数据模型

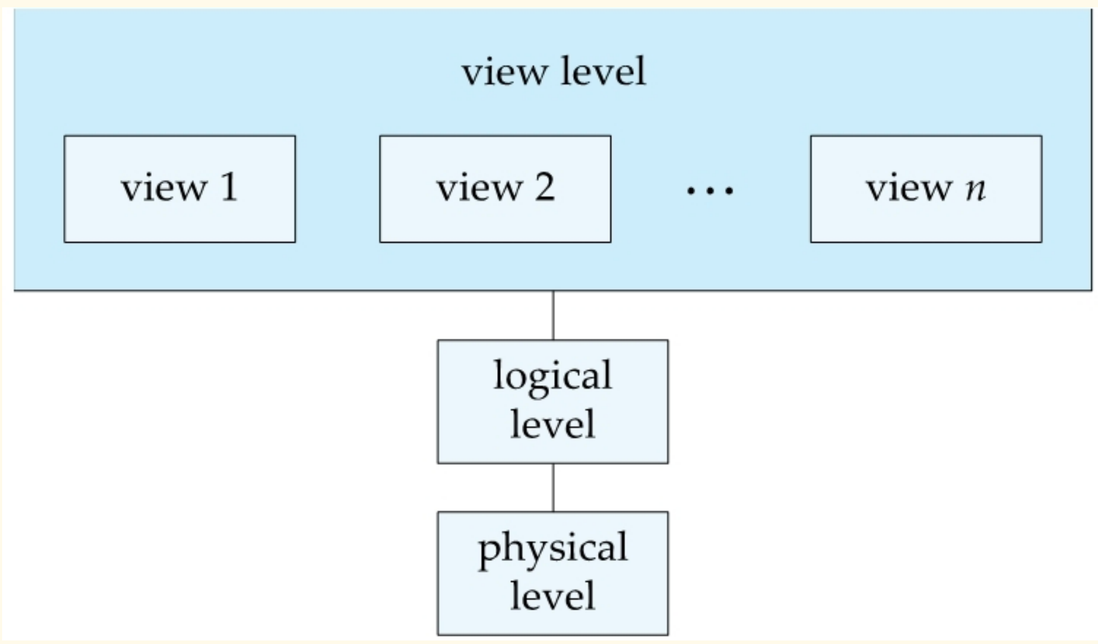
① 关系模型 ☆

② 实体联系模型 ☆ (Entity-Relationship)

③ 基于对象的数据模型 (Object-Relational)

④ 半结构化数据模型 (XML)

数据抽象



逻辑模式] (程序与数据间的)逻辑数据独立性

物理模式] 物理数据独立性 ☆

模式 → 实例

数据定义语言 DDL (针对模式)

DDL结果存在数据字典，数据字典包含元数据

元数据只能由系统本身访问与修改 〈 数据库模式 / 完整性约束 / 授权

数据操纵语言/查询语言 DML (针对实例)

DML 〈 过程化DML / 申明式DML (如SQL，并不是图灵机等效的)

⇓

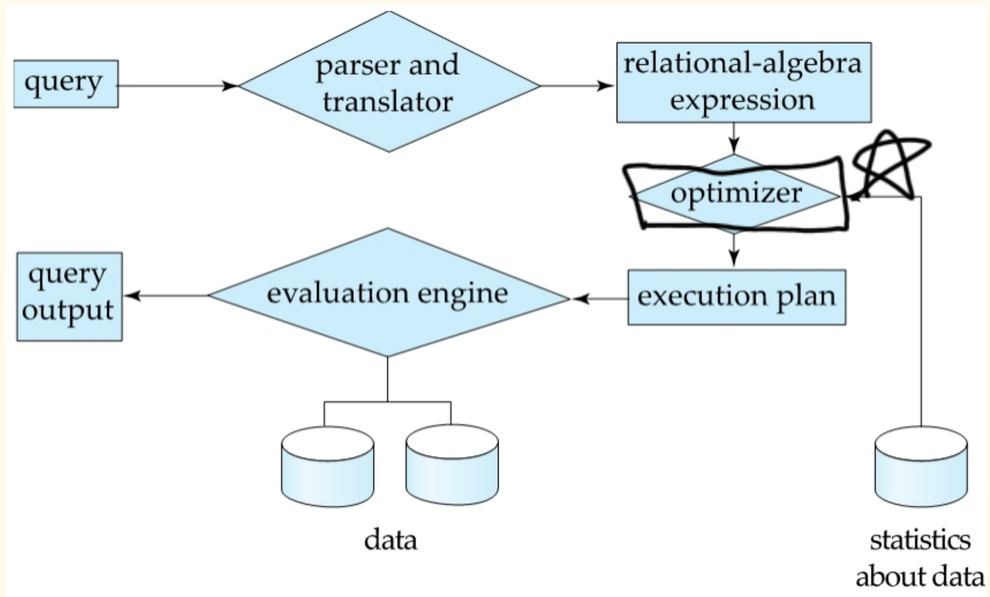
通过ODBC/JDBC与高级语言互通

⇑

应用程序使用宿主语言 (C/C++/JAVA，嵌入了SQL)

数据库设计与引擎

- 设计
  - 逻辑设计
  - 物理设计
- 引擎
  - 存储管理
  - 查询处理 ☆
  - 事务管理 ☆



关系数据库的结构

属性 $A_1 \cdots A_n$（表中的列）

属性取值的集合是域，每个域中均有空值(null)

域中每个元素均有原子性

关系模式 $R=(A_1, A_2, \cdots, A_n)$（表头）

数据库模式是库中所有关系模式集合

关系实例 $r(R)$

元组 $t \in r$（表中的行）

元组顺序无关紧要（行、列顺序Unordered）

码

超码 $K \subseteq R$，K对R具有唯一标识性

如{ID}是超码，增加任意属性后{ID, name}仍超码

候选码为最小超码，没有非标识的额外属性

一定存在但不一定唯一

例如选课（流水号, ID, 课程id, 课序, 学期, 年, 成绩）

候选码1　　候选码2

主码 从候选码中选择一个

有且仅有一个

复合主码各属性不能取空值

外码 一个关系中的值必须出现在另一关系中

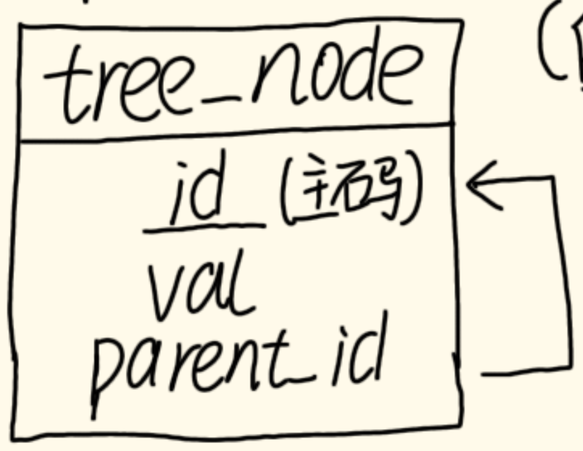
参照关系　　被参照关系

外码是 { 空值
被参照关系中的主码（不可是主码一部分）

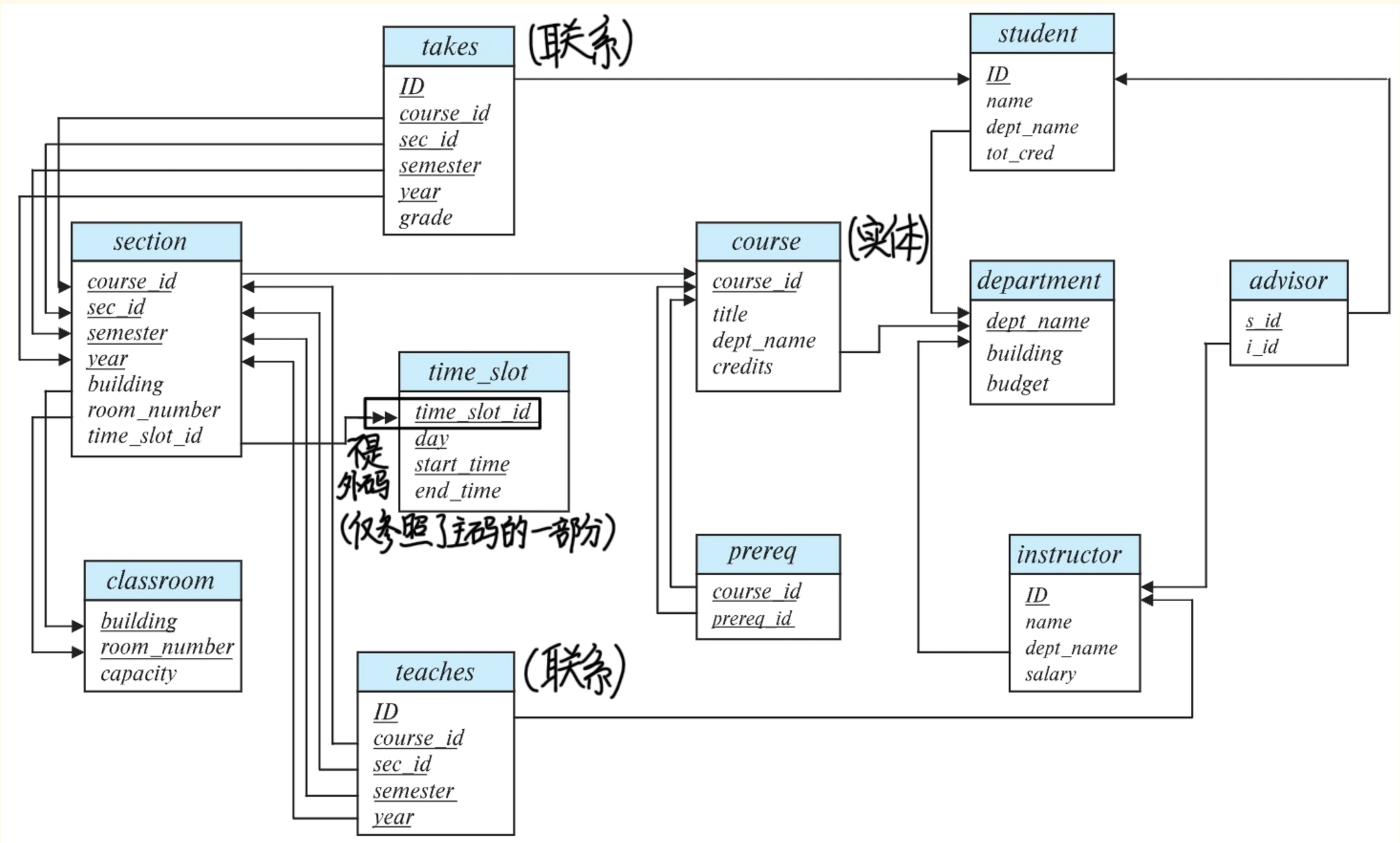
外码和被参照关系中的主码可以不同名

如教师表中的"部门"是外码，匹配部门表的主码

如树(id, val, parent_id)中 parent_id是外码



（自表引用）



关系查询语言

三种等价的语言：关系代数

元组关系演算

域关系演算

关系代数基本运算：选择σ　投影Π　并∪

差−　笛卡尔积×　更名ρ

(共6种基本的)

选择 $\sigma_p(r)$，p为选择谓词 (选择逐行进行操作)

可用 $=, \neq, >, \geq, <, \leq$

易错 $\wedge, \vee, \neg$

如

ID	name	dept_name	salary
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

$\sigma_{dept\_name = \text{"Physics"} \wedge salary \geq 90000}(instructor)$

投影 $\Pi_{A_1 A_2 \cdots A_K}(r)$，$A_i$为属性名

按列取出 $A_1 \sim A_k$ 属性

结果中重复的行会被去掉

笛卡尔积 $r_1 \times r_2$

如 instructor × teachers

instructor.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2018
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

当主码与外码等同

$\sigma_{instructor.id = teacher.id}(instructor \times teacher)$

或使用连接符 $\bowtie$：$r \bowtie_\theta s = \sigma_\theta(r \times s)$

$instructor \bowtie_{instructor.id = teacher.id} teacher$

作用：增补被参照关系的属性

instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
32343	El Said	History	60000	32343	HIS-351	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-101	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-319	1	Spring	2018
76766	Crick	Biology	72000	76766	BIO-101	1	Summer	2017
76766	Crick	Biology	72000	76766	BIO-301	1	Summer	2018
83821	Brandt	Comp. Sci.	92000	83821	CS-190	1	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-190	2	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-319	2	Spring	2018
98345	Kim	Elec. Eng.	80000	98345	EE-181	1	Spring	2017

并 $r \cup s$

① r, s 属性数相同

② 属性域兼容

交 $r \cap s$

差 $r - s$

赋值 $var \leftarrow expression$

更名 $\rho_x(E)$

如 $instructor \times instructor$ 会出错

(id属性同名)

$\rho_{new\_name}(instructor) \times instructor$

SQL属性类型

char(n)
varchar(n)
int (int长度取决于机器字长)
numeric(总位数, 小数位数)
smallint
real, double precision
float(n)
SQL不区分大小写, SQL字符串用单引号(关系代数"")

创建表

```
create table instructor (
        ID          char(5),
        name        varchar(20) not null,
        dept_name   varchar(20),
        salary      numeric(8,2),
        primary key (ID),
        foreign key (dept_name) references department);
```

} 完整性约束

完整性约束类型:
- primary key($A_1 \cdots A_n$)
- foreign key($A_m \cdots A_n$) references r
- not null

表更新

insert into student values('10211', 'Sam')
delete from student where … (删除tuple数据)
drop table student (删除表)

alter table student drop name(字段)
alter table student add grade
(新增加属性的值为null)

基本查询

select $A_1, A_2, \ldots, A_n$(属性)
from $r_1, r_2, \ldots, r_m$(关系表)
where P(谓词)

作用相当于投影 $\Pi_{A_1A_2\cdots A_n} S_p(r_1 \times r_2 \times \cdots \times r_n)$，但不去重

保留重复元素：
select all name from student;(默认)

去掉重复元素：
select distinct name from student;

选择所有属性：
select * from student;

验证查询：
select 'aa'; ⇒

'aa'

select 'aa' as AttributeName;
←——命名——→

广义投影：
select 'aa' from student;
N行

⇒

'aa'
'aa'
……
'aa'

N行

运算：

select grade*1.1 from student

连接：

```
select name, course_id
from instructor , teaches
where instructor.ID = teaches.ID
```

(为ins增补teach关系属性,数量与外码关系相同)

自然连接：

select * from instructor natural join teaches;

效果为等值连接,且共同属性仅保留一个属性

同名异义的属性会产生问题

Incorrect version (makes course.dept_name = instructor.dept_name)

```
select name, title
from instructor natural join teaches natural join course;
```

修正：select name, title from

(instructor natural join teaches)

join course using(course_id);

题：

select distinct T.name

from instructor as T, instructor as S

where T.salary < S.salary (找出工资非最高的)

在from子句中,as可有略为空格(select中不可者)

字符串操作

```
select name
from instructor
where name like '%dar%'
```

like 'W_G100?%77%' escape '?'

匹配一个字符　转义符　匹配任意长度字符串　定义转义符

'G' || 'Q' 连接

upper('g')　lower('G')

排序

select name from student

order by name desc/asc

只能放在select from后　二选一

介于

```
select name
from instructor
where salary between 90000 and 100000
```

[,]闭区间

元组比较

```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
```

集合运算

Find courses that ran in Fall 2017 or in Spring 2018

并

```
(select course_id from section where sem = 'Fall' and year = 2017)
union
(select course_id from section where sem = 'Spring' and year = 2018)
```

Find courses that ran in Fall 2017 and in Spring 2018

交

```
(select course_id from section where sem = 'Fall' and year = 2017)
intersect
(select course_id from section where sem = 'Spring' and year = 2018)
```

Find courses that ran in Fall 2017 but not in Spring 2018

差

```
(select course_id from section where sem = 'Fall' and year = 2017)
except
(select course_id from section where sem = 'Spring' and year = 2018)
```

union all
intersect all
except all
} 不建重

空值

任何有null的算术运算结果均为null

5 + **null** returns **null**

任何有null的比较运算结果均为unknown

=, >, <

5 < **null** or **null** <> **null** or **null** = **null**

判定null

select *name*
from *instructor*
where *salary* **is null**

< is null
is not null

逻辑运算

and : (*true* **and** *unknown*) = *unknown*,
(*false* **and** *unknown*) = *false*,
(*unknown* **and** *unknown*) = *unknown*

or: (*unknown* **or** *true*) = *true*,
(*unknown* **or** *false*) = *unknown*
(*unknown* **or** *unknown*) = *unknown*

对where语句，unknown视作false
对order by，null视作最大的

聚合函数

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

(运算会忽略空值)
(不能嵌套使用☆)

例如：

select avg (*salary*)
from *instructor*
where *dept_name*= 'Comp. Sci.';

select count (**distinct** *ID*)
from *teaches*
where *semester* = 'Spring' **and** *year* = 2018;

分组

```
select dept_name, avg (salary) as avg_salary
from instructor
group by dept_name;
```

空值会被分至空值组

☆除了聚集函数，select后的属性必须出现在group by之后

如：select name, avg(grade)
from student natural join take_course
group by ID, name;
不可只有ID

分组限制

```
select dept_name, avg (salary) as avg_salary
from instructor
group by dept_name
having avg (salary) > 42000;
```

相当于先分组后，再选择

having后属性必须是聚合函数或group by的

嵌套查询

①成员判定

in / not in

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
        course_id  not in (select course_id
                           from section
                           where semester = 'Spring' and year= 2018);
```

```
select distinct name
from instructor
where  name not in ('Mozart', 'Einstein')
```
元组用(, ,)

②集合比较

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept name = 'Biology');
```

(5 < **some** | 0 / 5 / 6 |) = true

(5 < **some** | 0 / 5 |) = false

(5 = **some** | 0 / 5 |) = true

(5 ≠ **some** | 0 / 5 |) = true (

注意，≠some(··)不是not in

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept name = 'Biology');
```

(5 < **all** | 0, 5, 6 |) = false

(5 < **all** | 6, 10 |) = true

(5 = **all** | 4, 5 |) = false

(5 ≠ **all** | 4, 6 |) = true

注意 ≠all(…)就是 not in
但 =all(…)不是 in

… >= all(select … from ……)

例：找最高平均工资部门

```
select dept-name, avg(salary)
from instructor
group by dept_name
having avg(salary)>=
    all(select avg(salary)
        from instructor
        group by dept_name
       );
```

③存在

exists r ⇔ $r \neq \emptyset$

not exists r ⇔ $r = \emptyset$

☆只能用在where子句中

相关查询的内层后于外层进行

例：

```
select course_id
from section as S
where semester = 'Fall' and year = 2017 and
        exists  (select *
                 from section as T
                 where semester = 'Spring' and year= 2018
                       and S.course_id = T.course_id);
```

相关子查询

例：

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course
                    where dept_name = 'Biology')
                   except
                    (select T.course_id
                     from takes as T
                     where S.ID = T.ID));
```

(学生选了生物系所有课程)

not exists (… except …)
⇔ $X - Y = \emptyset$
⇔ $X \subseteq Y$

☆用于判断集合包含☆

④ 嵌套

```
select T.course_id
 from course as T
 where unique ( select R.course_id
                from section as R
                where T.course_id= R.course_id
                      and R.year = 2017);
```

⑤ 嵌套也可放在 from (…) as T
↑这里面

```
select dept_name, avg_salary
from ( select dept_name, avg (salary)
       from instructor
       group by dept_name)
       as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

需使用as命名(嵌套子查询时不可省)

⑥ 嵌套用with实现

```
with max_budget (value) as
     (select max(budget)
      from department)
select department.name
from department, max_budget
where department.budget = max_budget.value;
```

标量，一行一列

```
with dept _total (dept_name, value) as
     (select dept_name, sum(salary)
      from instructor
      group by dept_name),
dept_total_avg(value) as
     (select avg(value)
     from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.value;
```

单行单列

with定义的临时关系仅在最近一个select中生效

⑦ select中嵌套(标量子查询)

```
select dept_name,
        ( select count(*)
          from instructor
          where department.dept_name = instructor.dept_name)
        as num_instructors
from department;
```

select到的单列单列值

(相关子查询)

⑧ 哑元关系

```
select ( select count(*) from student
         /
         select count(*) from instructor)
from dual
```

更新语句

删除

```
delete from instructor
```

只删实例，不删关系模式（drop删两者）

```
delete from instructor
where dept_name= 'Finance';
```

```
delete from instructor
where salary < (select avg (salary)
                from instructor);
```

插入

```
insert into course (course_id, title, dept_name, credits)
    values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

```
insert into student
    values ('3003', 'Green', 'Finance', null);
```

```
insert into instructor
    select ID, name, dept_name, 18000
    from   student
    where  dept_name = 'Music' and total_cred > 144;
```

```
insert into table1 select * from table1
```
（将表中内容复制增加一份）

如需自增序列号

```
create table t( ID serial, val char(20), primary key(ID));
insert into t select val from t;
```

（用于增加试验数据）

更新

```
update instructor
   set salary = salary * 1.05
```

```
update instructor
    set salary = salary * 1.05
   where salary < 70000;
```

```
update instructor
set salary = salary * 1.05
where salary <  (select avg (salary)
                 from instructor);
```

慎用2条update，因为顺序很重要，如

```
update instructor
   set salary = salary * 1.03
   where salary > 100000;
update instructor
    set salary = salary * 1.05
    where salary <= 100000;
```

否则99999先×1.05，又再×1.03

```
update instructor
   set salary = case
                    when salary <= 100000 then salary * 1.05
                    else salary * 1.03
                   end
```

set子句的子查询结果必须为标量

```
update student S
set tot_cred = (select sum(credits)
                from takes, course
                where takes.course_id = course.course_id  and
                        S.ID= takes.ID.and
                        takes.grade <> 'F' and
                        takes.grade is not null);
```

连接

自然连接

```
select name, course_id
from student natural join takes;
```

相同元组仅保留一份、左结合

```
select A1, A2, … An
from r1 natural join r2 natural join .. natural join rn
where P ;
```

在共有属性上做等值、存在同名异义的危险

内连接

不要求同名属性列，可用using指定，如

from table1 inner join table2 on table1.ID = table2.UserID

若同名，可用 using(ID)

外连接

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

外连接不舍弃失配元组

左外连接

course **natural left outer join** *prereq*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

In relational algebra: *course* ⟕ *prereq*

保留左表失配元组

右外连接

course **natural right outer join** *prereq*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

In relational algebra: *course* ⟖ *prereq*

全外连接

course **natural full outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

In relational algebra: *course* ⟗ *prereq*

连接方式与条件

Join types
inner join
left outer join
right outer join
full outer join

Join conditions
natural (同名属性上等值，只保留相同属性的一份)
on <predicate> (保留多份相同属性)
using $(A_1, A_2, \ldots, A_n)$ (同名属性的子集上等值，只保留相同属性的一份)

视图

视图为虚关系，关系和数据不存于数据库中，仅供可视化

create view *v* **as** < query expression >

只能drop，不能alter

行列子集视图(允许更新)

```
create view faculty as
        select ID, name, dept_name
        from instructor
```

分组视图(无法更新)

```
create view departments_total_salary(dept_name, total_salary) as
    select dept_name, sum (salary)
    from instructor
   group by dept_name;
```

依赖

V_1直接依赖于V_2

```
create view physics_fall_2017_watson as
    select course_id, room_number
    from physics_fall_2017
    where building= 'Watson';
```

V_1间接依赖于V_2

V_1递归 $\Longleftrightarrow$ V_1依赖于V_1

视图扩展

将依赖的视图替换为其定义的基本表

```
create view physics_fall_2017_watson  as
    select course_id, room_number
    from physics_fall_2017
    where building= 'Watson'
```

⇒

```
create view physics_fall_2017_watson as
    select course_id, room_number
    from (select course.course_id, building, room_number
        from course, section
        where course.course_id = section.course_id
            and course.dept_name = 'Physics'
            and section.semester = 'Fall'
            and section.year = '2017')
    where building= 'Watson';
```

物化视图

基本表改变，物化视图也需刷新

存储数据元组

更新视图

```
insert into faculty
    values ('30765', 'Green', 'Music');
```

事务

原子性

begin transactions;

……(sql语句)

$\underbrace{\text{commit}}_{\text{全做}}$ / $\underbrace{\text{rollback}}_{\text{全不做}}$;

隔离性

多用户执行时，相互隔离

约束

完整性约束

not null

```
name varchar(20) not null
budget numeric(12,2) not null
```

primary key

不允许取null

unique

允许取null

不允许属性值重复

check

只拒绝假，允许真和不定（null in (…)）

```
create table section
    (course_id varchar (8),
     sec_id varchar (8),
     semester varchar (6),
     year numeric (4,0),
     building varchar (15),
     room_number varchar (7),
     time slot id varchar (4),
     primary key (course_id, sec_id, semester, year),
     check (semester in ('Fall', 'Winter', 'Spring', 'Summer')))
```

而where只允许真

check后不允许子查询（check(…in (子查询))）X，

用trigger实现

违反约束，拒绝操作

参照完整性约束

参照表（外码的表）违约，拒绝操作

被参照表违约，可拒绝、置空、级联修改

- on delete/update cascade
- set null
- set default

```
create table course (
    (…
     dept_name varchar(20),
     foreign key (dept_name) references department
        on delete cascade
        on update cascade,
     . . .)
```

应用：

```
create table person (
    ID  char(10),
    name char(40),
    mother char(10),
    father  char(10),
    primary key ID,
    foreign key (father) references person,
    foreign key (mother) references  person)
```

先插人 or 先插入父母？

① 约束 set null，将父母初始置空

② 推迟外码约束

如：两表相互参照时

```
alter table employee
  add constraint employee_fk_dno foreign key (dno)
    references department initially deferred;
alter table department
  add constraint department_fk_mgr_ssn foreign key
    (mgr_ssn) references employee(ssn) initially
    deferred;
```

断言

```
create assertion <assertion-name> check (<predicate>);
```

SQL中的特殊数据

日期数据

date:

- Example: **date** '2005-7-27'

time:

- Example: **time** '09:00:30' **time** '09:00:30.75'

timestamp: (通过now( )函数返回)

- Example: **timestamp** '2005-7-27 09:00:30.75'

interval: (可与date/time/timestamp加减)

- Example: interval '1 day'

大对象

blob：二进制大对象

clob：字符大对象

用户自定义类型

create type *Dollars* **as numeric (12,2) final**

type：强类型检查 (最终类，不能被继承)

(dollars与cny不可比)

create domain *person_name* **char**(20) **not null** 约束

create domain *degree_level* **varchar**(10)
constraint *degree_level_test*
check (**value in** ('Bachelors', 'Masters', 'Doctorate'));

domain：弱类型检查 (dollars与cny都为numeric，可比)
可以带约束

索引

create index <name> **on** <relation-name> (attribute);

通过建B+树，加速查找

create table *student*
(*ID* **varchar** (5),
name **varchar** (20) **not null**,
dept_name **varchar** (20),
tot_cred **numeric** (3,0) **default** 0,
primary key (*ID*))

create index *studentID_index* **on** *student*(*ID*)

授权

关系模式权限

Index - allows creation and deletion of indices.

Resources - allows creation of new relations. (允许被参照)

Alteration - allows addition or deletion of attributes in a relation.

Drop - allows deletion of relations.

关系表权限

Read - allows reading, but not modification of data.

Insert - allows insertion of new data, but not modification of existing data.

Update - allows modification, but not deletion of data.

Delete - allows deletion of data.

grant语句授权

grant select on *instructor* **to** U_1, U_2, U_3

增删改查，
all privilleges

用户id，角色，public

revoke语句回收权限

revoke select on *student* **from** U_1, U_2, U_3

角色授权

create role *dean*;

grant *instructor* **to** *dean*; (对角色授权)

grant *dean* **to** Satoshi; (对用户赋予角色)

视图授权

```
create view  geo_instructor as
(select *
from instructor
where dept_name = 'Geology');
grant select on geo_instructor to  geo_staff
```

SQL接口(*不考)

- API(如JDBC)
- 嵌入式SQL(如SQLJ)

JDBC

创建连接→创建statement→作为语句查询

prepare statement：使用其生成sql语句

```
PreparedStatement pStmt = conn.prepareStatement(
                                    "insert into instructor values(?,?,?,?)");
```

而不是串拼接生成sql语句，可将SQL注入风险降为0

```
"insert into instructor values(' " + ID + " ', ' " + name + " ', " + " ' + dept
name + " ', " ' balance + ')“
```

'或"会引发问题

元数据：逆向工程

SQLJ

在编译时进行检查，而不是运行才报错

函数和过程(*不考)

PL/SQL

```
create function dept_count (dept_name varchar(20))
    returns integer 声明返回类型
    begin
    declare d_count  integer; 定义局部变量
        select count (* ) into d_count
        from instructor
        where instructor.dept_name = dept_name
    return d_count;
end
```

函数体

为单值赋予局部变量

```
create function instructor_of (dept_name char(20))
        returns table  (ID varchar(5),
                        name varchar(20),
                        dept_name varchar(20),
                        salary numeric(8,2))
        return table
                (select ID, name, dept_name, salary
                 from instructor
                 where instructor.dept_name = instructor_of.dept_name)
```

定义返回表的关系模式

For循环

```
declare n  integer default 0;
for r  as
      select budget from department
do
            set n = n + r.budget
end for
```

(最常用)

If语句

```
if boolean  expression
    then statement or compound statement
 elseif boolean  expression
    then statement or compound statement
 else statement or compound statement
 end if
```

While循环

```
while boolean expression  do
          sequence of statements ;
 end while
```

使用外部程序

```
create function dept_count(dept_name varchar(20))
returns integer
language C
external name '/usr/avi/bin/dept_count'
```

为.out文件

触发器

```
create trigger credits_earned after update of takes on (grade)
referencing new row as nrow
referencing old row as orow
for each row
when nrow.grade <> 'F' and nrow.grade is not null
   and (orow.grade = 'F' or orow.grade is null)
begin atomic
    update student
    set tot_cred= tot_cred +
         (select credits
          from course
          where course.course_id= nrow.course_id)
    where student.id = nrow.id;
end;
```

或 insert / delete

referencing old row as : for deletes and updates
referencing new row as : for inserts and updates

行级 FOR EACH ROW 执行指定操作的触发器为操作修改的每一行都调用一次。行级触发器 — old row → new row

语句级 FOR EACH STATEMENT 的触发器只是为任何操作执行一次，不管有多少行被修改。语句级触发器 — old table → new table

应用：管理统计数据（行级）
监控修改记录（语句级）

递归查询

```
with recursive rec_prereq(course_id, prereq_id) as (
    select course_id, prereq_id
    from prereq
  union
    select rec_prereq.course_id, prereq.prereq_id,
    from rec_rereq, prereq
    where rec_prereq.prereq_id = prereq.course_id
  )
select *
from rec_prereq;
```

子 爷] 基查询 ⇒ 父子关系

子 爷

子 子 爷

单调递增有上界 ⇒ 直到不动点

高级聚集函数

排名

rank() over(order by … desc/asc)

```
select ID, rank() over (order by GPA desc) as s_rank
from student_grades
order by s_rank
```

dense_rank()	90	85	85	80
密排名	1	2	3	4
排名	1	2	2	4

```
select ID, dept_name,
    rank () over (partition by dept_name order by GPA desc)
        as dept_rank
from dept_grades
order by dept_name, dept_rank;
```

分区排名

```
select ID,
        rank ( ) over (order by GPA desc nulls last) as s_rank
from student_grades
```

空值在最后（默认最前）

```
select ID, ntile(4) over (order by GPA desc) as quartile
    from student_grades;
```

仅排为1,2,3,4（四等分）

滑动窗口

```
select date, sum(value) over
    (order by date between rows 1 preceding and 1 following)
from sales
```

或

between rows unbounded preceding and current (前面所有行和当前行)

rows unbounded preceding (前面所有行)

```
select account_number, date_time,
   sum (value) over
            (partition by account_number
            order by date_time
            rows unbounded preceding)
  as balance
from transaction
order by account_number, date_time
```

分区

关系代数

基本运算

select: σ

project: Π

union: $\cup$

set difference: –

Cartesian product: x

rename: ρ

→ 关系需同构

附加运算

Set intersection $r \cap s$

Natural join $r \bowtie s$ (当r,s无共同属性时,做笛卡积)

Assignment $\leftarrow$

指定 theta join: $r \bowtie_\theta s$

Outer join

- 左外连 $⟕$
- 右外连 $⟖$
- 全外连 $⟗$

eg.

r

ID	name	dept_name
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

s

ID	course_id
10101	CS-101
12121	FIN-201
76766	BIO-101

$r \bowtie s$

ID	name	dept_name	course_id
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*

$r ⟕ s$

ID	name	dept_name	course_id
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
76766	null	null	BIO-101

商运算

满足逆乘不溢出的最大关系,如 $5/2=2$

$5/3=1$

$r/s=t \Rightarrow t \times s \subseteq r$

要求 $r \div s$, s的关系模式必须真包含于r

÷用于解决嵌套无法在关系代数中表示的问题

↳ 集合包含

eg.

ID	course_id
10101	CS-101
10101	CS-147
10101	CS-191
12121	CS-101
12121	CS-191
76766	BIO-101

r

course_id
CS-101
CS-147
CS-191

s

关系模式作差

数据项逆乘不涵出 ⟹

ID
10101

即该ID对应的course_id集合 ⊇ 课程号表集合

用途：选了所有生物系课程的学生

$$\Pi_{name,id}(STU) \bowtie \left(\Pi_{ID,course}(Takes) \div \Pi_{course}\, \sigma_{dept=\text{"Bio"}}(Course)\right)$$

总结：

$temp1 \leftarrow \Pi_{R-S}(r)$ 全部数据项

$temp2 \leftarrow \Pi_{R-S}((temp1 \times s) - \Pi_{R-S,S}(r))$ 逆乘涵出项

$result = temp1 - temp2$ 商

聚集函数

分组 $_{G_1,G_2,\ldots,G_n}\mathcal{G}_{F(A)}$

聚集函数

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values (**count-distinct**)

eg. $\mathcal{G}_{avg(salary)}(Instructor)$

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

⟹

dept_name	avg_sal
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

$_{dept\_name}\mathcal{G}_{avg(salary)\ as\ avg\_sal}(Instructor)$

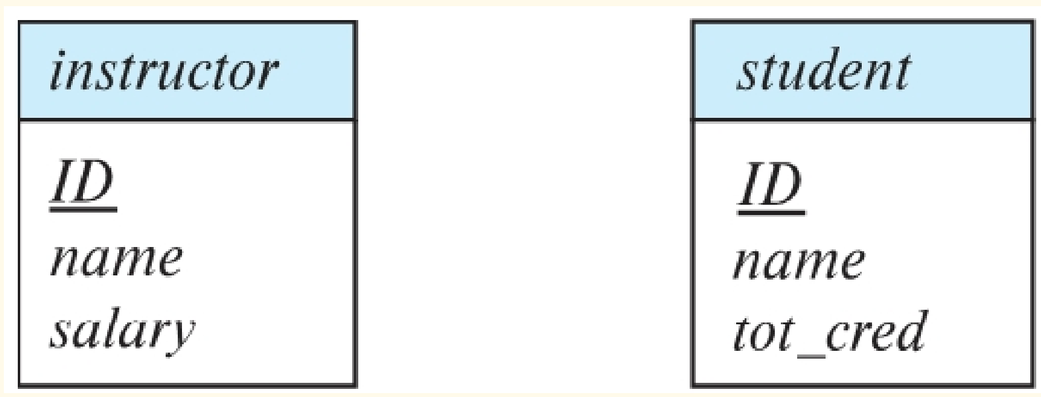
E-R模型

E-R
- 实体集
- 联系集
- 属性

Entity Set

实体集内共享相同属性

属性子集形成主码

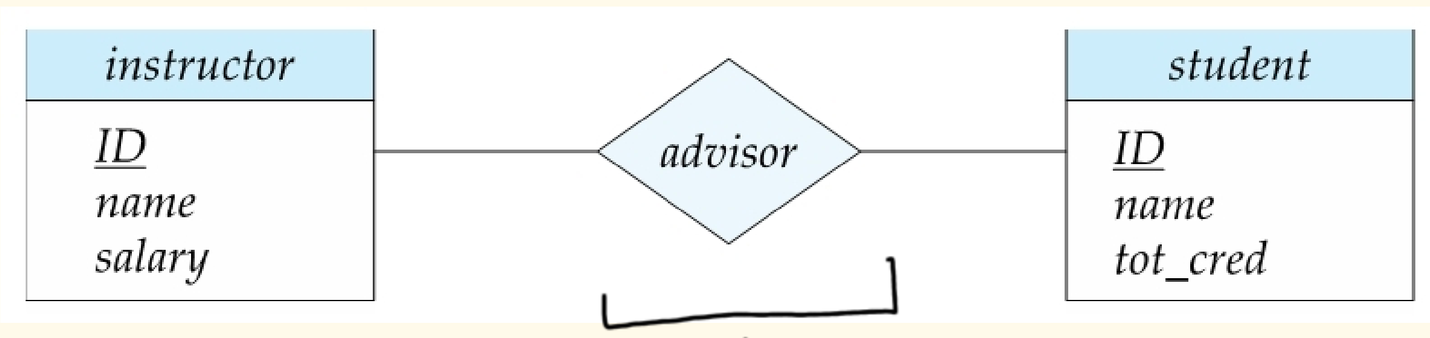


Relationship Set

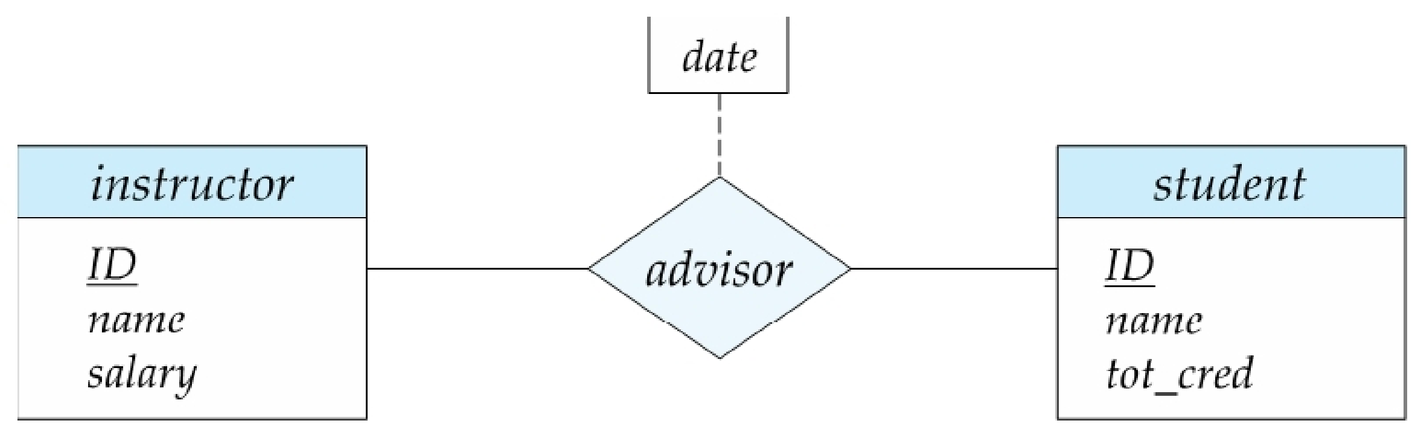
同类联系的集合

$$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

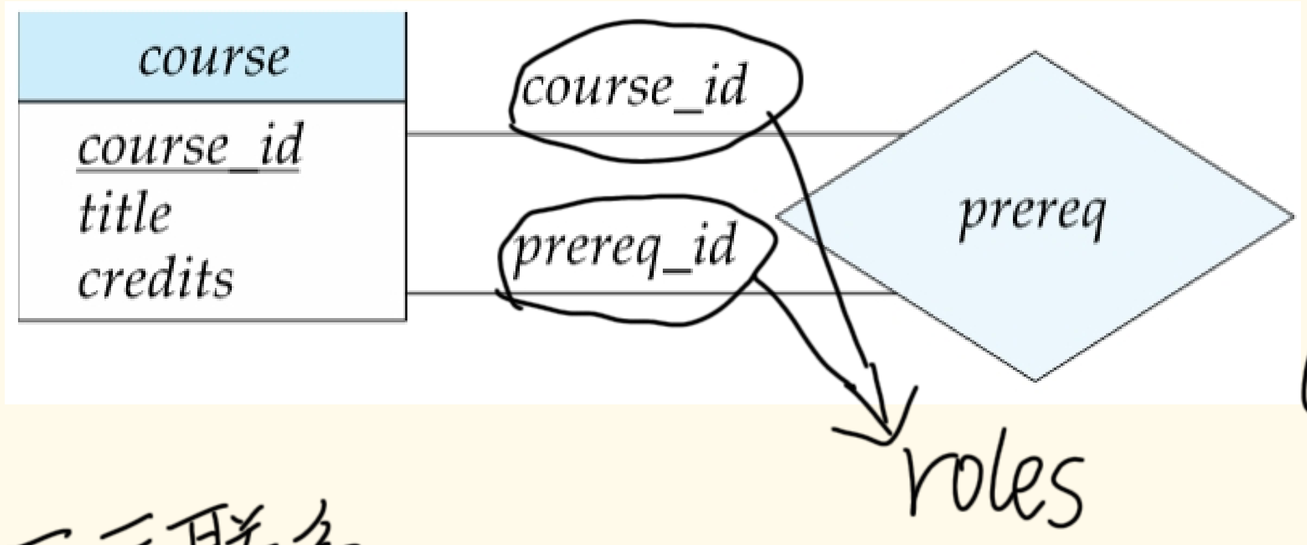
eg. (45565, 22222) ∈ advisor



联系集可以有属性

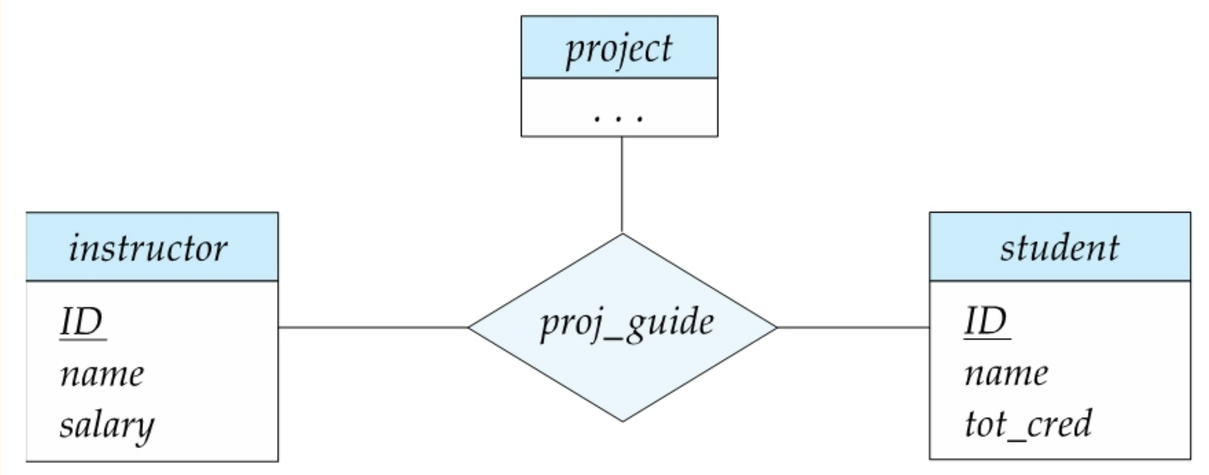


角色：在递归联系中必须出现



也是二元联系，
degree = 2
(课程先修课程是两类实体)

三元联系



实体

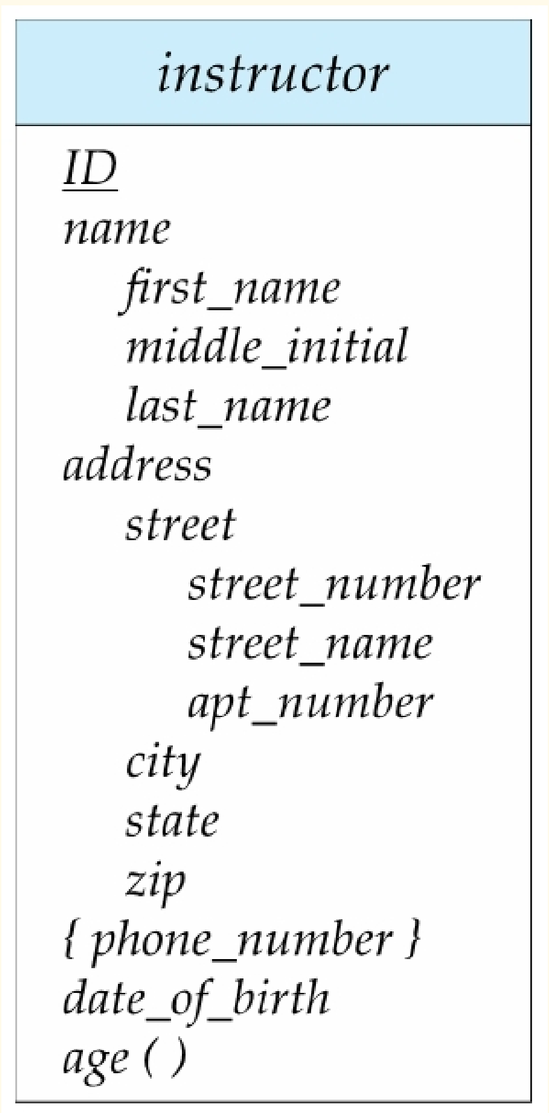
属性

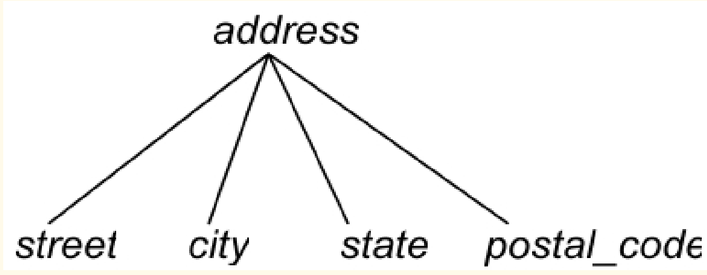
- 复合属性(有子属性) eg.
- 简单属性

- 多值属性(属性同时取多值)
- 单值属性

导出属性(由其他属性计算得到)

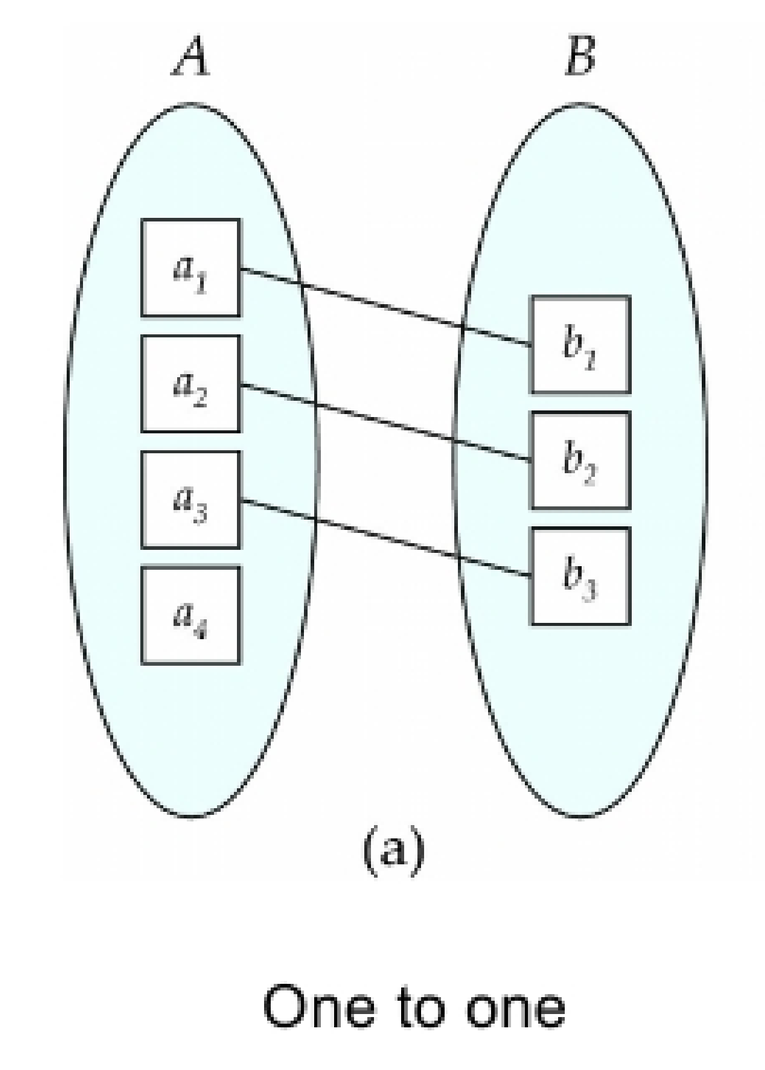
在E-R图表示：

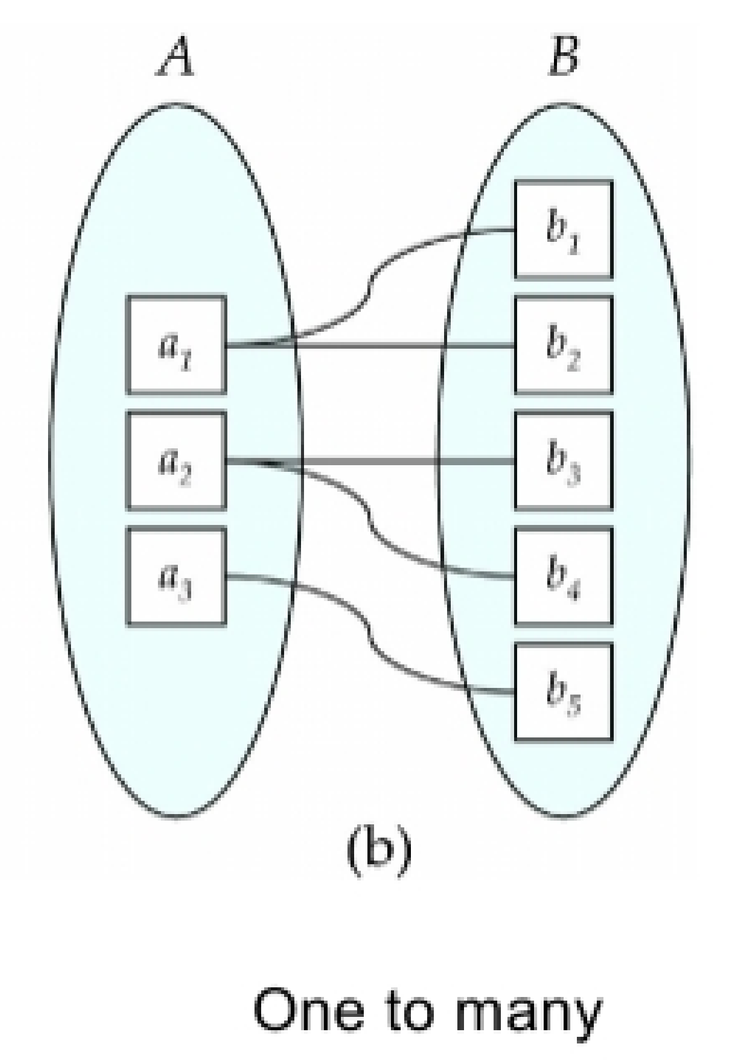


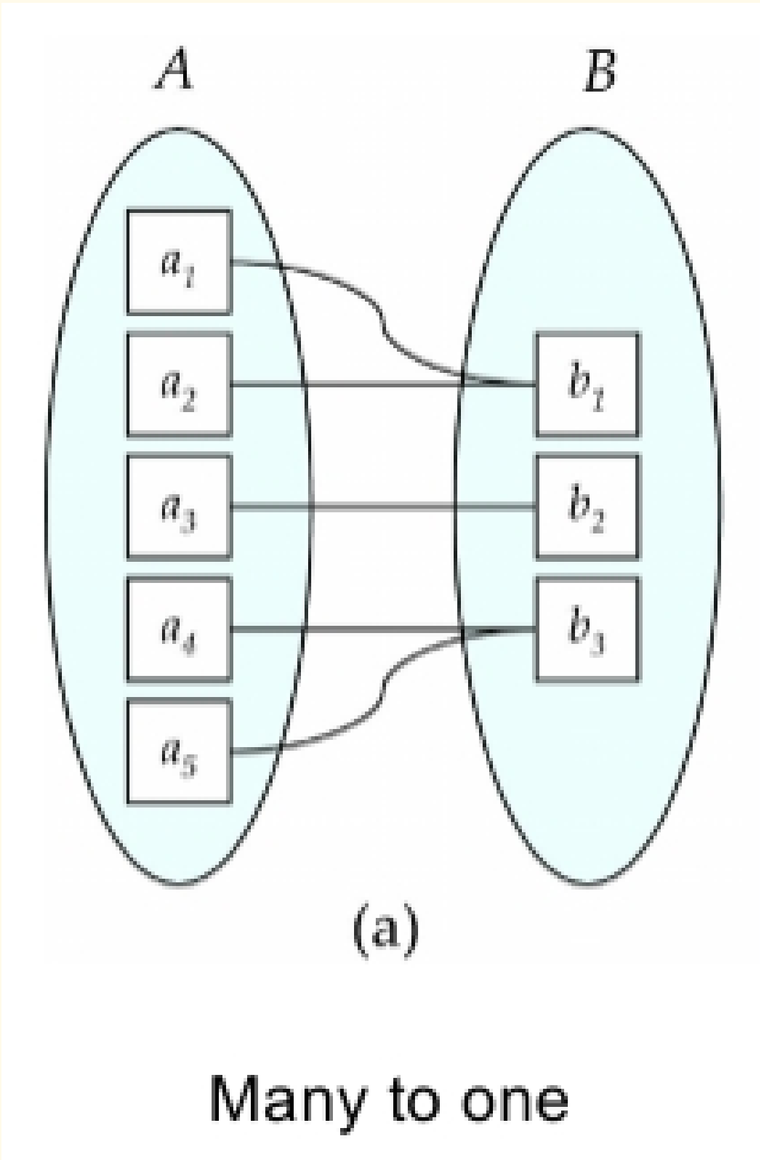


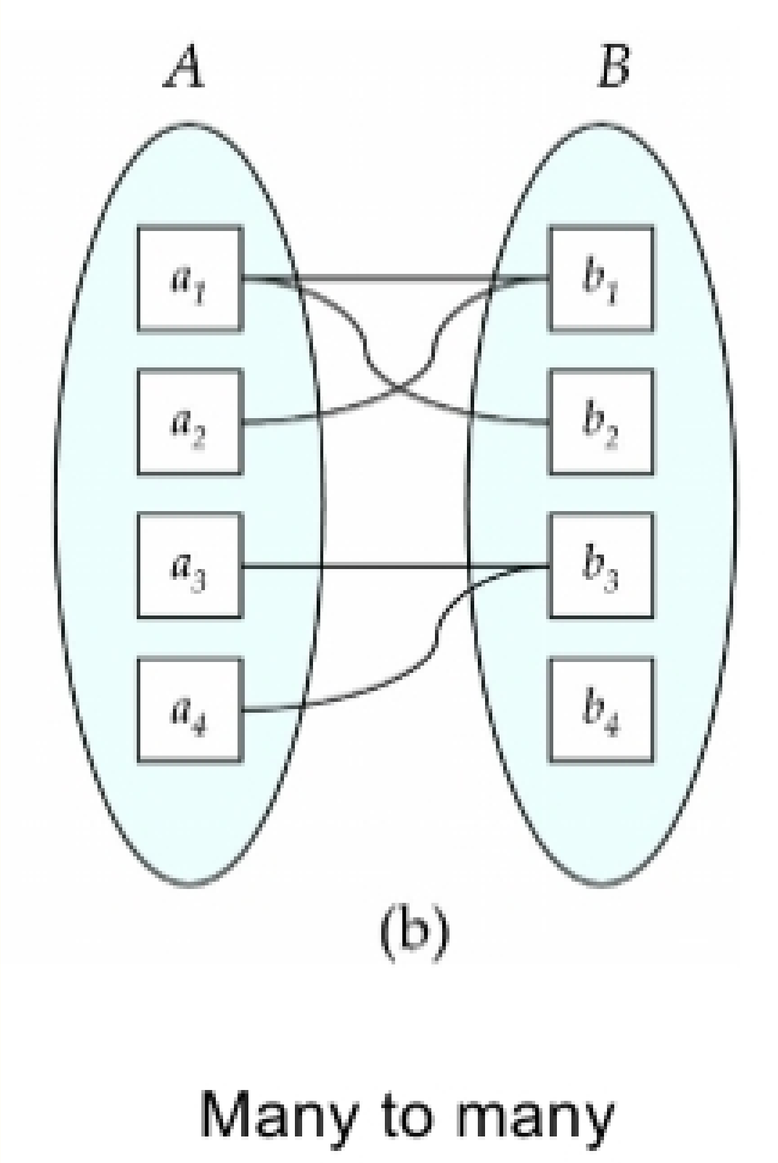
关系

映射基数

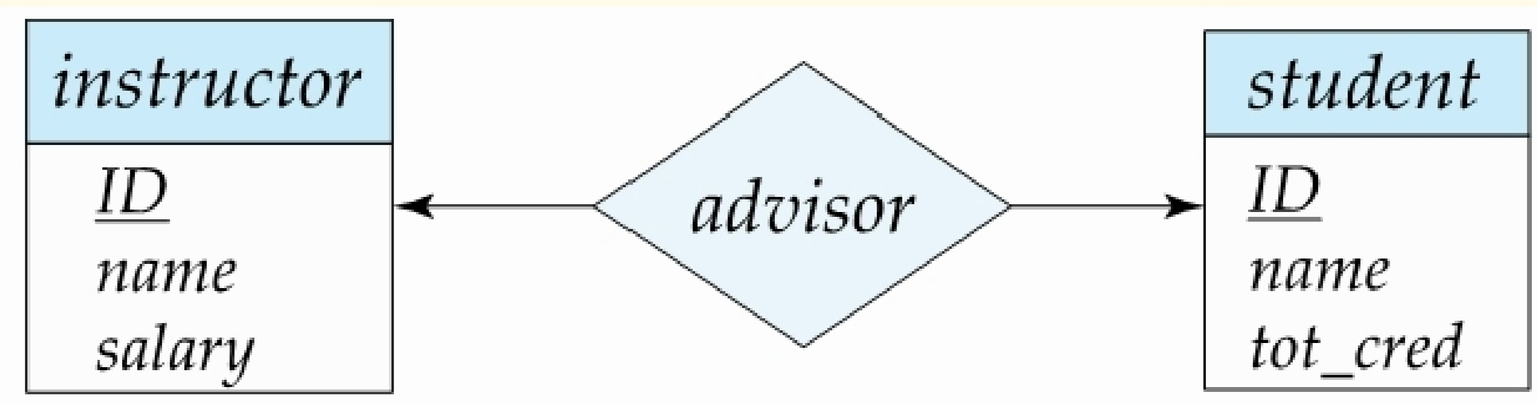




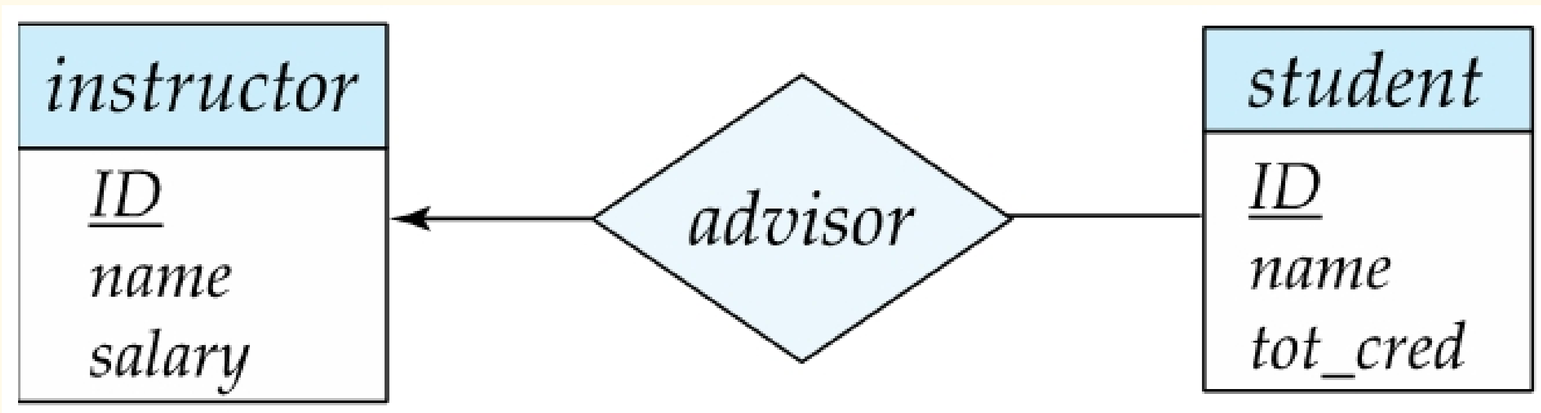




在E-R图中表示

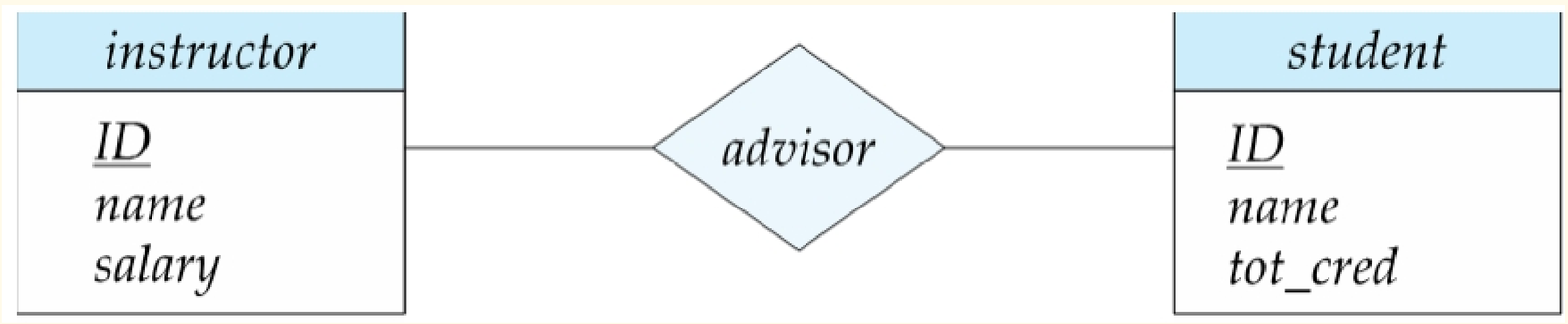


一 对 一



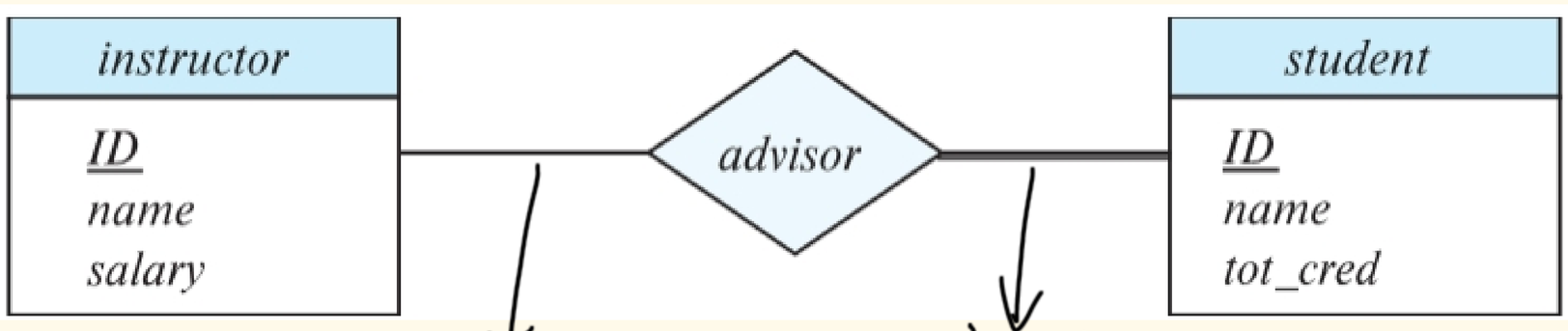
一 对 多

多 对 一



多 对 多

参与性



部份参与 全部参与

最小最大基数法表示：

一端 多端

0..1 1..*

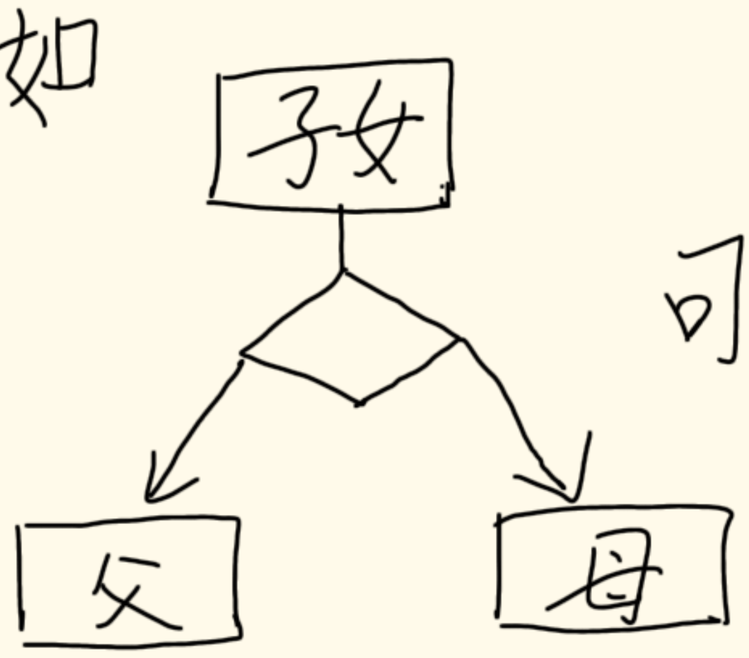
部份参与 全部参与

注：

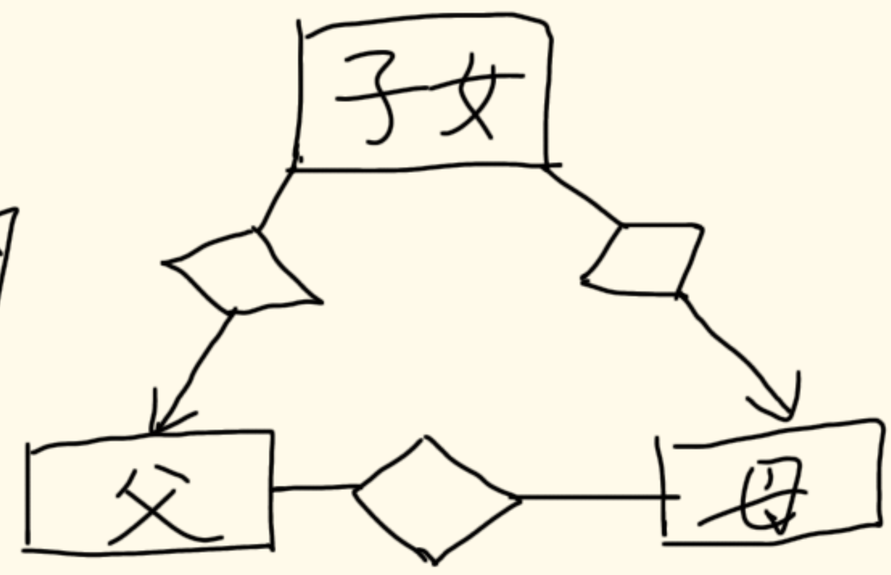
二元联系中一对实体最多一个联系！(但可有多个联系的属性)

多元联系中最多一个一端！

如



可修改为



但3个多端无法转化为三个二元联系

二元联系的主码

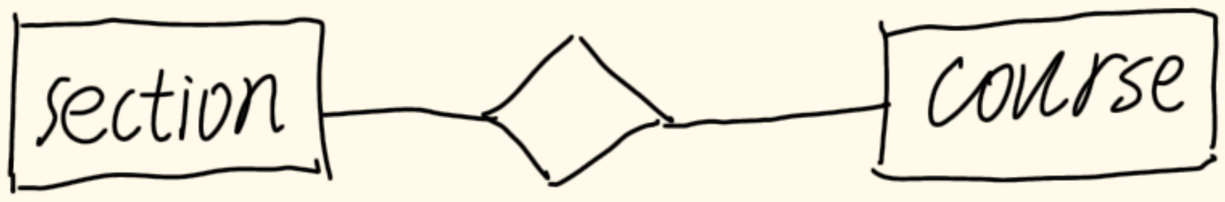
多对多：两方主码并集

一对多：多端主码

多对一：多端主码

一对一：两个一端中的一端主码(不取空)

弱实体集



两方都有course_id，已冗余

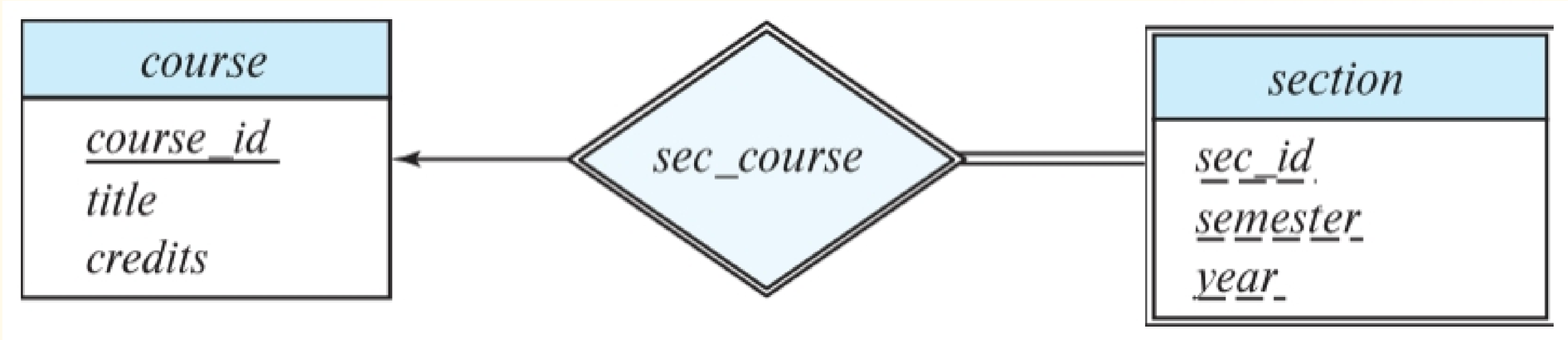
弱实体集：去掉冗余的course_id，就无法唯一标识(主码不够)

主码=部分码+从标识实体集借的属性

必须全部参与！

标识实体集：未去掉course_id

必须为一端！

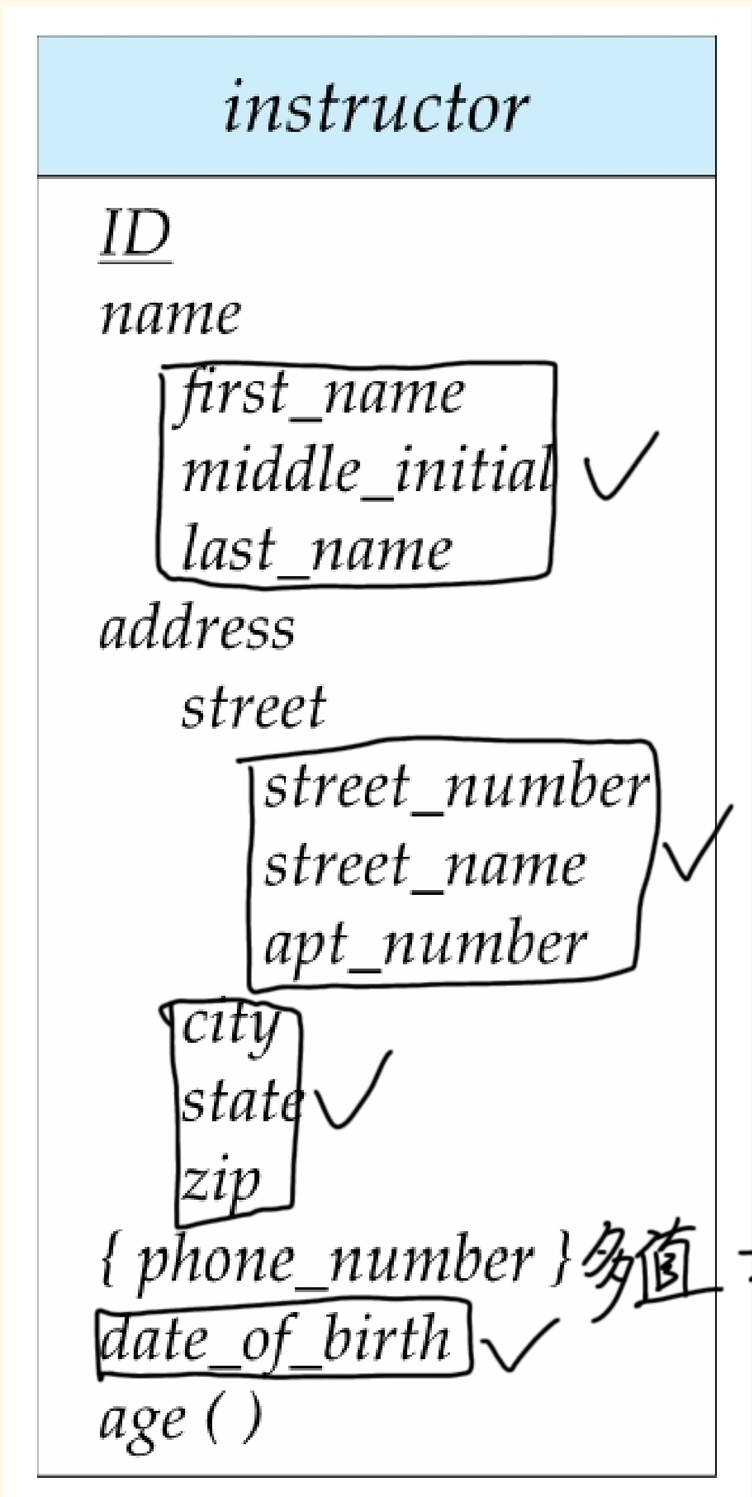


如何转换E-R为关系模式

①转换强实体集

复合属性只取子属性

忽略各多值属性(既可在实体,又可在联系)



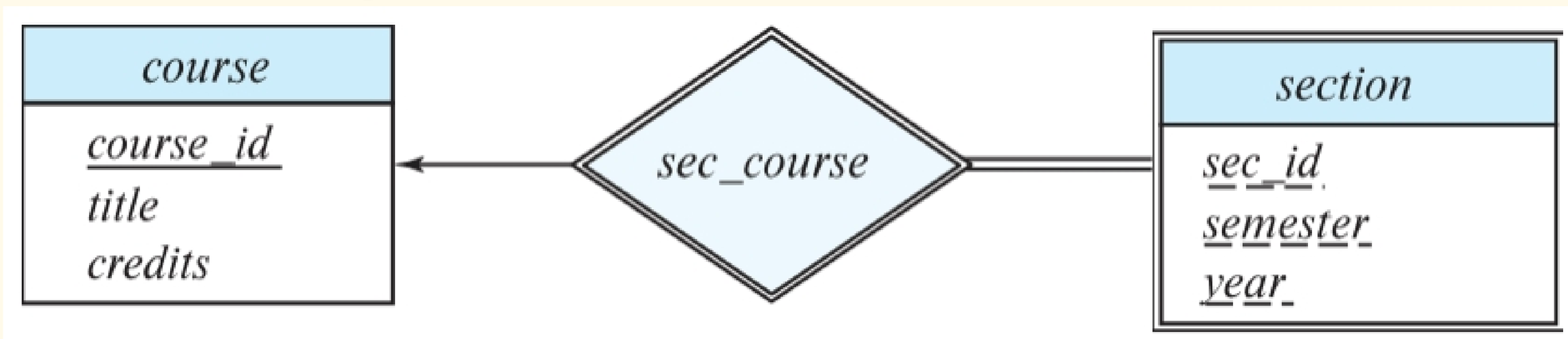
转换

instructor(ID, first_name, middle_initial, last_name,
street_number, street_name,
apt_number, city, state, zip_code, date_of_birth)

→ 多值属性单独用一个实体表示

inst_phone = (ID, phone-number)

②转换弱实体集(为弱集添加主码)

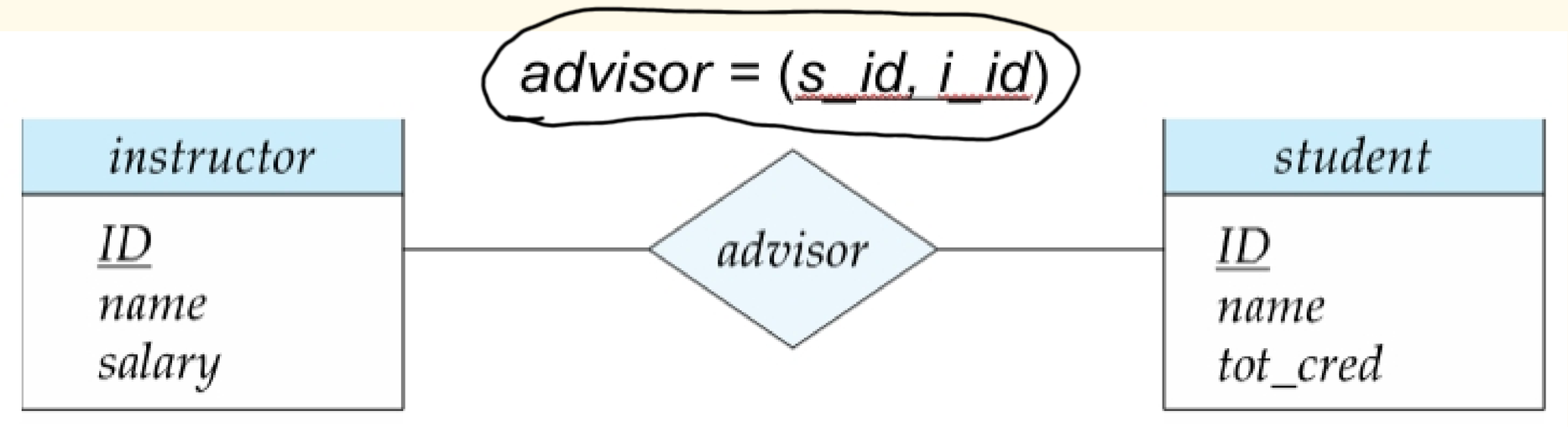


eg. section(course_id, sec_id, semester, year)

course(course_id, title, credits)

③转换关系集

1)多对多 ⇒ 联系+关系的多个属性



2)多对一、一对多 ⇒ 将一端的主码加到多端，作为外码

如：

转为 instructor(ID, name, salary)

student(ID, name, tot_cred, instr.ID)

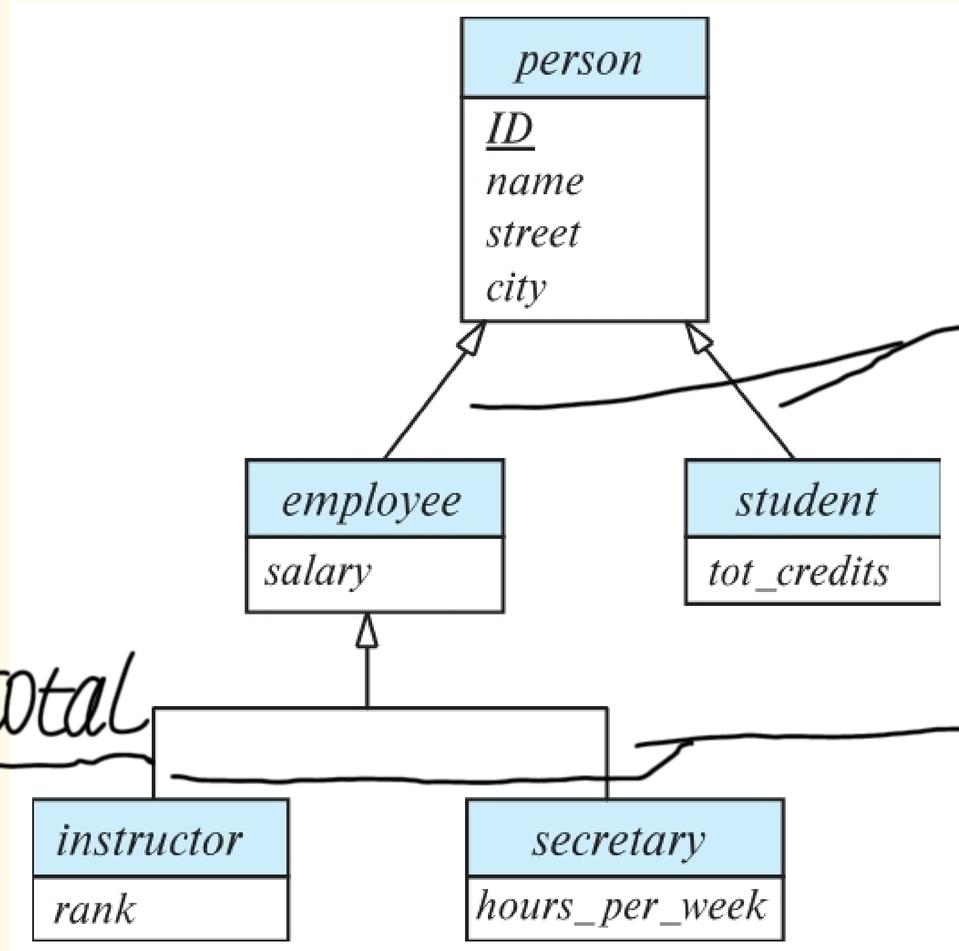
FK: instr.ID (外码约束)

3)一对一 ⇒ 选择全参与一端为"多端"

扩展E-R模型

特化、聚集

特化



相交约束（stu也可以同时为emp）

total

全特化

（默认为部分特化）

非相交约束

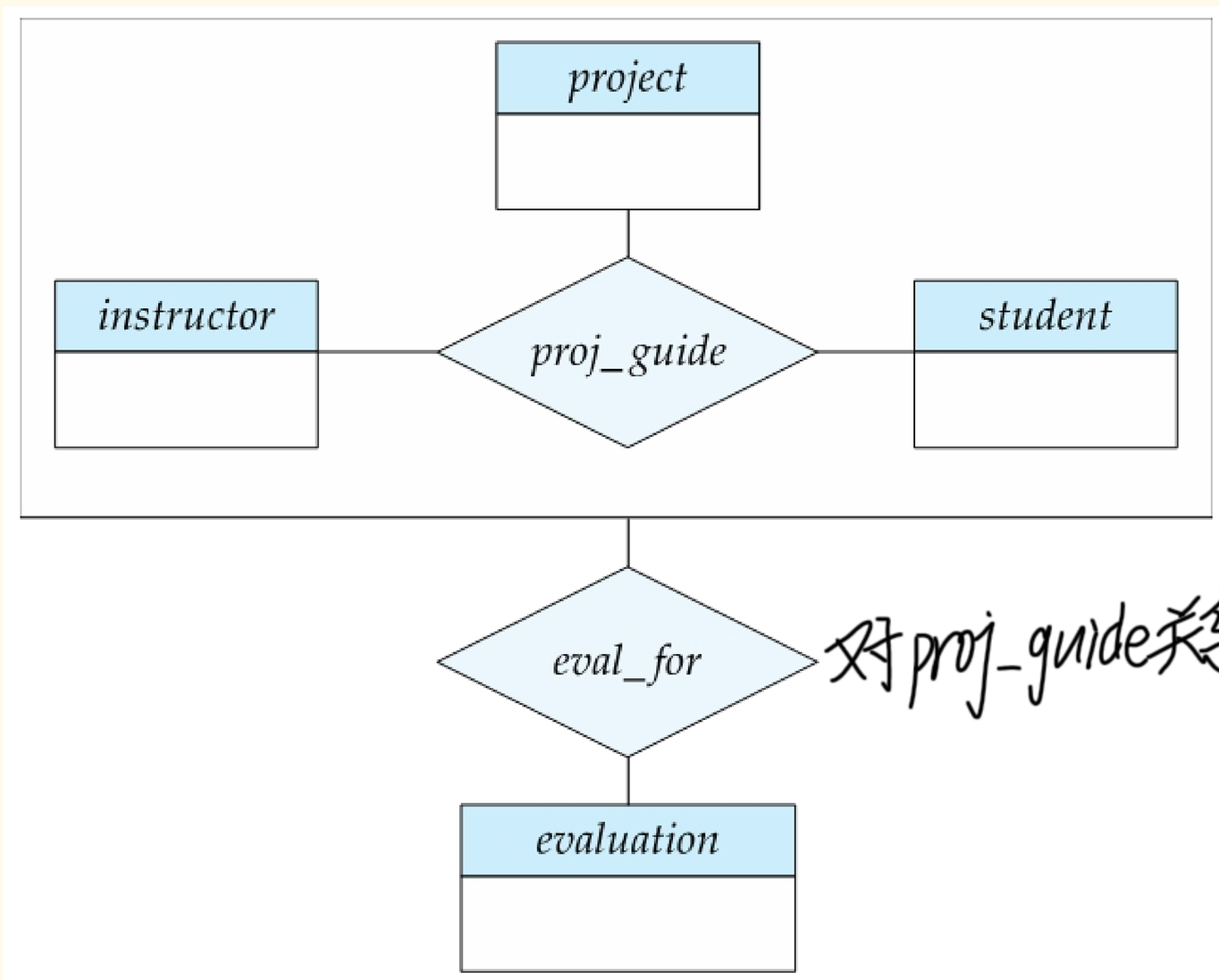
如何转化：下层实体继承上层主码

schema	attributes
person	ID, name, street, city
student	ID, tot_cred
employee	ID, salary

下层实体继承上层全部信息

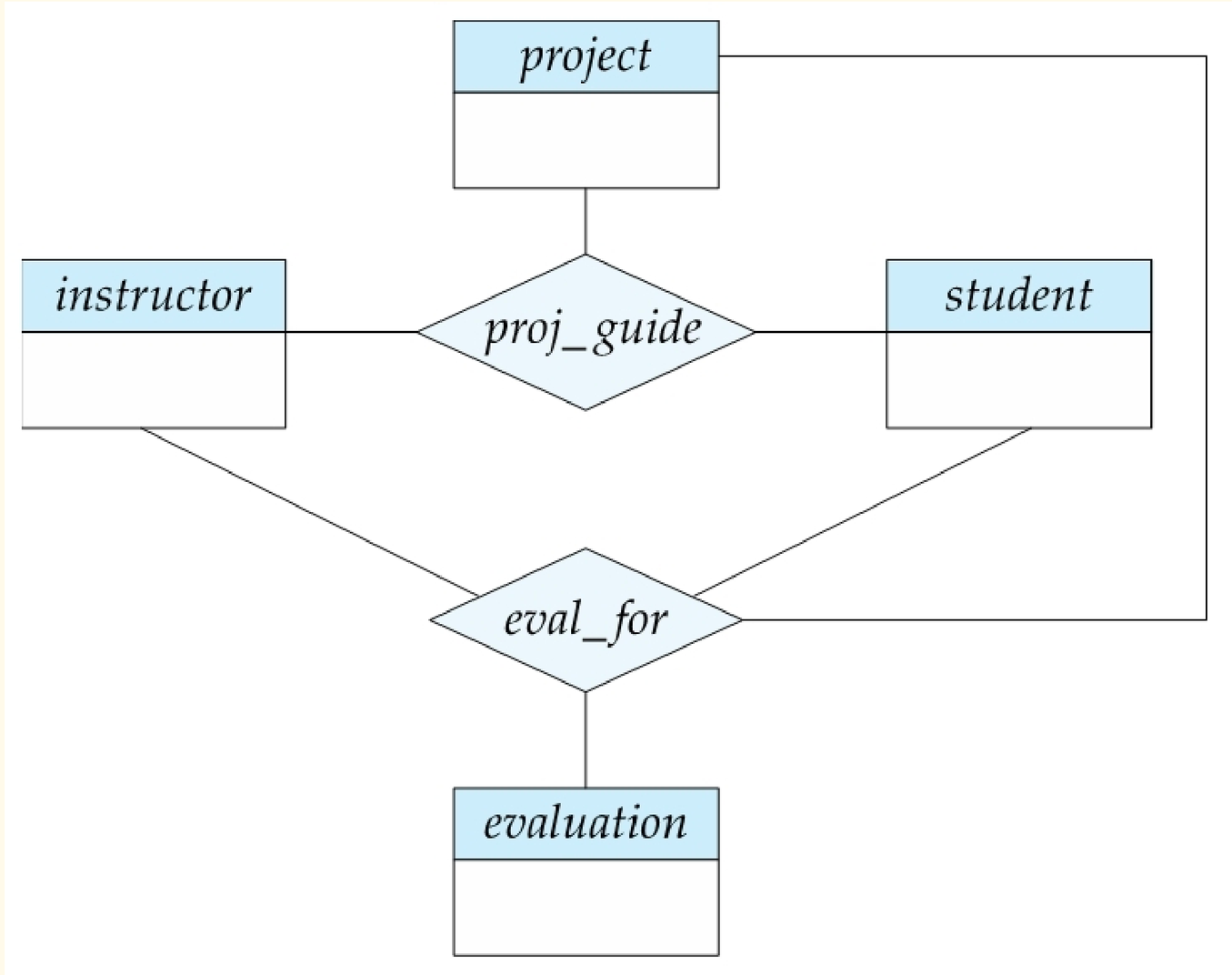
schema	attributes
person	ID, name, street, city
student	ID, name, street, city, tot_cred
employee	ID, name, street, city, salary

聚集

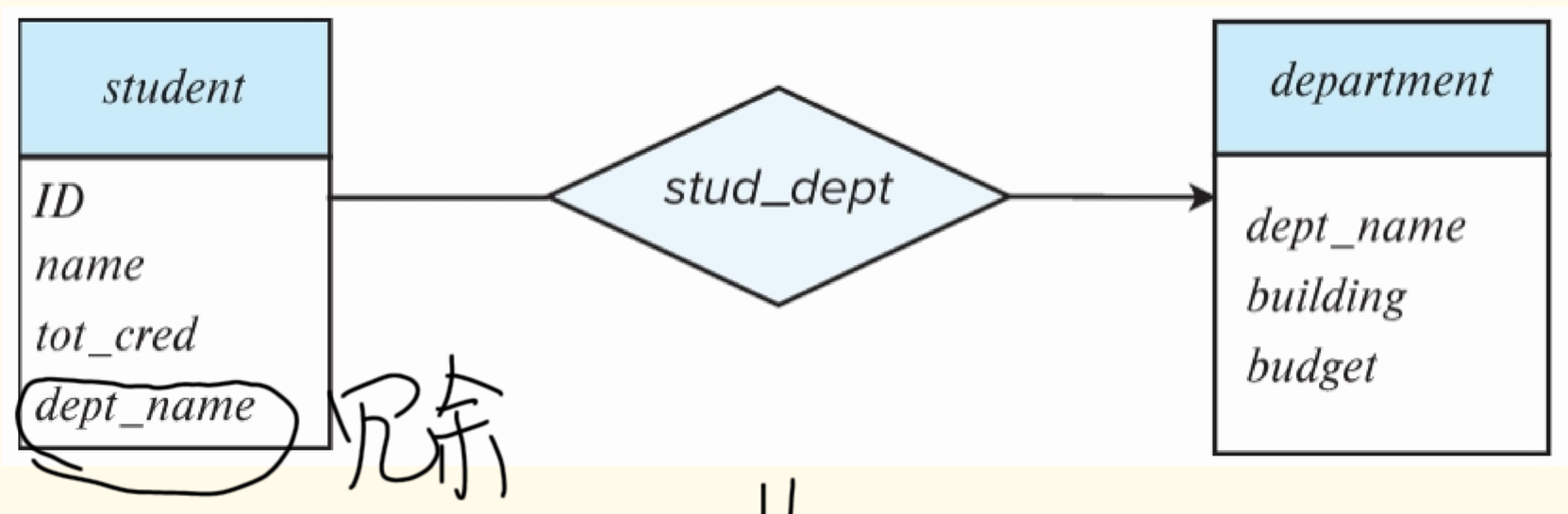


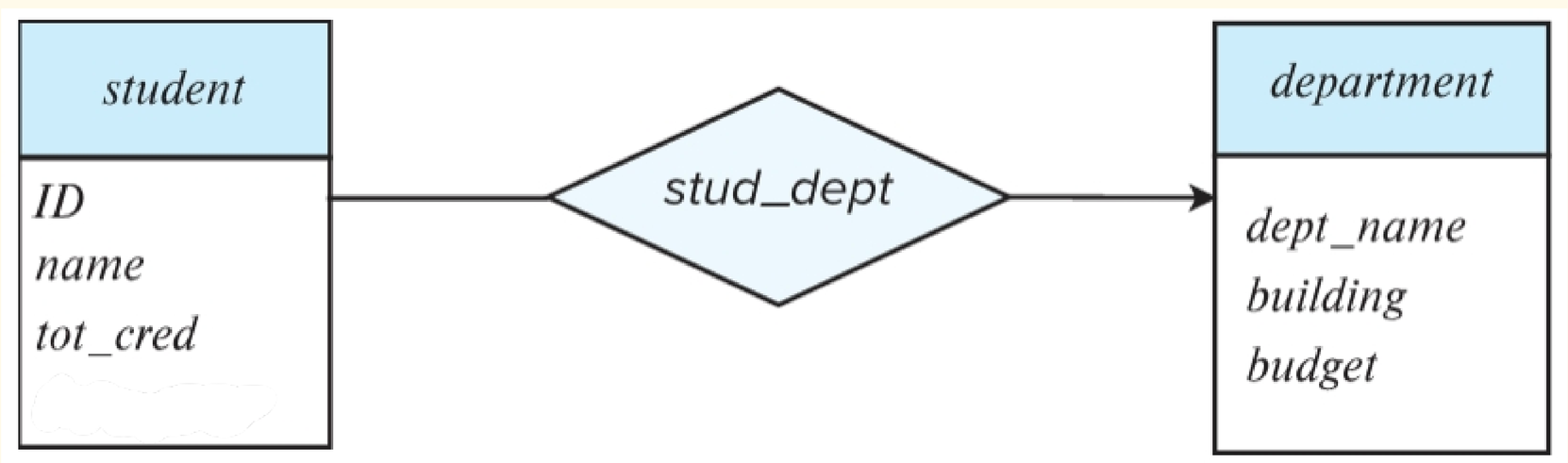
对proj_guide关系中的元组进行evaluation

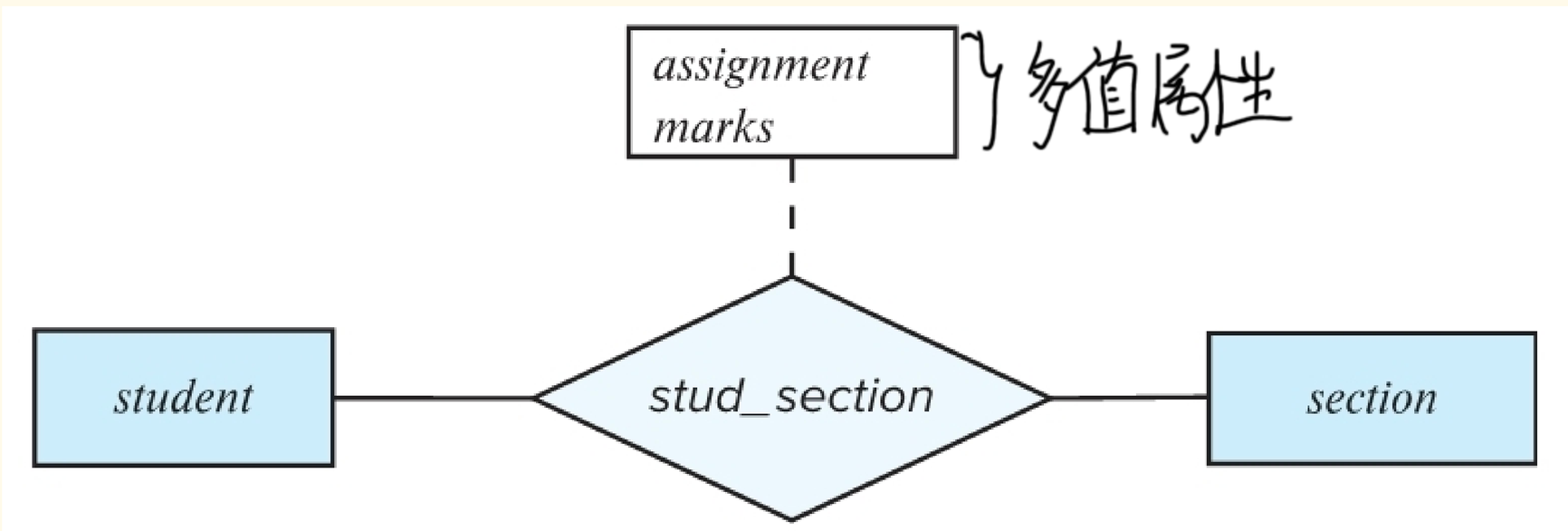
注意，并行时

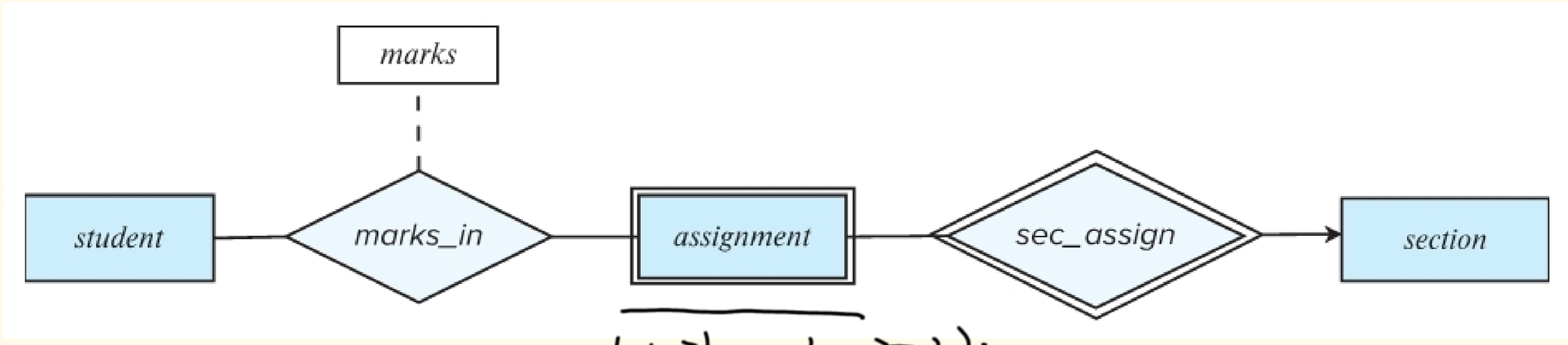


E-R图常见错误



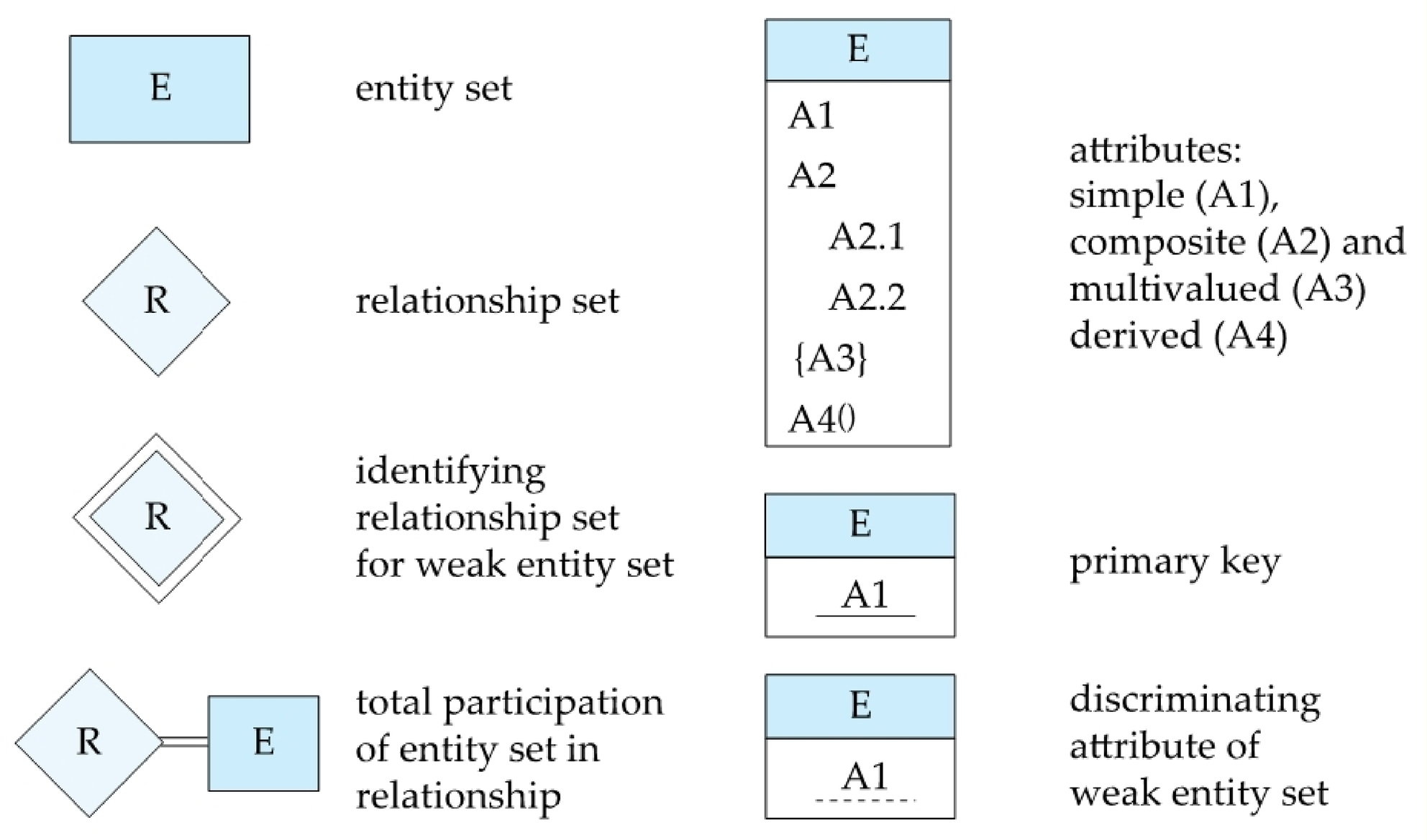


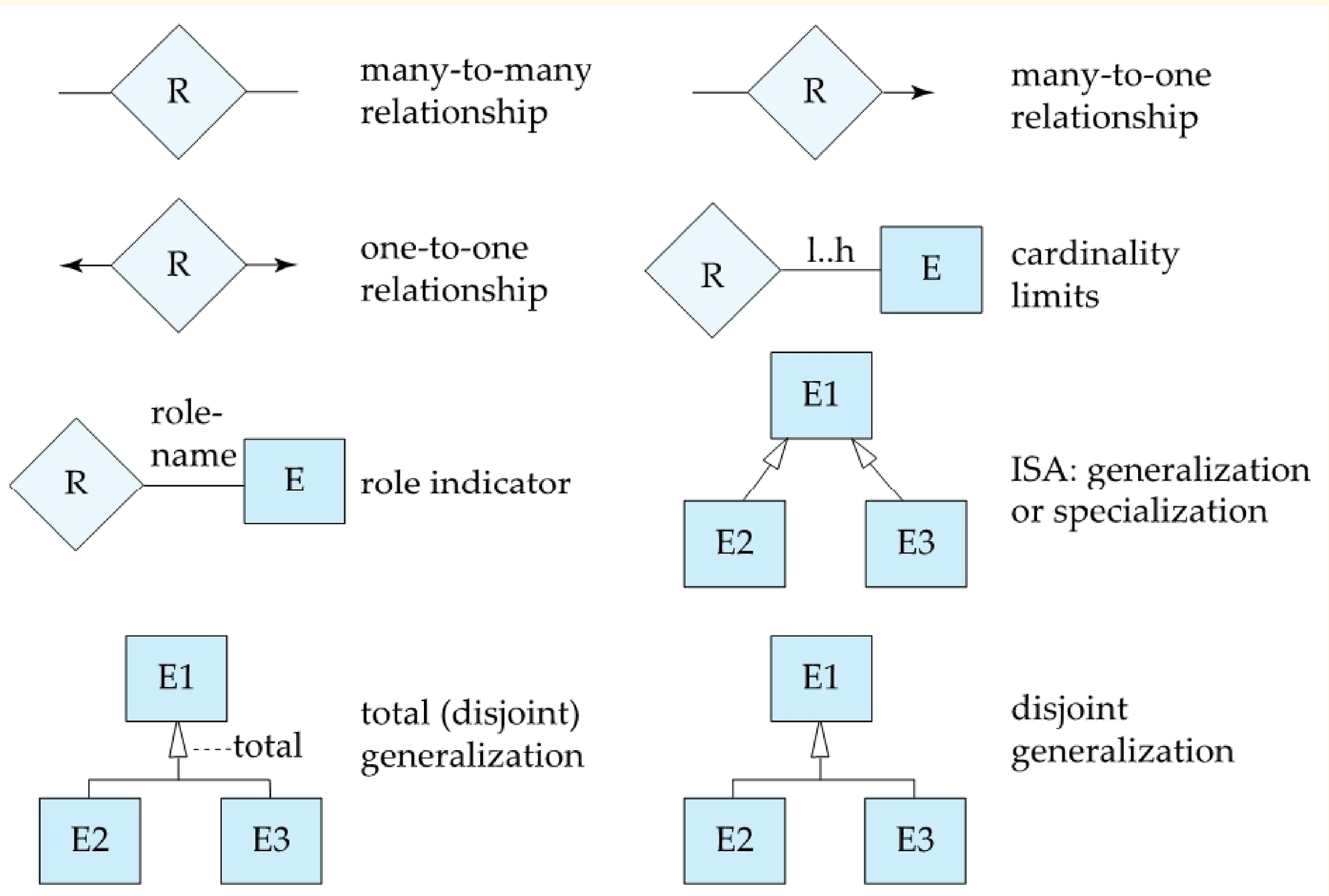




转为一个实体

总结：E-R图



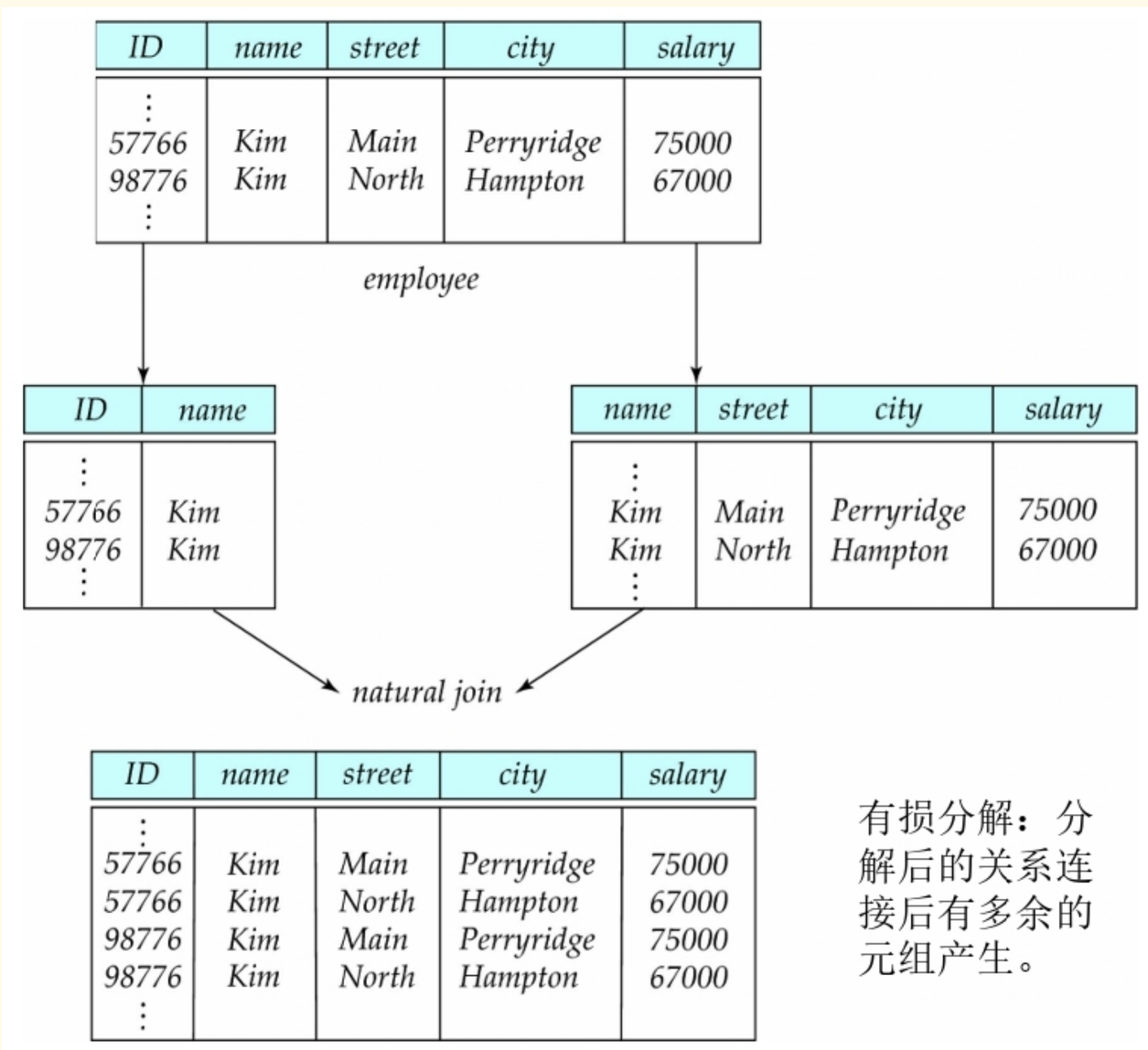


Ch7 关系数据库设计

分解

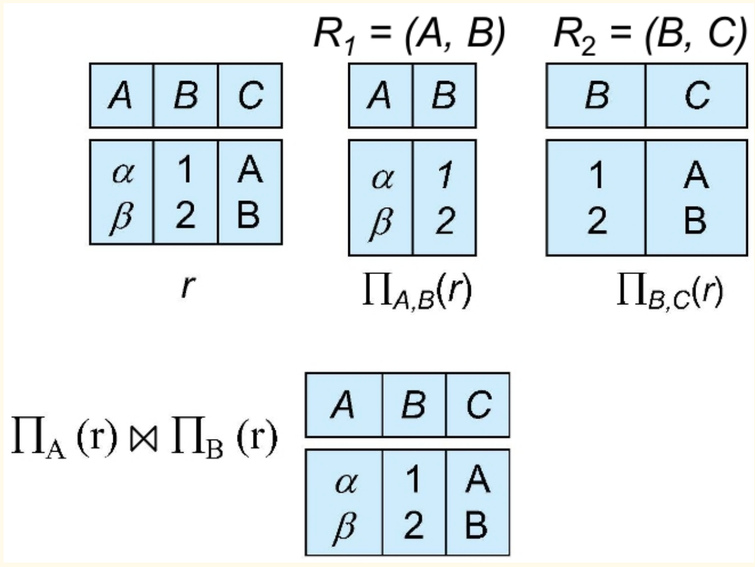
属性过多造成冗余

有损分解：$r \subseteq \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$



有损分解：分解后的关系连接后有多余的元组产生。

无损分解：$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$



函数依赖

Let R be a relation schema

$$\alpha \subseteq R \text{ and } \beta \subseteq R$$

The **functional dependency**

$$\alpha \rightarrow \beta$$

holds on R if and only if for any legal relations r(R), whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β. That is,

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

eg、 Consider r(A,B) with the following instance of r.

1	4
1	5
3	7

On this instance, $B \rightarrow A$ hold; $A \rightarrow B$ does **NOT** hold,

函数依赖集闭包

If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$

所有逻辑上的函数依赖集为闭包 F^+

平凡函数依赖

$\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

eg、 *ID, name* $\rightarrow$ *ID*

name $\rightarrow$ *name*

无损分解定理

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

$$\Leftrightarrow$$

$R_1 \cap R_2 \rightarrow R_1$ 或 R_2

eg、

$R = (A, B, C)$
$F = \{A \rightarrow B, B \rightarrow C)$

$R_1 = (A, B)$, $R_2 = (B, C)$

- Lossless decomposition:

  $R_1 \cap R_2 = \{B\}$ and $B \rightarrow \{B, C\}$

$R_1 = (A, B)$, $R_2 = (A, C)$

- Lossless decomposition:

  $R_1 \cap R_2 = \{A\}$ and $A \rightarrow \{A, B\}$

分解必须保持无损连接
尽量保持函数依赖(否则需用trigger)

BC范式

A relation schema R is in BCNF with respect to a set F of functional dependencies if for $\boxed{\text{all}}$ functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, <u>at least one</u> of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a superkey for R

eg.

Example schema that is ***not*** in BCNF:

in_dep (*ID, name, salary, dept_name, building, budget*)

because :

- *dept_name*→ *building, budget*
  - holds on *in_dep*
  - but
- *dept_name* is not a superkey

分解到BCNF，保证无损、无冗余，但不保证函数依赖

Let R be a schema R that is not in BCNF. Let $\alpha \rightarrow \beta$ be the FD that causes a violation of BCNF.

We decompose R into:

- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$

} 分解方式

分解到二目关系一定满足BCNF

eg、$R = (A, B)$

若 $F=\{\emptyset, A\to A, B\to B\}$，$R\in BCNF$

若 $F=\{A\to A, B\to B, A\to B, B\to A\}$，A或B一定为超码，$R\in BCNF$

第三范式

A relation schema R is in **third normal form (3NF)** if for all:

$$\alpha \to \beta \text{ in } F^+$$

at least one of the following holds:

- $\alpha \to \beta$ is trivial (i.e., $\beta \in \alpha$)
- α is a superkey for R
- Each attribute A in $\beta - \alpha$ is contained in a candidate key for R

放松了一条件
保持了函数依赖
但可能带来冗余

eg、 $R = (J, K, L)$
$F = \{JK \to L, L \to K\}$
And an instance table:

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

第二范式

A relation schema R is in ~~third~~ 2nd **normal form (2NF)** if for all:

$$\alpha \to \beta \text{ in } F^+$$

at least one of the following holds:

- $\alpha \to \beta$ is trivial (i.e., $\beta \in \alpha$)
- α is a superkey for R
- Each attribute A in $\beta - \alpha$ is contained in a candidate key for R
- *α不是任何候选码的真子集*

eg. $R = (A, B, C, D)$
$F = \{A \to B, C \to D\}$
Candidate Key $= (A, C)$

最高满足1NF

码

超码

K is a superkey for relation schema R if and only if $K \rightarrow R$

候选码

K is a candidate key for R if and only if

- $K \rightarrow R$, and
- for no $\alpha \subset K$, $\alpha \rightarrow R$

总结

1NF：数据项原子性

2NF：主键唯一标识性

3NF：非主属性间无函数依赖（有冗余）

BCNF：主属性与其他主属性子集无函数依赖（不保持函数依赖）

4NF：主属性只能有一个多值属性

逻辑蕴含

If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$

函数依赖集闭包：F^+

阿姆斯特朗公理系统

Reflexive rule: if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ 自反律

Augmentation rule: if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$ 增广律

Transitivity rule: if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ 传递律

Sound -- generate only functional dependencies that actually hold, and 正确的（推出来的都正确）

Complete -- generate all functional dependencies that hold. 完备的（蕴含的都推得出来）

eg、$R = (A, B, C, G, H, I)$

$F = \{ A \rightarrow B$
$A \rightarrow C$
$CG \rightarrow H$
$CG \rightarrow I$
$B \rightarrow H\}$

① $A \rightarrow B, B \rightarrow H \Rightarrow A \rightarrow H$

② $A \rightarrow C, CG \rightarrow H \Rightarrow AG \rightarrow H$

③ $CG \rightarrow H, CG \rightarrow I \Rightarrow CGI \rightarrow HI, CGCG \rightarrow CGI$
$\Rightarrow CGCG \rightarrow HI$
$\Rightarrow CG \rightarrow HI$

Union rule: If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds. 合并规则

Decomposition rule: If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds. 分解规则

Pseudotransitivity rule:If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds. 伪传递规则

求函数依赖集闭包

滚雪球算法（不常用）

$F^+ = F$
repeat
  for each functional dependency f in F^+
    apply reflexivity and augmentation rules on f
    add the resulting functional dependencies to F^+ > 自反、增广
  for each pair of functional dependencies f_1 and f_2 in F^+
    if f_1 and f_2 can be combined using transitivity
      then add the resulting functional dependency to F^+ > 传递
until F^+ does not change any further

属性集闭包

作用：判断 $\alpha \rightarrow \beta$ 是否成立

$\alpha \rightarrow \alpha^+, \beta \subseteq \alpha^+ \Rightarrow \alpha \rightarrow \beta$

判断K为候选码

$K \rightarrow R \Rightarrow K^+ = R$

$K' \subset K, K'^+ \neq R$

计算函数依赖集闭包

For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

方法：

result := α;
while (changes to *result*) **do**
  for each $\beta \rightarrow \gamma$ **in** *F* **do**
    begin
      if $\beta \subseteq$ *result* **then** *result* := *result* $\cup \gamma$
    end

eg.

$R = (A, B, C, G, H, I)$

$F = \{A \rightarrow B$
$A \rightarrow C$
$CG \rightarrow H$
$CG \rightarrow I$
$B \rightarrow H\}$

$(AG)^+$

1. $result = AG$
2. $result = ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
3. $result = ABCGH$ $(CG \rightarrow H$ and $CG \subseteq AGBC)$
4. $result = ABCGHI$ $(CG \rightarrow I$ and $CG \subseteq AGBCH)$

通过属性集闭包确定候选码

① $K^+ = R$

② 对任何 $K' \subset K$，都有 $K'^+ \subsetneq R$，则 K 为候选码

正则覆盖

多余属性

验证B多余：$F = \{AB \rightarrow C, A \rightarrow D, D \rightarrow C\}$

$F_t = \{A \rightarrow C, A \rightarrow D, D \rightarrow C\}$

若 $C \subseteq A^+_F$，则B多余

验证C多余：$F = \{AB \rightarrow CD, A \rightarrow C\}$

$F_t = \{AB \rightarrow D, A \rightarrow C\}$

若 $C \subseteq (AB)^+_{F_t}$，则C多余

正则覆盖

No functional dependency in F_c contains an extraneous attribute, and

Each left side of functional dependency in F_c is unique. That is, there are no two dependencies in F_c

- $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ such that
- $\alpha_1 = \alpha_2$

计算正则覆盖集 F_c

repeat

Use the union rule to replace any dependencies in F of the form

$\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$ } 对左部相同的，合并

Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β

/* Note: test for extraneous attributes done using F_c, not F*/

If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$ } 去除左右部无关属性

until (F_c not change

eg. $R = (A, B, C)$
$F = \{A \rightarrow BC$
$B \rightarrow C$
$A \rightarrow B$
$AB \rightarrow C\}$

① 合并 $A \rightarrow BC$ 和 $A \rightarrow B$

故 $F = \{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$

② $A \rightarrow BC$ 中，C 冗余

$F_t = \{A \rightarrow B, B \rightarrow C, AB \rightarrow C\}$

$C \subseteq A^+_{F_t} = ABC$，故 C 冗余

故 $F = \{A \rightarrow B, B \rightarrow C, AB \rightarrow C\}$

③ $AB \rightarrow C$ 中，B 冗余

$F_t = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

$C \subseteq A^+_F = ABC$，故 B 冗余

故 $F = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

④ $A \rightarrow B, B \rightarrow C \Rightarrow A \rightarrow C$ 传递冗余

故 $F = \{A \rightarrow B, B \rightarrow C\}$

函数依赖保留

A decomposition is **dependency preserving**, if

$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$

方法：$\alpha \rightarrow \beta$

验证 $\beta \subseteq \alpha^+_{(F_1 \cup \cdots \cup F_n)}$

result = α
repeat
  for each R_i in the decomposition
    result = *result* $\cup$ (*result* $\cap$ *Ri*)+ $\cap$ *Ri*
  until (*result* does not change)

If *result* contains all attributes in β, then the functional dependency $\alpha \rightarrow \beta$ is preserved.

分解至BCNF

例如：

Consider $R = (A, B, C, D, E)$, with $F = \{ A \rightarrow B, BC \rightarrow D\}$

- Decompose R into $R_1 = (A,B)$ and $R_2 = (A,C,D, E)$
- Neither of the dependencies in F contain only attributes from (A,C,D,E) so we might be mislead into thinking R_2 satisfies BCNF.
- In fact, dependency $AC \rightarrow D$ in F^+ shows R_2 is not in BCNF.

分解算法

result := {*R* };
done := false;
compute F^+;
while (not *done)* **do** } 子表未必满足BCNF，可能要继续分解
  if (there is a schema R_i in *result* that is not in BCNF)
    then begin
      let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that holds on R_i such that $\alpha \rightarrow R_i$ is not in F^+, and $\alpha \cap \beta = \varnothing$; } $\alpha \rightarrow \beta$违反BCNF 且$\alpha \cap \beta = \varnothing$
      result := (*result* – R_i) $\cup$ ($R_i - \beta$) $\cup$ (α, β); } 一分为二
      end
    else *done* := **true;**

eg、*class* (*course_id*, *title*, *dept_name*, *credits*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)

Functional dependencies:

- ① *course_id*→ *title*, *dept_name*, *credits*
- ② *building*, *room_number*→*capacity*
- ③ *course_id*, *sec_id*, *semester*, *year*→*building*, *room_number*, *time_slot_id*

A candidate key {*course_id*, *sec_id*, *semester*, *year*}.

BCNF Decomposition:

- *course_id*→ *title*, *dept_name*, *credits* holds
  - but *course_id* is not a superkey.
- We replace *class* by:
  - *course*(*course_id*, *title*, *dept_name*, *credits*)
  - *class-1* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)

只出现在→左边属性L

①在L中找到属性(组合)使其闭包为自己，与L，R两边均出现的属性组合

②在L中找不到，则找闭包为全部属性，其为唯一候选码

约束②不满足BCNF ⇒ 继续分解

course is in BCNF

- How do we know this?

building, *room_number*→*capacity* holds on *class-1*

- but {*building*, *room_number*} is not a superkey for *class-1*.
- We replace *class-1* by:
  - *classroom* (*building*, *room_number*, *capacity*)
  - *section* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)

classroom and *section* are in BCNF.

分解至3NF

Let F_c be a canonical cover for F; 求正则覆盖集
i := 0;
for each functional dependency $\alpha \rightarrow \beta$ in F_c **do**
  if none of the schemas R_j, $1 \le j \le i$ contains $\alpha\ \beta$
    then begin
      $i := i + 1$;
      $R_i := \alpha\ \beta$
    end

FC中，一箭头分为一小表
$\alpha \rightarrow \beta \Rightarrow R_i = (\alpha, \beta)$

if none of the schemas R_j, $1 \le j \le i$ contains a candidate key for R
  then begin
    $i := i + 1$;
    R_i := any candidate key for R;
  end
/* Optionally, remove redundant relations */

补码

repeat
if any schema R_j is contained in another schema R_k
  then /* delete R_j */
    $R_j = R$;;
    $i=i-1$;

删除被包含的冗余小表

return $(R_1, R_2, ..., R_i)$ 结果一定满足3NF

eg、Relation schema:

cust_banker_branch = (*customer_id, employee_id*, *branch_name*, *type*)

The functional dependencies for this relation schema are:

- *customer_id*, *employee_id* → *branch_name*, *type*
- *employee_id* → *branch_name*
- *customer_id*, *branch_name* → *employee_id*

① 计算函数依赖的正则覆盖集

合并相同左项

去除无关属性

We first compute a canonical cover

- *branch_name* is extraneous in the r.h.s. of the 1st dependency
- No other attribute is extraneous, so we get F_C =

  customer_id, employee_id → *type*
  employee_id → *branch_name*
  customer_id, branch_name → *employee_id*

②补码

The **for** loop generates following 3NF schema:

(*customer_id, employee_id, type*)

(*employee_id, branch_name*)

(*customer_id, branch_name, employee_id*)

- Observe that (*customer_id, employee_id, type*) contains a candidate key of the original schema, so no further relation schema needs be added

③删除冗余表
(employee_id, branch_name)已被包含在
(customer, branch_name, employee_id)

At end of for loop, detect and delete schemas, such as (*employee_id, branch_name*), which are subsets of other schemas

- result will not depend on the order in which FDs are considered

The resultant simplified 3NF schema is:

(*customer_id, employee_id, type*)

(*customer_id, branch_name, employee_id*)

多值依赖(*,难)

Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The **multivalued dependency**

$$\alpha \rightarrow\rightarrow \beta$$

holds on R if in any legal relation $r(R)$, for all pairs for tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that:

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R - \beta] = t_2[R - \beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R - \beta] = t_1[R - \beta]$$

eg.完全搭配

Tabular representation of $\alpha \rightarrow\rightarrow \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

函数依赖 $\subsetneq$ 多值依赖

If $\alpha \rightarrow \beta$, then $\alpha \rightarrow\rightarrow \beta$

第4范式

A relation schema R is in **4NF** with respect to a set D of functional and multivalued dependencies if for all multivalued dependencies in D^+ of the form $\alpha \rightarrow\rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:

- $\alpha \rightarrow\rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
- α is a superkey for schema R

If a relation is in 4NF it is in BCNF

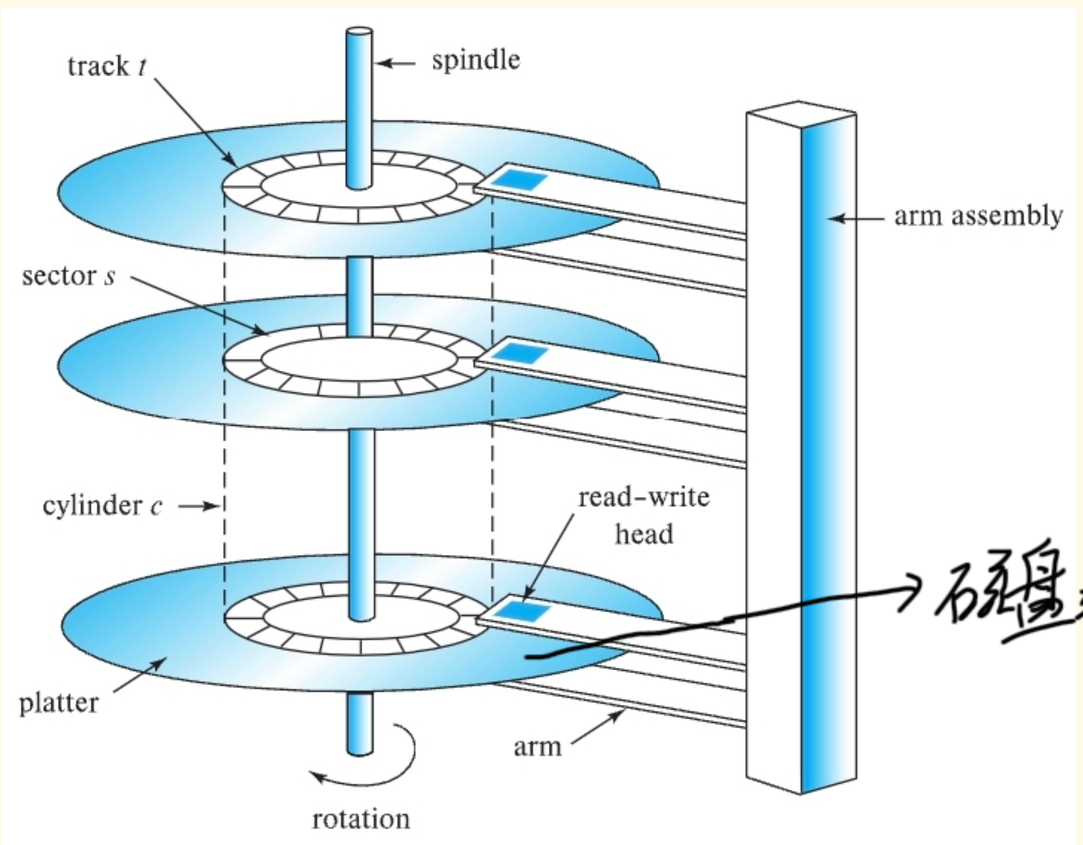
物理存储(*,考)

- 易失存储：RAM
- 非易失存储：ROM

分级
- 一级：主存、cache
- 二级：闪存、磁盘
- 三级：移动硬盘、光盘、磁带

存储接口
- Disk接口标准：SATA、SAS、NVMe
- SAN (Storage Area Networks)
- NAS (Network Attached Storage)

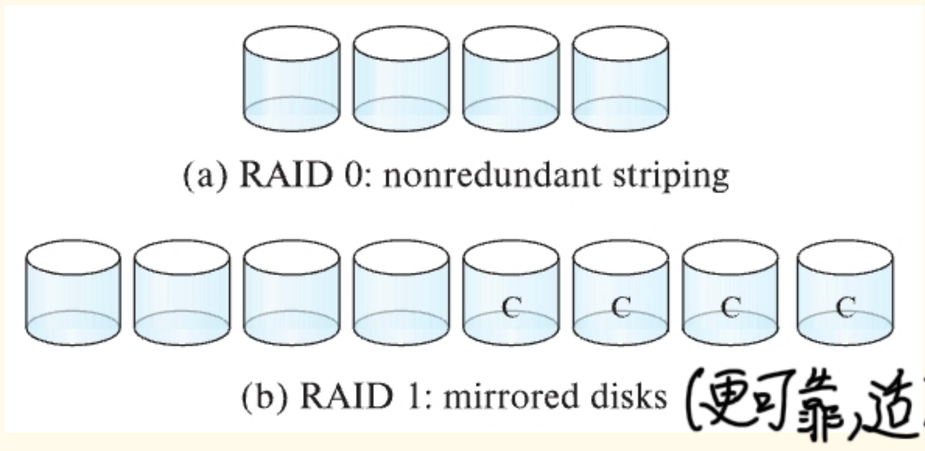
磁盘



磁盘块(最小单元)

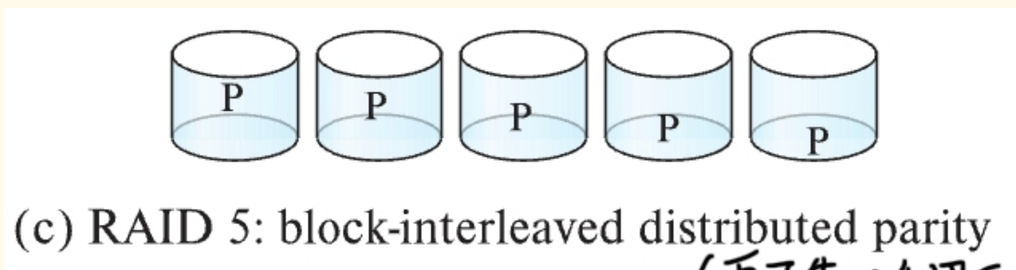
寻道时间、传输时间
连续存放、随机存放
每秒IO操作(200 IOPS)
平均失效时间(3~5年MTTF)
↓
统计数据，非个体指标

RAID：冗余多个盘



(a) RAID 0: nonredundant striping

(b) RAID 1: mirrored disks (更可靠，适用于写得多)



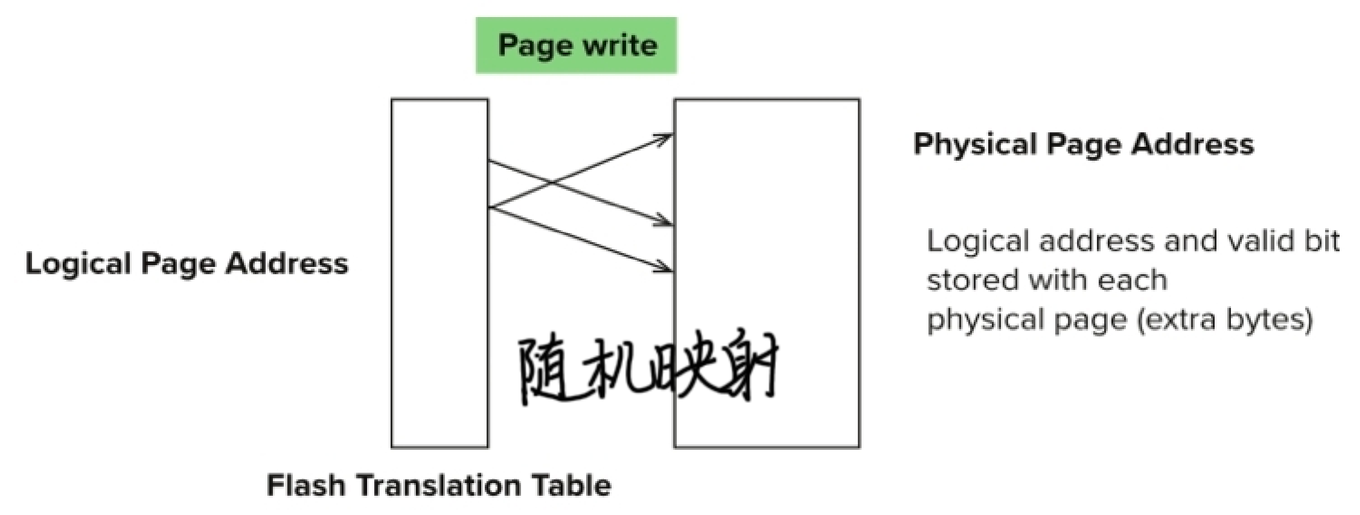
(c) RAID 5: block-interleaved distributed parity

(更可靠，适用于读得多)

闪存

无法覆盖写入，需先擦除

擦除损伤大 ⇒ 擦除均衡 ⇓



文件组织 (*,不考)

1个文件 → 1种记录

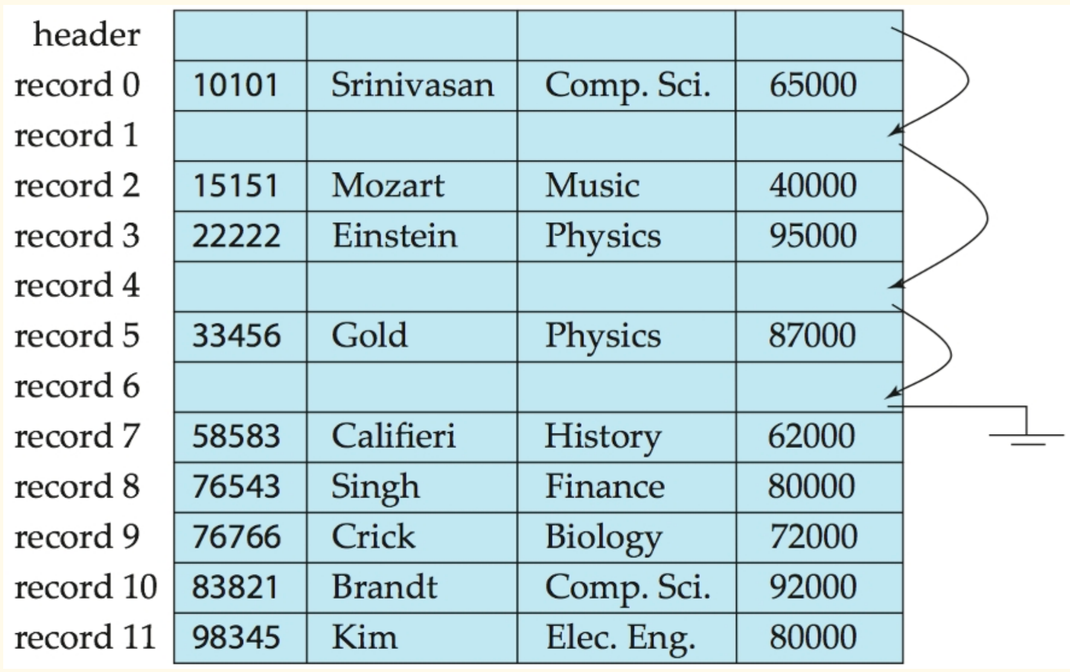
物理上组织方式

① 连续存放

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
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

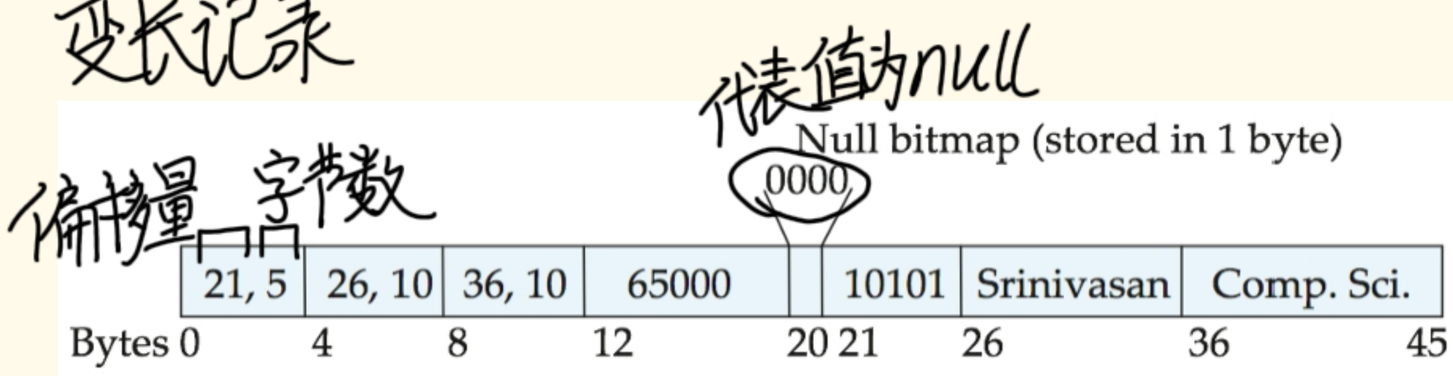
维护(增删)成本高

② 空闲空间链

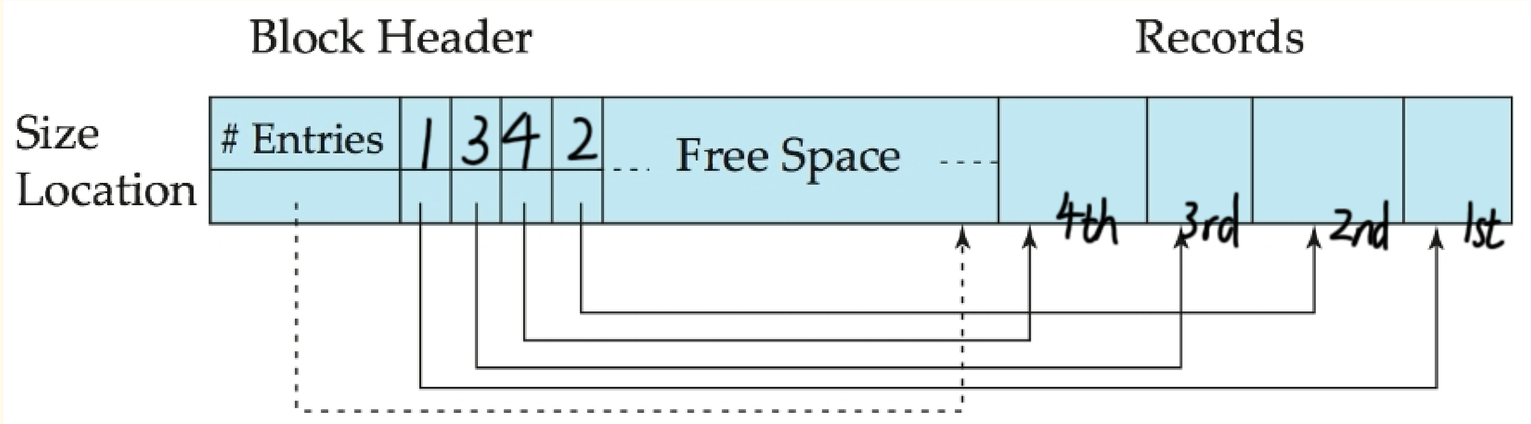


③ 分槽页结构

变长记录



分槽页



逻辑上组织方式

① 堆

随机存放

② 顺序

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	
32222	Verdi	Music	48000	

顺序指针

搜索码(排序依据,只能有一个)

③ 多表聚簇

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

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

Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

均为 Comp. Sci. 的教师

④ 哈希

数据字典：储存元数据

Buffer管理 (*,考)

最近最少使用原则 LRU (常用)
最近最多使用原则 MRU
立即丢弃
固定块 (正在写入的块)
强制输出 (日志、恢复)

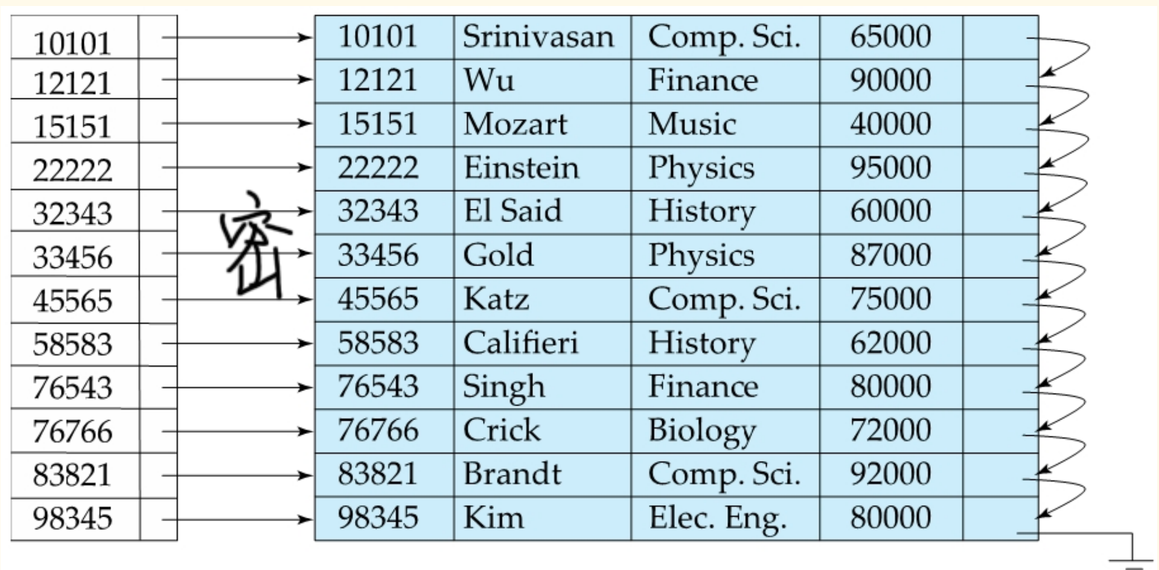
索引 (*,考)

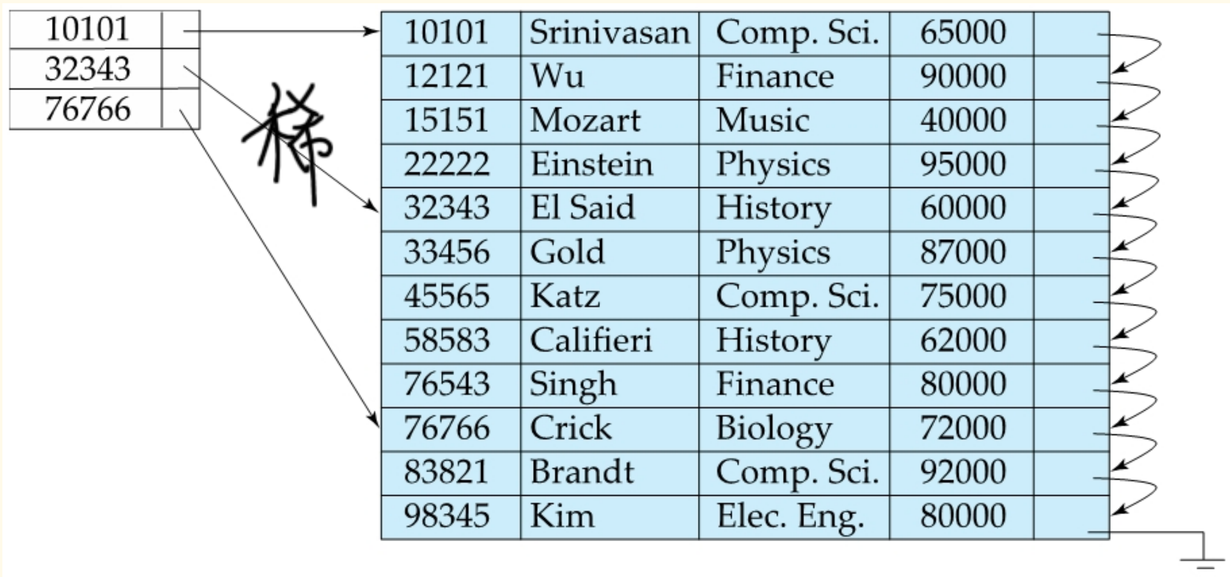
索引文件：索引项集合

search-key	pointer

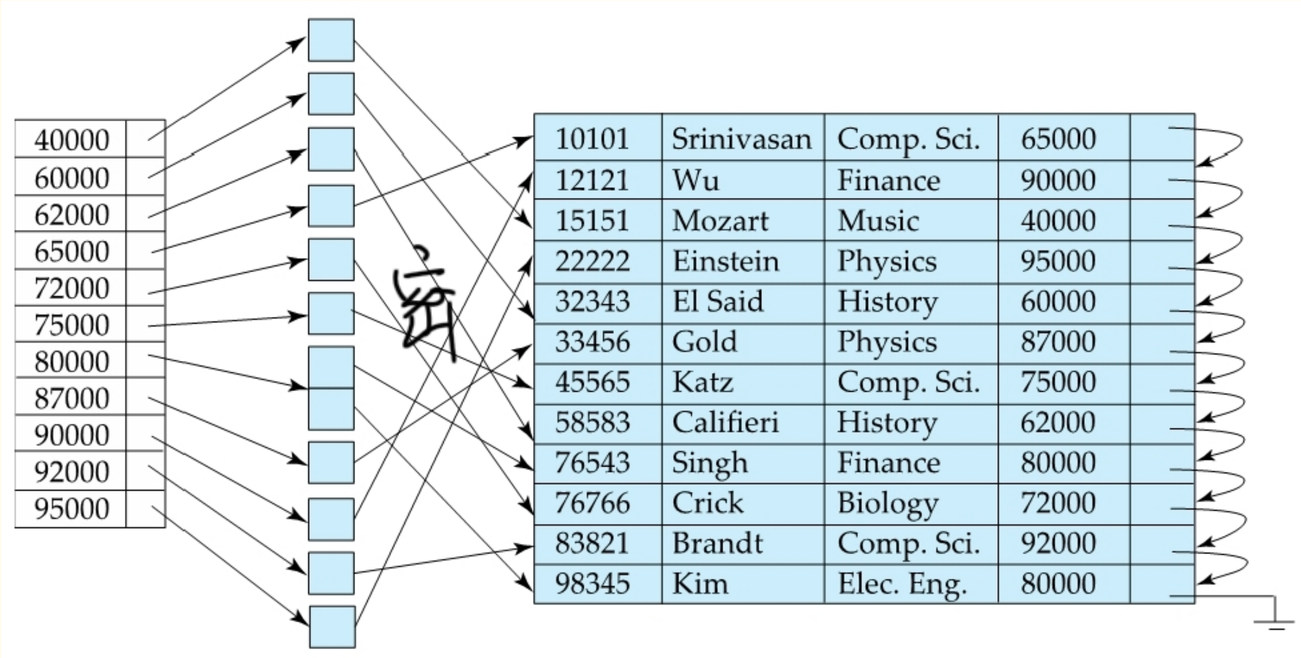
索引分类：顺序索引
哈希索引

顺序索引：① 聚集索引
一张表最多只有一个
小有序管大有序
可稀疏可稠密

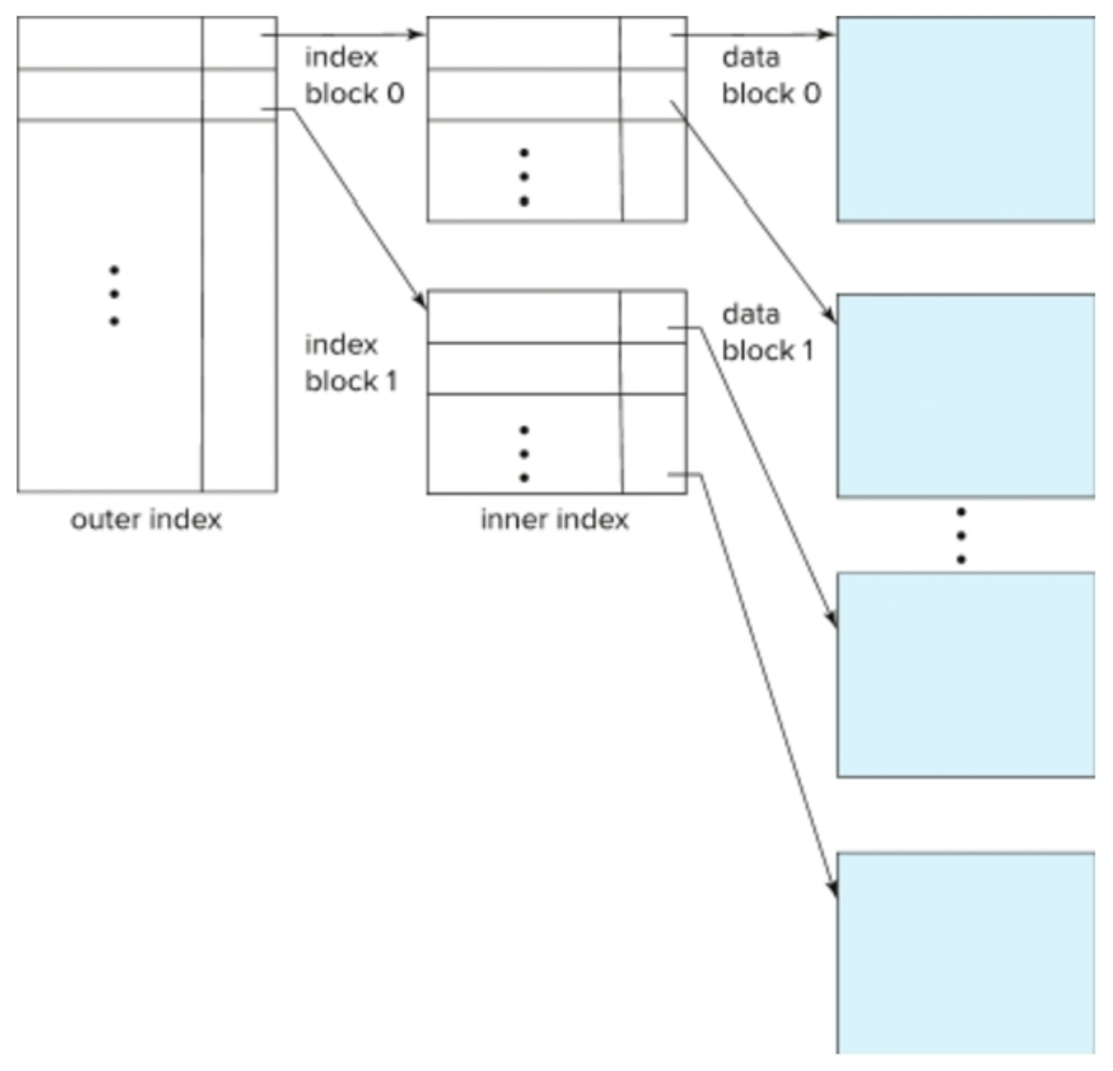




② 辅助索引
对一张表可以建很多个
小有序管大无序
只能为密索引



多级索引



B+树

三个性质：平、半、序

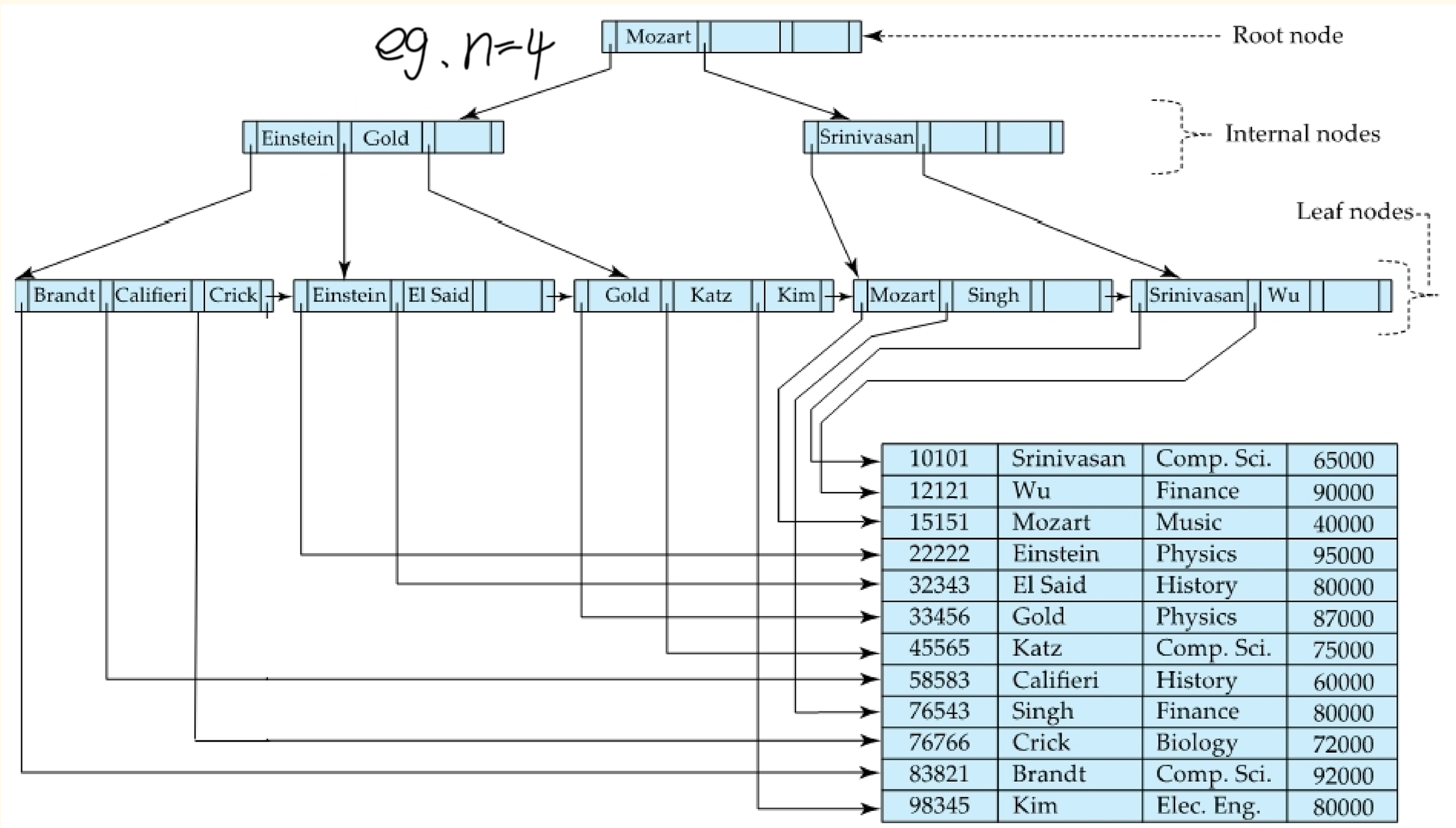
① 叶根距一致

② 过半性：指针数 n，非空指针数 $\geq \lceil \frac{n}{2} \rceil$

③ 每个记录均有序

有多少层，就要多少次块的访问 + 一次记录文件访问

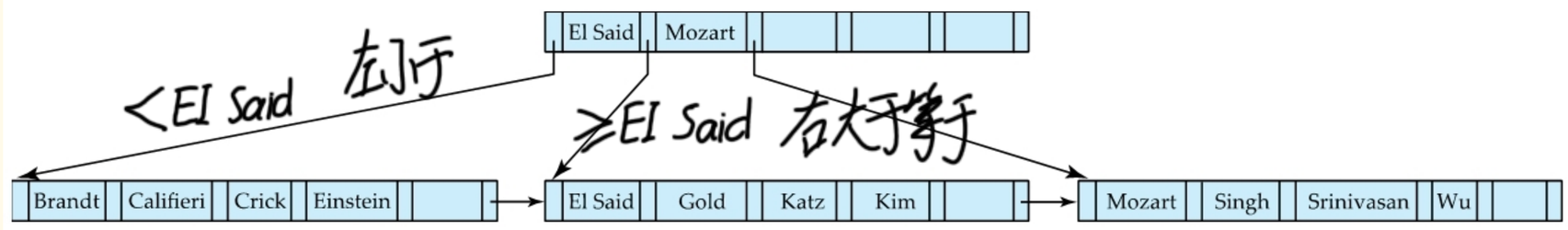
$\Rightarrow (h(\text{树高}) + 1) \times (T_{search} + T_{transit})$



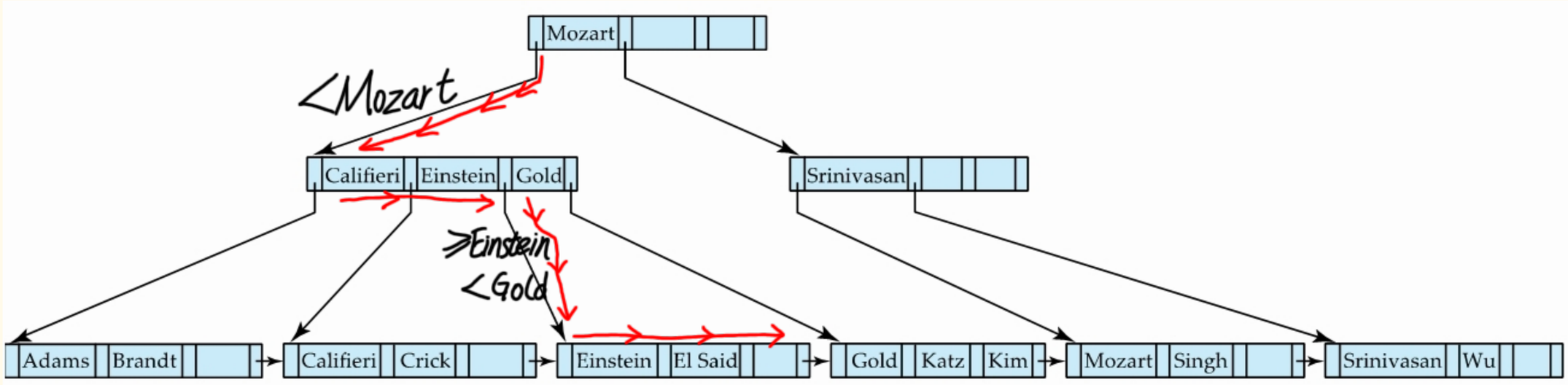
树高 $\leq \log_{\lceil \frac{n}{2} \rceil}(K)$

B+树搜索

- B+-tree for *instructor* file (n = 6)

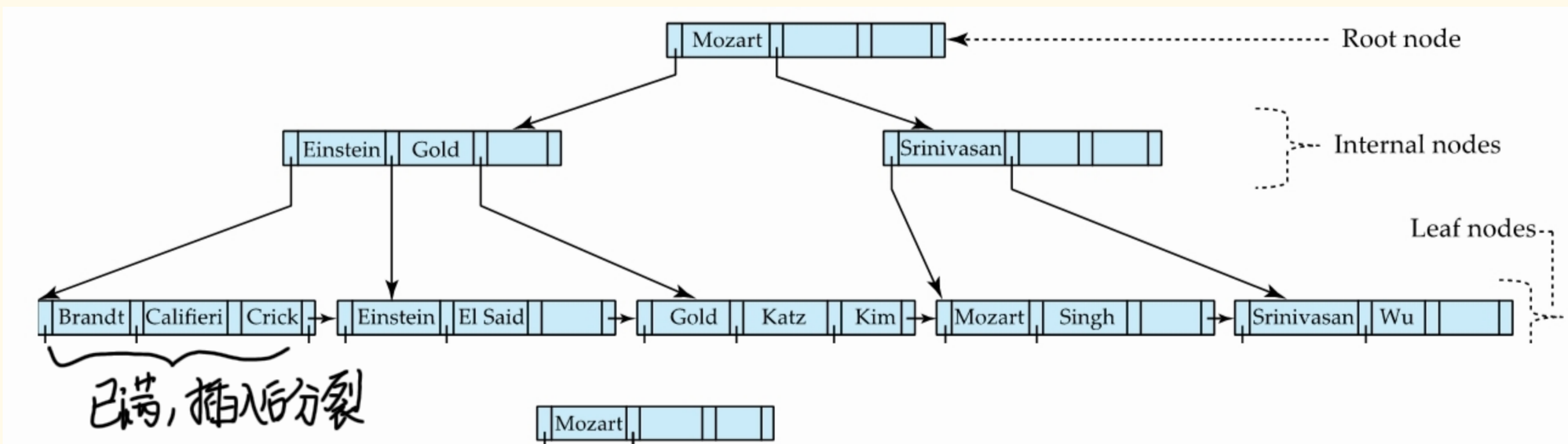


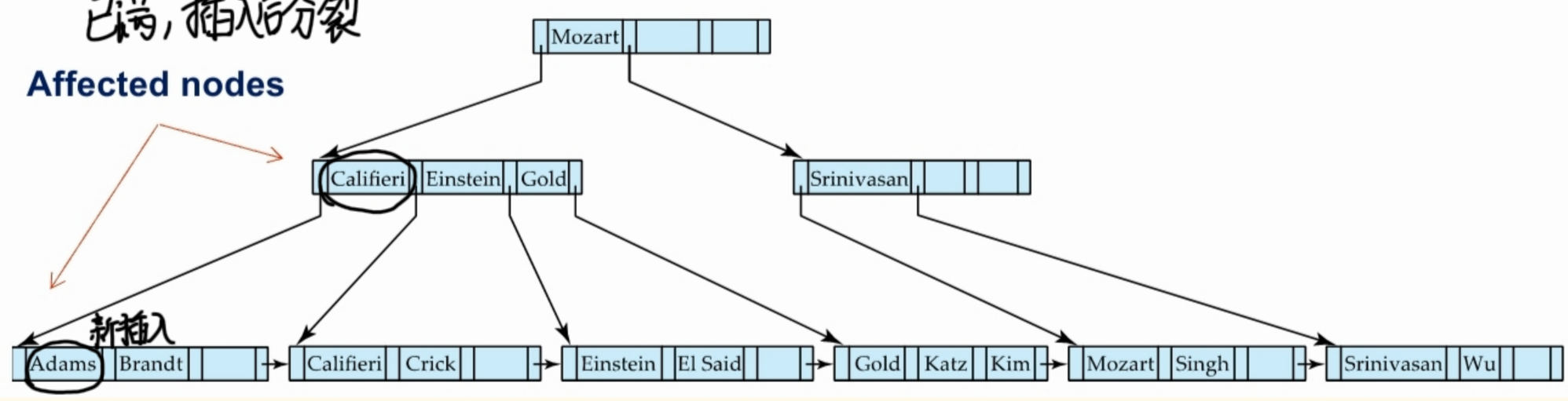
eg.

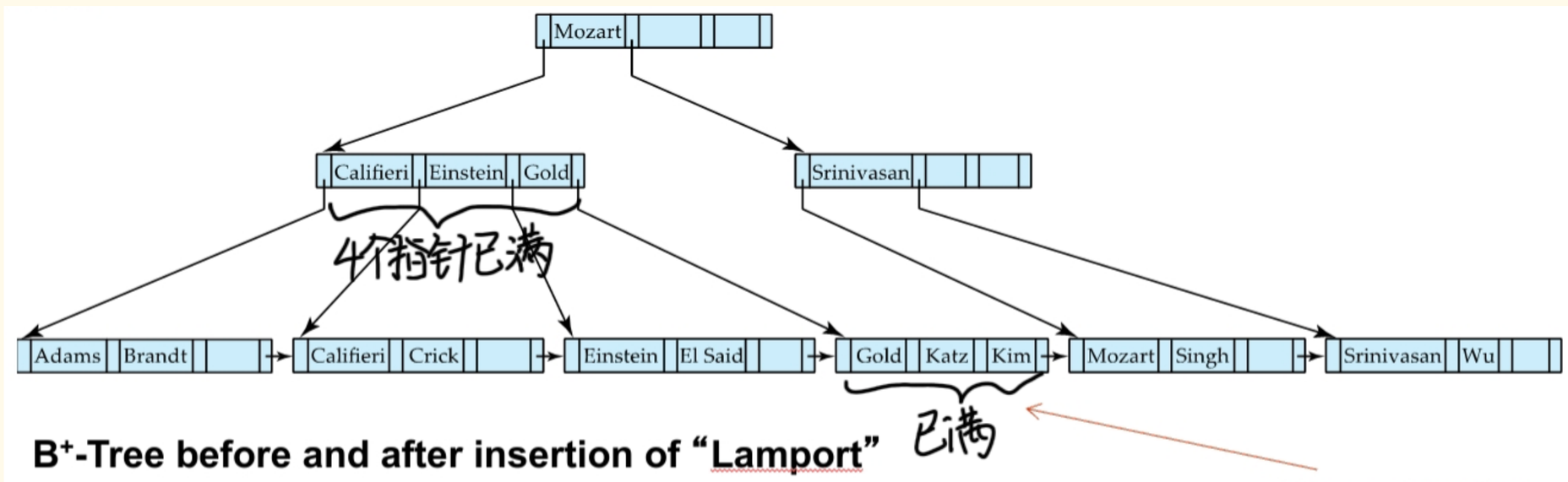


插入

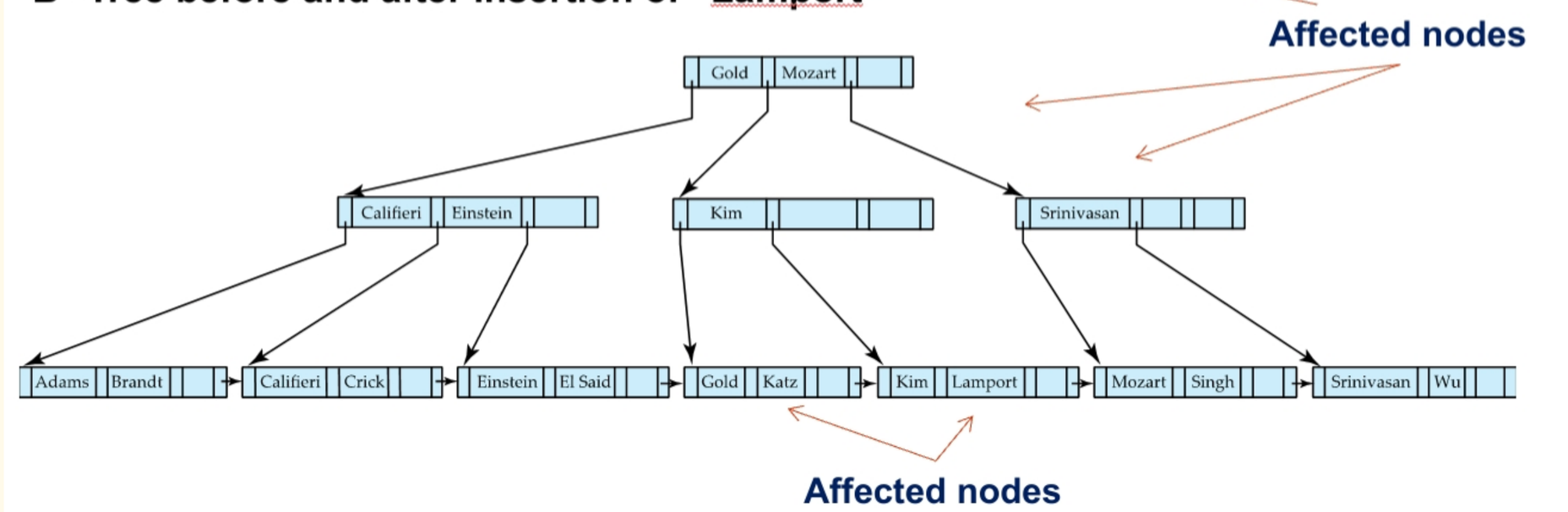
从下至上，按$\lceil \frac{n}{2} \rceil$, $n-\lceil \frac{n}{2} \rceil$分裂



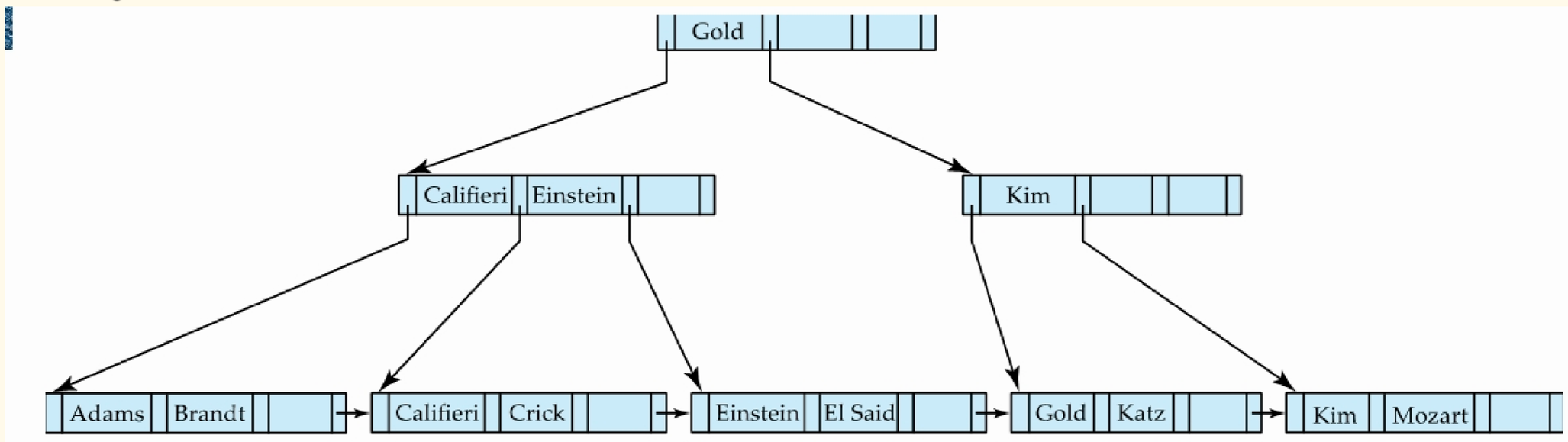




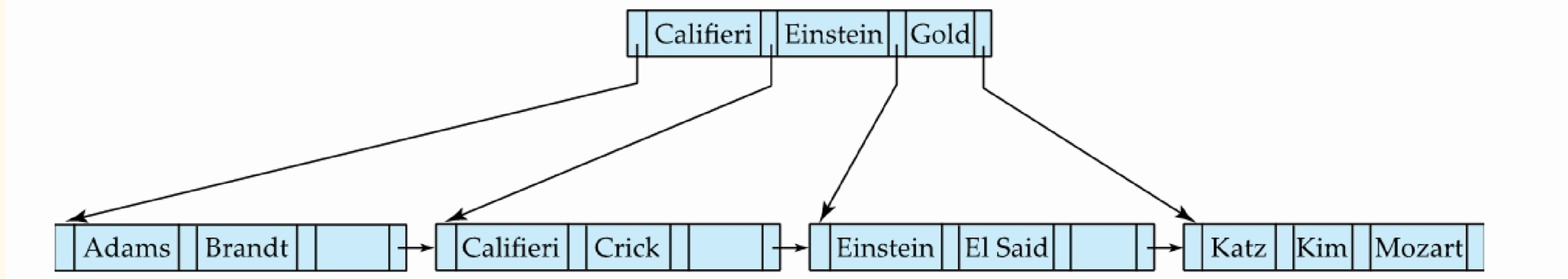
B+-Tree before and after insertion of "Lamport"



删除



Before and after deletion of "Gold"



增改复杂度

$\leq O(\log_{\lceil \frac{n}{2} \rceil}(K))$，n为指针数，K为结点记录数

平均结点占用

取决于一系列结点插入顺序

随机顺序：$\frac{2}{3}$

有序插入：$\frac{1}{2}$（更省存储空间）

B^+树加载与构建：对有序结点自底向上构建

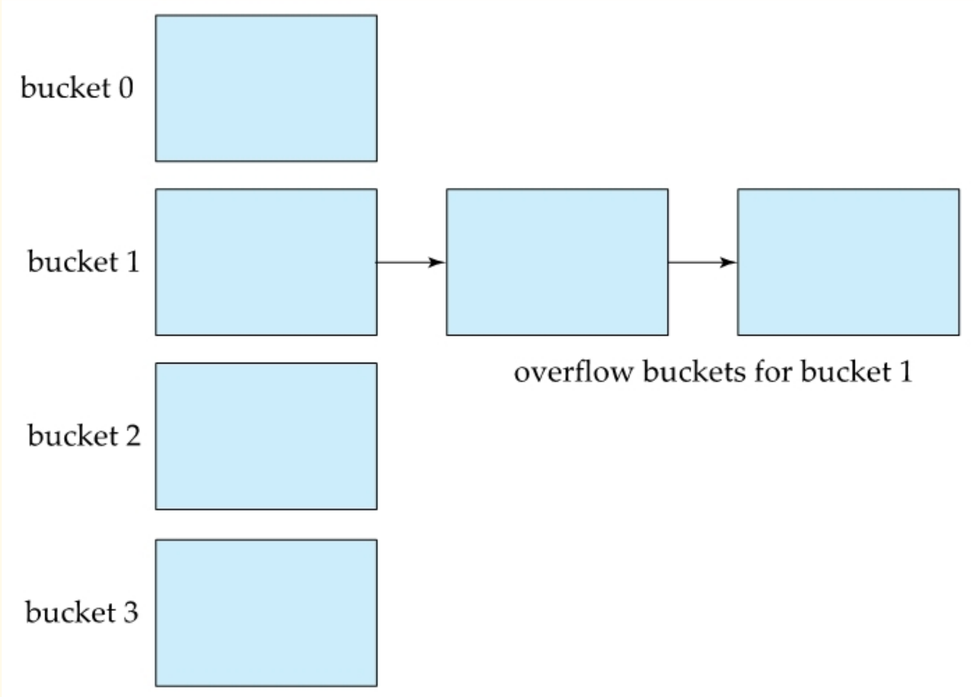
字符串索引

Prefix compression 前缀压缩

- Key values at internal nodes can be prefixes of full key
  - Keep enough characters to distinguish entries in the subtrees separated by the key value
    - E.g., "Silas" and "Silberschatz" can be separated by "Silb"

哈希

采用有链溢出



静态散列：桶数量固定

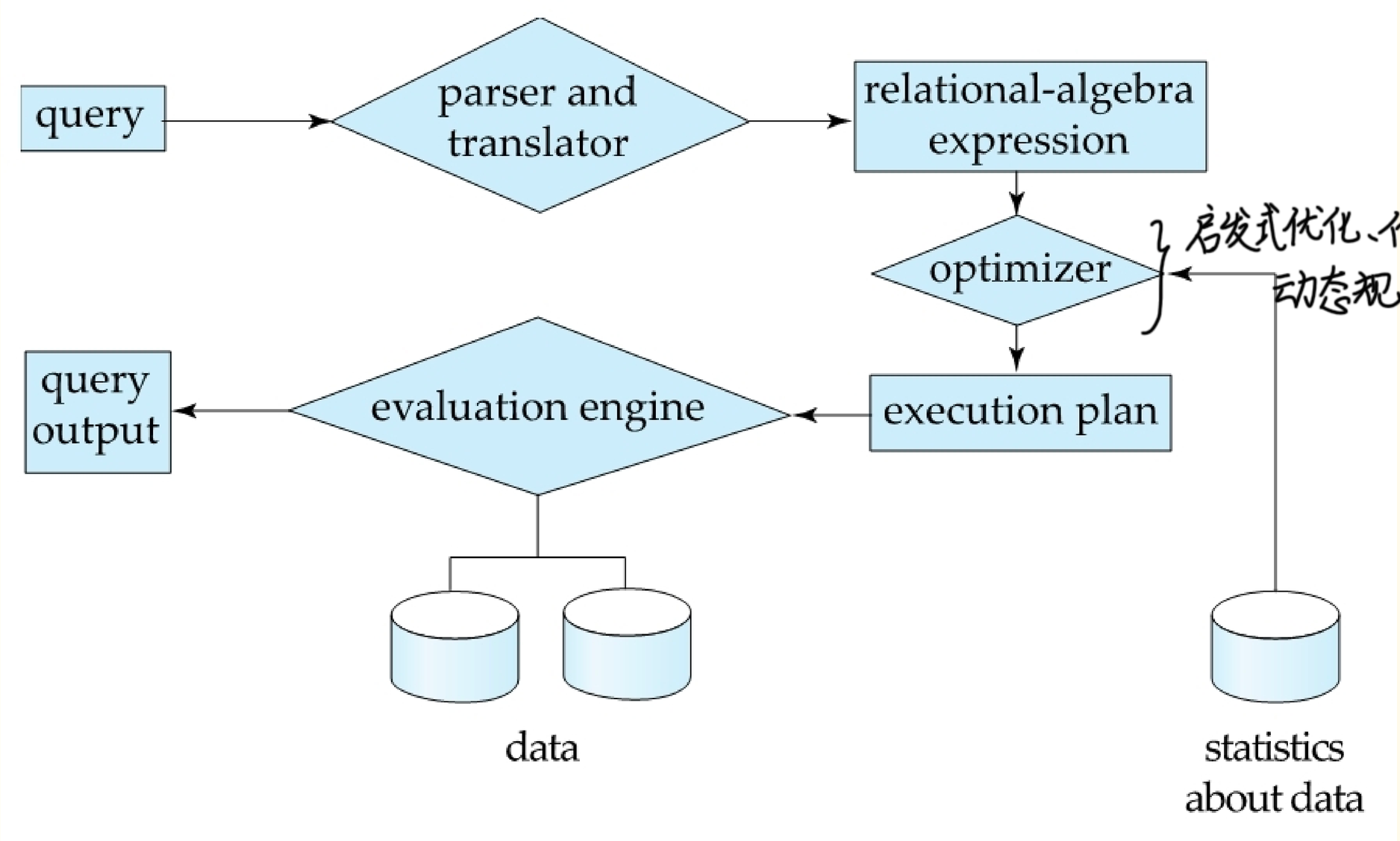
动态散列：桶数量不定

散列经实践证明性能低，SQL中采用顺序索引（B^+树）

创建索引

```
create index takes_pk on takes (ID,course_ID, year, semester, section)
drop index  takes_pk
```

查询处理



查询代价(*)

$cost = t_{seek} + t_{transfer}$

选择

① 线性搜索

② 索引搜索

(1) 等值查询

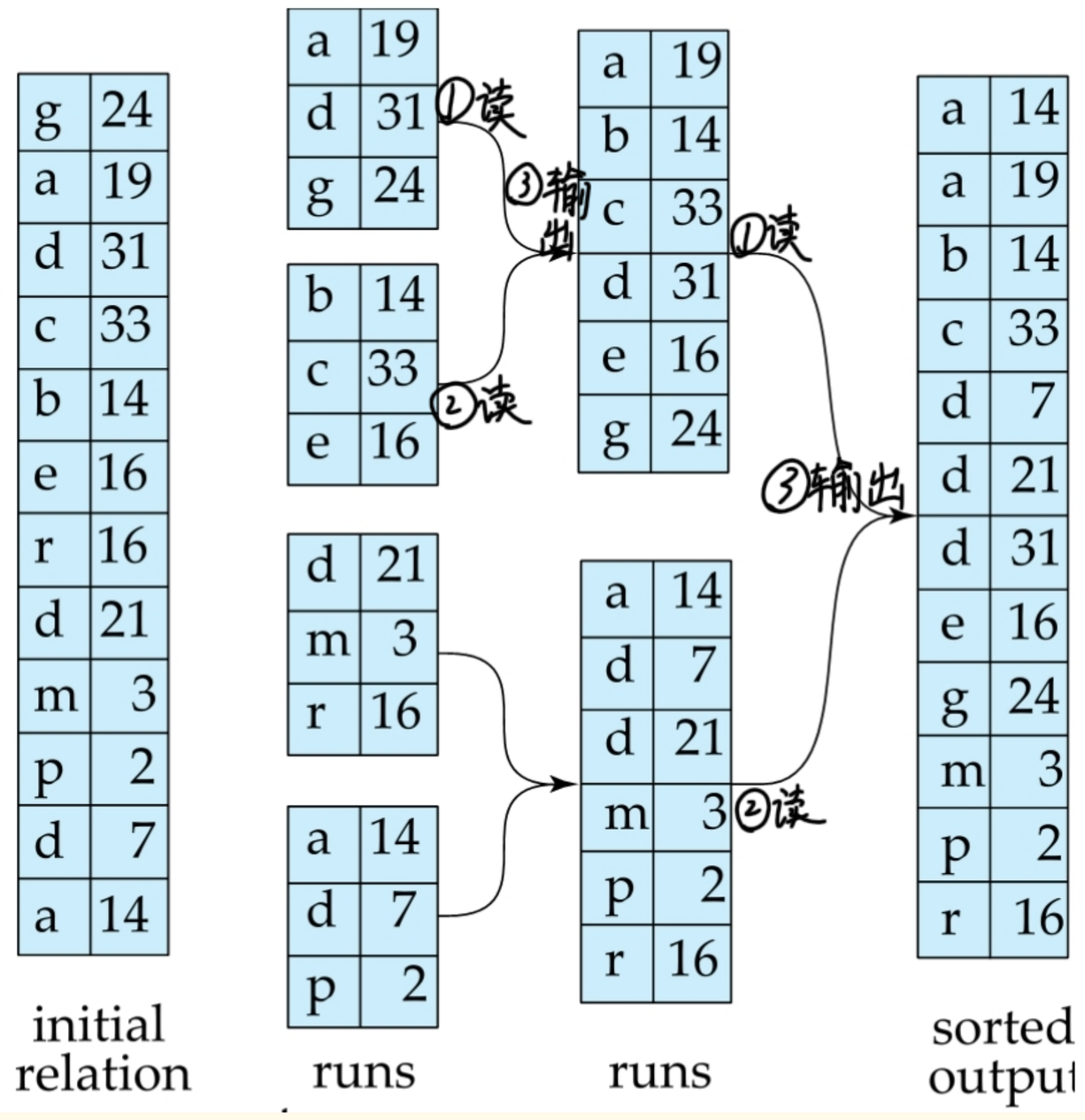
聚集索引/辅助索引 + 主码搜索/非主码搜索

(2) 范围比较

聚集索引/辅助索引

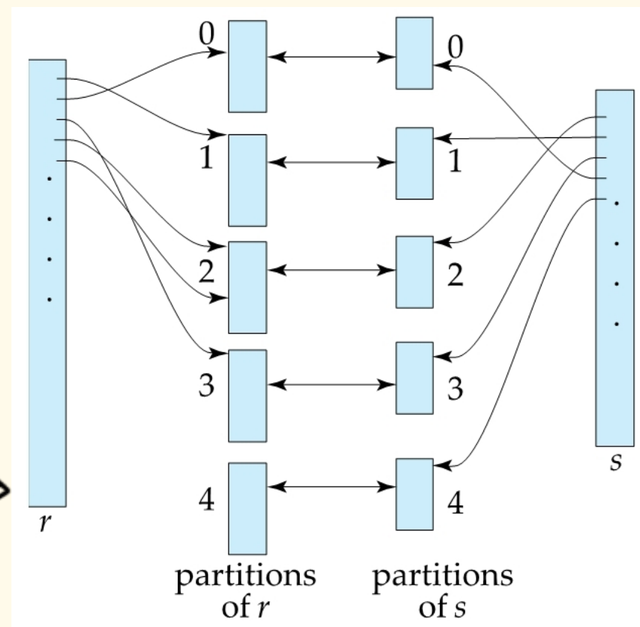
外部归并排序

假设只能同时处理3块



连接

- Nested-loop join
- Block nested-loop join (常用)
- Indexed nested-loop join
- Merge-join
- Hash-join (常用)



查询优化

等价规则 (重点: 1-8)

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections.

$$\sigma_{\theta_1 \wedge \theta_2}(E) \equiv \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. Selection operations are commutative.

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) \equiv \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

3. Only the last in a sequence of projection operations is needed, the others can be omitted.

$$\Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{L_n}(E))\ldots)) \equiv \Pi_{L_1}(E)$$

where $L_1 \subseteq L_2 \ldots \subseteq L_n$

4. Selections can be combined with Cartesian products and theta joins.
   a. $\sigma_\theta(E_1 \times E_2) \equiv E_1 \bowtie_\theta E_2$ (笛卡尔积+选择=连接)
   b. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) \equiv E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

5. Theta-join operations (and natural joins) are commutative.

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$

(⋈可交换)

6. (a) Natural join operations are associative: (⋈可结合)

$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$$

(b) Theta joins are associative in the following manner:

$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 \equiv E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$$

where θ_2 involves attributes from only E_2 and E_3.

7. The selection operation distributes over the theta join operation under the following two conditions: (选择早做)
   (a) When all the attributes in θ_0 involve only the attributes of one of the expressions (E_1) being joined.

$$\sigma_{\theta_0}(E_1 \bowtie_\theta E_2) \equiv (\sigma_{\theta_0}(E_1)) \bowtie_\theta E_2$$

   (b) When θ_1 involves only the attributes of E_1 and θ_2 involves only the attributes of E_2.

$$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_\theta E_2) \equiv (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2))$$

8. The projection operation distributes over the theta join operation as follows: (投影早做)
   (a) if θ involves only attributes from $L_1 \cup L_2$:

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) \equiv \Pi_{L_1}(E_1) \bowtie_\theta \Pi_{L_2}(E_2)$$

   (b) In general, consider a join $E_1 \bowtie_\theta E_2$.
   - Let L_1 and L_2 be sets of attributes from E_1 and E_2, respectively.
   - Let L_3 be attributes of E_1 that are involved in join condition θ, but are not in $L_1 \cup L_2$, and
   - let L_4 be attributes of E_2 that are involved in join condition θ, but are not in $L_1 \cup L_2$.

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) \equiv \Pi_{L_1 \cup L_2}(\Pi_{L_1 \cup L_3}(E_1) \bowtie_\theta \Pi_{L_2 \cup L_4}(E_2))$$

9. The set operations union and intersection are commutative

$E_1 \cup E_2 \equiv E_2 \cup E_1$

$E_1 \cap E_2 \equiv E_2 \cap E_1$

(set difference is not commutative).

10. Set union and intersection are associative.

$(E_1 \cup E_2) \cup E_3 \equiv E_1 \cup (E_2 \cup E_3)$

$(E_1 \cap E_2) \cap E_3 \equiv E_1 \cap (E_2 \cap E_3)$

11. The selection operation distributes over $\cup$, $\cap$ and –.

a. $\sigma_\theta(E_1 \cup E_2) \equiv \sigma_\theta(E_1) \cup \sigma_\theta(E_2)$

b. $\sigma_\theta(E_1 \cap E_2) \equiv \sigma_\theta(E_1) \cap \sigma_\theta(E_2)$

c. $\sigma_\theta(E_1 - E_2) \equiv \sigma_\theta(E_1) - \sigma_\theta(E_2)$

d. $\sigma_\theta(E_1 \cap E_2) \equiv \sigma_\theta(E_1) \cap E_2$

e. $\sigma_\theta(E_1 - E_2) \equiv \sigma_\theta(E_1) - E_2$

preceding equivalence does not hold for $\cup$

eg、选择早做

$$\Pi_{name,\ title}(\sigma_{dept\_name=\text{'Music'}}(instructor \bowtie (teaches \bowtie \Pi_{course\_id,\ title}(course))))$$

$$\Downarrow$$

$$\Pi_{name,\ title}((\sigma_{dept\_name=\text{'Music'}}(instructor)) \bowtie (teaches \bowtie \Pi_{course\_id,\ title}(course)))$$

总结

对于选择，① $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$

② $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$

③ $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta_2} E_2$ 选择早做

$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta_3} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta_3} (\sigma_{\theta_2}(E_2))$

对于投影，④ $\Pi_{L_1}(\Pi_{L_2}(\cdots \Pi_{L_n}(E))) = \Pi_{L_1}(E)$, $L_1 \subseteq L_2 \cdots \subseteq L_n$

⑤ $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$ 投影早做

对于连接，⑥ $E_1 \bowtie E_2 = E_2 \bowtie E_1$

⑦ $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$

⑧ $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$ 笛卡尔积+选择=连接

$\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

查询优化应用

例：

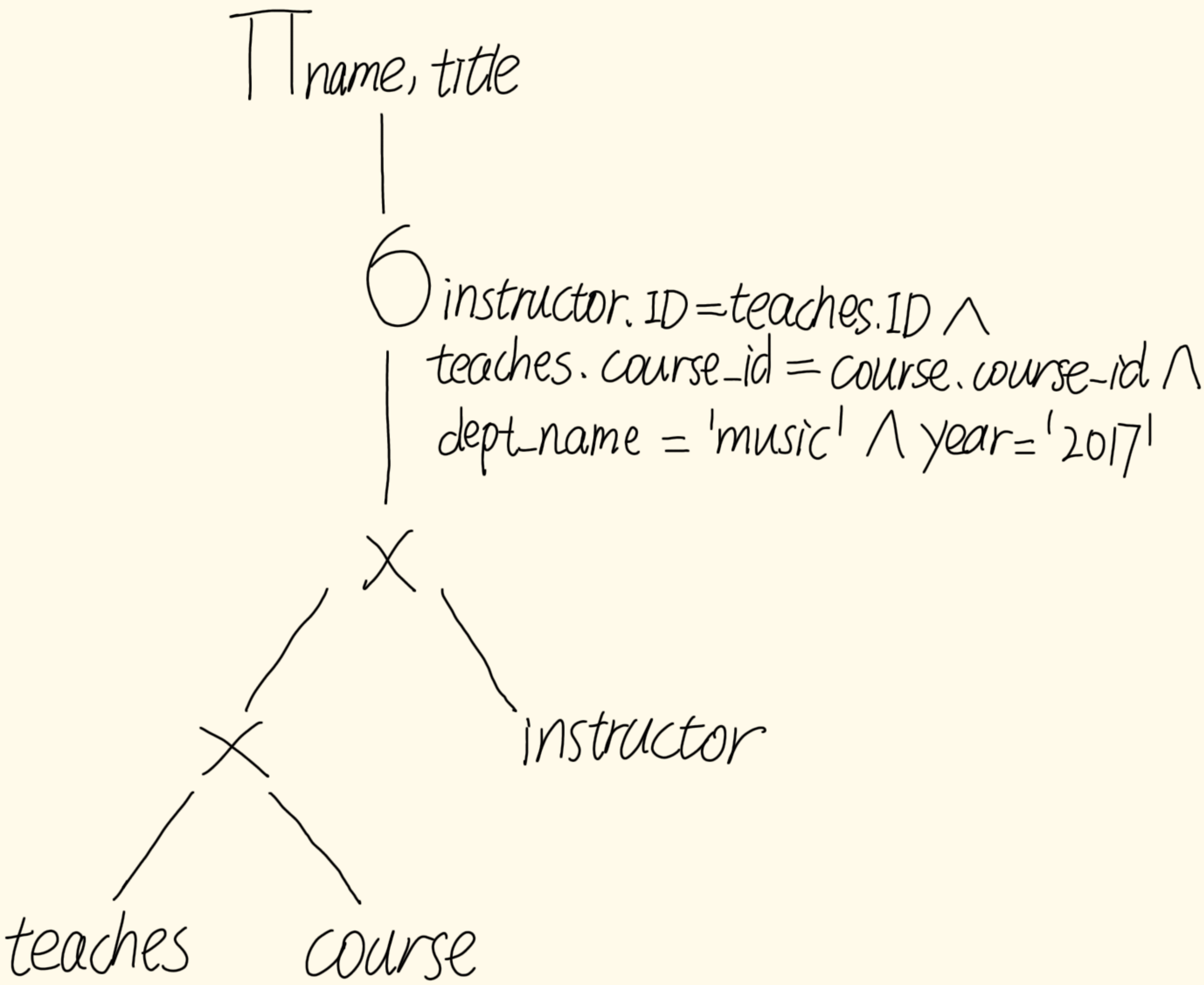
```
select name,title
from instructor, teaches, course
where instructor.ID=teaches.ID and teaches.course_id =
  course.course_id and dept_name='music' and year='2017';
```

① SQL转为关系代数

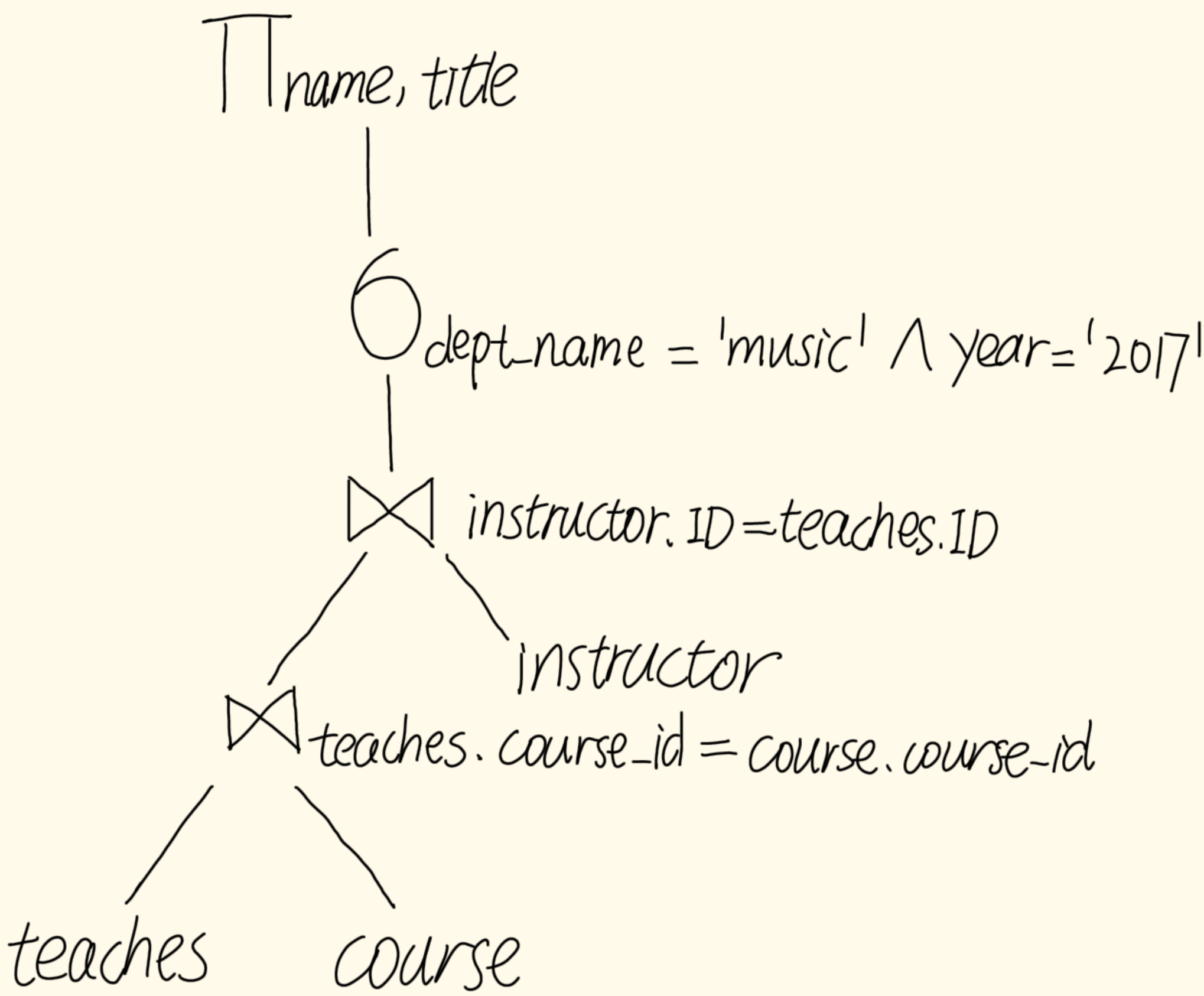
$$\Pi_{name,\ title}\left(\sigma_{instructor.ID=teaches.ID\ \wedge\ teaches.course\_id=course.course\_id\ \wedge\ dept\_name='music'\ \wedge\ year='2017'}\left(instructor \times teaches \times course\right)\right)$$

② 初始语法树

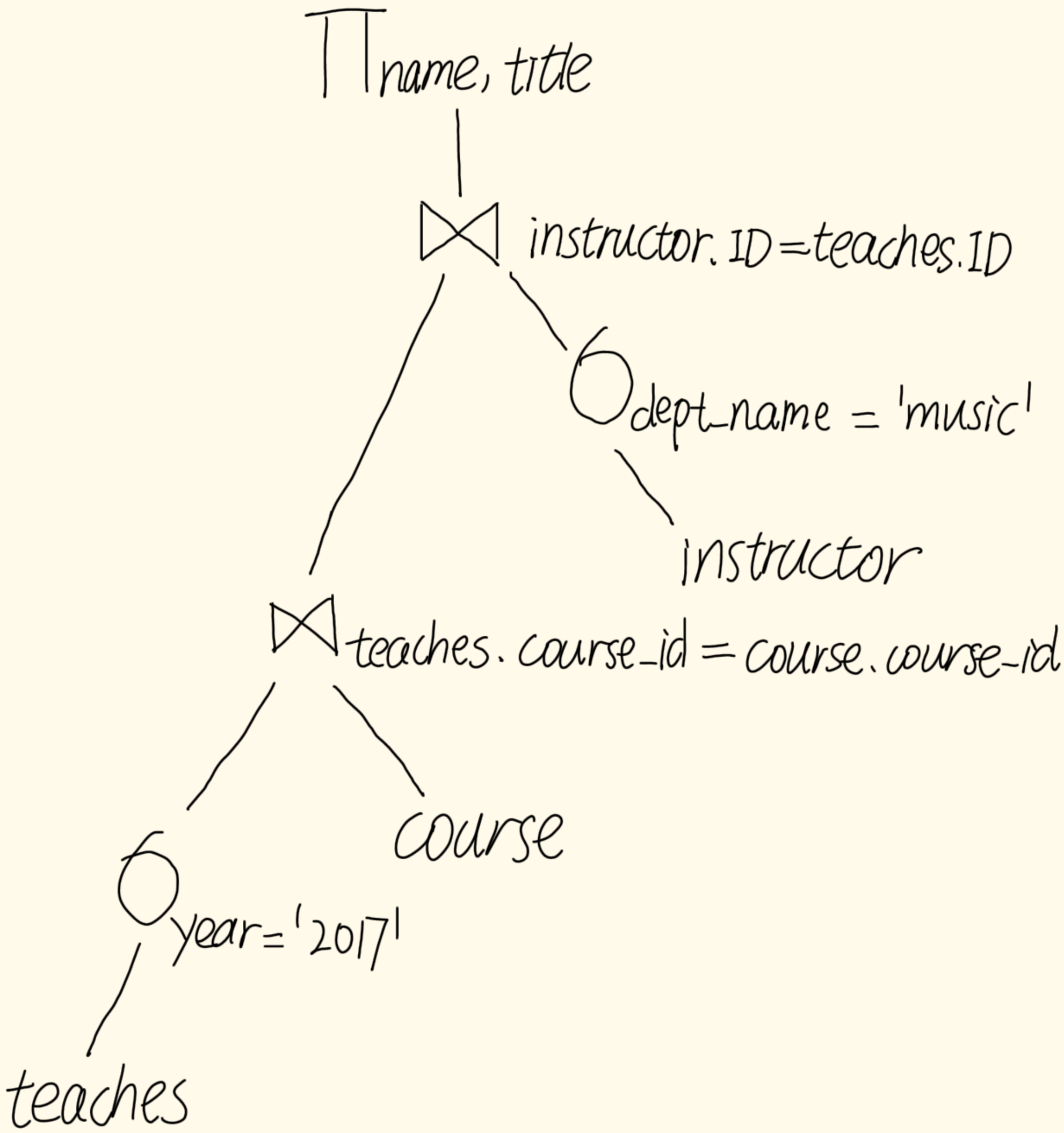
（复合的全放在左子树）


Π name, title
σ instructor.ID = teaches.ID ∧
teaches.course_id = course.course_id ∧
dept_name = 'music' ∧ year = '2017'
×
×
instructor
teaches
course

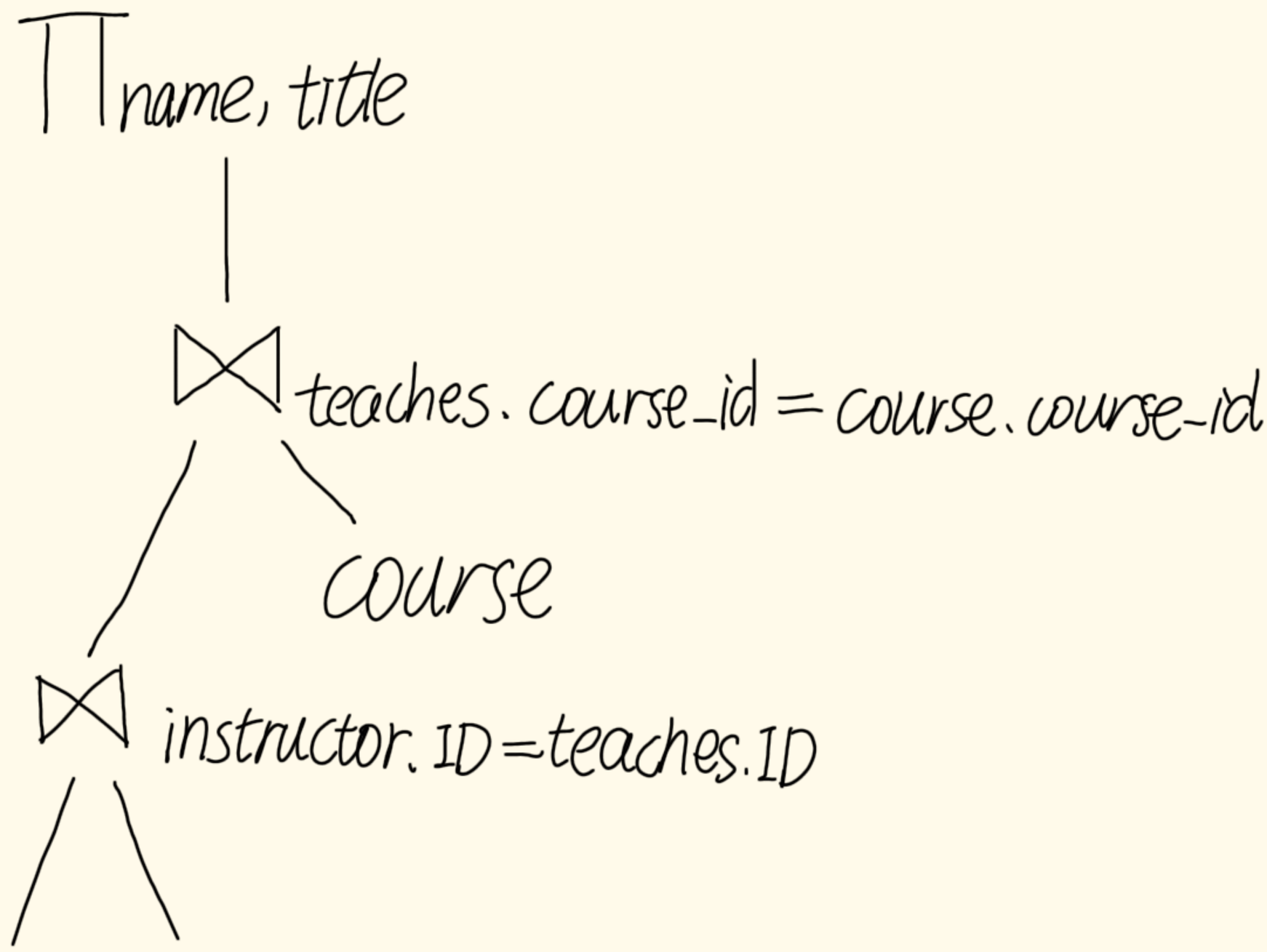
③ 第一步：σ(×)转为⋈


Π name, title
σ dept_name = 'music' ∧ year = '2017'
⋈ instructor.ID = teaches.ID
instructor
⋈ teaches.course_id = course.course_id
teaches
course

④ 第二步：选择早做

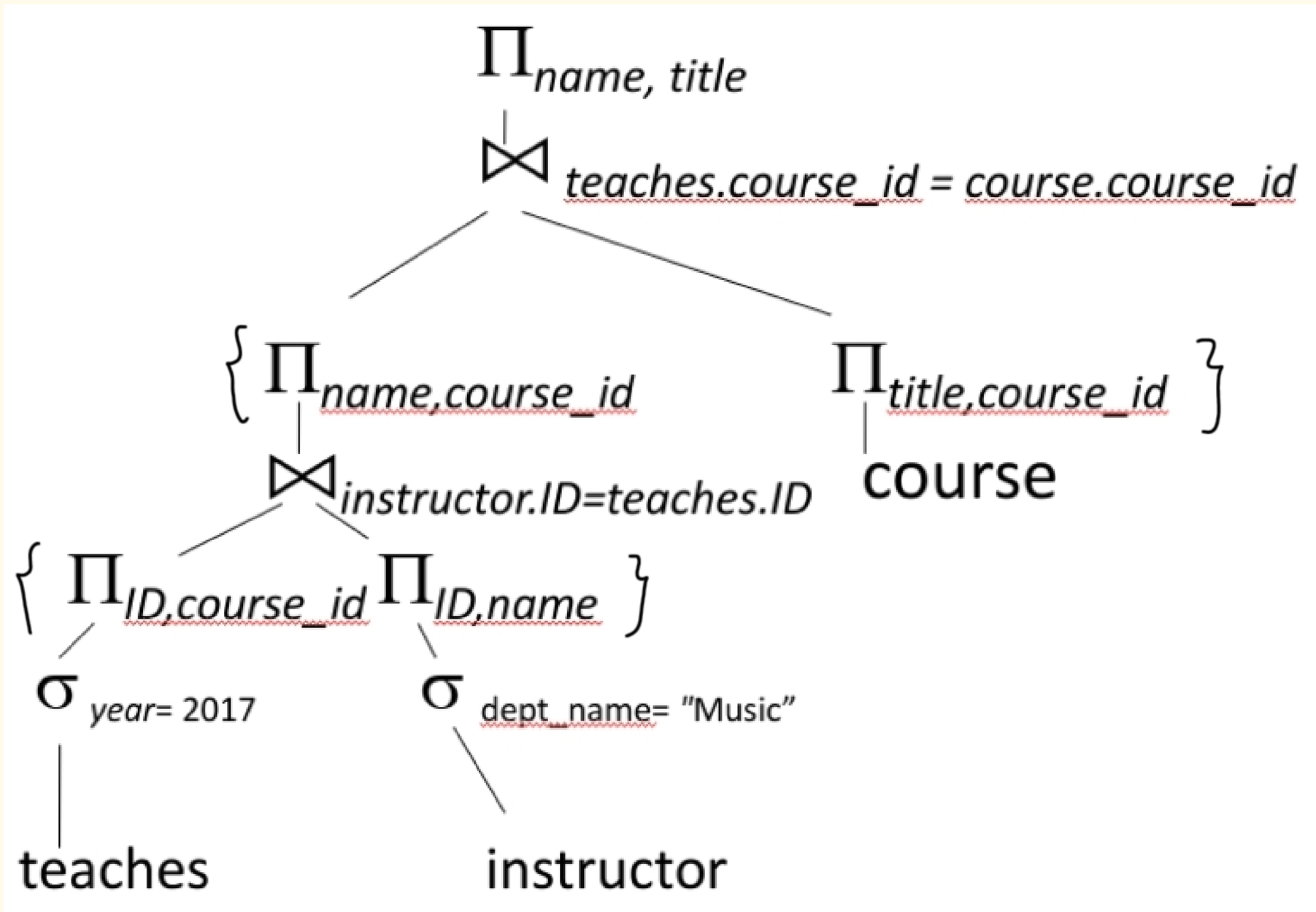


⑤ 第三步：⋈ 可交换（目标为左深树）



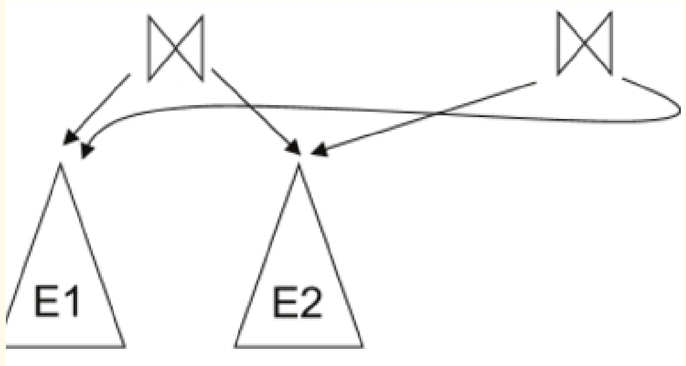

year='2017'
teaches
dept_name = 'music'
instructor

⑥无中生有Π，投影早做


Π name, title
⋈ teaches.course_id = course.course_id
Π name,course_id
Π title,course_id
course
⋈ instructor.ID=teaches.ID
Π ID,course_id
Π ID,name
σ year= 2017
σ dept_name= "Music"
teaches
instructor

启发式优化 (*,不考)

空间: 共享指针



时间: 动态编程(基于代价的优化)

$r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$ ⇒ 排列组合太多 ⇒ GA, 动态规划

关系集 S ⇒ $S = \underline{S_1} \bowtie \underline{(S - S_1)}$

递归解决

启发式规则:

Perform selection early

Perform projection early

Perform most restrictive selection and join operations

事务

概念

事务：一个访问和可能改变多个数据项的执行程序单元

原子性

eg、E.g., transaction to transfer $50 from account A to account B:

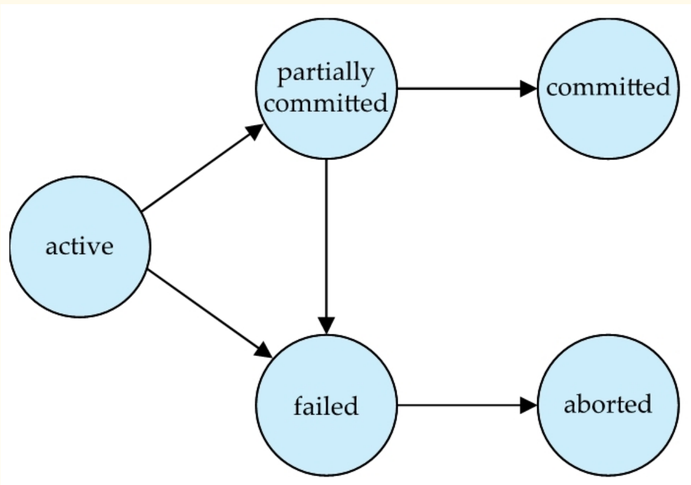
1. **read**(*A*)
2. *A* := *A* – 50
3. **write**(*A*)
4. **read**(*B*)
5. *B* := *B* + 50
6. **write**(*B*)

性质：原子性、持久性（事务完成后，DB持久不再改变）、一致性（事务将一致状态的DB转换到一致的另一状态）、隔离性（不受其他影响，并行时也像串行）

（ACID特性）

- **Atomicity.** Either all operations of the transaction are properly reflected in the database or none are. 原
- **Consistency.** Execution of a transaction in isolation preserves the consistency of the database. 致
- **Isolation.** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions. 隔
  - That is, for every pair of transactions T_i and T_j, it appears to T_i that either T_j, finished execution before T_i started, or T_j started execution after T_i finished.
- **Durability.** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures. 久

事务的状态



并发执行

① 两段锁

② 多版本

调度：不能改变一个事务内指令执行顺序

① 串行调度

T_1	T_2
read (A)	
A := A – 50	
write (A)	
read (B)	
B := B + 50	
write (B)	
commit	
	read (A)
	temp := A * 0.1
	A := A - *temp*
	write (A)
	read (B)
	B := B + *temp*
	write (B)
	commit

等价的

T_1	T_2
	read (A)
	temp := A * 0.1
	A := A - *temp*
	write (A)
	read (B)
	B := B + *temp*
	write (B)
	commit
read (A)	
A := A – 50	
write (A)	
read (B)	
B := B + 50	
write (B)	
commit	

不等价

② 并发调度

T_1	T_2
read (A)	
A := A – 50	
write (A)	
	read (A)
	temp := A * 0.1
	A := A - *temp*
	write (A)
read (B)	
B := B + 50	
write (B)	
commit	
	read (B)
	B := B + *temp*
	write (B)
	commit

T_1	T_2
read (A)	
A := A – 50	
	read (A)
	temp := A * 0.1
	A := A - *temp*
	write (A)
	read (B)
write (A)	
read (B)	
B := B + 50	
write (B)	
commit	
	B := B + *temp*
	write (B)
	commit

冲突可串行化

两指令冲突：

1. I_i = **read**(Q), I_j = **read**(Q). I_i and I_j don't conflict.
2. I_i = **read**(Q), I_j = **write**(Q). They conflict.
3. I_i = **write**(Q), I_j = **read**(Q). They conflict
4. I_i = **write**(Q), I_j = **write**(Q). They conflict

交换非冲突指令 ⟹ 冲突等价 ⟹ 冲突可串行化

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

视图可串行化

视图等价：① 视图读的初值相同

② 过程相同

③ 最后的写入相同

如，盲写

T_{27}	T_{28}	T_{29}
read (Q)		
	write (Q)	
write (Q)		
		write (Q)

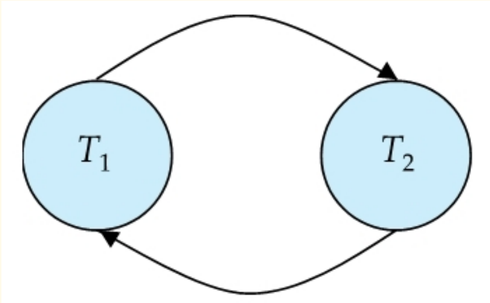
如，

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

冲突可串行化为视图可串行化的特例

判断冲突可串行化



T_1~T_2冲突，T_1先出现冲突 ⇒ 画$T_1 \to T_2$线

无环 ⇒ 可串行化，执行顺序为图拓扑序

可恢复调度

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	

已提前提交T8中间数据

⇒ T8无法回滚，不可恢复

级联回滚坍塌

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

若T10回滚，则T11，T12必须回滚

隔离级别(3解)

无读取幻觉

无写入错误

(最常用)

Serializable — default

Repeatable read — only committed records to be read.

- Repeated reads of same record must return same value.
- However, a transaction may not be serializable – it may find some records inserted by a transaction but not find others.

Read committed — only committed records can be read.

- Successive reads of record may return different (but committed) values.

Read uncommitted — even uncommitted records may be read.

某些场景，降低要求限制，提高性能

实现方式：两段锁协议
时间戳协议
多版本控制(常用)

基于锁的协议

- 独占锁（X，exclusive）→ 可读可写
- 共享锁（S，shared）→ 只能读

相容性矩阵

T_j \ T_i	S	X
S	true	false
X	false	false

死锁

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B) 等待T_3
lock-X(A) 等待T_4	

相互等待

解决：kill一个transaction

饿死

① 等待 X-lock item，但一系列事务都在对item加 S-lock ⇒ 引入代价

② kill一个事务时，反复挑出来杀掉（解决死锁时）

预防死锁

① 一次性封锁所有资源

② 顺序封锁资源

③ wait-die调度（老事务等新事务资源：wait，新事务等老事务资源：die）[非抢占式]

④ wound-wait调度（老事务等新事务资源：kill，新事务等老事务资源：wait）[抢占式]

必要条件

检测死锁

① 超时机制

② 等待图

死锁 $\xrightarrow{\text{充分}}$ 等待图中出现环

死锁 $\xleftarrow{\text{必要}}$ 等待图中出现环

死锁解决

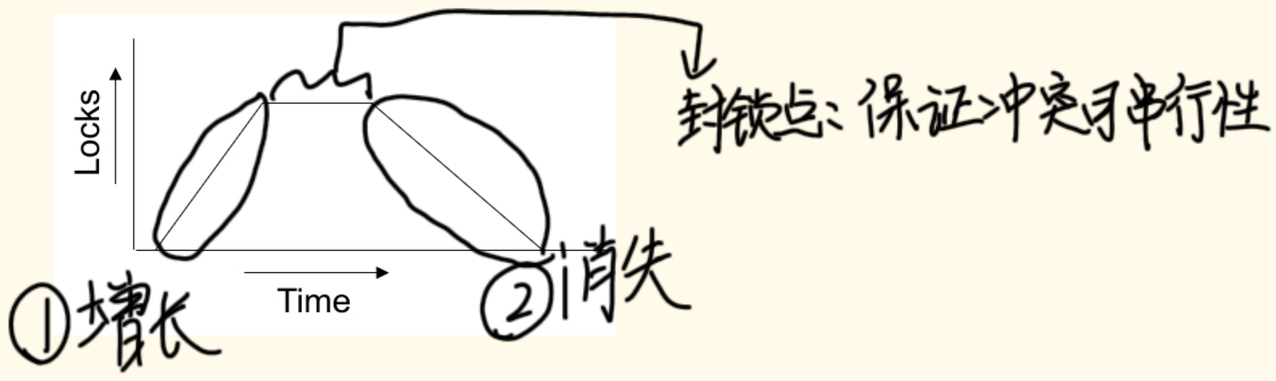
① 选 victim 事务杀掉

② 部分回滚

锁协议两阶段

两段锁：

T_1	T_2	concurrency-control manager
lock-X(B)		
		grant-X(B, T_1)
read(B)		
$B := B - 50$		
write(B)		
unlock(B)		
	lock-S(A)	
		grant-S(A, T_2)
	read(A)	
	unlock(A)	
	lock-S(B)	
		grant-S(B, T_2)
	read(B)	
	unlock(B)	
	display($A + B$)	
lock-X(A)		
		grant-X(A, T_1)
read(A)		
$A := A + 50$		
write(A)		
unlock(A)		



Strict two-phase locking: a transaction must hold all its exclusive locks till it commits/aborts. 严格两段锁 提交才释放 X-lock

- Ensures recoverability and avoids cascading roll-backs

Rigorous two-phase locking: a transaction must hold *all* locks till commit/abort. 严酷两段锁 提交才释放 X-lock 和 S-lock（常见）

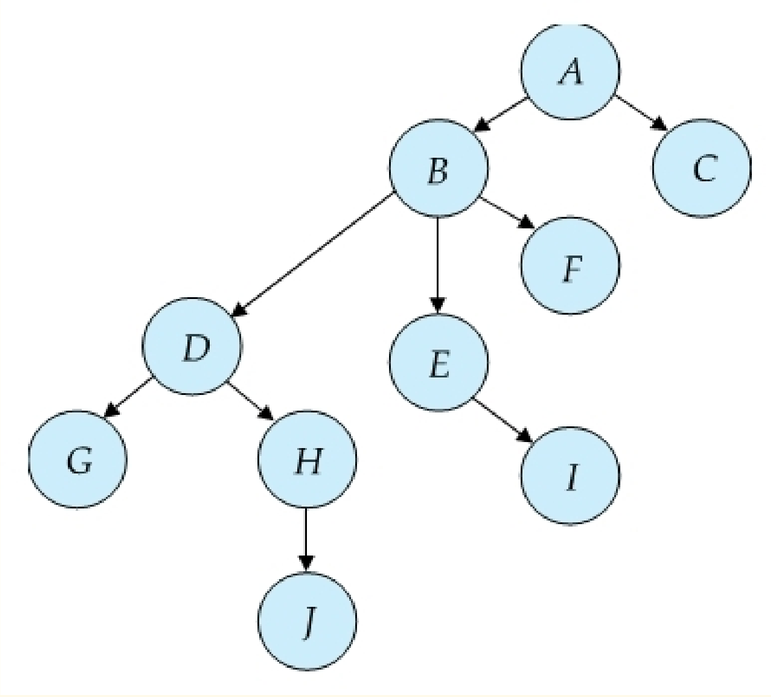
- Transactions can be serialized in the order in which they commit.

> 不存在提交顺序依赖要求
> 一定可恢复

升级和降级

- Growing Phase:
  - can acquire a lock-S on item
  - can acquire a lock-X on item
  - can **convert** a lock-S to a lock-X (**upgrade**)
- Shrinking Phase:
  - can release a lock-S
  - can release a lock-X
  - can convert a lock-X to a lock-S (**downgrade**)

基于图的协议（*，不考）



只能加独占锁

必须先对父项加锁，才能向子项加锁（需形成完整偏序路径）

不满足两段锁（解锁时可对其他分支加锁）

⇓

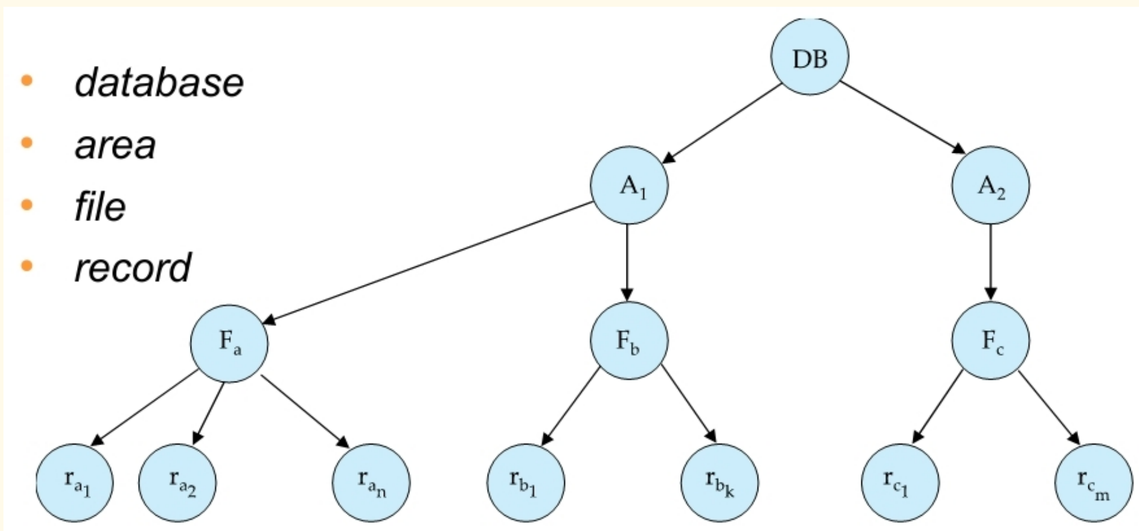
冲突可串行化、不会死锁（定义了顺序）

存在提交顺序依赖要求、以及级联回滚

偏序关系难以确定

多粒度锁

- database
- area
- file
- record



从根开始加锁

欲加下层锁，上层必有适当锁

意向锁：

- ***intention-shared*** (IS)
- ***intention-exclusive*** (IX)
- ***shared and intention-exclusive*** (SIX)

	IS	IX	S	SIX	X
IS	true	true	true	true	false
IX	true	true	false	false	false
S	true	false	true	false	false
SIX	true	false	false	false	false
X	false	false	false	false	false

时间戳顺序协议（*，不考）

W-timestamp(*Q*) is the largest time-stamp of any transaction that executed **write**(*Q*) successfully.

R-timestamp(*Q*) is the largest time-stamp of any transaction that executed **read**(*Q*) successfully.

读指令：$TS(T_i) < W\text{-}timestamp(Q) \Rightarrow T_i$ 回滚（别人即将修改）

$TS(T_i) > W\text{-}timestamp(Q) \Rightarrow T_i$ 执行，并修改Q读时戳

写指令：$TS(T_i) < R\text{-}timestamp(Q) \Rightarrow T_i$ 回滚（别人已读旧数据）

$TS(T_i) < W\text{-}timestamp(Q) \Rightarrow T_i$ 回滚（别人已写）

其他 $\Rightarrow T_i$ 执行，并修改 Q 写时戳

T_{27}	T_{28}
read(Q)	
	write(Q)
write(Q)	

TSO 协议下，T_{27} 的 write(Q) 失败

引入托马斯写规则后，write(Q) 可执行

基于验证的协议（*，不考）

- 读和执行
- 验证
- 写

- **StartTS**(T_i) : the time when T_i started its execution
- **ValidationTS**(T_i): the time when T_i entered its validation phase
- **FinishTS**(T_i) : the time when T_i finished its write phase

只有 $startTS(T) < finishTS(T_i) < validation(T)$，$T_i$ 才可能影响 T

T_{25}	T_{26}
read(B)	
	read(B)
	$B := B - 50$
	read(A)
	$A := A + 50$
read(A)	
<validate>	
display($A + B$)	
	<validate>
	write(B)
	write(A)

T_{25} 未写入，故 T_{26} 不受 T_{25} 影响，T_{26} 无需回滚

问题：出现大量回滚

多版本协议 MVCC

① 多版本时间戳顺序协议

- **Content** -- the value of version Q_k.
- **W-timestamp**(Q_k) -- timestamp of the transaction that created (wrote) version Q_k
- **R-timestamp**(Q_k) -- largest timestamp of a transaction that successfully read version Q_k

为版本赋予时间戳

② 多版本两段锁

需区分只读事务和更新事务

③ 快照隔离(SI)

Better solution: snapshot isolation level 在快照隔离级别下，一个事务执行的查询操作会获取一个快照，即在该事务开始时数据库的一个镜像，这个快照会在整个事务结束前保持不变。因此，事务在执行时不会看到并发事务对数据库的修改。

⇒ 因此仅当写入相同数据项时，才会发生冲突

⇓

加X锁

先提交者胜利

SI好处：

读不阻塞、无幻读、不死锁、不丢失更新

SI缺陷：

并非总是串行执行

故障分类

(1) 事务故障

① 逻辑错误（如，除零）

② 系统错误（如，死锁）

⇒ 利用日志恢复

(2) 系统崩溃

障停假设：崩溃时，数据不变

(3) 磁盘故障

⇒ 备份存储

存储结构

(1) 易失存储

(2) 非易失存储

(3) 稳定存储

日志恢复

日志：顺序记录事务操作，存于稳定存储中

① 立即更新策略（故障时未完成事务，恢复后需将其恢复为none，故障时已commit但未写入磁盘，恢复为all）

日志中记录的事务一定是完整且已提交的！

A transaction is said to have committed when its commit log record is output to stable storage 当一个事务的commit日志记录被输出到稳定存储器后，则该事务已提交

② 延迟更新策略（仅提交时才从缓存输至磁盘）

不常用

假设：修改时按严格两段锁，T_i修改数据项g后，T_j必须等T_i提交后才能修改g!!!（也太严酷）

We assume that *if a transaction T_i has modified an item, no other transaction can modify the same item until T_i has committed or aborted*

可恢复：undo（逆向读日志）⇒保证了原子性、一致性，补偿日志 $\langle T_i, X, X_{prev} \rangle$（只可用于redo）

[对未完成的事务]：有始无终

Contains the record <T_i **start**>,

But does not contain either the record <T_i **commit**> *or* <T_i **abort**>.

redo（正向）⇒补偿日志，可多次重做

[对已完成的事务]：有始有终

Contains the records <T_i **start**>

And contains the record <T_i **commit**> *or* <T_i **abort**>

eg.

<T_0 start>	<T_0 start>	<T_0 start>
<T_0, A, 1000, 950>	<T_0, A, 1000, 950>	<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>
	<T_0 commit>	<T_0 commit>
	<T_1 start>	<T_1 start>
	<T_1, C, 700, 600>	<T_1, C, 700, 600>
		<T_1 commit>
(a) 然后故障	(b) 然后故障	(c) 然后故障

(a)：undo，<T_0, B, 2000>，<T_0, A, 1000>，<T_0, abort>

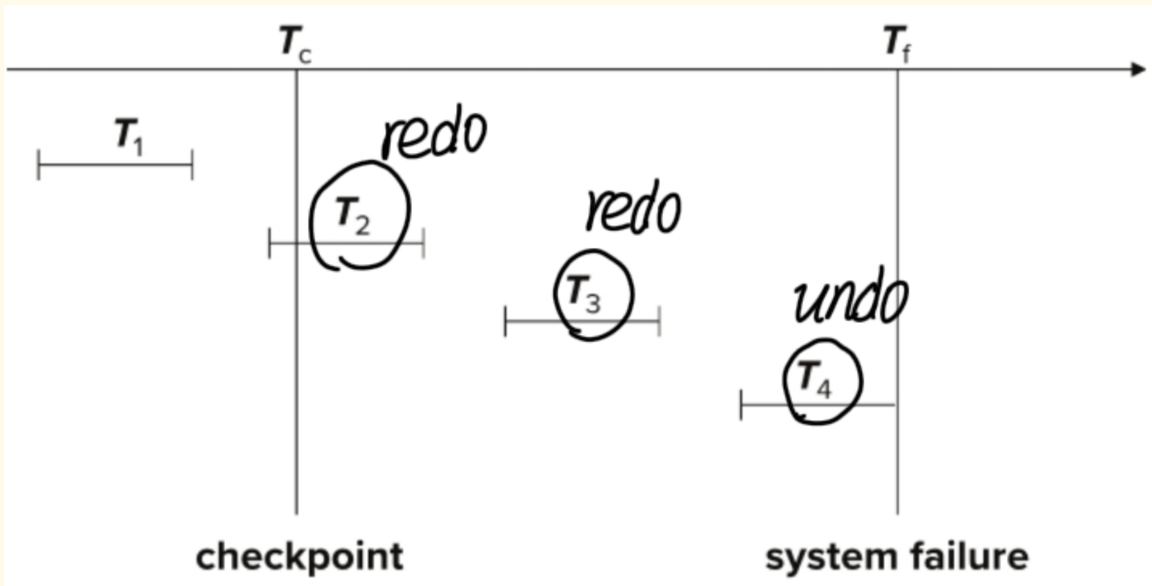
(b)：redo，<T_0, A, 950>，<T_0, B, 2050>，<T_0, commit>

undo，<T_1, C, 700>，<T_1, abort>　　不必要的redo ⇓ checkpoints

检查点：输出日志至稳定存储
输出缓存数据至磁盘
日志<checkpoint L>，L是当前的事务列表
} 执行时update暂停

恢复时，redo列表L中的事务，再对检查点之后事务re/undo

eg.



redo阶段　　undo阶段

恢复算法

两阶段

Redo phase: replay updates of **all** transactions, whether they committed, aborted, or are incomplete

Undo phase: undo all incomplete transactions

↓ 正向 redo　　↑ 逆向 undo

① redo 阶段

从checkpoint record正向扫描：

$\langle T_i, X_i, V \rangle \Rightarrow$ redo

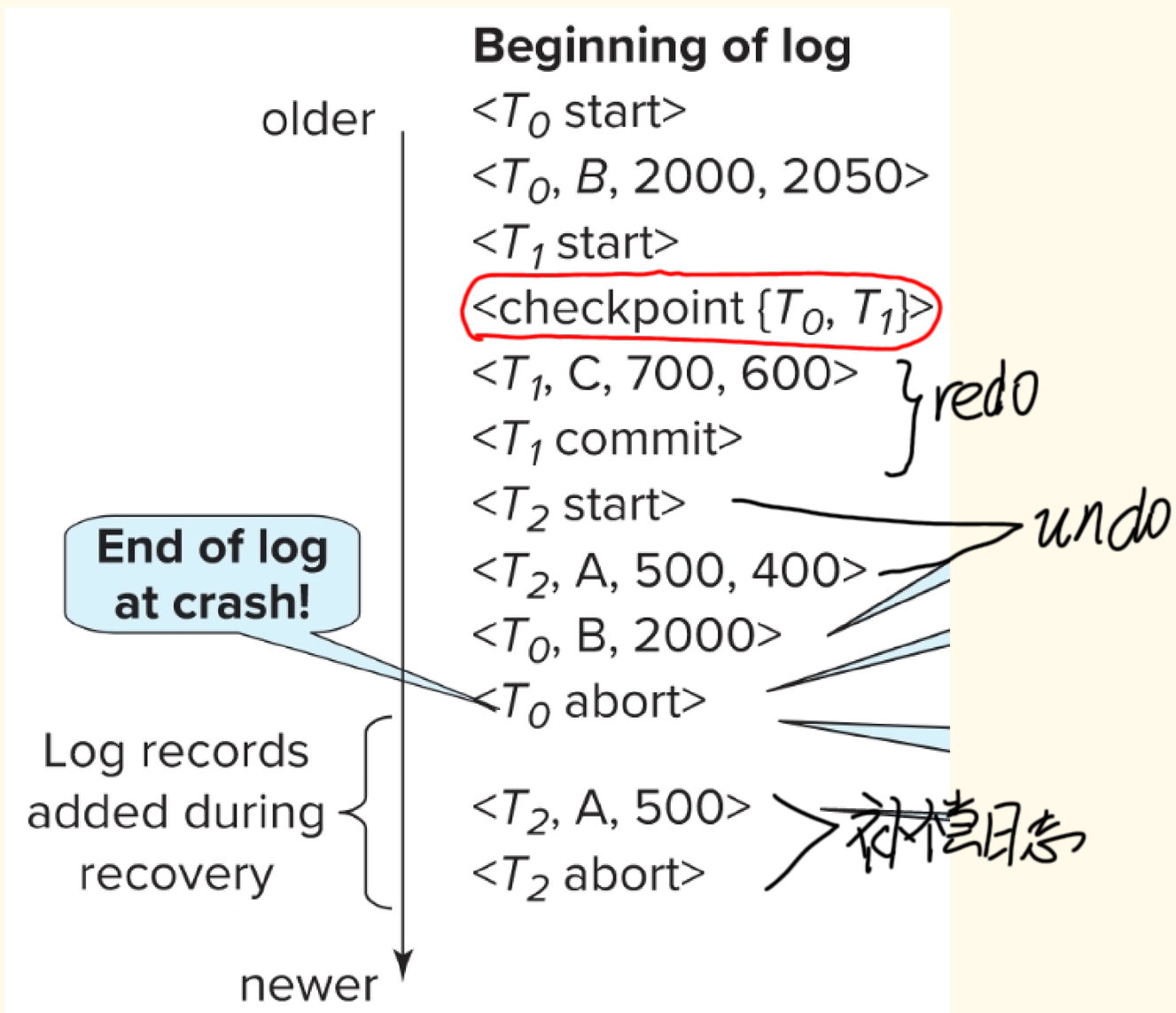
$\langle T_i, \text{start} \rangle \Rightarrow T_i$加至undo-list

$\langle T_i, \text{commit/abort} \rangle \Rightarrow T_i$移出undo-list

② undo阶段

反向扫描，undo-list中事务undo

eg.



日志 buffering

向存储器输出：WAL，日志在前

write-ahead logging or **WAL** rule 日志在前

数据库buffering

非强制策略：commit之后才可

抢占策略：commit之前就可写入磁盘

Fuzzy Checkpoints

在检查点时才向磁盘输出modified块

有上一次checkpoint指针，若有问题，可恢复至上次

2021－2022

1. 若关系模式 R 仅有两个属性，则下列范式级别中，R 最高能满足（　　）。

A. 1NF　　B. 2NF　　C. 3NF　　D. BCNF

$R=(A,B)$ $F=\{A\to B\}$ 或 $\{B\to A\}$ ⇒ 一定满足 BCNF

A为主码　　B为主码

2. DBMS 中的并发控制子系统主要保证事务的哪个特性（　　）。

A．原子性　　B．一致性 隔离性　　C．隔离性　　D．一致性

3. 设关系 r 和 s 的元组数分别为 nr（nr >= 1）和 ns（ns >= 1），则 r⋈ s 的元组数（　　）。

A. >=0 且<=nr×ns　　B. >=1 且<=nr×ns

C. >=0 且<nr×ns　　D. >=1 且< nr×ns

设 $r=(a,b)$，$s=(b,c)$

若 $r(b)$ 与 $s(b)$ 无相同元素，则 $r\bowtie s$ 元组数为0

若 $r=\{(0,1)\}$，$s=\{(1,2),(1,3)\}$，则 $r\bowtie s=\{(0,1,2),(0,1,3)\}$，

元组数为 $nr\times ns$

4. 如果仅需避免“读脏”（dirty read），可将事务隔离级别最低设置为（　）。

A. READ UNCOMMITTED　　B. READ COMMITTED 无读脏

C. REPEATABLE READ　　D. SERIALIZABLE

无读脏、可重复提交　　无读脏、可重复提交、无幻读

数据库系统的三级模式结构分别是怎样的，它如何保证数据独立性？

应用程序交互的外模式、逻辑模式、存储结构相关的内模式

逻辑独立性　　物理独立性

2. 考虑如下表所示的 T1 和 T2 两个事务的并发调度：（10 分）

T1	T2
read (A)	
	read (B)
	read (A)
	write (A)
	write (B)
write (A)	
	commit
commit	

（1）判断该调度是否是冲突可串行化的调度，并给出简单理由。

不是，无法通过仅交换非冲突指令，与串行调度等价

（2）请为 T1，T2 加上适当的加减锁指令，使之满足两段锁协议（2PL）。T1,T2 加上加减锁指令后，对其进行并发调度是否可能产生死锁？

不会，T1仅需A，不会产生相互等待。

某小型图书馆管理数据库系统包含如下关系模式：

reader(ID, name, DoB, address) --读者(ID,姓名，出生日期，住址)

title(call_number, ISBN, name, publisher, year) --书目（分类号，ISBN，书名，出版社，出版年份）

book(book_ID, call_number, price) --图书（编号，书目分类号，购价）

borrow(reader_ID, book_ID, borrowing_time, return_time) --借阅（读者ID，图书ID，借出时间，归还时间）

注意，图书馆可馆藏同一书目的多本图书。 ☆ book-ID指定了具体哪一本书
call_number指定了什么书，可能对应多个book_ID

请用**关系代数**完成查询1-3：

3. 查询借过“S10123”号读者所借过全部书目的读者ID；

不能用 $\Pi_{reader\_ID,\ book\_ID} \div \Pi_{book\_ID}$

$$\Pi_{borrow.reader\_ID,\ title.call\_number}(borrow \bowtie borrow \bowtie title) \div$$

$$\Pi_{title}(\sigma_{borrow.reader\_ID=\text{"S10123"}}(borrow \bowtie borrow \bowtie title))$$

7. 按借阅图书数量从高到低地列出所有借过图书的读者的ID，姓名和借阅图书的数量。

```
select reader.ID, reader.name, count(distinct borrow.book_ID) as num
from reader, borrow
where reader.ID = borrow.reader_ID
group by reader.ID, reader.name
order by num desc;
```

防止一个人多次借同一本书

select的属性必须全部出现在group by中

order by必须放最后

2020-2021

3. 五种基本关系代数运算是（　　）。

$A \cap B = A \cup B - (A-B) - (B-A)$

A．并，差，交，选择，连接　　B．并，差，交，选择，投影

C．并，差，选择，投影，自然连接　　D．并，差，选择，投影，笛卡尔积

连接可用θ和X代替

4 数据库中产生数据不一致的根本原因是（　　）。

A.未对数据进行完整性控制　　B.数据存储量过大　　C.数据冗余　　D.数据保护不严格

5. 对由 SELECT-FROM-WHERE-GROUP-ORDER 组成的 SQL 语句，其在被 DBMS 处理时，各子句的执行次序为（　　）。

A. FROM-WHERE-GROUP-SELECT-ORDER　　B. FROM-SELECT-WHERE-GROUP-ORDER

C. SELECT-FROM-GROUP-WHERE-ORDER　　D. SELECT-FROM-WHERE-GROUP-ORDER

设有表（下划线标识的字段为主码）：

员工(emp)｛姓名(eName)，年龄(eAge)，级别(eLvl)｝　　其中级别为从低到高 1～9 的整数

项目(prj){项目号(pNo)，项目名(pName)，项目地点(pCity)，项目金额(pMoney)}

参与(pe){姓名(eName)，项目号(pNo)，岗位(pos)，报酬(pay)}

6．查询参与了“张三”参与的所有项目，但没有参与“李四”参与的任何项目的员工的姓名。

该员工参与的项目与李四参与的项目无交集

```
select e.eName from emp as e
where
    not exists (
        (select p1.pNo from pe as p1 where p1.eName='张三')
        except
        (select p2.pNo from pe as p2 where p2.eName = e.eName)
    )
and
    not exists (
        (select p3.pNo from pe as p3 where p3.eName = e.eName)
        intersect
        (select p4.pNo from pe as p4 where p4.eName='李四')
    );
```

某教材有若干知识点（点号，名称，视频链接），若干题目（题号，题干，难度，答案），若干小节（节号，标题，字数）；每个题目使用一或多个知识点，一个知识点可被多个题目使用；一个知识点属于且只属于一个小节；一个小节包含一到多个知识点。今要建立该教材的数据库。（15分）

知识点（点号，名称，视频链接，节号）

FK: 节号→小节（节号）

小节（节号，标题，字数）

题目（题号，题干，难度，答案）

使用（点号，题号）

3．基于上述的关系模式，用 SQL 写出如下查询：(5 分)

求相关题目数量不足 30 的小节的节号和标题。（“相关题目”指使用了小节包含的知识点的题目）

```
select 节号,标题 from 小节 as X
where 30 > (
    select count(distinct Y.题号)
    from 小节 natural join 使用 as Y
    where Y.节号 = X.节号
)
```

一个题目可能使用了同一小节的多个知识点，有多条记录，需按题号去重来统计题数

2019－2020

1. 在关系数据库设计中，设计关系模式是（ C ）的任务：

A. 需求分析阶段　　B. 概念设计阶段

C. 逻辑设计阶段　　D. 物理设计阶段

需求分析 → 概念设计(E-R图) → 逻辑设计(关系模式)
→ 物理设计 → 数据库实施 → 运行和维护

2. 并发操作会带来哪些数据不一致性（ D ）：

A. 丢失修改、不可重复读、脏读、死锁　　B. 不可重复读、脏读、死锁

C. 丢失修改、脏读、死锁　　D. 丢失修改、不可重复读、脏读

丢失修改：T_1，T_2先后对同一数据修改但未存磁盘，T_1修改被T_2覆盖
脏读：T_2读取T_1修改后数据之后，T_1回滚
不可重复读：由于T_2的修改，T_1先后两次读同一数据内容不相同
幻读：由于T_2的插入操作，T_1先后两次读同一数据的记录数增加

读未提交
读已提交
可重复读
可串行化

3. 数据库系统的核心是（ B ）：

A. 数据库　　B. 数据库管理系统

C. 数据模型　　D. 软件工具

2. 考虑下列两个事务及其调度：

T1	T2
	Read(A)
	Write(A)
Read(A)	
Read(B)	
	Read(B)
	Write(B)

（2） 请使用两段锁来保证上述两事务的冲突可串行化，使用 Lock(A)来表示对数据 A 加锁，使用 ULock(A)表示对数据 A 释放锁。（5 分）

	T_1	T_2
加锁	Lock(A)	
	Read(A)	
	Lock(B)	
	Read(B)	
		Lock(A)
		Lock(B)
解锁	ULock(A)	
	ULock(B)	
		………
		ULock(A)
		ULock(B)

此时T_2等待T_1释放锁