数据库作业答案

助教

2015-01

中山大學

SUN YAT-SEN UNIVERSITY

推荐阅读书目

- 《Oracle PL/SQL最佳实践》，机械工业出版社
- 《**数据挖掘概念与技术**》韩家炜，机械工业出版社
- 《Building the Data Warehouse》W.H.Inmon

第一章

1.8 列出文件处理系统和DBMS的四个主要区别

- Both systems contain a collection of data and a set of programs which access that data. A database management system coordinates both the physical and the logical access to the data, whereas a file-processing system coordinates only the physical access.
- A database management system reduces the amount of data duplication by ensuring that a physical piece of data is available to all programs authorized to have access to it, whereas data written by one program in a file-processing system may not be readable by another program.
- A database management system is designed to allow flexible access to data (i.e., queries), whereas a file-processing system is designed to allow predetermined access to data (i.e., compiled programs).
- A database management system is designed to coordinate multiple users accessing the same data at the same time. A file-processing system is usually designed to allow one or more programs to access different data files at the same time. In a file-processing system, a file can be accessed by two programs concurrently only if both programs have read-only access to the file.

第一章

- **1.9解释物理数据独立性的概念，以及它在数据库系统中的重要性。**
- 物理数据独立性：物理独立性是指用户的应用程序与存储在磁盘上的数据库中数据是相互独立的。即，数据在磁盘上怎样存储由DBMS管理，用户程序不需要了解，应用程序要处理的只是数据的逻辑结构，这样当数据的物理存储改变了，应用程序不用改变。
- 重要性：实现应用程序与存储在磁盘上的数据相分离，应用程序不依赖与物理模式，不随物理模式的改变而改变。

第二章

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

- **2.9考虑图2-15所示银行数据库系统**
- （a）适当的主码是什么？
- branch(branch_name, branch_city, asset
- customer (customer_name, customer_street, customer_city)
- loan (loan_number, branch_name, amount)
- borrower (customer_name, loan_number)
- account (account_number, branch_name, balance)
- depositor (customer_name, account_number)
- （b）给出你选择的主码，确定适当的外码。
- loan: branch_name references branch
- account: branch_name references branch
- borrower: customer_name references customer
- loan_number references loan
- depositor: customer_name references customer
- account_number references account

第二章

- **2.10 考虑图2-8所示advisor关系，advisor的主码是s_id。假设一个学生可以有多位指导老师。那么s_id还是advisor关系的主码吗?如果不是，advisor的主码会是什么呢？**
- **答案**：不能，s_id不再是advisor的主码。因为可能存在多个元组有着相同的s_id，此时s_id不能用来区别不同的元组。advisor的主码应该是s_id,i_id。

- **2.11 解释术语关系和关系模型在意义上的区别**
- **答案**：关系模式（relation schema）是一种定义类型，关系（relation）是这种类型的一个实例

- **2.12考虑图2-14所示关系数据库。给出关系代数表达式来表示下列每一个查询：**

employee(*person-name*, *street*, *city*)
works(person-name, *company-name*, *salary*)
company(*company-name*, *city*)

- **a.找出为"First Bank Corporation"工作的所有员工姓名。**
- **b. 找出为 "First Bank Corporation"工作的所有员工的姓名和居住城市**
- **c. 找出为 "First Bank Corporation"工作且挣钱超过10 000美元的所有员工的姓名、街道地址和居住城市。**

第二章

- 答案：

a. $\Pi_{person\text{-}name}(\sigma_{company\text{-}name = \text{“First Bank Corporation”}}(works))$

b. $\Pi_{person\text{-}name,\, city}(employee \bowtie (\sigma_{company\text{-}name = \text{“First Bank Corporation”}}(works)))$

c. $\Pi_{person\text{-}name,\, street,\, city}(\sigma_{(company\text{-}name = \text{“First Bank Corporation”} \wedge salary > 10000)} works \bowtie employee)$

- **2.13 考虑2-15所示银行数据库。对于下列每个查询，给出一个关系代数表达式：**
- **a. 找出贷款额度超过10 000美元的所有贷款号**
- **b. 找出所有这样的存款人姓名，他拥有一个存款额大于6000美元的账户**
- **c. 找出所有这样的存款人的姓名，他在“Uptown”支行拥有一个存款额大于6000美元的账户**

第二章

- **答案：**
- a.$\Pi_{loan\_number}(\sigma_{amount>10000}(loan))$
- b.$\Pi_{customer\_name}(\sigma_{balance>6000}(depositor \bowtie account))$
- c.$\Pi_{customer\_name}(\sigma_{(branch\_name=”Uptown” \wedge balance>6000)}(depositor \bowtie account))$

第三章

- **3.11 使用大学模式, 用SQL写出如下查询.**
- **a. 找出所有至少选修了一门 Comp.Sci.课程的学生姓名，保证结果中没有重复的姓名**
- **b. 找出所有没有选修在 2009 年春季之前开设的任何课程的学生的ID和姓名**
- **c. 找出每个系老师的最高工资值。可以假设每个系至少有一位老师**
- **d. 从前述查询所计算出的每个系最高工资中选出最低值**
- 答案：
- a. select name
- from student natural join takes natural join course
- where course.dept = 'Comp. Sci.' ;
- b. select id, name from student
- except
- select id, name from student natural join takes
- where year < 2009 ;
- C. select dept, max(salary)
- from instructor
- group by dept ;
- d. select min(maxsalary)
- from (select dept, max(salary) as maxsalary from instructor
- group by dept) ;

第三章

- **3.12 使用大学模式，用SQL写出如下查询。**
- **a. 创建一门课程“CS-001”，其名称为“Weekly Seminar”，学分为0**
- **b. 创建该课程在2009年秋季的一个课程段，sec_id为1**
- **c. 让Comp.Sci. 系的每个学生选修上述课程段**
- **d. 删除名为Chavez的学生选修上述课程段的信息**
- **e. 删除课程CS-001。如果在运行此删除语句之前，没有先删除这门课的授课信息（课程段），会发生什么事？**
- **f. 删除课程名称中包含“database”的任意课程的任意课程段所对应的所有takes元组，在课程名的匹配中忽略大小写**
- 答案：
- a. insert into course
- values (’CS-001’, ’Weekly Seminar’, ’Comp. Sci.’, 0) ;
- b. insert into section
- values (’CS-001’, 1, ’Autumn’, 2009, null, null, null) ;
- C. insert into takes
- select id, ’CS-001’, 1, ’Autumn’, 2009, null
- from student
- where dept name = ’Comp. Sci.’ ;

第三章

- d. delete from takes
- where course id= ’CS-001’ and section id = 1 and
- year = 2009 and semester = ’Autumn’ and
- id in (select id from student where name = ’Chavez’) ;
- e. delete from course
- where course_id = ‘CS-001’;
- (会出现违反外 键规则。因为section有外键指向course，如果直接从course删除CS-001的信息，此时section中关于CS-001的字段在course中找不到关联)
- f. delete from takes
- where course id in
- (select course id from course
- where lower(title) like ’%database%’) ;

- **3.13 写出对应于图3-18中模式的SQL DDL.在数据类型上做合理的假设,确保声明主码和外码.**

person (*driver_id*, *name*, *address*)
car (*license*, *model*, *year*)
accident (*report_number*, *date*, *location*)
owns (*driver_id*, *license*)
participated (*report_number*, *license*, *driver_id*, *damage_amount*)

第三章

- **3.13 写出对应于图3-18中模式的SQL DDL.在数据类型上做合理的假设,确保声明主码和外码.**
- **答案：**

```
create table person
  (driver_id varchar(50),
  name varchar(50),
  address varchar(50),
  primary key (driver_id));
create table car
  (license varchar(50),
  model varchar(50),
  year integer,
  primary key (license));
create table accident
  (report_number integer,
  date date,
  location varchar(50),
  primary key (report_number));
```

```
create table owns
  (driver_id varchar(50),
  license varchar(50),
  primary key (driver_id,license)
  foriegn key (driver_id)
  references person
  foriegn key (license) references car);
create table participated
  (report_number integer,
  license varchar(50),
  driver_id varchar(50),
  damage_amount integer,
  primary key (report_number,license)
  foriegn key (license) references car
  foriegn key (report_number)
  references accident));
```

第三章

- **3.14 考虑图3-18中的保险公司数据库，其中加下划线的是主码。对这个关系数据库构造如下的SQL查询：**
- **a. 找出在“John Smith”的车有关的交通事故的数量**
- **b. 对事故报告编号为“AR2197”中的车牌是“AABB2000”的车辆损坏保险费用更新到3000美元**
- **答案：**
- a.select count(*)
- from accident
- where exits(
- select *
- from participated,owns,person
- where owns.driver_id=person.driver_id
- and person.name='John Smith'
- and owns.license=participated.license
- and accident.report_number=participated.report_number)
- b. update participated
- set damage_amount=3000
- where report_number=”AR2197” and license=”AABB2000”

第三章

- **3.15 考虑图3-19中的银行数据库，其中加下划线的是主码。对这个关系数据库构造如下的SQL查询：**
- **a. 找出在“Brooklyn”的所有支行都有账户的所有客户**
- **b. 找出银行的所有贷款额的总和**
- **c. 找出总资产至少比位于Brooklyn的某一家支行要多的所有支行名字**
- **答案：**
- a.with branchcount as
- (select count(*) from Branch Where branch_city=’Brooklyn’)
- select customer_name from customer c where branchcount=
- (select count (distinct branch_name) from(customer natural join depositor natural join account natural join branch)as d where d.customer_name= c.customre_name)
- b. select sum(amount) from loan
- c.select branch_name
- from branch where assets > some
- (select assets
- from branch
- where branch_city=’Brooklyn’)

第四章

- **4.12 对于图4-11中的数据库,写出一个查询来找到那些没有经理的雇员。注意一个雇员可能只是没有列出其经理，或者可能有null经理。使用外连接书写查询，然后不用外连接再重写查询。**
- **答案：**
- 使用外连接：
- select employee_name
- from employee **natural left outer join** manages
- where manager_name is null;
-
- 不使用外连接：
- select emplyee_name
- from employee
- where not exists
- (select employee_name
- from manages
- where manages.employee_name = emplyee.employee_name
- and manages.manager_name is not null);

第四章

- **4.12 对于图4-11中的数据库,写出一个查询来找到那些没有经理的雇员。注意一个雇员可能只是没有列出其经理，或者可能有null经理。使用外连接书写查询，然后不用外连接再重写查询。**
- **答案：**
- 使用外连接：
- select employee_name
- from employee **natural left outer join** manages
- where manager_name is null;
-
- 不使用外连接：
- select emplyee_name
- from employee
- where not exists
- (select employee_name
- from manages
- where manages.employee_name = emplyee.employee_name
- and manages.manager_name is not null);

第四章

- **4.13 在什么情况下，查询**
- **select ***
- **from student natural full outer join takes natural full outer join course**
- **将包含在属性titles上取空值的元组？**
- **答案：**
- 情况一：course元组中存在title属性为null的元组
- 情况二：存在一个学生，没有上任何课(即takes里面没有此学生的ID)

- **4.14 给定学生每年修到的学分总数，如何定义视图tot_credits(year, num_credits)**
- **答案：**
- **create view** tot_credits(year, num_credits)
- **as**
- (select year, sum(credits)
- from takes natural join course
- group by year)

第六章

6.10

- a. ∏name (student ⋈takes ⋈ ∏course_id (ϭdept_name = ‘Comp.Sci.’(course)))
- b. ∏ID,name (student) −∏ID,name (ϭyear<2009(student⋈takes))
- c. dept_namegmax(salary)(instructor)
- d. gmin(maxsal)(dept_namegmax(salary) as maxsal(instructor))

6.11

- a. ∏person_name (ϭcompany_name = “First Bank Corporation” (works))
- b. ∏person_name, city (employee ⋈ (ϭcompany_name = “First Bank Corporation” (works)))
- c. ∏person_name, street, city(ϭ (company_name =“First Bank Corporation”∧ salary > 10000)(works ⋈ employee))
- d. ∏person_name (employee ⋈ works ⋈ company)
- e. ∏company_name(company÷(∏city(ϭcompany_name = “Small Bank Corporation” (company))))

6.12

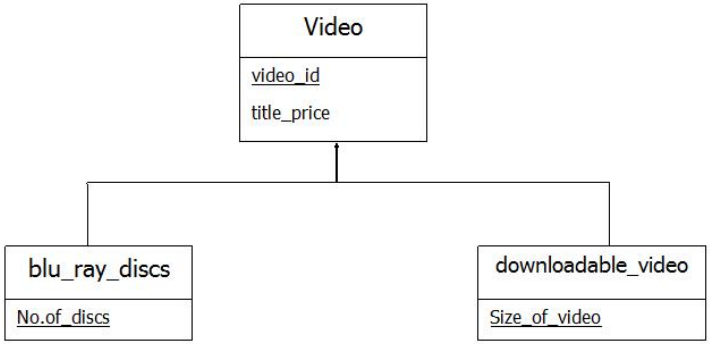
- a.ϭinstrcnt>1 (course_id,section_id,year ,semestergcount(*) as instrcnt(teaches))
- b.∏course_id,section_id,year,semester(ϭID<>other_ID(takes⋈ptakes1(ID2,course_id,section_id,year ,semester)(takes)))

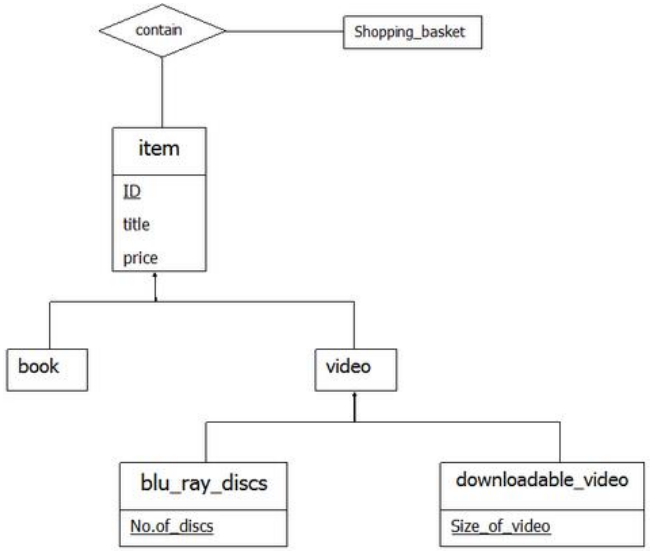
第七章

7.20 (a)

- author (name, address, URL)
- book (ISBN, title, year, price)
- customer (email, name, address, phone)
- publisher (name, address, phone, URL)
- shopping_basket (basket_id)
- warehouse (code, address, phone)

7.20 (b)(c)





第八章

- 8.6
- 因为A->BC,所以有A->B和A->C；
- 因为A->B 和B->D,所以 A->D；
- 因为 A->CD和CD->E ,所以 A->E；
- 又因为 A->A，所以综上有A->ABCDE；
- 因为 E->A，所以由传递性E->ABCDE；
- 因为 CD->E ,同理有 CD->ABCDE；
- 因为 B->D和BC->CD,所以 BC->ABCDE
- 还有C->C，D->D, BD->D。
- 所以F的闭包有BD->B，BD->D, C->C，D->D, BD->BD, B->D,B->B，B->BD
- R的候选码为A, BC, CD, E。
- 8.27
- 由result = {B}，以及F中的类似β → γ形式的FDs，发现满足β ⊆ result的是B → B和B → D。所以result = {B，D}。所以B+ = {B，D}。
- 8.30
- (1)无损连接分解，维持数据库的正确性。
- (2)保持依赖分解，快速检查更新的正确性
- (3)最小化重复信息，尽可能地使用最小的空
- 8.31
- 因为BCNF并不总是保持依赖的，有些情况下BCNF的分解会妨碍对某些函数依赖的高效检查，BCNF也可能会保留一些冗余，所以有时候也会选择其它范式，比如3NF。
-

第十四章

- 14.12
- 原子性：原子性的用途在于保证程序执行的步骤集合作为一个单一的、不可分割的单元出现，使事务不可分割，要么执行其全部内容，要么不执行。
- 隔离性：该特性确保事务正常执行而不被来自并发执行的数据库语句所干扰。
- 持久性：持久性使得事务的操作在系统崩溃后也是持久的。
- 一致性：一致性要求一个事物作为原子从一个一致的数据库状态开始独立地运行，且事务结束时数据库也必须再次是一致的。
- 14.13
- （1）活动状态 -> 部分提交状态 -> 提交状态
- 事务开始执行，执行完最后一条语句，并成功完成输出。
- （2）活动状态 -> 部分提交状态 -> 失败状态 -> 中止状态
- 事务开始执行，执行完最后一条语句，发生硬件或逻辑故障使其无法输出，正常执行不能继续，事务回滚到初始状态。
- （3）活动状态 -> 失败状态 -> 中止状态
- 事务开始执行，在执行语句操作中发生硬件或逻辑错误，正常的执行不能继续，事务回滚到初始状态。

第十四章

- 14.14
- 串行调度是指一次执行一个事务，每个事务仅当前一个事务执行完成后才开始。同一时刻内只有一件事务处于执行中。
- 可串行化调度是指，在并发执行中，通过保证所执行的任何调度的效果都与没有并发执行的调度效果一样，使得调度在结果上等价于串行调度。其本质仍然是并行的。
- 14.15
- a. 无论T13或T14哪个事务先执行，所有可能令A或B不为0的操作的执行条件，都是A或B其中一个的值为0。因此两个事务的每一个串行执行都满足一致性需求，因此保持数据库的一致性。
- b .

T13	T14
Read(A)	
	Read(B) Read(A)
Read(B) If A=0 then B=B+1	
	If B=0 then A=A+1 Write(A)
Write(B)	

- c.不存在。从a题我们知道一个可串行化调度产生A=0VB=0.假设我们从T13的read(A)指令开始，当调度完成后，无论什么时候执行 T14，B都为1.再假设我们先执行T14。那么T14的Read(B)将读取一个0.所以当T14结束时，A=1。于是A=0VB=0≡FVF≡F。从 T14开始同理。

第十五章

15.2

有可能引起死锁：

T34	T35
lock-S(A);	
read(A);	
	lock-S(B);
	read(B);
	lock-X(A);
lock-X(B)	

15.3

- 强封锁协议带来一定的好处，两阶段封锁协议实现了事务集的串行化调度，但同时，一个事务的失败可能会引起一连串事务的回滚。为避免这种情况的发生，强封锁协议可以进一步加强对两阶段封锁协议的控制。
- 强两阶段封锁协议，要求事务提交之前不得释放任何锁。使用锁机制的数据库系统，要么使用严格两阶段封锁协议，要么使用强两阶段封锁协议。而严格两阶段封锁 协议除了要求封锁是两阶段之外，还要求事务持有的所有排它锁必须在事务提交之后方可释放。这个要求保证未提交事务所写的任何数据，在该事务提交之前均以排 它锁封锁，防止其他事务读取这些数据。

15.20：

- 严格两阶段封锁协议的好处是保证未提交事务所写的任何数据，在该事务提交之前均以排它锁封锁，防止其他事务读取这些数据。弊端是并行度不高。

15.23：

- 当我们可能会进入不一致的状态时，避免死锁比允许死锁发生然后检测的方式代价更小。因为死锁可以通过回滚事务加以解决，而不一致状态可能引起现实中的问题，这是数据库系统不能处理的。

15.25

- 当事务给一个节点加显示锁，其所有后代节点都被加上隐式锁，两种锁的加锁方式不同，检测锁是否存在的方式不同，锁本身的效果和功用没有区别。

Thanks for Your Attention!