QUESTION 1. [10 marks]
Consider the database schema library, and answer the questions that follow it.

library.student(sid, firstName, lastName)
This records the basic information of each student.

library.book(bid, isbn, category, name, status)
This records the basic information of each book in the school library. ISBN is the International Standard Book Number. Status is either 'out' or 'in', and indicates whether this book is currently in the library (assuming all books are for circulation).

library.borrow(sid, bid, startDate, endDate)
This records the situation each time a student borrows a book from the library.

PART (A) [1 mark]
Suggest a natural way to record the fact that a given book is currently borrowed by a given student.

SOLUTION: Record an endDate of null.

PART (B) [1 mark]
Is it possible to remove bid from book, and use isbn as the primary key? Explain.

SOLUTION: No, isbn identifies the book edition, but there may be several copies in the library, and they must be distinguished by their bid.

PART (C) [2 marks]
There is some redundancy in the information recorded in relational schemas book and borrow. Identify the redundancy and suggest a modification of the university library database schema that eliminates it. Are there any drawbacks to your suggested schema? Explain.

SOLUTION: The status attribute of book is redundant, since we can derive it by seeing whether the tuple in borrow with the same bid has a null endDate. A drawback is that if we remove the status attribute, there is an added time to access the borrow table to look it up.

PART (D) [2 marks]
No book can be simultaneously borrowed by two different students. Write a constraint in relational algebra that guarantees this.

SOLUTION: Here's a constraint:

\[ b_1 = \rho_{B_1}(\text{borrow}) \quad b_2 = \rho_{B_2}(\text{borrow}) \]

\[ \sigma_{b_1.bid=b_2.bid \land b_1.sid!=b_2.sid \land b_1.startDate\leq b_2.startDate \land (b_1.endDate \lor b_2.startDate \leq b_1.endDate)} (b_1 \times b_2) = \emptyset \]
PART (E) [2 marks]
Write a relational algebra query that finds the name of all students who have borrowed more than one book in CS category.

SOLUTION: Here’s a query. Notice that it leaves open the possibility that students borrowed different copies of the same book.

\[
C_1 = \sigma_{\text{category} = \text{CS}}(\text{borrow} \bowtie \text{book}) \\
C_2 = C_1 \\
C_3 = \sigma_{C_1:\text{bid} \neq C_2:\text{bid} \land C_1:\text{sid} = C_2:\text{sid}}(C_1 \times C_2) \\
\text{Answer} = \pi_{\text{firstName}, \text{lastName}}(\text{student} \bowtie_{\text{sid} = C_1:\text{sid}} C_3)
\]

PART (F) [2 marks]
Write an SQL query that lists all the students who have never borrowed books from two different categories.

SOLUTION: Here’s a query.

\[
B_2 = B_1 = \text{borrow} \\
B_3 = B_1 \bowtie B_1.\text{category} <> B_2.\text{category} \land B_1.\text{sid} = B_2.\text{sid} \land B_2 \\
S = \pi_{\text{sid}}(\text{student}) \\
S_2 = S - \rho_{B_1.\text{sid} \rightarrow \text{sid}}(\pi_{B_1\text{sid}}(B_3)) \\
\text{Answer} = \pi_{\text{firstName}, \text{lastName}}(\text{student} \bowtie S_2)
\]

QUESTION 2. [10 marks]
Consider the following schema, and then provide SQL queries for the questions that follow it.

```sql
-- eid: unique employee id
-- name: employee’s name
-- city: employee’s city of residence
Employee(_eid_, name, city)

-- cid: unique company id
-- name: company name
-- city: city where company is located
Company(_cid_, name, city)

-- eid: unique employee id
-- cid: unique company id
-- an employee can have worked for a company at most once
-- salary: employee’s salary for said company in CAD
-- startDate, endDate: First and last day of employment.
-- endDate can be NULL, signifying ‘employee is currently employed at company’
WorksFor(_eid_, _cid_, salary, startDate, endDate)

-- eid: unique employee id
-- cid: unique company id
-- mid: employee id of employee’s manager
Manages(_eid_, _cid_, mid)
```
PART (A)  [2 marks]

Write an SQL query that returns the names of all currently commuting employees (i.e. employees who work in a different city than the one they live in)

SOLUTION:

```
select e.name
from Employee e, Company c, WorksFor w
where e.eid = w.eid and c.cid = w.cid and (w.endDate is null or
w.endDate = current_date) and e.city != c.city
```

PART (B)  [3 marks]

Write an SQL query that returns the names of all managers, along with the average salaries of all employees currently managed by them.

SOLUTION:

```
create temp view ManagerSalary as ( 
    select avg(w.salary) as sal, m.mid as mid 
    from Employee e, Manages m, WorksFor w 
    where m.eid = e.eid and w.cid = 
    m.cid and m.eid = w.eid and (w.endDate is null or w.endDate = 
    current_date) 
    group by m.mid); 
create temp view AllManagerSalary as 
    (select * from ManagerSalary) union (select 0, m.mid from Manages m 
    where m.mid not in ( select ms.mid from ManagerSalary ms ) );

select e.name, m.salary from AllManagerSalary m, Employee e where m.mid = e.eid;
```
PART (C)  [5 marks]

Write an SQL query that returns the names of all managers, along with the highest number of employees they have managed in any single year (a manager has managed an employee in a given year, if she has managed the employee during at least one day in said year). HINT: you may use the Postgresql function EXTRACT ( YEAR FROM x ), Example:

\[
\text{SELECT EXTRACT( YEAR FROM DATE '2001-01-01')}\]

returns 2001

Solution: -- Employee start and end dates for each manager

create temp view ManagerEmployees as ( 
select EXTRACT(YEAR FROM w.startDate) as startYear, EXTRACT(YEAR FROM IF(w.endDate is null, current_date, w.endDate)) as endYear, e.eid, 
m.mid as mid 
from Employee e, Manages m, WorksFor w where m.eid = e.eid and w.cid = 
m.cid and m.eid = w.eid and (w.endDate is null or w.endDate = 
current_date));

--Now note that the max will be obtained for a year that is an endYear 
(equivalently, a startYear)

create temp view ManagerEmployeeCount as 
(select mec.mid, max(mec.emplCount) as employees from 
(select count(distinct e.eid) as emplCount, me.mid as mid, years.endYear as year 
from ManagerEmployees me, (select distinct me2.endYear as endYear from 
ManagerEmployees me2) as years 
where me.startYear <= years.endYear and me.endYear >= years.endYear 
group by me.mid, year ) as ManagerEmployeesYears mec 
group by mec.mid );

create temp view AllManagerCount as 
(select * from ManagerEmployeeCount) union (select m.mid,0 from 
Manages m where m.mid not in (select ms.mid from ManagerEmployeeCount 
ms) );

select e.name, m.employees from AllManagerCount m, Employee e where 
m.mid = e.eid;
QUESTION 3.  [13 marks]

A university department has a database containing data about all lectures during a study period. The database has the following schema:

Schedule(C, D, T, R, P, A)

...where C stands for course, D for day (weekday), T for time, R for room, P for professor, and A for head TA. The set of functional dependencies, F, are:

\[ F = \{ RDT \rightarrow P, RDT \rightarrow C, C \rightarrow A, PDT \rightarrow R, PDT \rightarrow C, CDT \rightarrow P, CDT \rightarrow R \} \]

Answer the following questions:

PART (A) [2 marks]

What do these functional dependencies mean? Express them in words.

Solution:

\( RDT \rightarrow P \): No two professors share the same room/day/time.

\( RDT \rightarrow C \): No two courses share the same room/day/time combination.

\( C \rightarrow A \): There is no more than one head TA per course.

\( PDT \rightarrow R \): No two rooms share the same professor/day/time combination.

\( PDT \rightarrow C \): No two courses share the same professor/day/time combination.

\( CDT \rightarrow P \): No two professors share the same course/day/time combination.

\( CDT \rightarrow R \): No two rooms share the same course/day/time combination.

PART (B) [2 marks]

Is the functional dependency \( RDT \rightarrow AP \) entailed by the functional dependencies in \( F \)? Explain.

Solution: Yes. \( RDT \rightarrow A \) is entailed by \( RDT \rightarrow C \) and \( C \rightarrow A \), by transitivity. Then the union rule says that \( RDT \rightarrow A \) and \( RDT \rightarrow P \) entail \( RDT \rightarrow PA \).

PART (C) [3 marks]

Compute the key(s) of relation Schedule.

Solution: \( D \) and \( T \) must be part of any key, since they do not occur on the right-hand side of any FDs. However \( (DT)^+ = DT \), so they need at least one more attribute to form a key. The triples \( RDT, PDT, \) and \( CDT \) all have attribute closure \( CDTRPA \), and hence are keys. \( ADT \) has closure \( ADT \), and hence gives no new possibilities. So the three keys are \( RDT, PDT, \) and \( CDT \).

PART (D) [3 marks]

Is Schedule in BCNF form? If not, decompose it into smaller relations that are in BCNF.

Solution: \( C \rightarrow A \) violates BCNF, so we decompose into \( R_1 = (CA, \{C \rightarrow A\}) \) and \( R_2 = (CDTRP, \{RDT \rightarrow P, RDT \rightarrow C, PDT \rightarrow R, PDT \rightarrow C, CDT \rightarrow P, CDT \rightarrow R\}) \). \( R_1 \) must be in BCNF because it has just two attributes, and \( R_2 \) is in BCNF because all the FDs that project onto it have a left-hand side that is a superkey for \( R_1 \), and hence \( R_2 \).
PART (E)  [3 marks]

Assume that we create a new relation Profs_Schedule by projecting some attributes of Schedule, Profs_Schedule = π_{DTP}(Schedule). Identify the functional dependencies that hold for Profs_Schedule.

SOLUTION: By checking the attribute closures \( D^+ \), \( T^+ \), \( P^+ \), \( (DT)^+ \), \( (DP)^+ \), and \( (TP)^+ \), it turns out that the new relation has no non-trivial FDs projected onto it.

QUESTION 4.  [5 marks]

Below are the contents of an XML file part1.xml and the associated DTD part1.dtd. Identify the mistakes in them, and suggest ways to fix them.

part1.xml:

```xml
<?xml version="1.0" ?>
<!DOCTYPE PersonList SYSTEM "part1.dtd">
<Department deptId="cs">
  <Title>Computer Science</Title>
  <Address>30 Maple Str, London, ON</Address>
  <Divisions>
    <Division>
      <DivName>Graphics</DivName>
      <DivHead>Ronald Roe</DivHead>
    </Division>
    <Division>
      <DivName>Theory</DivName>
      <DivHead>N. Owen</DivHead>
    </Division>
  </Divisions>
</Department>

<Department deptId="ece">
  <Title>Electrical and Computer Engineering</Title>
  <Address>32 Maple Str., London, ON</Address>
  <Divisions>
    <Division>
      <DivHead>John Doe</DivHead>
      <DivName>Photonics</DivName>
    </Division>
    <Division>
      <DivHead>John Doe, Jr</DivHead>
      <DivName>Electronics</DivName>
    </Division>
    <Division Title="Systems">
      <Division>
        <DivName>Robotics</DivName>
        <DivHead>John Smith</DivHead>
      </Division>
      <Division>
        <DivName>Telecommunications</DivName>
        <DivHead>Nicole Smith</DivHead>
      </Division>
    </Division>
  </Divisions>
</Department>
```


part1.dtd:

```xml
<!ELEMENT Department(Title, Address, Divisions?)>
<!ELEMENT Title(#PCDATA)>
<!ELEMENT Address(#PCDATA)>
<!ELEMENT Divisions(Division+)>
<!ELEMENT Division(DivName, DivHead)>
<!ELEMENT DivName(#PCDATA)>
<!ELEMENT DivHead(#PCDATA)>
<!ATTLIST Department deptId ID #REQUIRED>
<!ATTLIST Division Title(#PCDATA) #IMPLIED>
```

**Solution:** Here are the mistakes

1. There is no single root element. Create one and add it to DTD.
2. Improper closing tag `</Title>`. Add the shilling slash `"/"`
3. Illegal nesting, `iDivision` within `iDivision`. Remove `iDivision Title="Systems"`.
4. `DivName` must precede `DivHead`, transpose them.
5. Division element can't be empty, add a `DivName` and `DivHead`
6. Attribute `Title` of Division should be `#CDATA`