(employee))$

b. $\Pi_{person\_name}\left(\sigma_{salary>100000}(works)\right)$

c. $\Pi_{person\_name}(\sigma_{salary>100000 \wedge city="Miami"}(employee \bowtie works))$

第一次作业 2.7

2.7 Consider the bank database of Figure 2.18. Give an expression in the relational algebra for each of the following queries:

a. Find the name of each branch located in "Chicago".

b. Find the ID of each borrower who has a loan in branch "Downtown".

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*)

Figure 2.18 Bank database.

答案:

a. $\Pi_{branch\_name}(\sigma_{branch\_city="Chicago"}(branch))$

b. $\Pi_{ID}(\sigma_{branch\_city="Downtown"}(borrwer \bowtie loan))$

第一次作业 2.8

employee (*person_name*, *street*, *city*)
works (*person_name*, *company_name*, *salary*)
company (*company_name*, *city*)

Figure 2.17 Employee database.

2.8 Consider the employee database of Figure 2.17. Give an expression in the relational algebra to express each of the following queries:

a. Find the ID and name of each employee who does not work for "BigBank".

b. Find the ID and name of each employee who earns at least as much as every employee in the database.

答案：

a. $\Pi_{ID,person\_name}(works) - \Pi_{ID,person\_name}\left(\sigma_{company\_name="BigBank"}(works)\right)$

b. $\Pi_{ID,person\_name}(works) - \Pi_{ID,person\_name}(\sigma_{works.salary<d.salary}(works \times \rho_d(\Pi_{salary}(works))))$

第一次作业 2.15

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*)

Figure 2.18 Bank database.

2.15 Consider the bank database of Figure 2.18. Give an expression in the relational algebra for each of the following queries:

a. Find each loan number with a loan amount greater than \$10000.

b. Find the ID of each depositor who has an account with a balance greater than \$6000.

c. Find the ID of each depositor who has an account with a balance greater than \$6000 at the "Uptown" branch.

答案：

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

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

c. $\Pi_{ID}(\sigma_{balance>6000 \wedge branch\_name="Uptown"}(depositor \bowtie account))$

第二次作业 3.8

3.8 Consider the bank database of Figure 3.18, where the primary keys are underlined. Construct the following SQL queries for this relational database.

branch(<u>*branch_name*</u>, *branch_city, assets*)
customer (<u>*ID*</u>, *customer_name*, *customer_street, customer_city*)
loan (<u>*loan_number*</u>, *branch_name, amount*)
borrower (<u>*ID*</u>, <u>*loan_number*</u>)
account (<u>*account_number*</u>, *branch_name, balance*)
depositor (<u>*ID*</u>, <u>*account_number*</u>)

Figure 3.18 Banking database.

a. Find the ID of each customer of the bank who has an account but not a loan.

(**select** *ID*

from *depositor)*

except

(**select** *ID*

from *borrower)*

第二次作业 3.8

b. Find the lD of each customer who lives on the same street and in the same city as customer '12345'.

select *F.ID*

from *customer as F, customer as S*

where *F.customer _street = S.customer_street*

and *F.customer_city = S.customer_city*

and *S.customer _id = '12345'*

c. Find the name of each branch that has at least one customer who has an account in the bank and who lives in “Harrison".

select *distinct branch_name*

from *account, depositor, customer*

where *customer.id = depositor.id*

and *depositor.account_number = account.account_number*

and *customer_city = 'Harrison'*

第二次作业 3.9

3.9 Consider the relational database of Figure 3.19, where the primary keys are underlined. Give an expression in SQL for each of the following queries.

a. Find the ID, name, and city of residence of each employee who works for "First Bank Corporation".

b. Find the ID, name, and city of residence of each employee who works for "First Bank Corporation" and earns more than $10000.

c. Find the ID of each employee who does not work for "First Bank Corporation".

d. Find the ID of each employee who earns more than every employee of "Small Bank Corporation".

e. Assume that companies may be located in several cities. Find the name of each company that is located in every city in which "Small Bank Corporation" is located.

f. Find the name of the company that has the most employees (or companies, in the case where there is a tie for the most).

g. Find the name of each company whose employees earn a higher salary, on average, than the average salary at "First Bank Corporation".

employee (*ID*, *person_name*, *street*, *city*)
works (*ID*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

Figure 3.19 Employee database.

a. ***select*** *E.ID, E.persone_name, E.city*
from *employee as E, works as W*
where *E.ID = W.ID and W.company_name = "First Bank Corporation"*

b. ***select*** *E.ID, E.persone_name, E.city*
from *employee as E, works as W*
where *E.ID = W.ID*
and W.company_name = "First Bank Corporation"
and W.salary>10000

c. ***select*** *ID*
from *works*
where *company_name < > "First Bank Corporation"*

d. ***select*** *ID*
from *works*
where *salary >all(* ***select*** *salary*
from *works*
where *company_name = "Small Bank Corporation"*

第二次作业 3.9

e. Assume that companies may be located in several cities. Find the name of each company that is located in every city in which "Small Bank Corporation" is located.

f. Find the name of the company that has the most employees (or companies, in the case where there is a tie for the most).

g. Find the name of each company whose employees earn a higher salary, on average, than the average salary at "First Bank Corporation".

e.
```
select  C.company_name
from    company as C
where not exists (select city
                  from company
                  where company_name = "Small Bank Corporation"
                  except  (select  D.city from company as D
                           where D.company_name = C.company_name))
```

f.
```
select  company_name
from    works
group by company_name
having count(distinct ID) >= all(
                select count(distinct ID) from works group by company_name)
```

g.
```
select  company_name
from    works
group by company_name
having avg(salary) > (select avg(salary) from works
                      where company_name = " First Bank Corporation")
```

第二次作业 3.15

3.15 Consider the bank database of Figure 3.18, where the primary keys are underlined. Construct the following SQL queries for this relational database.

a. Find each customer who has an account at *every* branch located in “Brooklyn”.

b. Find the total sum of all loan amounts in the bank.

c. Find the names of all branches that have assets greater than those of at least one branch located in “Brooklyn”.

---

branch(<u>*branch_name*</u>, *branch_city, assets*)
customer (<u>*ID*</u>, *customer_name*, *customer_street, customer_city*)
loan (<u>*loan_number*</u>, *branch_name, amount*)
borrower (<u>*ID*</u>, <u>*loan_number*</u>)
account (<u>*account_number*</u>, *branch_name, balance*)
depositor (<u>*ID*</u>, <u>*account_number*</u>)

Figure 3.18 Banking database.

第二次作业 3.15

a. Find the ID of each customer of the bank who has an account but not a loan.

b. Find the ID of each customer who lives on the same street and in the same city as customer '12345'.

c. Find the name of each branch that has at least one customer who has an account in the bank and who lives in “Harrison”.

a.
```
select distinct C.customer_name
from customer as C
where not exists (
    (select branch_name
     from branch
     where branch_city = 'Brooklyn')
        except
        (select branch_name
         from account join depositor using(account_number)
         where C.ID = depositor.ID))
```

b.
```
select sum(amount)
from loan
```

c.
```
select branch_name
from branch
where assets > some
    (select assets
     from branch
     where branch_city = "Brooklyn")
```

第二次作业 3.16

3.16 Consider the employee database of Figure 3.19, where the primary keys are underlined. Give an expression in SQL for each of the following queries.

a. Find ID and name of each employee who lives in the same city as the location of the company for which the employee works.

b. Find ID and name of each employee who lives in the same city and on the same street as does her or his manager.

c. Find ID and name of each employee who earns more than the average salary of all employees of her or his company.

d. Find the company that has the smallest payroll.

employee (<u>*ID*</u>, *person_name*, *street*, *city*)
works (<u>*ID*</u>, *company_name*, *salary*)
company (<u>*company_name*</u>, *city*)
manages (<u>*ID*</u>, *manager_id*)

Figure 3.19 Employee database.

a.
```
select ID, person_name
from (employee natural join works) natural join
company
```

b.
```
select E.ID, E.person_name
from employee as E, employee as D, manages
where E.ID = manages.ID
      and D.ID = manages.manager_id
      and E.street = D.street
      and E.city = D.city
```

c.
```
select ID, person_name
from works as A
where salary > (select avg(salary)
                from works as W
                where W. company_name = A. company_name)
```

d.
```
select company_name
from works
group by company_name
having sum(salary) <= all(select sum(salary)
                          from works
                          group by company_name)
```

第二次作业 3.17

3.17 Consider the employee database of Figure 3.19. Give an expression in SQL for each of the following queries.

a. Give all employees of "First Bank Corporation" a 10 percent raise.

b. Give all managers of "First Bank Corporation" a 10 percent raise.

c. Delete all tuples in the *works* relation for employees of "Small Bank Corporation".

employee (*ID*, *person_name*, *street*, *city*)
works (*ID*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

Figure 3.19 Employee database.

a. ***update*** *works*
set *salary = salary * 1.1*
where *company_name = "First Bank Corporation"*

b. ***update*** *works*
set *salary = salary * 1.1*
where *works.company_name = "First Bank Corporation"*
*and ID in (**select** distinct manager_id*
from *manages)*

c. ***delete from*** *works*
where *company_name ="Small Bank Corporation"*

第二次作业 3.21

3.21 Consider the library database of Figure 3.20. Write the following queries in SQL.

a. Find the member number and name of each member who has borrowed at least one book published by "McGraw-Hill".

b. Find the member number and name of each member who has borrowed every book published by "McGraw-Hill".

c. For each publisher, find the member number and name of each member who has borrowed more than five books of that publisher.

d. Find the average number of books borrowed per member. Take into account that if a member does not borrow any books, then that member does not appear in the *borrowed* relation at all, but that member still counts in the average.

member(*memb_no*, *name*)
book(*isbn*, *title*, *authors*, *publisher*)
borrowed(*memb_no*, *isbn*, *date*)

Figure 3.20 Library database.

a.
```
select  M.memb_no, M.name
from    member M, book B, borrowed C
where C.memb_no = M.memb_no
        and B.isbn = C.isbn
        and B.publisher = "McGraw-Hill"
```

b.
```
select  M.memb_no, M.name
from    member M
where no exist (select isbn  from book
                        where B.publisher ="McGraw-Hill")
except (select isbn from borrowed E
             where E. memb_no = M.memb_no)
```

c.
```
select M.memb_no, M.name, B.publisher
from    member M, book B, borrowed C
where   C.memb_no = M.memb_no
        and B.isbn = C.isbn
group by  B.publisher, M.memb_no, M.name
having count(*) > 5
```

d.
```
Select avg(borrowed_count) as avg_borrowed_count
 from    (select ,count(b.isbn) as borrowed_count
           from member as M
           left join borrowed as B on M.memb_no = B.memb_no
           group by M. memb_no
 )as member_borrowed_count
```

第三次作业 4.7

employee (*<u>ID</u>*, *person_name*, *street*, *city*)
works (*<u>ID</u>*, *company_name*, *salary*)
company (*<u>company_name</u>*, *city*)
manages (*<u>ID</u>*, *manager_id*)

Figure 4.12 Employee database.

1. "employee"表，包括ID、人名、街道和城市列。ID是主键。
2. "works"表包括ID、公司名和薪资列。ID是一个外键，引用"employee"表，company_name是一个外键，引用"company"表。这确保了员工和所在公司之间的引用完整性。
3. "company"表包含公司名和城市列。公司名是主键。
4. "manages"表用于跟踪管理关系。它包括ID和manager_id列，都是外键，引用"employee"表。

第三次作业 4.7

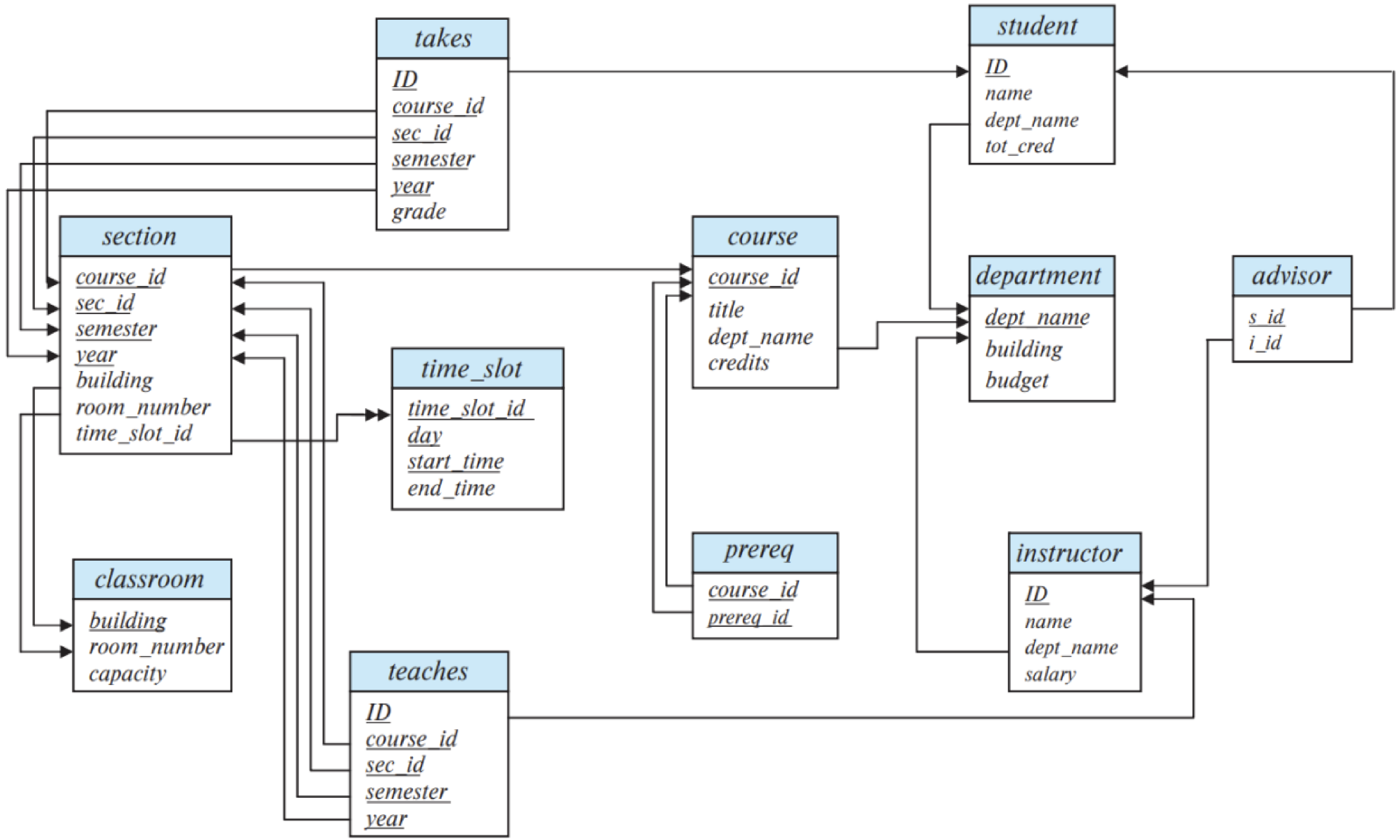
```sql
CREATE TABLE employee(
    ID INT PRIMARY KEY,
    person_name VARCHAR(128) NOT NULL,
    street VARCHAR(128),
    city VARCHAR(64)
)

CREATE TABLE works(
    ID INT PRIMARY KEY,
    company_name VARCHAR(128) NOT NULL,
    salary INT NOT NULL,
    FOREIGN KEY (company_name) REFERENCES company(company_name)
)


CREATE TABLE company(
    company_name VARCHAR(128) PRIMARY KEY,
    city VARCHAR(64)
)

CREATE TABLE manages(
    ID INT PRIMARY KEY,
    manager_id INT,
    FOREIGN KEY (manager_id) REFERENCES employee(ID)
)
```

第三次作业 4.16

4.16 Write an SQL query using the university schema to find the ID of each student who has never taken a course at the university. Do this using no subqueries and no set operations (use an outer join).

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

第三次作业 4.16




```sql
SELECT s.ID
FROM student s LEFT OUTER JOIN takes t
    ON s.ID = t.ID
WHERE t.ID IS NULL;
```

第四次作业 7.1

7.1 Suppose that we decompose the schema $R = (A, B, C, D, E)$ into

$$(A, B, C)$$
$$(A, D, E).$$

Show that this decomposition is a lossless decomposition if the following set F of functional dependencies holds:

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

根据无损分解的定义 $R_1 \cap R_2 \rightarrow R_1$ 或者 $R_1 \cap R_2 \rightarrow R_2$

因为 $R_1 = \{A, B, C\}$, $R_2 = \{A, D, E\}$, 所以 $R_1 \cap R_2 = \{A\}$

因为有 $A \rightarrow BC$, 所以可以得到 $A \rightarrow B$ 与 $A \rightarrow C$

因为 $A \rightarrow B$ 与 $B \rightarrow D$, 所以有 $A \rightarrow D$

因为 $A \rightarrow CD$ 与 $CD \rightarrow E$, 所以有 $A \rightarrow E$

综上所述 $A \rightarrow ABCDE$

结合 $R_1 \cap R_2 = \{A\}$

有 $R_1 \cap R_2 \rightarrow R_1$

所以这是一个无损分解

第四次作业 7.6

7.6 Compute the closure of the following set F of functional dependencies for relation schema $R = (A, B, C, D, E)$.

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

List the candidate keys for R.

因为有 $A \rightarrow BC$, 所以可以得到 $A \rightarrow B$ 与 $A \rightarrow C$

因为 $A \rightarrow B$ 与 $B \rightarrow D$, 所以有 $A \rightarrow D$

因为 $A \rightarrow CD$ 与 $CD \rightarrow E$, 所以有 $A \rightarrow E$

综上所述 $A \rightarrow ABCDE$

因为 $E \rightarrow A$, 所以 $E \rightarrow ABCDE$

因为 $CD \rightarrow E$, 所以 $CD \rightarrow ABCDE$

因为 $B \rightarrow D$ 且 $BC \rightarrow CD$, 所以有 $BC \rightarrow ABCDE$

综上所述，候选码有：A, E, CD, BC

第四次作业 7.27

7.27 Use Armstrong's axioms to prove the soundness of the decomposition rule.

- **Decomposition rule**. If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds.

of rules is called **Armstrong's axioms** in honor of the person who first proposed it.

- **Reflexivity rule**. If α is a set of attributes and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ holds.
- **Augmentation rule**. If $\alpha \rightarrow \beta$ holds and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$ holds.
- **Transitivity rule**. If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds.

因为 $\alpha \rightarrow \beta\gamma$

根据增强规则：有 $\alpha\beta \rightarrow \beta\gamma$

根据反射性规则：因为有 $\beta \subseteq \gamma\beta$，所以 $\gamma\beta \rightarrow \beta$

根据传递性规则，因为 $\alpha \rightarrow \beta\gamma$ 且 $\gamma\beta \rightarrow \beta$，所以有 $\alpha \rightarrow \beta$

证明 $\alpha \rightarrow \gamma$ 时同理，故不再赘述。

第四次作业 7.30

7.30 Consider the following set F of functional dependencies on the relation schema $(A,\ B,\ C,\ D, E, G)$:

$$A \rightarrow BCD$$
$$BC \rightarrow DE$$
$$B \rightarrow D$$
$$D \rightarrow A$$

a. Compute B^+.

因为 $B \rightarrow D$ 且 $D \rightarrow A$ 所以有 $B \rightarrow ABD$

因为 $A \rightarrow BCD$，所以有 $B \rightarrow ABCD$

因为 $BC \rightarrow DE$， 所以有 $B \rightarrow ABCDE$

综上所述，$B^+ = \{A, B, C, D, E\}$

第四次作业 7.30

b. Prove (using Armstrong's axioms) that AG is a superkey.

$A \to BCD$，由分解律：$A \to BC$

因为 $BC \to DE$，由传递律：$A \to DE$

由合并律：$A \to BCDE$

由增广律：$AG \to ABCDEG$

所以 AG 是超码

第四次作业 7.30

c. Compute a canonical cover for this set of functional dependencies F; give each step of your derivation with an explanation.

- 对于 $A \to BCD$，D 是无关属性，因为在 $(F - \{A \to BCD\}) \cup (A \to BC)$ 中，$A \to BC$ 和 $B \to D$ 可以推出 $A \to D$
- 对于 $BC \to DE$，D 是无关属性，因为在 $(F - \{BC \to DE\}) \cup (BC \to E)$ 中，$B \to D$ 可以推出 $BC \to D$
- 对于 $BC \to E$，C 是无关属性，因为由 $B \to D$、$D \to A$、$A \to BC$ 可以推出 $B \to C$，而由增广律可以得到 $B \to BC$，再由合并律即可得到 $B \to E$，因此 C 是无关属性
- 此时 F 可被简化为 $\{A \to BC, B \to E, B \to D, D \to A\}$
- 由于有两个左侧为 B 的函数依赖，则将其合并，可以得到 F 的正则覆盖为：

$$A \to BC$$
$$B \to DE$$
$$D \to A$$

第四次作业 7.30

d. Give a 3NF decomposition of the given schema based on a canonical cover.

正则覆盖中没有多余的函数依赖

所以属性集是正则覆盖中其它函数依赖组成的子集

所以三个函数依赖都有自己的关系

$$R_1(A,B,C)$$

$$R_2(B,D,E)$$

$$R_3(D,A)$$

AG是超码且上述关系没有原关系的超码，所以需要添加

$$R_4(A,G)$$

最终结果为

$$R_1(A,B,C)$$

$$R_1(B,D,E)$$

$$R_3(D,A)$$

$$R_4(A,G)$$

第四次作业 7.30

- Given the relational schema R<U, F>, U={A,B,C,D,E },
  F={AC→BD, B →C, C →D, B →E}
  a) Use Armstrong axioms and related rules to prove the functional dependency AC → E
  b) Compute $(A)^+$ and $(AC)^+$
  c) Find a canonical cover F_c of F
  d) Find all candidate keys, and point out R is in which normal form
  e) Decompose R into 3NF, which the decomposition is lossless-join and dependency preserving.
  f) Give related explanation or proof that the above decomposition is lossless-join and dependency preserving
  g) *Decompose the relation into relations in BCNF

a)

$$\because AC \rightarrow BD$$
$$\therefore AC \rightarrow B$$
$$\because 条件 B \rightarrow E$$
$$\therefore AC \rightarrow E$$

b)

$$(A)^+ = \{A\}$$

$$\because AC \rightarrow BD \; B \rightarrow E \therefore AC \rightarrow E$$
$$\therefore (AC)^+ = \{A, B, C, D, E\}$$

第四次作业 7.30

c)

将$AC \to BD$分解为$AC \to B$和$AC \to D$

得到：$AC \to B\ AC \to D\ B \to C\ C \to D\ B \to E$

由$AC \to D$和$C \to D$可以去掉A：$C \to D$

因此正则覆盖为：

$$\{AC \to B, B \to CE, C \to D\}$$

d)

$A^+ = \{A\}$

$B^+ = \{B, C, D, E\}$

$C^+ = \{C, D\}$

$D^+ = \{D\}$

$E^+ = \{E, A\}$

$AB^+ = \{A, B, C, D, E\}$

$AC^+ = \{A, C, B, D, E\}$

满足$1F$范式

e)

$R_1 = (A, B, C)$

$R_2 = (C, D)$

$R_3 = (B, C, E)$

第四次作业 7.30

f)

	A	B	C	D	E
ABC	a_1	a_2	a_3	$\cancel{b_{14}}$ a_4	$\cancel{b_{15}}$ a_5
CD	b_{21}	b_{22}	a_3	a_4	b_{25}
BCE	b_{31}	a_2	a_3	$\cancel{b_{34}}$ a_4	a_5

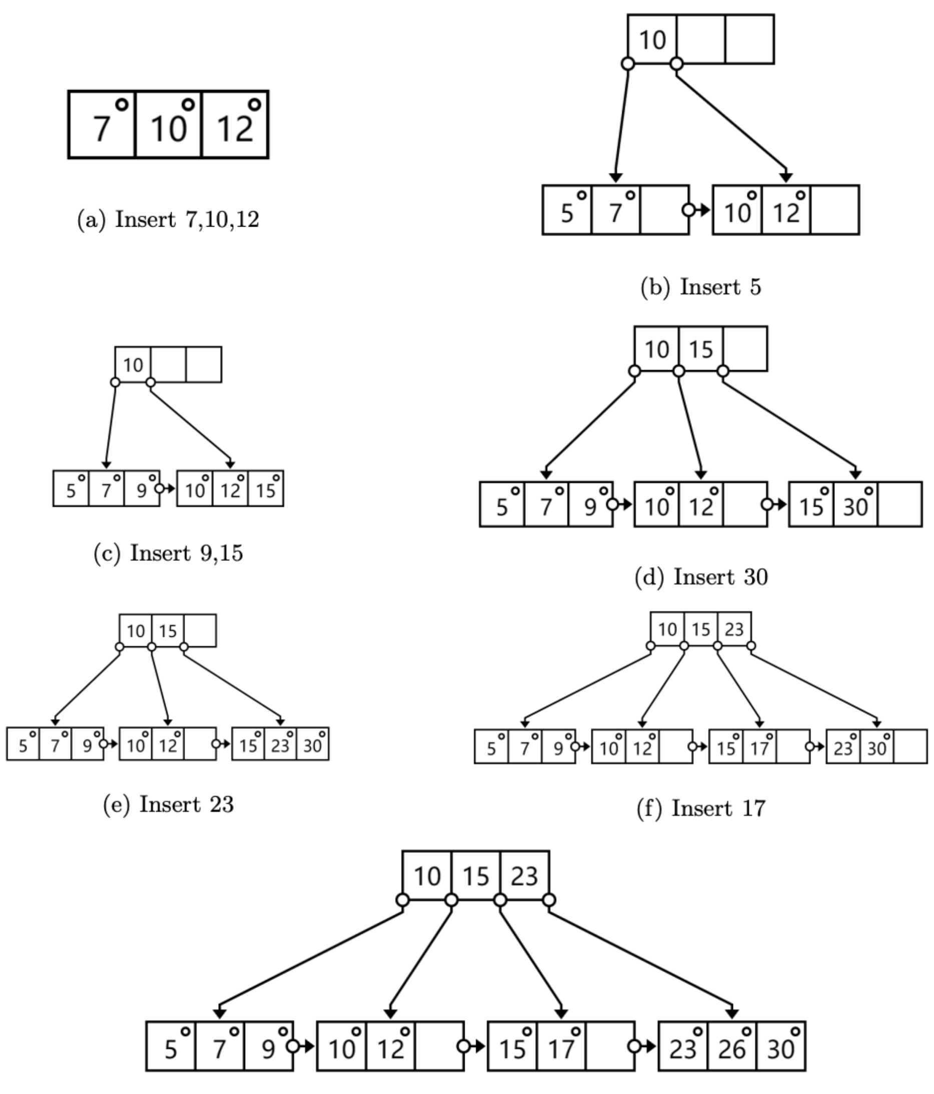
$B \to E$: $b_{15} \to a_5$

$C \to D$: $b_{14} \to a_4$ $\quad b_{34} \to a_4$

e) 分解结果为$\{A, B, C\}, \{B, C\}, \{B, E\}, \{C, D\}$

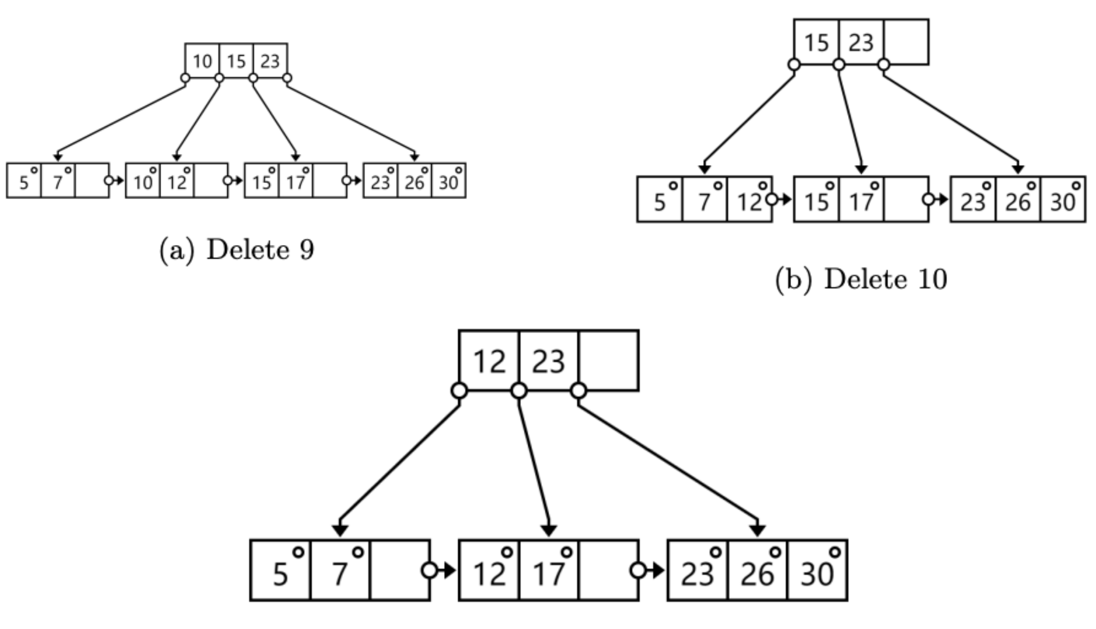
第五次作业

- **Q1**: Construct a B⁺-tree from an empty tree. Each node can hold four pointers
  - The sequential values to be inserted are: 10, 7, 12, 5, 9, 15, 30, 23, 17, 26
  - Then delete 9, 10, 15, respectively
  - Please give the B⁺ trees after each insertion and each deletion



(a) Insert 7,10,12 (b) Insert 5 (c) Insert 9,15 (d) Insert 30 (e) Insert 23 (f) Insert 17 (g) Insert 26

Figure 1.1.1: B^+ tree after sequential insertion (Part I)



(a) Delete 9 (b) Delete 10 (c) Delete 15

Figure 1.1.2: B^+ tree after sequential deletion (Part I)

第五次作业

- Q2: Compare B⁺-tree and B-tree and describe their difference

1. 节点存储数据的方式：

B-树：内部节点和叶子节点都可以存储数据。每个节点中的关键字信息包含关键码和指向子节点的指针，这些关键码和指针用于索引和查找数据。

B⁺树：内部节点不存储数据，只作为索引存在。所有数据都存储在叶子节点中，且叶子节点之间通过链表相连，这使得范围查询等操作更为高效。

2. 查找过程：

B-树：在查找过程中，如果找到了具体的数值，就会结束查找。

B⁺树：由于所有数据都存储在叶子节点中，所以查找过程会一直进行到叶子节点，通过索引找到叶子节点中的数据才结束。

3. 关键字出现频率：

B-树：中任何一个关键字出现且只出现在一个节点中。

B⁺树：由于叶子节点之间存在链表连接，所以关键字可能在多个叶子节点中出现。

4. 磁盘 I/O 与查询时间复杂度：

B-树：查询时间复杂度不固定，与 Key 在树中的位置有关。在某些情况下，如果 Key 恰好位于根节点或接近根节点的位置，查询时间可能会非常快（O(1)）。

B⁺树：由于叶子节点的数据都是使用链表连接起来的，并且在磁盘中是顺序存储的，所以当读到某个值时，磁盘预读原理会提前把这些数据都读进内存。这使得范围查询和排序等操作非常高效。同时，B⁺树的查询时间复杂度是固定的，为 O(logn)。

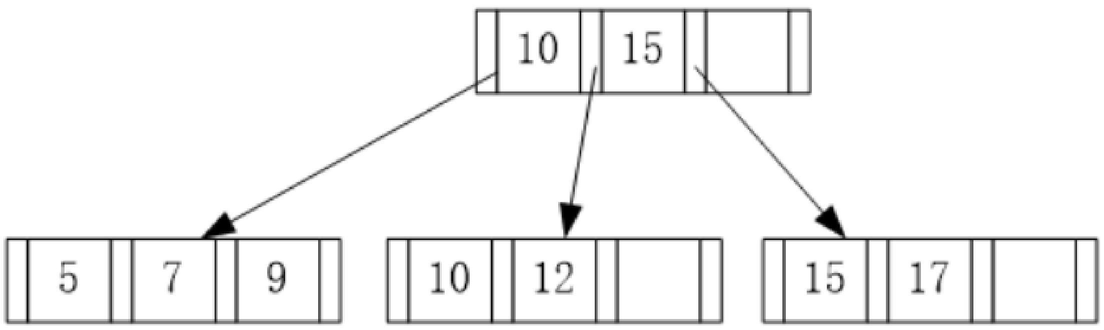
5. 应用场景：

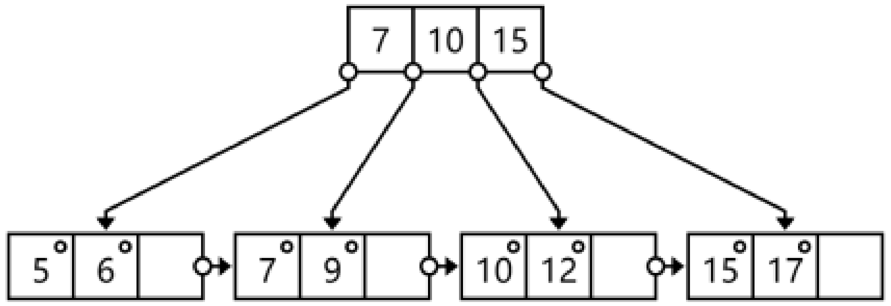
B-树：由于内部节点也可以存储数据，所以 B-树在需要频繁进行插入、删除等操作的场景下表现较好。它也被广泛应用于文件系统和数据库索引中。

B⁺树：由于所有数据都存储在叶子节点中，并且叶子节点之间通过链表相连，这使得 B⁺树在范围查询和排序等操作中表现出色。因此，B⁺树更适合用于需要频繁进行范围查询和排序的场景，如数据库索引等。

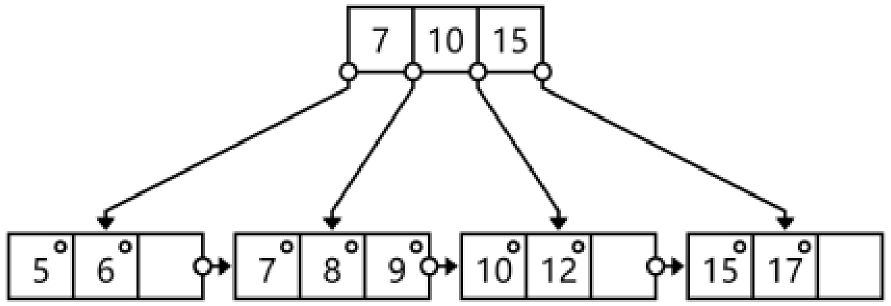
第五次作业

- For the B+ tree index, please give the B+ trees after insertion of search key 6 and 8:





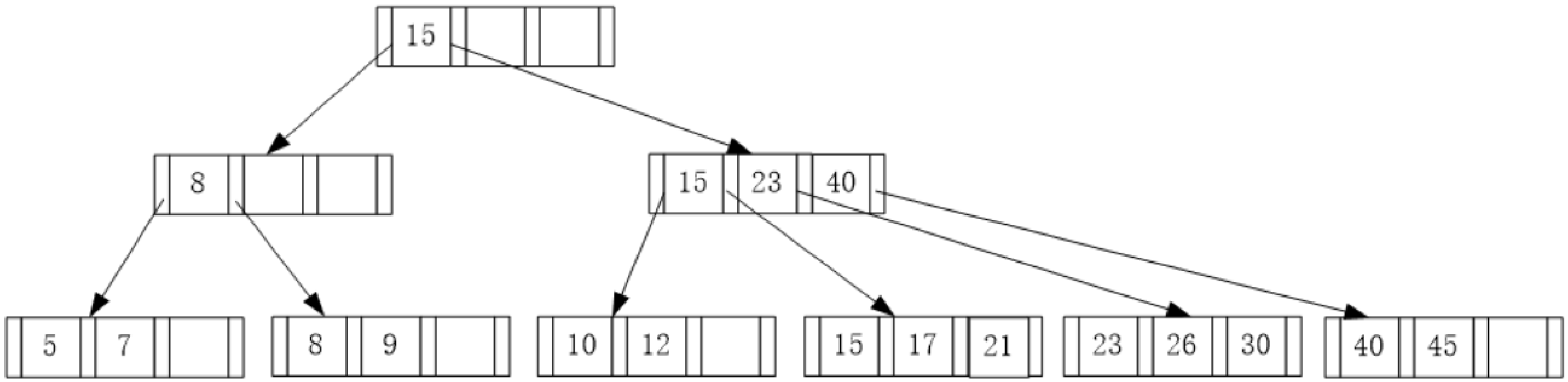
(a) Insert 6

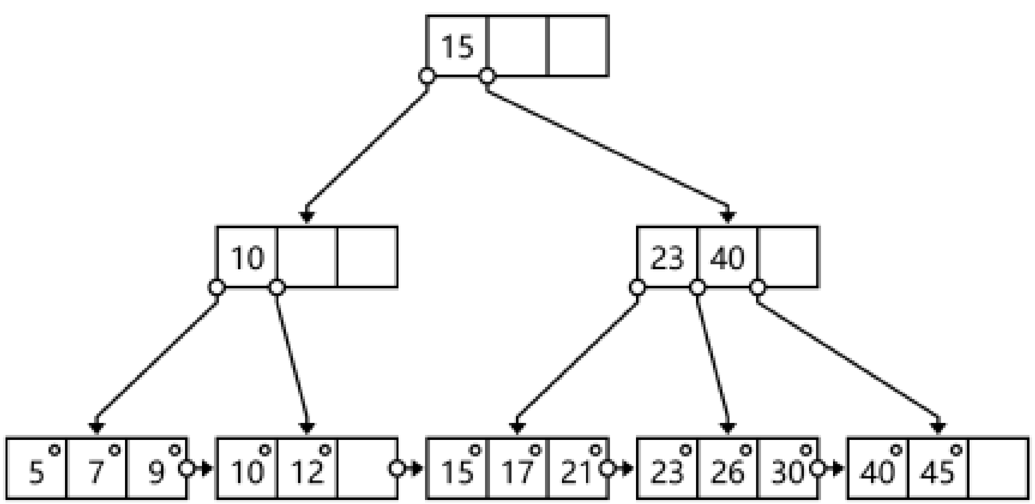


(b) Insert 8

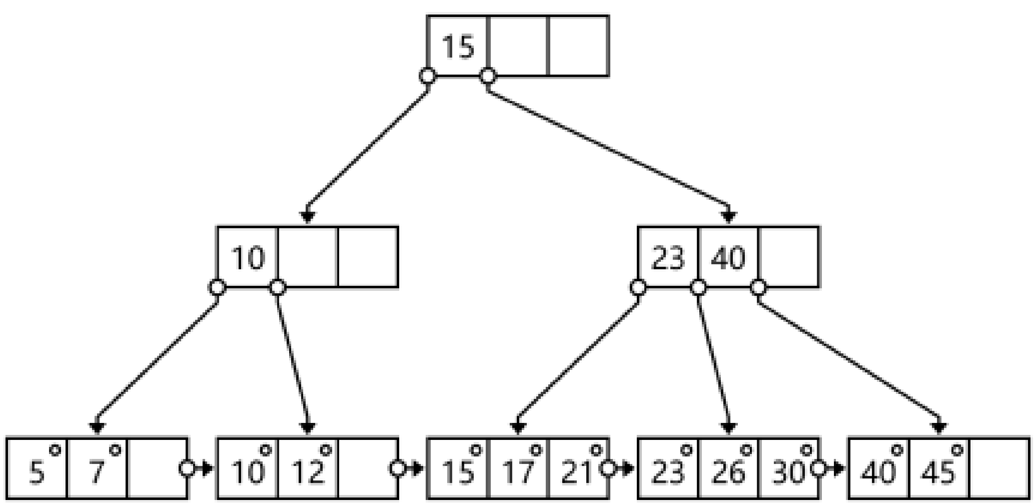
第五次作业

- Pease give the B+ trees after deletion of search keys both 8 and 9:





(a) Delete 8



(b) Delete 9

第六次作业

16.16 Suppose that a B$^+$-tree index on (*dept_name*, *building*) is available on relation *department*. What would be the best way to handle the following selection?

$$\sigma_{(building\ <\ \text{"Watson"})\ \wedge\ (budget\ <\ 55000)\ \wedge\ (dept\_name\ =\ \text{"Music"})}(department)$$

---

branch(<u>*branch_name*</u>, *branch_city*, *assets*)
customer (<u>*customer_name*</u>, *customer_street*, *customer_city*)
loan (<u>*loan_number*</u>, *branch_name*, *amount*)
borrower (<u>*customer_name*</u>, <u>*loan_number*</u>)
account (<u>*account_number*</u>, *branch_name*, *balance*)
depositor (<u>*customer_name*</u>, <u>*account_number*</u>)

Figure 16.9 Banking database.

第六次作业

在给定的条件下，最好的处理方式是利用 B+-树索引来优化查询。这是因为 B+-树索引在处理范围查询和精确匹配查询时非常高效。

假设有一个 B+-树索引存在于(dept name, building)上，可以采取以下步骤来处理查询 σ (building < “Watson”) ∧ (budget < 55000) ∧ (dept name = “Music”)(department)：

1. 索引扫描：首先利用 dept name 列来进行精确匹配查询，因为 B+-树索引的首要列是 dept name。查询 dept name = "Music"，找到所有属于 Music 系的记录。

2. 范围查询：对于第一步中匹配的记录，使用 building < "Watson"条件进行范围查询。由于 building 是索引的第二个列，可以在索引上高效地执行这个范围查询。

3. 过滤其他条件：对满足前两个条件的记录，在主表中提取出来，然后再检查 budget < 55000 的条件。这一步可能需要访问主表，但由于前两步已经大大减少了需要检查的记录数，这个操作会相对高效。