# 浙江大学 2020 - 2021 学年 春夏 学期

## 《数据库系统》课程期末考试试卷参考答案和评分标准

课程号：21121350， 开课学院：计算机学院

考试试卷：√A 卷、B 卷（请在选定项上打√）

考试形式：√闭、开卷（请在选定项上打√），允许带一张 A4 纸笔记入场

考试日期：2021 年 7 月 2 日，考试时间：120 分钟

**诚信考试，沉着应考，杜绝违纪。**

**考生姓名：_____学号：_____所属院系：_____**

| 题序 | 一 | 二 | 三 | 四 | 五 | 六 | 七 | 八 | 总 分 |
|---|---|---|---|---|---|---|---|---|---|
| 得分 | | | | | | | | | |
| 评卷人 | | | | | | | | | |

## Problem 1: Relational Model and SQL (18 points)

Following are the relational schemas of a SRTP (Student Research Training Program) project database.

student (sId, sName, dId)
teacher (tId, tName, dId)
department (dId, dName)
project (pId, pName, tId, startTime, endTime)
participate (pId, sId, role)

The underlined attributes are primary keys, and foreign keys are listed as follows:
"dId" in "student" references "department";
"dId" in "teacher" references "department";
"tId" in "project" references "teacher";
"pId" and "sId" in "participate" reference "project" and "student", respectively.
In "participate", only two different roles are permitted: "leader" and "member". Based on the above relational schemas, please answer the following questions:

(1) Write a relational algebra expression to find the names of the projects that are instructed by a teacher from the department "Computer Science". (4 points)
(2) Write SQL statements to create tables project and participate with all the necessary

constraints (Note: Tables student, teacher, and department have already been created and can be referenced). (6 points)

(3) Write a SQL statement to find the names of the teachers that instruct at least one project started in the year 2020. (4 points)

(4) Write a SQL statement to find the names of the students participating more than 2 projects. (4 points)

**Answers of Problem 1:**

(1)

$\prod_{pName}(project \bowtie teacher \bowtie (\sigma_{dName=\text{“Computer Science”}}(department))$

(2)

```
CREATE TABLE project
    (pId    char(10),
    pName    varchar(20),
    tId    char(10),
    startTime    date,
    endTime    date,
    primary key (pId),
    foreign key (tId) references teacher);

CREATE TABLE participate
    (pId    char(10),
    sId    char(10),
    role    varchar(20),
    primary key (pId, sId),
    foreign key (pId) references project,
    foreign key (sId) references student,
    check (role=“leader” or role=“member”));
```

(3)

```
select distinct tName
from project, teacher
where project.tId=teacher.tId and startTime between '2020-01-01' and '2020-12-31'
```

(4)

```
select sName
from student
where sId in
    (select sId
    from participate
    group by sId
    having count(pId) > 2)
```

## Problem 2: E-R Model (11 points)

Based on the SRTP project management scenario in Problem 1, some new requirements are added as follows:

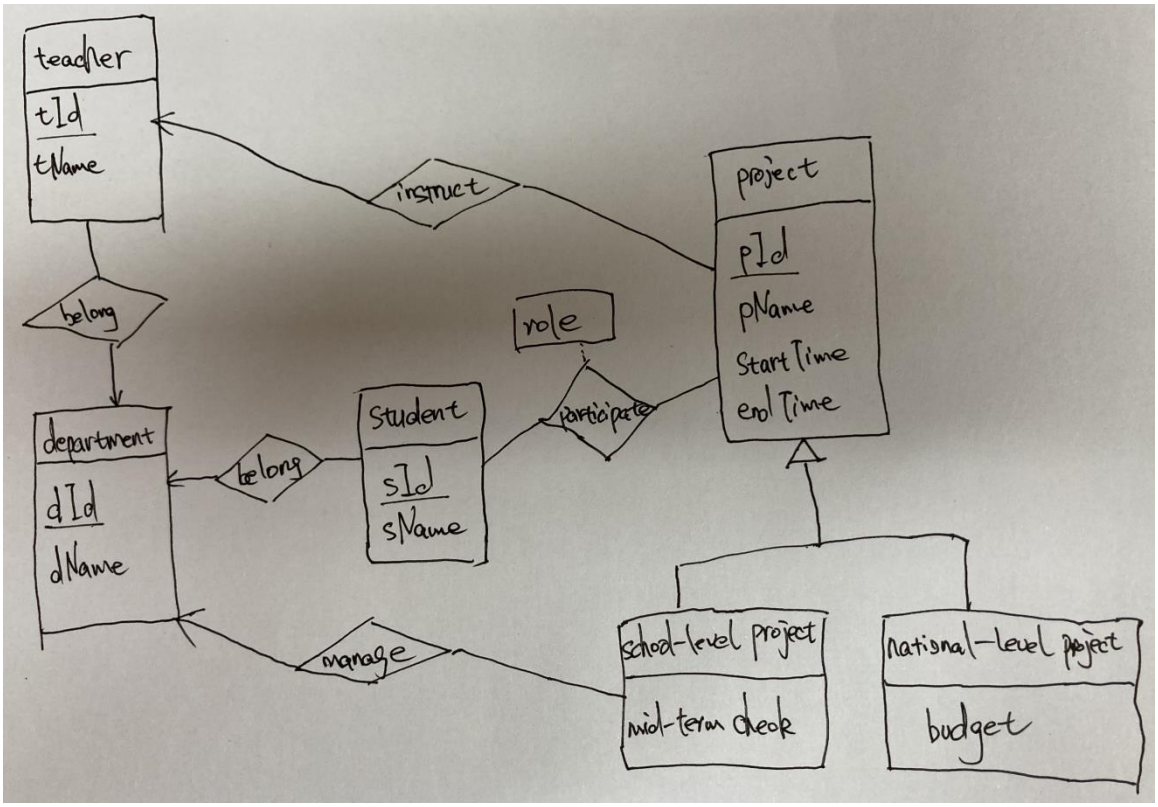
(1) There are two kinds of SRTP projects, i.e., school-level projects and national-level projects, and a project is either school-level or national-level.

(2) National-level projects have budget information, and school-level projects have mid-term check information.

(3) A school-level project is associated with exactly a department that is in charge of the management of the project.

Please draw an E-R diagram for the scenario.

**Answers of Problem 2:**




## Problem 3: Relational Formalization (12 points)

For relation schema R (A, B, C, D, E, F) with functional dependencies set F = {A->B, A->C, B->C, D->E, D->F, EF->D}. Answer the following questions:

(1) Find all the candidate keys. (3 points)

(2) Find the canonical cover Fc. (3 points)

(3) If R is not in BCNF, decompose it into BCNF schemas. (4 points) Is this decomposition dependency preserving? (2 points)

**Answers of Problem 3:**

(1)
AD AEF

(2)
A->B, B->C, D->EF, EF->D

(3)
There are different decomposition results and the following is just an example.
R1=(A, B), R2=(A, C, D, E, F) (A->B)
R21=(A, C), R22=(A, D, E, F) (A->C)
R221=(D, E), R222(A, D, F) (D->E)
R2221(D, F), R2222(A, D) (D->F)
This decomposition is not dependency preserving (e.g., B->C is not preserved).

Following is another solution:
R1=(B, C), R2=(A, B, D, E, F) (B->C)
R21=(A, B), R22=(A, D, E, F) (A->B)
R221=(D, E, F), R222(A, D) (D->EF)
This decomposition is dependency preserving, because A->B can be checked on R21, B->C can be checked on R1, D->EF and EF->D can be checked on R221.

## Problem 4: XML (8 points)

The following is a simplified DTD for the SRTP project database given in Problem 1:

```
<!DOCTYPE SRTP[
    <!ELEMENT SRTP(department+, teacher+, student+, project*)>
    <!ELEMENT department (dname)>
    <!ATTLIST department dId ID #REQUIRED>
    <!ELEMENT teacher (tname)>
    <!ATTLIST teacher
        tId ID #REQUIRED
        dId IDREF #REQUIRED>
    <!ELEMENT student (sname)>
    <!ATTLIST student
        sId ID #REQUIRED
        dId IDREF #REQUIRED>
    <!ELEMENT project (pname, starttime, endtime)>
    <!ATTLIST project
        pId ID #REQUIRED
        tId IDREF #REQUIRED
        sIds IDREFS #REQUIRED >
```

```
        <!ELEMENT     dname (#PCDATA)>
        <!ELEMENT     tname (#PCDATA)>
        <!ELEMENT     sname (#PCDATA)>
        <!ELEMENT     pname(#PCDATA)>
        <!ELEMENT     starttime(#PCDATA)>
        <!ELEMENT     endtime(#PCDATA)>
    ]>
```

Please answer the following questions:

(1) Give an XPath expression to return the names of all the teachers who supervise SRTP projects. (4 points)

(2) Give an XQuery expression to return all the projects and their corresponding instructors, in the form of project_instructor elements that have a project subelement and a teacher subelement. (4 points)

**Answers of Problem 4:**

(1)

```
/SRTP/project/id(@tId)/tname/text()
```

(2)

```
for  $p in /SRTP/project,
     $t in /SRTP/teacher,
where $p/@tId= $t/@tId
return <project_instructor> { $p $t } </project_instructor>
```

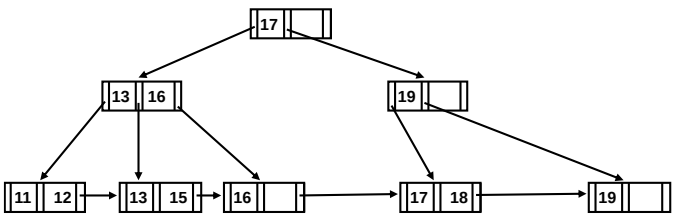
## Problem 5: B+ -Tree and Query Processing (10 points)

Table student in Problem 1 is stored sequentially on sId. The following B+-tree is built for the table on attribute dId. Please answer the following questions:

(1) Is the built index a primary index? Why? (2 points)

(2) Draw the B+-tree after inserting entry 14. (4 points)

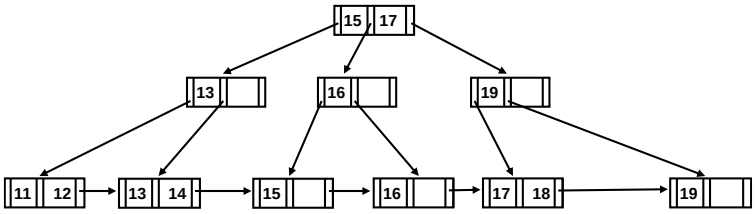
(3) Draw the B+-tree after deleting entry 19 from the original B+-tree. (4 points)



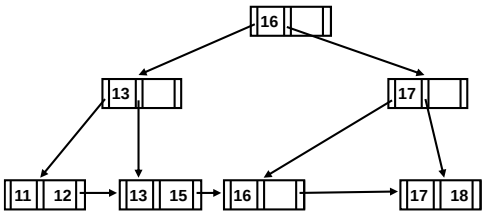
**Answers of Problem 5:**

(1)
The built index is not a primary index, as the search key of the index is not the search key of the sequentially ordered data file.

(2)




(3)



## Problem 6: Query Processing (14 points)

There are two relations *r* (100 blocks) and *s* (20 blocks), and hash-join algorithm is used to perform natural join between these two relations (memory size M=6 blocks). Please answer the following questions:

(1) How many partitions can be constructed? Why? (3 points)
(2) Which relation is best to choose as the build relation? Why? (3 points)
(3) Is recursive partition needed? Why? (3 points)
(4) Please compute the cost (numbers of seeks and block transfers) of the hash-join. (5 points)

**Answers of Problem 6:**

(1)
5 partitions, as the number of partitions is M-1.

(2)
Relation *s*, as relation *s* is smaller than relation *r*.

(3)
Recursive partition is not needed, as the size of the partitions of relation *s* (i.e., 4) is less than or equal to M-2 (i.e., 4).

(4)
Number of block transfers: $3\times(100+20)+4\times5$
Note: $4\times5$ is not necessary, which considers partially filled blocks.
Number of seeks: $2\times(100+20)+2\times5$

- If recursive partitioning is not required: cost of hash join is
  $3(b_r + b_s) + 4 * n_h$ block transfers +
  $2(\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil)$ seeks
- If recursive partitioning required:
  - number of passes required for partitioning build relation $s$ to less than $M$ blocks per partition is $\lceil \log_{\lfloor M/bb \rfloor - 1}(b_s/M) \rceil$
  - best to choose the smaller relation as the build relation.
  - Total cost estimate is:
    $2(b_r + b_s)\lceil \log_{\lfloor M/bb \rfloor - 1}(b_s/M) \rceil + b_r + b_s$ block transfers +
    $2(\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil)\lceil \log_{\lfloor M/bb \rfloor - 1}(b_s/M) \rceil$ seeks
- If the entire build input can be kept in main memory no partitioning is required
  - Cost estimate goes down to $b_r + b_s$.

## Problem 7: Concurrency Control (13 points)

Given the following schedule, please answer the following questions:

(1) Draw the precedence graph for the schedule. (3 points)

(2) Is the schedule conflict serializable? Why? (2 points)

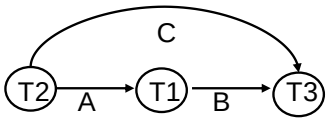
(3) Is it possible that the schedule is generated by the 2PL protocol with lock conversions? Explain. (5 points)

(4) Which conditions should be satisfied if we want the schedule to be recoverable? (3 points)

| T1 | T2 | T3 |
|---|---|---|
| | read C | |
| read B | | |
| | write C | |
| | read A | |
| | | read C |
| | write A | |
| read A | | |
| | | write C |
| write B | | |
| | | read B |

**Answers of Problem 7:**

(1)




(2)

The schedule is conflict serializable, as the precedence graph is acyclic.

(3)

It is possible that the schedule is generated by the 2PL protocol with lock conversions.

| T1 | T2 | T3 |
|---|---|---|
| | Lock-S (C)<br>read C | |
| Lock-S (B)<br>read B | | |
| | Upgrade (C)<br>write C | |
| | Lock-X (A)<br>read A | |
| | UL (C) | Lock-S (C)<br>read C |
| | write A | |
| | UL (A) | |
| Lock-S (A)<br>read A | | |
| | | Upgrade (C)<br>write C |
| Upgrade (B)<br>write B | | |
| UL (A, B) | | Lock-S (B)<br>read B |
| | | UL (B, C) |

(4)

T1 must commit before T3 does.

T2 must commit before T1 does.

T2 must commit before T3 does.

## Problem 8: Recovery (14 points)

Given the following log file that supports logical undo, please answer the following questions:

(1) The system crashes just after the last log record. What are the values of B and C in the database after system crash? (3 points)
(2) Which transactions should redo and undo, respectively? (3 points)
(3) What are the start and end points for redo and undo, respectively? (3 points)
(4) What are the log records added during recovery? (5 points)

1 <$T_0$ start>
2 <$T_0$, B, 2000, 2050>
3 <$T_1$ start>
4 <$T_1$, B, 2050, 2100>
5 <$T_1$, $O_1$, operation-begin>
6 <checkpoint {$T_0$, $T_1$}>
7 <$T_1$, C, 700, 400>
8 <$T_0$ commit>
9 <$T_1$, $O_1$, operation-end, (C, +300)>
10 <$T_2$ start>
11 <$T_2$, $O_2$, operation-begin>
12 <$T_2$, C, 400, 300>
13 <$T_2$, $O_2$, operation-end, (C, +100)>
14 <$T_2$, commit>

**Answers of Problem 8:**

(1)
B=2100
C= 300 or 400 or 700

(2)
redo: $T_0$ and $T_2$ undo: $T_1$

(3)
redo: 7-14 undo: 14-3

(4)
<$T_1$, C, 600>
<$T_1$, $O_1$, operation-abort>
<$T_1$, B, 2050>
<$T_1$, abort>