Question 1. [12 MARKS]

Consider the following schema for tracking customers' ratings of books for a bookstore website.

- Books(<u>ISBN</u>, title, author, year, length). ISBN is a string used internationally for identifying books; length is an integer that represents the number of pages in a book.
- Authors(<u>AID</u>, name, city)
- Customers(<u>CID</u>, name)
- Rates(<u>ISBN</u>, <u>CID</u>, rating). Rating is an integer.

The following inclusion dependencies hold:

- Books(author) $\subseteq$ Authors(AID), Rates(ISBN) $\subseteq$ Books(ISBN), Rates(CID) $\subseteq$ Customers(CID)

Write the following queries in relational algebra using only the basic operators $\Pi, \sigma, \times, \bowtie, \cap, \cup, -, \rho, :=$. Assume the set semantics (not bag semantics) for Relational Algebra.

1. Find the CID(s) of customers who have rated both Aristotle's "Metaphysics" and Plato's "The Republic".

   $R := \Pi_{CID}(Customers \bowtie Rates \bowtie \sigma_{name=\text{``Aristotle''} \wedge title=\text{``Metaphysics''}}(Authors \bowtie Books))$

   $S := \Pi_{CID}(Customers \bowtie Rates \bowtie \sigma_{name=\text{``Plato''} \wedge title=\text{``The Republic''}}(Authors \bowtie Books))$

   $T := R \cap S$

2. Find the title(s) of the book(s) with the highest rating. If there is a tie, report them all.

   $R := \Pi_{rating}(Rates) - \Pi_{rating}(\sigma_{rating<rating2}(Rates \bowtie \rho_{Rates(ISBN2,CID2,rating2)}(Rates)))$

   $S := \Pi_{title}(Books \bowtie Rates \bowtie R)$

3. Find the AID(s) and name(s) of all authors who have at least two books with more than 500 pages.

   $R := \sigma_{length>500}(Books) \bowtie \sigma_{length2>500}(\rho_{Books(ISBN2,title2,author2,year2,length2)}(Books)$

   $S := \sigma_{ISBN \neq ISBN2 \wedge author=author2}(R)$

   $T := \Pi_{AID,name}\sigma_{AID=author}(Authors \times S)$

Question 2. [17 MARKS]

Consider the following schema:

Employee(eid, name, city)
– eid: unique employee id
– name: employee's name
– city: employee's city of residence

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

WorksFor(eid, cid, salary)
– eid: foreign key of Employee
– cid: foreign key of Company
– (An employee can have worked for a company at most once)
– salary: employee's salary at that company

Manages(eid, cid, mid) ← boss' sid
– eid: foreign key of Employee
– cid: foreign key of Company
– mid: employee id of eid's manager at that company (foreign key of Employee)

Write the following SQL queries.

1. For each employee-manager pair, report the employee's ID, the manager's ID and the company name.
   Solution:

```
select eid, mid, name
from manages natural join company;
```

2. Find all employees who make more than their manager. Report the company id, the name of the employee, and the name of the manager. Employees who don't have a manager should be excluded.
   Solution:

```
-- Involves: Self-join, plus joining back to get more than IDs.
-- Tricky spots:
--     Can't use natural join because boss's eid is called mid
--     Must make cids match also.
-- Of course, it is not necessary to use a view (in any of these queries).
create view leapfrog as
select manages.eid as peon, manages.mid as boss
from manages, worksfor w1, worksfor w2
where manages.eid = w1.eid and manages.mid = w2.eid and
manages.cid = w1.cid and manages.cid = w2.cid
and w1.salary > w2.salary;

select peon, e1.name, boss, e2.name
from leapfrog, employee e1, employee e2
where e1.eid = peon and e2.eid = boss;
```

Arity

3. For each company with more than 10 employees, report the company ID, company name, and number of employees.
   Solution:

```
-- Involves: group by, having, and joining back to get more than IDs.
-- Tricky spots: Can't select company name at time of group-by
```

```
create view bigCompanies as select cid, count(*) as size
from Company natural join worksfor
group by cid having count(*) > 10;

select cid, name, size from
bigcompanies natural join company;
```

4. Delete all employees who have not worked for any company.
   Solution:

```
-- Involves: delete and not exists
-- Could be done with natural join instead of a where clause.
delete from employee
where not exists (
    select * from worksfor
    where employee.eid = worksfor.eid
);
```

5. For each manager, report the manager's ID and the average salary of those employees who they manage. **Solution:**

```
-- Select-project-join with grouping.
select mid, avg(salary)
from manages, worksfor
where manages.eid = worksfor.eid
group by mid;
```

Question 3. [10 marks]

Assume a database named "db" with only one table named "people" that contains four columns: "id", "name", "city" and "age". Your table contains the following records:

id	name	city	age
1	Bob	Toronto	50
2	Ted	Toronto	51
3	Fred	Toronto	19
4	Steve	Toronto	17
5	John	Toronto	15
6	Luis	Toronto	53

Part (a) [2 marks]

Write an SQL query to find the possible dad/son pairs in the database assuming that a dad/ son relationship is possible if two persons have at least 20 years of age difference. For example the row "Bob, Steve" should be returned from your query, since the age difference of Bob (50) and Steve (17) is 33 years and 33 is larger than 20.

Solution:

```
SELECT P1.name AS dad, P2.name AS son
FROM People P1, People P2
WHERE P1.city = P2.city
AND P1.age > P2.age + 20;
```

Part (b) [2 marks]

What should be the output of your query over the above database instance (list the tuples that should be returned)?

Solution:

Bob, Fred
Ted, Fred
Luis, Fred
Bob, Steve
Ted, Steve
Luis, Steve
Bob, John
Ted, John
Luis, John

Part (c) [6 MARKS]

Write server side code (java/jdbc) that connects to the database, creates a valid SQL query statement, executes the query, prints the dad/son pairs on screen, one-pair per line, and finally closes the database connection. **Note**: Marking will be tolerant on syntax errors; if you don't know the exact name of a java method, that's fine.

// Connect to the database "db" using credentials (login: "admin", password: "admin")

```
String url = "jdbc:postgresql://localhost/db";
Connection conn = DriverManager.getConnection(url, "admin", "admin");
Statement st = conn.createStatement();
```

// Create a valid SQL query statement (use the query of (a))

```
String query = "SELECT P1.name AS dad, P2.name AS son
FROM People P1, People P2
WHERE P1.city = P2.city
AND P1.age > P2.age + 20;";
```

// Execute query and retrieve results

```
ResultSet rs = st.executeQuery(query);
```

// Print results (one line per pair)

```
while (rs.next()) {
String dad = rs.getString("dad");
String son = rs.getString ("son");
System.out.println(dad + ", " + son);
}
```

```
// Close connection
rs.close();
conn.close();
```

Question 4. [6 marks]

Consider the following SQL statements:

```
CREATE TABLE S (
   P INT PRIMARY KEY,
   Q INT CONSTRAINT positive CHECK (Q > 1),
   CONSTRAINT smaller CHECK (P < Q)
);

CREATE TABLE R (
   A INT PRIMARY KEY,
   B INT UNIQUE,
   C INT CONSTRAINT notBig CHECK (C < 100),
   FOREIGN KEY (B) REFERENCES S(P)
);

insert into S values
(2, 140),
(9, 49),
(8, 42),
(6, 140),
(5, 99);

insert into R values
(11, 2, 79),
(15, 8, 71),
(18, 9, 75);
```

When imported into postgres, they create the following messages:

```
psql:schema.sql:5: NOTICE:  CREATE TABLE / PRIMARY KEY will create
    implicit index "s_pkey" for table "s"
CREATE TABLE
psql:schema.sql:12: NOTICE:  CREATE TABLE / PRIMARY KEY will create
    implicit index "r_pkey" for table "r"
psql:schema.sql:12: NOTICE:  CREATE TABLE / UNIQUE will create
    implicit index "r_b_key" for table "r"
CREATE TABLE
```

For each error message on the next page, write an `insert` statement that will cause it. Treat each independently: assume you are starting with the database in the state described above, with no changes made to it.

Note: Of course there are multiple solutions to each of these questions.

1. `ERROR:  duplicate key value violates unique constraint "s_pkey"`

   Solution:

   ```
   insert into s values (9, 22);
   ```

2. `ERROR:  new row for relation "s" violates check constraint "smaller"`

   Solution:

   ```
   insert into s values (-5, -10);
   ```

3. `ERROR:  new row for relation "s" violates check constraint "positive"`

   Solution:

   ```
   insert into s values (-50, -10);
   ```

4. `ERROR:  duplicate key value violates unique constraint "r_b_key"`

   Solution:

   ```
   insert into r values(10, 8, 50);
   ```

5. `ERROR:  new row for relation "r" violates check constraint "notbig"`

   Solution:

   ```
   insert into r values(15, 6, 101);
   ```

6.
   ```
   ERROR:  insert or update on table "r" violates foreign key constraint "r_b_fkey"
   DETAIL:  Key (b)=(16) is not present in table "s".
   ```

   Solution:

   ```
   insert into r values(25, 16, 10);
   ```

Question 5. [17 marks]

Part (a) [5 marks]

The following is a valid XML file, but we've removed the part of the DTD that declares `Book`:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE Booklist [
    <!ELEMENT Booklist (Book*)>
    <!ELEMENT Book ....remainder of DTD omitted....>
]>
<Booklist>
     <Book>
         <Title>DDD</Title> <ISBN>1357</ISBN>
         <Author>Blake</Author> <Author>Atwood</Author> <Author>Seuss</Author>
     </Book>
     <Book>
         <Title>AAA</Title> <ISBN>7777</ISBN>
         <Author>Seuss</Author> <Author>Ashbe</Author>
     </Book>
     <Book>
         <Title>BBB</Title> <ISBN>4567</ISBN>
         <Author>Reid</Author> <Author>Oppel</Author> <Author>Petersen</Author>
     </Book>
     <Book>
         <Title>CCC</Title> <ISBN>2222</ISBN>
         <Author>Seuss</Author>
     </Book>
</Booklist>
```

Each declaration below is syntactically correct. Could it be the declaration for `Book`? **Do not guess.** Each part is worth 1 mark if correct, -1 if incorrect, and 0 if you don't respond. Your minimum mark is 0.

1. Could this be part of the declaration for `Book`? [✓ Yes] No

   ```
   <!ELEMENT Book (Title, ISBN, Author*)>
   ```

2. Could this be part of the declaration for `Book`? Yes [✓ No]

   ```
   <!ELEMENT Book (ISBN, Title+, Author*)>
   ```

3. Could this be part of the declaration for `Book`? [✓ Yes] No

   ```
   <!ELEMENT Book (Title, ISBN+, Author+, Date*)>
   ```

4. Could this be part of the declaration for `Book`? Yes [✓ No]

   ```
   <!ELEMENT Book (Title, ISBN, Date+, Author*)>
   ```

5. Could this be part of the declaration for `Book`? [✓ Yes] No

   ```
   <!ELEMENT Book (Title, ISBN, Author*)>
   <!ELEMENT Title (#PCDATA)>
   <!ATTLIST Title edition CDATA #IMPLIED>
   ```

Part (b) [12 MARKS]

Suppose the xml file above is called `booklist.xml`. Each of the XQuery queries below and on the next page runs without errors. Show the output. We will not be grading the whitespace in your answer, so format it as you wish.

1. `doc("booklist.xml")//Book/Author[.="Seuss"]`

   Solution:

   ```
   <Author>Seuss</Author>, <Author>Seuss</Author>, <Author>Seuss</Author>
   ```

2.
   ```
   for $x in doc("booklist.xml")//Book
   where $x/Author = "Seuss"
   return $x/Author
   ```

   Solution:

   ```
   <Author>Blake</Author>,
   <Author>Atwood</Author>,
   <Author>Seuss</Author>,
   <Author>Seuss</Author>,
   <Author>Ashbe</Author>,
   <Author>Seuss</Author>
   ```

3.
   ```
   for $x in doc("booklist.xml")//Book
   where $x/Author = "Seuss"
   return if ($x/Author = "Seuss") then () else ($x/Author)
   ```

   Solution:

   ```
   ()
   ```

For convenience, here is the contents of file `booklist.xml` again:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE Booklist [
    <!ELEMENT Booklist (Book*)>
    <!ELEMENT Book ....remainder of DTD omitted....>
]>
<Booklist>
     <Book>
         <Title>DDD</Title> <ISBN>1357</ISBN>
         <Author>Blake</Author> <Author>Atwood</Author> <Author>Seuss</Author>
     </Book>
     <Book>
         <Title>AAA</Title> <ISBN>7777</ISBN>
         <Author>Seuss</Author> <Author>Ashbe</Author>
     </Book>
     <Book>
         <Title>BBB</Title> <ISBN>4567</ISBN>
         <Author>Reid</Author> <Author>Oppel</Author> <Author>Petersen</Author>
     </Book>
     <Book>
         <Title>CCC</Title> <ISBN>2222</ISBN>
         <Author>Seuss</Author>
     </Book>
</Booklist>
```

On the next page, show the output of the remaining queries.

4. `doc("booklist.xml")//Title[. > (doc("booklist.xml")//Title)]`

   Solution:

   ```
   <Title>DDD</Title>, <Title>BBB</Title>, <Title>CCC</Title>
   ```

5.
   ```
   let $x := doc("booklist.xml")//Book/Title
   where $x = "AAA"
   return $x
   ```

   Solution:

   ```
   <Title>DDD</Title>,
   <Title>AAA</Title>,
   <Title>BBB</Title>,
   <Title>CCC</Title>
   ```

6.
   ```
   let $d := doc("booklist.xml")
   for $x in $d/Booklist/Book
   where count($x/Author) > 2
   return <Thing> { $x/ISBN } </Thing>
   ```

   Solution:

   ```
   <Thing><ISBN>1357</ISBN></Thing>, <Thing><ISBN>4567</ISBN></Thing>
   ```

Question 6. [11 marks]

Suppose we have a file called `grades.xml` that is valid with respect to the following DTD:

```
   <!ELEMENT GRADES (ITEMS, STUDENTS)>
   <!ATTLIST GRADES course CDATA #REQUIRED>
   <!ATTLIST GRADES term CDATA #REQUIRED>
   <!ELEMENT ITEMS (ITEM*)>
   <!ELEMENT ITEM EMPTY>
   <!ATTLIST ITEM name CDATA #REQUIRED>
   <!ATTLIST ITEM outof CDATA #REQUIRED>
   <!ATTLIST ITEM weight CDATA #REQUIRED>
   <!ELEMENT STUDENTS (STUDENT*)>
   <!ELEMENT STUDENT (MARK*)>
   <!ATTLIST STUDENT id CDATA #REQUIRED>
   <!ATTLIST STUDENT status (enrolled|dropped) #REQUIRED>
   <!ATTLIST STUDENT name CDATA #REQUIRED>
   <!ELEMENT MARK (#PCDATA)>
   <!ATTLIST MARK item CDATA #REQUIRED>
```

Part (a) [1 mark]

Define a small xml file that is valid with respect to this DTD and includes one `ITEM` and one `STUDENT`, who has one `MARK`.

Solution:

```
<?xml version = '1.0' standalone = 'no' ?>
<!DOCTYPE GRADES SYSTEM "grades.dtd">
<GRADES course="csc343" term="20119">
<ITEMS>
   <ITEM name="A1" outof="100" weight="10"/>
</ITEMS>
<STUDENTS>
   <STUDENT id="12345" status="enrolled" name = "John Hancock">
         <MARK item="A1">83</MARK>
   </STUDENT>
</STUDENTS>
</GRADES>
```

Part (b) [5 marks]

Write an query in XQuery that returns the ID and name of every student with a mark of at least 85 on a course item with weight at least 10%.

Solution:

```
let $d := doc("grades.xml")
let $heavyCourses :=
        for $item in $d//ITEM
        where $item/@weight >= "10"
        return $item/@name
for $student in $d//STUDENT
where exists (
        for $mark in $student/MARK
        where $mark > "76" and exists(index-of($heavyCourses, $mark/@item))
        return $mark
)
return ($student/@id, $student/@name)

Notes:
- attribute and element names are case-sensitive
- the weight and grade values are strings; must quote them
- don't want duplicates of student who had several of these
- MUST have brackets on the multi-part return value
```

Part (c) [5 MARKS]

Write an query in XQuery that returns xml containing the following information: for each student whose status is "dropped", their student id and the number of marks that they have in the file `grades.xml`. The xml generated must conform to the following DTD:

```
<!ELEMENT DROPS (STUDENT*)>
<!ELEMENT STUDENT (#PCDATA)>
<!ATTLIST STUDENT nummarks CDATA #REQUIRED>
```

Generate only the xml elements, not the "preamble" (the part that declares the xml version etc.).

Solution:

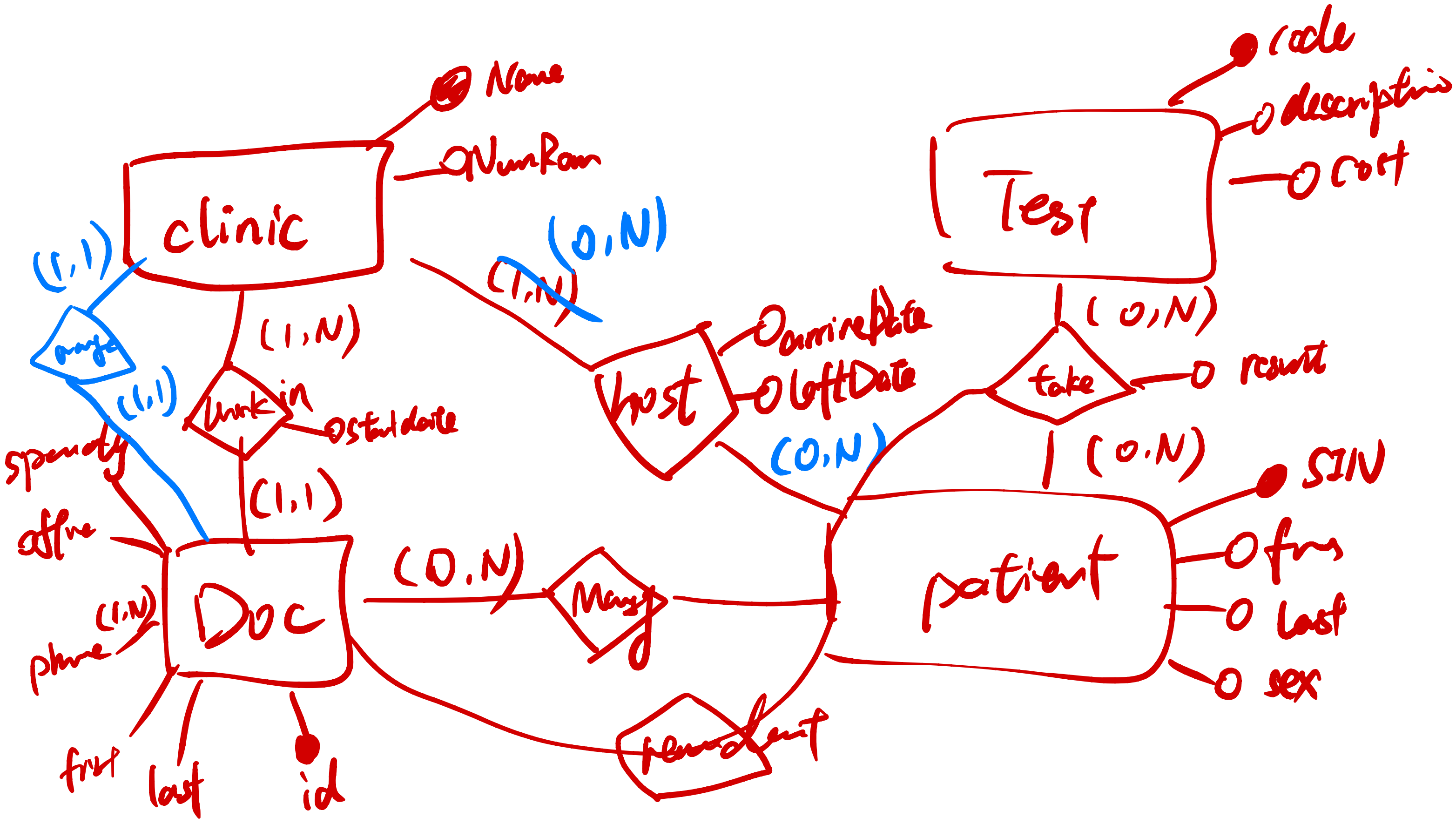
```
let $d := doc("grades.xml")
return <DROPS> {
        for $student in $d//STUDENT
        let $numGrades := count($student/MARK)
        where $student/@status = "dropped"
        return <STUDENT nummarks = "{$numGrades}"> {$student/@id} </STUDENT>
} </DROPS>

Notes
- There must be at least one for or let.
  So for instance, the first line can't be inside the return.
```

Question 7. [16 MARKS]

General introduction

A hospital consists of a number of specialized clinics (such as Maternity, Paediatrics, Cardiology, etc). Each clinic has a number of doctors who work in it. Doctors are specialists in some branch of medicine and can be associated with at most one clinic of the hospital. Each clinic hosts a number of patients. On admission, the personal details of every patient are recorded. We also want to store information about medical tests undertaken by a patient, after recommendation of a specialist. A number of tests may be conducted for each patient. In addition, we would like to keep the following information:

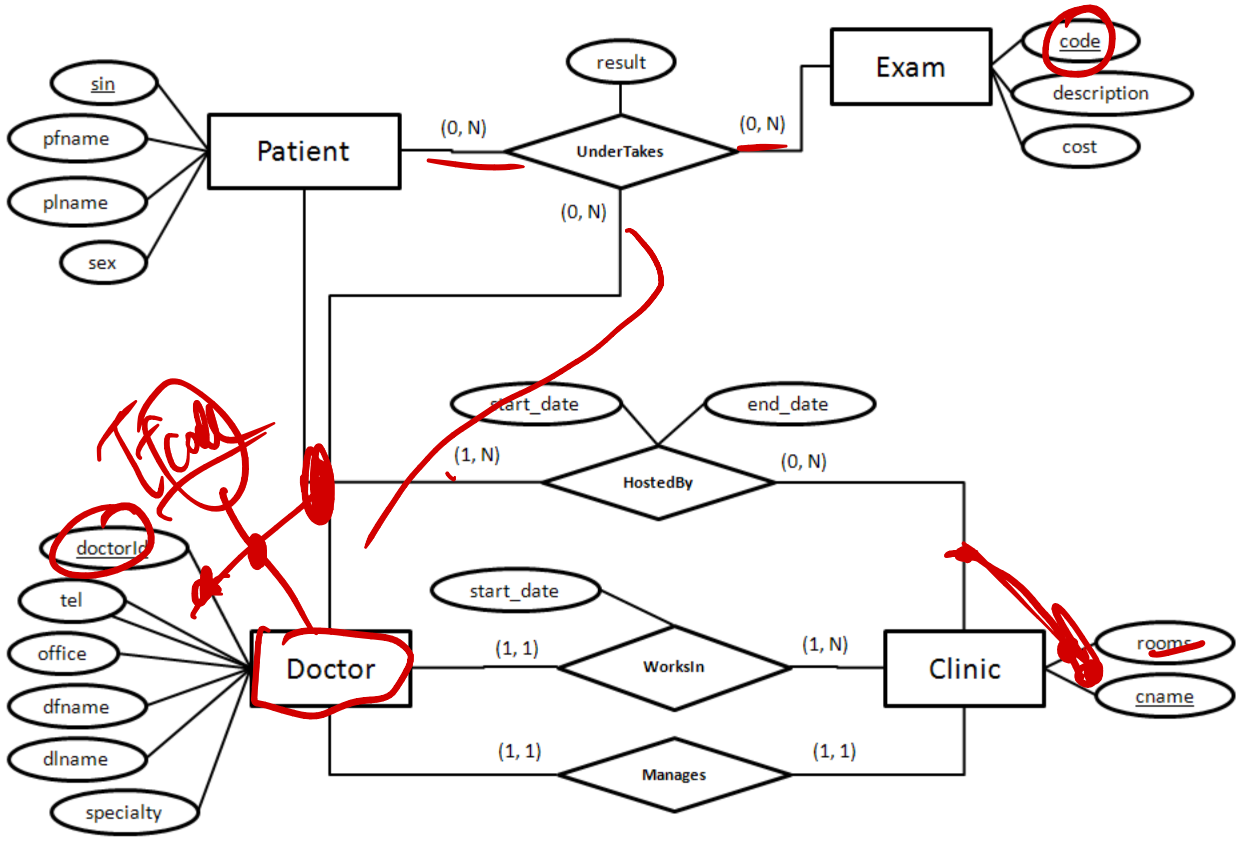
- For each clinic, a name that uniquely identifies it and the total number of rooms in the clinic. A clinic can have more than one doctor who works in it, can have more than one patient, and is managed by exactly one doctor.
- For each doctor, a licence id which uniquely identifies the doctor, their first and last name, their telephone number(s) (there can be more than one), their office, and their specialty. A doctor works in only one clinic. We want to keep, as well, the date when the doctor started working in the clinic.
- For each patient, a SIN (social insurance number) which uniquely identifies a patient, their first and last name, and their sex. A patient can undertake many medical tests and a medical test can be undertaken by many patients. For each patient hosted in a clinic, we want to keep, as well, the date when the patient arrived at the clinic and the date when they left the clinic.
- For each medical test, a test code that uniquely identifies it, a description, a cost, and the result of the test for each patient who has the test. The result of a test can be either negative or positive.



Part (a) [8 marks]

Design an Entity-Relationship Model (ER Diagram) for the hospital. Use entitities (rectangles), relationships (diamonds) and attributes (ovals) to model the information above. Clearly indicate primary keys (underlined attributes) and the cardinalities with which an entity participates in a relationship (express the participation of an entity to a relationship with a pair of (minimum, maximum) or alternatively, you can use the arrow notation of the textbook).

Solution:



Part (b) [8 MARKS]

Translate your Entity-Relationship Model (ER Diagram) from the question above into a logical model (DB Schema). For each relation in your schema, provide its name, attributes and keys (underlined attributes).

Solution:

Doctor

doctorId	dfname	dlname	office	specialty	cname	start_date

Tel

doctorId	tel

Patient

sin	pfname	plname	sex

Exam

code	description	cost

Clinic

cname	rooms	doctorId

Undertakes

sin	code	doctorId	result

HostedBy

sin	cname	start_date	end_date

3NF

$A \rightarrow B$

$\uparrow \quad \uparrow$

Question 8. [8 MARKS]

Consider the relation R on attributes $ABCDE$, with the following functional dependencies:

$$A \to BC, \quad CD \to E, \quad B \to D, \quad E \to A.$$

This question is about performing BCNF decomposition on R.

1. For each functional dependency, indicate whether it violates BCNF for this relation.

Does $A \to BC$ violate BCNF?	Yes	[✓ No]
Does $CD \to E$ violate BCNF?	Yes	[✓ No]
Does $B \to D$ violate BCNF?	[✓ Yes]	No
Does $E \to A$ violate BCNF?	Yes	[✓ No]

Explanation:

- $A^+ = ABCDE$, therefore $A \to BC$ does not violate BCNF.
- $CD^+ = CDEAB$, therefore $CD \to E$ does not violate BCNF.
- $B^+ = BD$, therefore $B \to D$ DOES violate BCNF.
- $E^+ = EABCDE$, therefore $E \to A$ does not violate BCNF.

2. Suppose we decompose relation R into two new relations: relation $R1$ on attributes BD and relation $R2$ on attributes $ACEB$. Project the functional dependencies of R onto $R1$.

Solution:

B	D	closure	FDs
+		$B^+ = D$	$B \to D$
	+	$D^+ = D$	

Therefore, the FDs of $R1$ on attributes BD are: $B \to D$.

3. Project the functional dependencies of R onto $R2$.

Solution:

A	C	E	B	closure	FDs
+				$A^+ = ABCDE$	$A \to CEB$
	+			$C^+ = C$	
		+		$E+ = EABCD$	$E \to ACB$
			+	$B^+ = BD$	
supersets of A					nothing we would keep
supersets of E					nothing we would keep
	+		+	$CB^+ = CBDEA$	$CB \to AE$

Therefore, the FDs of $R2$ on attributes $ACEB$ are $A \to CEB$, $E \to ACB$ and $CB \to AE$.

4. For each new relation, indicate whether it is in BCNF.

   Is $R1$ in BCNF? [✓ Yes] No

   Is $R2$ in BCNF? [✓ Yes] No

Question 9. [8 MARKS]

Consider the relation R on attributes $ABCDEGHI$, with the following functional dependencies:

$$H \rightarrow GD, \quad E \rightarrow D, \quad HD \rightarrow CE, \quad BD \rightarrow A.$$

This question is about performing 3NF synthesis on R.

1. Compute a minimal basis for the functional dependencies of relation R.

Solution:

(a) Split the FDs so that they have singleton right-hand sides:

$$H \rightarrow G, \quad H \rightarrow D, \quad E \rightarrow D, \quad HD \rightarrow C, \quad HD \rightarrow E, \quad BD \rightarrow A.$$

Call this set of functional dependencies S.

(b) Check if any of the FDs are redundant.

- $(H)^+_{S-\{H \rightarrow G\}} = HCDE$, which does not include G, so the FD $H \rightarrow G$ is NOT redundant.
- $(H)^+_{S-\{H \rightarrow D\}} = HG$, which does not include D, so the FD $H \rightarrow D$ is NOT redundant.
- $(E)^+_{S-\{E \rightarrow D\}} = E$, so the FD $E \rightarrow D$ is NOT redundant.
- $(HD)^+_{S-\{HD \rightarrow C\}} = HGDE$, which does not include C, so the FD $HD \rightarrow C$ is NOT redundant.
- $(HD)^+_{S-\{HD \rightarrow E\}} = HDG$, which does not include E, so the FD $HD \rightarrow E$ is NOT redundant.
- $(BD)^+_{S-\{BD \rightarrow A\}} = BD$, which does not include A, so the FD $BD \rightarrow A$ is NOT redundant.

This leaves us with the FDs:

$$H \rightarrow G, \quad H \rightarrow D, \quad E \rightarrow D, \quad HD \rightarrow C, \quad HD \rightarrow E, \quad BD \rightarrow A.$$

(c) Check whether any of the left-hand sides can be reduced.

- In FD $HD \rightarrow C$, D is redundant because $H^+ = HGDCE$, which includes C.
- In FD $HD \rightarrow E$, D is redundant because $H^+ = HGDCE$, which includes E.
- In FD $BD \rightarrow A$, neither attribute is redundant.

Therefore, a minimal basis is:

$$H \rightarrow G, \quad H \rightarrow D, \quad E \rightarrow D, \quad H \rightarrow C, \quad H \rightarrow E, \quad BD \rightarrow A.$$

2. What set of relation schemas would the 3NF synthesis algorithm generate from this minimal basis?
   Hint: Don't forget to make sure that some key of the original relation R is included in some relation schema.

   Solution:

   (a) The relation schemas generated directly from the minimal cover are: HG, ED, HC, HE, BDA.
   Or, if we combine FDs with common left-hand sides, we get this set of schemas: $HGCE, ED, BDA$.

   (b) Check whether any of them contains a key.

   - We could generate all the keys, but we can save time by observing that any key must include HBI (since these don't occur on the RHS of any FD).
   - None of these schemas includes HBI, so certainly none of them includes a key. So we need to add a relation whose schema is some key. It doesn't matter which key.
   - HBI itself is a key, since $HBI^+ = HBIGDECA$. So add the relation HBI.

   (c) The final schema is: $HG, HD, ED, HC, HE, BDA, HBI$
   or if we have combined FDs with common left-hand sides (as above): $HDGCE, ED, BDA, HBI$.

CONT'D...