关系代数

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.

---

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

Figure 2.17 Employee database.

---

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

Figure 2.17 Employee database.

答案:

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

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

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

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. To find employees who do not work for BigBank, we first find all those who *do* work for BigBank. Those are exactly the employees *not* part of the desired result. We then use set difference to find the set of all employees minus those employees that should not be in the result.

$$\Pi_{ID,person\_name}(employee) -$$
$$\Pi_{ID,person\_name}$$
$$(employee \bowtie_{employee.ID=works.ID} (\sigma_{company\_name=\}\text{BigBank}''}(works)))$$

b. We use the same approach as in part *a* by first finding those employess who do not earn the highest salary, or, said differently, for whom some other employee earns more. Since this involves comparing two employee salary values, we need to reference the *employee* relation twice and therefore use renaming.

$$\Pi_{ID,person\_name}(employee) -$$
$$\Pi_{A.ID,A.person\_name}(\rho_A(employee) \bowtie_{A.salary<B.salary} \rho_B(employee))$$

SQL

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.

答案:

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

```
select e.ID, e.person_name, city
from employee as e, works as w
where w.company_name = 'First Bank Corporation' and
      w.ID = e.ID
```

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

```
select *
from employee
where ID in
   (select ID
    from works
    where company_name = 'First Bank Corporation' and salary > 10000)
```

This could be written also in the style of the answer to part *a*.

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

```
select ID
from works
where company_name <> 'First Bank Corporation'
```

If one allows people to appear in *employee* without appearing also in *works*, the solution is slightly more complicated. An outer join as discussed in Chapter 4 could be used as well.

```
select ID
from employee
where ID not in
   (select ID
    from works
    where company_name = 'First Bank Corporation')
```

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

```
select ID
from works
where salary > all
   (select salary
    from works
    where company_name = 'Small Bank Corporation')
```

If people may work for several companies and we wish to consider the *total* earnings of each person, the problem is more complex. But note that the fact that ID is the primary key for *works* implies that this cannot be the case.

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.

```
select S.company_name
from company as S
where not exists ((select city
                   from company
                   where company_name = 'Small Bank Corporation')
                  except
                   (select city
                    from company as T
                    where S.company_name = T.company_name))
```

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).

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

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

3.10 Consider the relational database of Figure 3.19. Give an expression in SQL for each of the following:

a. Modify the database so that the employee whose ID is '12345' now lives in "Newtown".

b. Give each manager of "First Bank Corporation" a 10 percent raise unless the salary becomes greater than $100000; in such cases, give only a 3 percent raise.

答案:

a. Modify the database so that the employee whose ID is '12345' now lives in "Newtown".

```
update employee
set city = 'Newtown'
where ID = '12345'
```

b. Give each manager of "First Bank Corporation" a 10 percent raise unless the salary becomes greater than $100000; in such cases, give only a 3 percent raise.

```
update works T
set T.salary = T.salary * 1.03
where T.ID in (select manager_id
                  from manages)
      and T.salary * 1.1 >  100000
      and T.company_name = 'First Bank Corporation'

update works T
set T.salary = T.salary * 1.1
where T.ID in (select manager_id
                  from manages)
      and T.salary * 1.1 <= 100000
      and T.company_name = 'First Bank Corporation'
```

The above updates would give different results if executed in the opposite order. We give below a safer solution using the **case** statement.

```
update works T
set T.salary = T.salary *
    (case
        when (T.salary * 1.1 > 100000) then 1.03
        else 1.1
    end)
where T.ID in (select manager_id
                from manages) and
      T.company_name = 'First Bank Corporation'
```

高级 SQL

4.3 Outer join expressions can be computed in SQL without using the SQL **outer join** operation. To illustrate this fact, show how to rewrite each of the following SQL queries without using the **outer join** expression.

a. **select * from** *student* **natural left outer join** *takes*

b. **select * from** *student* **natural full outer join** *takes*

---

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*)

Figure 2.8 Schema of the university database.

答案:

a. **select** * **from** *student* **natural left outer join** *takes*
can be rewritten as:

```
select * from student natural join takes
union
select ID, name, dept_name, tot_cred, null, null, null, null, null
from student S1 where not exists
    (select ID from takes T1 where T1.id = S1.id)
```

b. **select** * **from** *student* **natural full outer join** *takes*
can be rewritten as:

```
(select * from student natural join takes)
union
(select ID, name, dept_name, tot_cred, null, null, null, null, null
from student S1
where not exists
    (select ID from takes T1 where T1.id = S1.id))
union
(select ID, null, null, null, course_id, sec_id, semester, year, grade
from takes T1
where not exists
    (select ID from student S1 whereT1.id = S1.id))
```

---

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

Figure 4.12 Employee database.

4.7 Consider the employee database of Figure 4.12. Give an SQL DDL definition of this database. Identify referential-integrity constraints that should hold, and include them in the DDL definition.

答案:

```sql
CREATE TABLE company  (
   company_name varchar(255),
   city varchar(255),
   PRIMARY KEY (company_name)
)
CREATE TABLE employee  (
   ID int,
   person_name varchar(255),
   street varchar(255),
   city varchar(255),
   PRIMARY KEY (ID)
)
```

```sql
CREATE TABLE manages  (
   ID int,
   manager_id int,
   PRIMARY KEY (ID),
   FOREIGN KEY (ID) REFERENCES employee(ID),
   FOREIGN KEY (manager_id) REFERENCES employee(ID)
)
CREATE TABLE works  (
   ID int,
   company_name varchar(255),
   salary decimal(10, 2),
   PRIMARY KEY (ID),
   FOREIGN KEY (ID) REFERENCES employee(ID),
   FOREIGN KEY (company_name) REFERENCES company(company_name)
)
```

4.9 SQL allows a foreign-key dependency to refer to the same relation, as in the following example:

```
create table manager
   (employee_ID      char(20),
    manager_ID       char(20),
    primary key employee_ID,
    foreign key (manager_ID) references manager(employee_ID)
                                  on delete cascade )
```

Here, *employee_ID* is a key to the table *manager*, meaning that each employee has at most one manager. The foreign-key clause requires that every manager also be an employee. Explain exactly what happens when a tuple in the relation *manager* is deleted.

答案:

The tuples of all employees of the manager, at all levels, get deleted as well! This happens in a series of steps. The initial deletion will trigger deletion of all the tuples corresponding to direct employees of the manager. These deletions will in turn cause deletions of second-level employee tuples, and so on, till all direct and indirect employee tuples are deleted.

E-R 模型

- A database used in an ordering (订货/订购) system contains information about *customers*, *products* and *orders*. The following information is to be included:
  - for each ***customer***
    - *customer-number, ship-to-addresses* (several per customer), *name, balance, discount*
  - for each ***order***（订单）
    - *order number, customer number, ship-to address* (one per order), *date of ordering* (including *year, month, and day*), *products* (several kinds per order), and *quantity* ordered of each product
  - for each kind of ***product***

    product-number, name, manufacturing-plant, quantity on hand
- Some constraints(i.e., integrity constraints) about this ordering system are as follows
  - a customer may *make* one or more orders, and some customers may make no orders
  - each order must be for one (and only one) customer
  - each order must *contain* at least one kind of products
  - a kind of product may appear in several orders, but some kinds of products may not be ordered by any custome

写出关系表。

答案:

customer (customer-number, name, balance, account)

customer-ship-to-addresses (customer-number, ship-to-address)

order (order-number, ship-to-address, year, month, day, customer-number)

product (product-number, name, manufacturing-plant, quantity-on-hand)

contain (order-number, product-number, quantity-of-product)

7.20 考虑图 7-29 中的 E-R 图，它对一家网上书店建模。

a. 列出实体集及其主码。

b. 假设书店向其展览的商品中增加了蓝光光盘和可下载视频。相同的商品可能在一种格式中存在，或在两种格式中都存在且具有不同的价格。扩展 E-R 图来为这个新增需求建模，忽略对购物篮的影响。

- 现在用概化来扩展 E-R 图，从而对包含书、蓝光光盘或可下载视频的任意组合的购物篮进行建模。

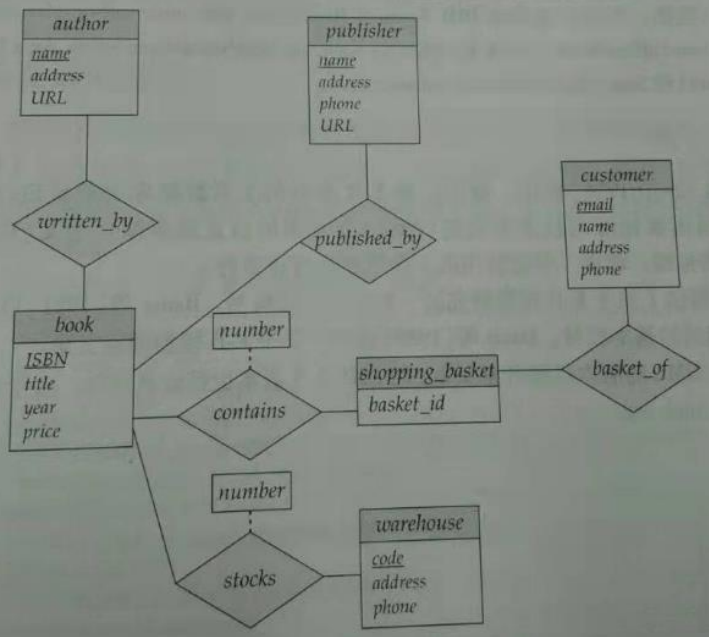


图 7-29 习题 7.20 的 E-R 图

写出关系表。

答案：

author (name, address, URL)

publisher (name, address, phone, URL)

book (ISBN, title, year, price)

shopping_basket (basket_id)

warehouse (code, address, phone)

customer (email, name, address, phone)

written_by (name. ISBN)

published_by (name, ISBN)

contains (ISBN, basket_id, number)

stocks (ISBN, code, number)

basket_of (email, basket_id)

函数依赖、范式与分解

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.

答案:

Note: It is not reasonable to expect students to enumerate all of F^+. Some shorthand representation of the result should be acceptable as long as the nontrivial members of F^+ are found.

Starting with $A \rightarrow BC$, we can conclude: $A \rightarrow B$ and $A \rightarrow C$.

Since $A \rightarrow B$ and $B \rightarrow D$, $A \rightarrow D$	(decomposition, transitive)
Since $A \rightarrow CD$ and $CD \rightarrow E$, $A \rightarrow E$	(union, decomposition, transitive)
Since $A \rightarrow A$, we have	(reflexive)
$A \rightarrow ABCDE$ from the above steps	(union)
Since $E \rightarrow A$, $E \rightarrow ABCDE$	(transitive)
Since $CD \rightarrow E$, $CD \rightarrow ABCDE$	(transitive)
Since $B \rightarrow D$ and $BC \rightarrow CD$, $BC \rightarrow ABCDE$	(augmentative, transitive)
Also, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow D$, etc.	

Therefore, any functional dependency with A, E, BC, or CD on the left-hand side of the arrow is in F^+, no matter which other attributes appear in the FD. Allow * to represent any set of attributes in R, then F^+ is $BD \rightarrow B$, $BD \rightarrow D$, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow BD$, $B \rightarrow D$, $B \rightarrow B$, $B \rightarrow BD$, and all FDs of the form $A \ast\!\!\rightarrow \alpha$, $BC \ast\!\!\rightarrow \alpha$, $CD \ast\!\!\rightarrow \alpha$, $E \ast\!\!\rightarrow \alpha$ where α is any subset of $\{A, B, C, D, E\}$. The candidate keys are A, BC, CD, and E.

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^+.

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

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

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

e. Give a BCNF decomposition of the given schema using the original set F of functional dependencies.

答案:

a. $B \rightarrow BD$ (third dependency)
$BD \rightarrow ABD$ (fourth dependency)
$ABD \rightarrow ABCD$ (first dependency)
$ABCD \rightarrow ABCDE$ *(second dependency)*

Thus, $B^+ = ABCDE$

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

$A \rightarrow BCD$ (Given)
$A \rightarrow ABCD$ (Augmentation with A)
$BC \rightarrow DE$ (Given)
$ABCD \rightarrow ABCDE$ (Augmentation with ABCD)
$A \rightarrow ABCDE$ (Transitivity)
$AF \rightarrow ABCDEF$ (Augmentation with F)

c. We see that D is extraneous in dep. 1 and 2, because of dep. 3. Removing these two, we get the new set of rules

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

Now notice that B^+ is $ABCDE$, and in particular, the FD $B \rightarrow E$ can be determined from this set. Thus, the attribute C is extraneous in

the third dependency. Removing this attribute, and combining with the third FD, we get the final canonical cover as :

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

Here, no attribute is extraneous in any FD.

d. We see that there is no FD in the canonical cover such that the set of attributes is a subset of any other FD in the canonical cover. Thus, each each FD gives rise to its own relation, giving

$$r_1(A, B, C)$$
$$r_2(B, D, E)$$
$$r_3(D, A)$$

Now the attribute F is not dependent on any attribute. Thus, it must be a part of every superkey. Also, none of the relations in the above schema have F, and hence, none of them have a superkey. Thus, we need to add a new relation with a superkey.

$$r_4(A, F)$$

e. We start with

$$r(A, B, C, D, E, F)$$

We see that the relation is not in BCNF because of the first FD. Hence, we decompose it accordingly to get

$$r_1(A, B, C, D)\ r_2(A, E, F)$$

Now we notice that $A \rightarrow E$ is an FD in F^+, and causes r_2 to violate BCNF. Once again, decomposing r_2 gives

$$r_1(A, B, C, D)\ r_2(A, F)\ r_3(A, E)$$

This schema is now in BCNF.

f. Can you get the same BCNF decomposition of r as above, using the canonical cover?
If we use the functional dependencies in the preceding canonical cover directly, we cannot get the above decomposition. However, we can infer the original dependencies from the canonical cover, and if we use those for BCNF decomposition, we would be able to get the same decomposition.

（5）R（U，F），U={A，B，C}，F ={A→B, B →A，C →A};

（6）R（U，F），U={A，B，C，D}，F ={A→C, D →B};

分别列出以上两题中 R 的候选码，并判断属于第几范式。

答案:

（5）候选码为 C，属于第二范式

（6）候选码为 AD，属于第一范式

- Considering the schema ***R***(C, T, H, R, S, G),
  and ***F***={CS →G, C →T, TH →R, HR →C, HS →R}
  , give a decomposition of ***R*** into BCNF

对 PPT 中上述题目做 3NF 分解，要求保持依赖且无损

答案：

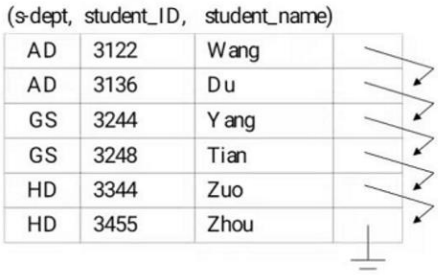
首先计算出 R 的候选码为 HS，然后计算 F 的正则依赖$F_c = F$，按照 F 进行 3NF 分解得到

$$R_1 = \{C, S, G\}$$
$$R_2 = \{C, T\}$$
$$R_3 = \{T, H, R\}$$
$$R_4 = \{H, R, C\}$$
$$R_5 = \{H, S, R\}$$

R_5中已包含候选键 HS，分解结束。

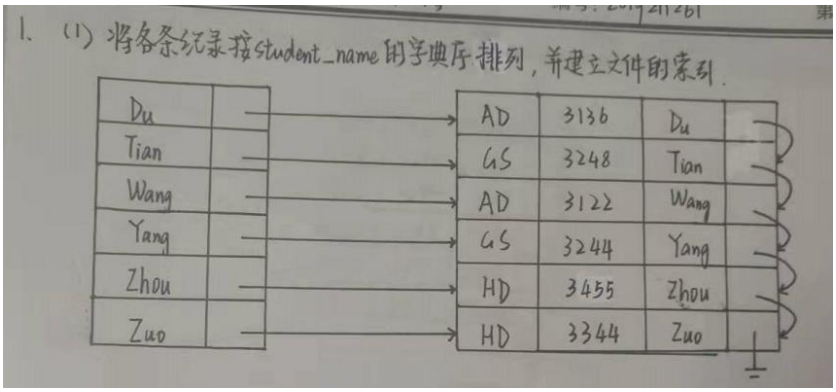
索引

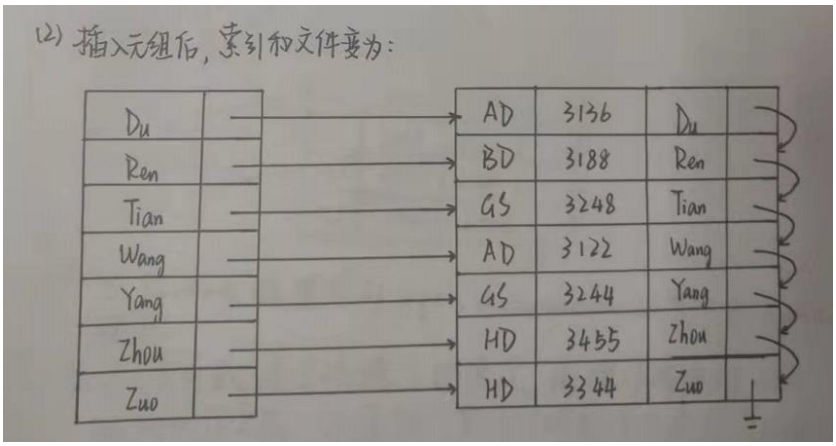
Give the data file ***teacher***(s_dept, student_ID, student_name) as shown below, which is organized as a sequential file, taking the attributes s_*dept* as the search key,
(1) *D*efine a *dense* and *clustering* index for the indexed file ***teacher*** on attribute student_name. It is required that the index file and index entries in the index file should be figured out.
(2) If a tuple (BD, 3188, Ren) is inserted into the indexed file, depict the indexed file and the index file.



(s-dept,	student_ID,	student_name)
AD	3122	Wang
AD	3136	Du
GS	3244	Yang
GS	3248	Tian
HD	3344	Zuo
HD	3455	Zhou

答案：





Consider the following tables in the database ***University***:
Student(*sid*, *name*, *department*, *age*)
Department(*dname*, *building*, *budget*)

- Consider the following SQL query. In addition to the existing primary indices on the primary keys of the tables, on which attributes in the tables the indices can be further defined to speed up the query?

```
select department, sum(sid)
from Student as A, Department as B
where building='T3' and age>20 and A.department= B.dname
group by department
```

(2) For the following query

```
update Student
set age=age+2
where sid between 211301 and 211318
```

A nonclustered index has been defined on the attribute *age*.
Does this index speed up or slow down the operation, and why?

答案：

（1）building，age，department

（2）减慢，更新索引增大开销。

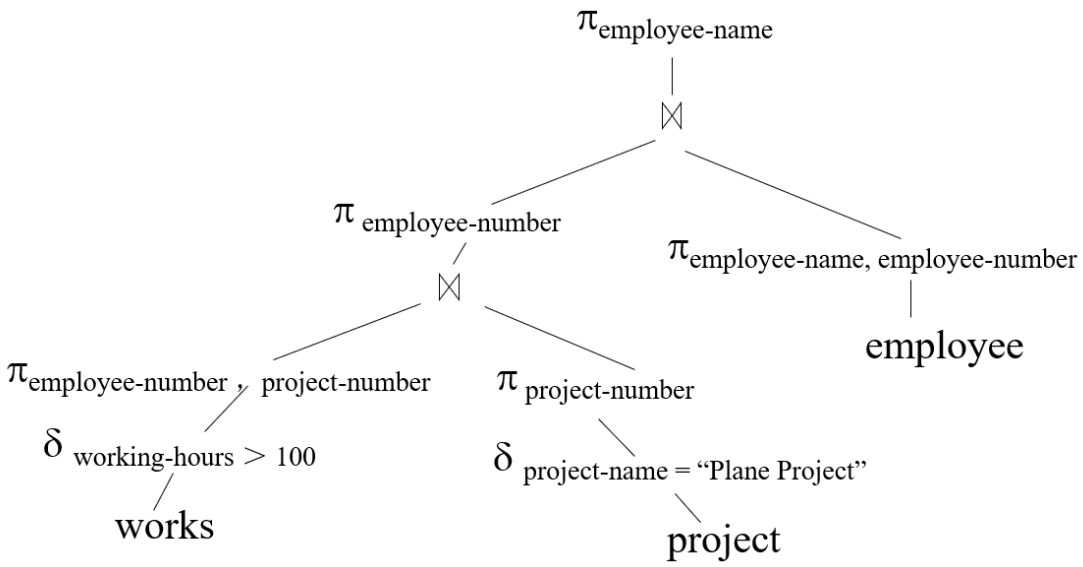
查询优化与事务

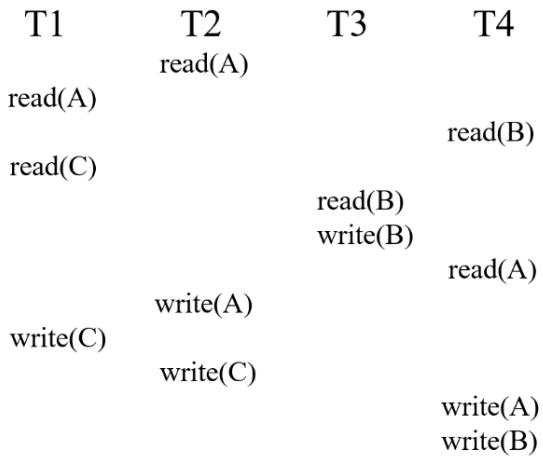
Heuristic Optimization

employee (employee-number, employee-name, sex, age, address, salary, department-number)
project(project-number ,project-name, address, department-number)
works(employee-number, project-number, working-hours)

Find the names of all employees who work for Plane Project and whose working hours is longer than 100 hours .

答案:

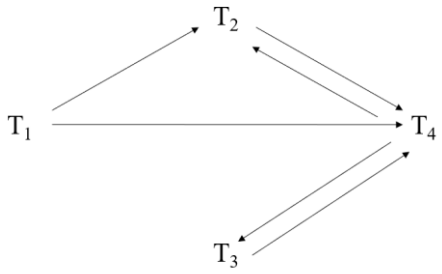




判断是否可以串行化。

答案:

构造调度的优先图。由调度图可以看出，存在环路，所以调度是非冲突可串行化的。



T_1	T_2	T_3	
	read(Y)		
read(X)			
	Y:=Y－20		
	write(Y)		
X:=X+10			
write(X)			
		read(X)	
	read(X)		
	X:=X－10		
commit			
	write(X)		
	commit		
		read(Y)	Is S a recoverable schedule? and why?
		X:=X+Y	
		write(X)	
		commit	Is S a cascadeless schedule? and why?

答案：

S 是可恢复调度。所有读写都满足 Tj 读取了 Ti 所写的数据项则 Ti 先于 Tj 提交。

S 不是无级联调度。T2 读取了 T1 所写的数据项 X，T1 未在 T2 读取前提交。