

VANCOUVER ISLAND UNIVERSITY
CSCI 370 — FINAL EXAMINATION
17 April 2009, 13:00 — 16:00

TO BE ANSWERED IN BOOKLETS

DURATION: 3 Hours

INSTRUCTOR: H. Liu

Instructions

- Students must count the number of pages in this examination paper before beginning to write, and report any discrepancy immediately to the invigilator.
- This examination paper consists of eight pages.
- This is a CLOSED BOOK — NO NOTES examination.
- Calculators are NOT permitted.
- Remember to state any assumptions and show rough work.
- Note carefully the weight of each question, and answer appropriately.
- Attempt all questions. All questions relate to material covered in the lectures, labs and assignments.

(This page is left empty intentionally.)

Note: Question 1, 2, 3, and 4 refer to the following relational schema, which describes a hypothetical database used by a university to manage the maintenance projects. In the schema, the primary key of each relation is underlined.

```
Buildings(bno, buildingName, constructionDate)
Companies(cid, companyName, contactInfo, specialty)
Projects(pno, bno, description, budget, dateProposed)
Contracts(cno, pno, cid, description, expense, startDate, completeDate)
```

- Each record in the relation **Buildings** describes a building. Each building is identified by a unique building number (bno), and has a name (buildingName), which is also unique, and a date indicating when the building was first constructed (constructionDate).
- Each record in the relation **Companies** describes a company. Each company has a unique id (cid), a name (companyName), its contact information (contactInfo) and a description about the company's specialties (specialty).
- Each record in the relation **Projects** describes a maintenance project. Each project is identified by a unique number (pno). Each project record also describes the details of this maintenance project (description), on which building (identified by bno) the project was or will be done, the budget for this project (budget), and the date the project was proposed (dateProposed). You can safely assume that each project has been either completed or just proposed (i.e., no contract has been recorded for it).
- One maintenance project could be done through several contracts. Each record in the relation **Contracts** records a contract between the university and a company (identified by cid). The contract record also shows a description of the details of the contract (description), the expense (expense) and the start and finish date of the contract (startDate and completeDate). Each contract is given a unique number (cno) and is linked to a maintenance project identified by pno.

You might find the following partial class declaration useful in the exam:

```
class Environment {
public:
    static Environment * createEnvironment();
    static void terminateEnvironment(Environment *env);
    Connection * createConnection(const string &userName,
                                const string &password,
                                const string &connectString = "");
    void terminateConnection(Connection *connection);
}

class Connection
{
public :
    Statement* createState( const string &sql = "");
    void terminateStatement(Statement *statement);
    void commit();
    void rollback();
}

class Statement
{
public:
    ResultSet * executeQuery( const string &sql = "");

    unsigned int executeUpdate( const string &sql = "");
    void closeResultSet(ResultSet *resultSet);
}

class ResultSet
{
public:
    bool next();
    int getInt(unsigned int colIndex);
    string getString(unsigned int colIndex);
}
```

1. (5 marks) In the relation *Building*, instead of *constructionDate*, we can also use *buildingAge* to store the same information. Which choice is better? Why?

Solution:

It is better to use *constructionDate* than *buildingAge*, because *buildingAge* changes every year while *constructionDate* stays the same. If the *buildingAge* is used, then the data stored in the database would need to be updated every year.

2. (25 marks) Express each of the following queries in a single SQL statement.
- (a) For each maintenance project done or proposed to be done on each building, list the building's name, the project's description and the project's budget. Order the result according to the building's name ascending first, then the budget descending.

Solution:

```
select buildingName, description, budget
from Projects P join Buildings B on P.bno = B.bno
order by buildingName, budget desc;
```

- (b) List the name of each building that has the word 'Science' in its name and that has never had any maintenance project done on or proposed for it.

Solution:

```
select buildingName
from Buildings
where buildingName like '%Science%'
and bno not in (select bno
                from Projects);
```

- (c) List the name of each building which is constructed in or after year 2000 and has already had at least one maintenance project done on it or proposed for it. Duplicates should be eliminated from the result.

Solution:

```
select buildingName
from Buildings
where to_char(constructionDate, 'yyyy') > '2000'
and bno in (select bno
            from Projects);
```

Or

```
select distinct buildingName
from Buildings B join Projects P on B.bno = P.bno
where to_char(constructionDate, 'yyyy') > '2000';
```

- (d) For each building, list the building's name and the number of maintenance projects done or proposed on it, and name the second column as `totalProjectNumber`. If the building does not have any project, list the number as 0.

Solution:

```
select buildingName, count(pno) as totalProjectNumber
from Buildings B left join Projects P on B.bno = P.bno
group by B.buildingName;
```

- (e) For each project that is over budget, list the building's name, the project's description and budget, and the total amount of money spent on this project (which should be the sum of the expenses of all the contracts linked to this project).

Solution:

```
select buildingName, P.description, budget, sum(expense) as totalExpenses
from Project P, Buildings B, Contracts C
where P.bno = B.bno and P.pno = C.pno
group by P.pno, B.buildingName, P.description, P.budget
having sum(expense) > budget;
```

3. (10 marks) Express the following query in relational algebra and Datalog respectively:

For each building that is newer than the building named 'Library' (compared by their constructionDate), list the name of the building, and the description, budget, and the date the project was proposed of all the maintenance projects done or proposed on the building.

Solution:

Note: P means project operator, S means select operator and
J means join operator

Relational Algebra query:

```
P_{buildingName, description, budget, dataProposed}
(Projects J_(bno) (Buildings J_(constructionDate > conDate)
(P_{constructionDate as conDate} S_(buildingName = 'Library') Buildings)))
```

Datalog query:

```
Result(buildingName, description, budget, dateProposed) :-
exists bno1, conDate1, conDate2,
Buildings(bno1, buildingName, conDate1) and Buildings(_, 'Library', conDate2)
and Projects(_, bno1, description, budget, dataProposed)
and conDate1 > conDate2);
```

4. (10 marks) The university just got some funding to upgrade campus buildings. And the university decided to start as many projects as possible

from the earliest proposed but not done project. Your task is to develop a C/C++ function, called `listProjects` to find the projects that can be funded by the given funding.

This function takes a database connection (*conn*), a double number indicating the amount of money received (*amount*), and a string that contains a date in the format of 'yyyy-mm-dd', (*date*) as its parameters. Then from the database, the function should find all the projects proposed after *date* (presumably those are the projects waiting to be started), order them according to the dates the projects are proposed, and as long as the remaining funding can still cover the whole budget of the project, show the project's description, the building's name and the budget. Finally, the function should show the remaining funding when the remaining funding is not enough to cover the next project's budget.

The prototype of the function is shown below:

```
void listProjects(Connection *conn, double amount, string date);
```

Solution:

```
void listProjects(Connection *conn, double amount, string date)
{
    string queryString = "select description, buildingName, budget ";
    queryString = queryString + "from Projects P, Buildings B "
        + "where P.dateProposed > to_date('" + date + "', 'yyyy-mm-dd') "
        + " and P.bno = B.bno "
        + "order by P.dateProposed";

    Statement * stmt = conn->createStatement();
    ResultSet * rs = stmt->executeQuery(queryString);

    string description, buildingName;
    double budget;
    cout << "The following project may be accomplished with the funding: \n";
    cout << "Project Description \t Building Name \t Budget \n";
    cout << "-----\n";
    while (rs->next()) {
        description = rs->getString(1);
        buildingName = rs->getString(2);
        budget = rs->getDouble(3);

        if (amount > budget) {
            cout << description << "\t"
                << buildingName << "\t"
                << budget << "\n";
            amount = amount - budget;
        } else {
            break;
        }
    }
}
```

```

    }
}
cout << "-----\n";
cout << "The remaining funding is " << amount << ".\n";

stmt->closeResultSet(rs);
conn->terminateStatement(stmt);
return;
}

```

5. (10 marks) Consider a relation with schema $R(A, B, C, D, E)$ and a set of FD's, $F = \{A \rightarrow D, BE \rightarrow C, E \rightarrow A\}$.

(a) What are all the candidate keys of R?

Solution:

There is only one candidate key of R, which is BE.

(b) R is not in BCNF. Why?

Solution:

Because $A \rightarrow D$ is not trivial and A is not a super key of R.

(c) find a lossless join decomposition to decompose R into collections of relations that are in BCNF.

Solution:

```

      ABCDE (using A->D)
     /  \
    AD  ABCE (using E->A)
      /  \
     EA  BCE

```

6. (10 marks) For each of the following two schedules:

i. $r_3(B); r_3(A); w_3(A); r_1(C); r_1(B); r_2(A); w_2(A); w_1(C); w_1(B); w_3(C);$

ii. $r_2(D); r_2(A); w_2(D); r_4(B); r_1(C); r_3(C); r_4(D); w_4(D); w_4(B); r_3(B); w_3(C); r_1(A); w_1(A);$

Answer the following questions:

(a) what is the precedence (serialization) graph for the schedule?

Solution:

i. T3 before T2, T3 before T1 and T1 before T3

ii. T2 before T1, T2 before T4, T1 before T3 and T4 before T3

(b) Is the schedule conflict-serializable? If so, list **all** the equivalent serial schedules.

Solution:

- i. The schedule is not conflict-serializable since there is a cycle in its precedence graph.
- ii. The schedule is conflict-serializable since its precedence graph is acyclic. The equivalent serialschedules are T2 T4 T1 T3 and T2 T1 T4 T3.

7. (10 marks)

- (a) Strict two phase locking (strict 2PL) requires each transaction to hold all locks to the end of the transaction. Schedules generated by scheduler using strict 2PL are guaranteed to be cascadeless. If we modify the strict 2PL slightly, so that each transaction is allowed to release shared locks early, but still required to acquire all locks before any lock is released and to hold the exclusive locks to the end. Are the schedules generated using this modified strict 2PL still cascadeless? If not, are they recoverable, or conflict serializable?

Solution:

The schedules should still be cascadeless since no transaction would be allowed to access the dirty data of another transaction (because this other transaction would hold its exclusive locks to the end).

- (b) If we modify the strict 2PL further and allow each transaction to acquire and release shared locks at any time as long as the transaction acquire and then hold all exclusive locks to the end, what kind of anomaly may happen in the schedules generated using such locking rules? Give an example of the anomaly.

Solution:

This locking protocol could cause unrepeatable read anomaly.

Example (E means exclusive(write) lock and S means shared(read) lock):

T1	T2
-----	-----
acquire E(x)	
acquire S(y)	
acquire E(z)	
read(y)	
read(x)	
x = x+y;	
write(x);	
release S(y)	
	acquire E(y)
	read(y)
	y = y * 2;
	write(y)
	commit and release E(y)
acquire S(y)	

```

read y
read z
z = z-y;
write(z)
commit and release
E(x), S(y), E(z)

```

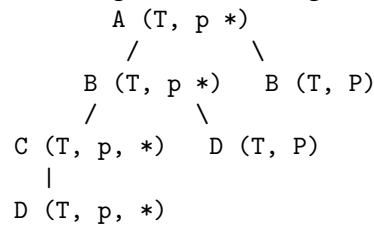
8. (10 marks) Show the grant diagrams after the sequence of actions listed in the following table. Assume A is the owner of the relation T to which privilege p refers.

Step	By	Action
1	A	Grant p To B On T With Grant Option
2	B	Grant p To C On T With Grant Option
3	B	Grant p To D On T
4	A	Grant p To B On T
5	C	Grant p To D On T With Grant Option

If A wanted nobody but B to have the privilege p on the relation T, what action(s) should A take?

Solution:

Grant diagram (* means grant option):



A can run the following SQL statement:

```
revoke grant option for p on T from B cascade;
```

9. (10 marks) The data pages of an instance of a hypothetical relation T with three attributes A, B and C are shown in the next page. Assume that a B+ tree internal node can store at most 4 values and 5 pointers and its leaf node can store at most 4 data items and 2 pointers (to doubly link the leaf nodes together). Also assume that there is already a clustered (sparse) B+ tree index built on the relation T using attribute C as the search key.

You are asked to build another B+ tree index on T using attribute A as the search key. Can you build this index as a clustered (sparse) index or a non-clustered (dense) index? Draw this B+ tree index. (You may use the next page to draw the index.)

A B C

1	2	34
3	18	32
5	26	30

7	16	28
9	4	26
11	24	24

13	12	22
15	32	20
17	6	18
19	14	16

21	30	14
23	8	12
25	20	10
27	34	8

29	10	6
31	28	4
33	22	2

===== END OF EXAM QUESTIONS =====