Q1:
1 /

Solution: See Figure 2

Note: Some of you reasoned that since there is not a double line from "Pharmaceutical Co." to "Phone", the latter would not be considered multi-valued attribute. The missing line was my error, and I was not looking for this as part of the answer. However, we did accept that correction as part of the solution.

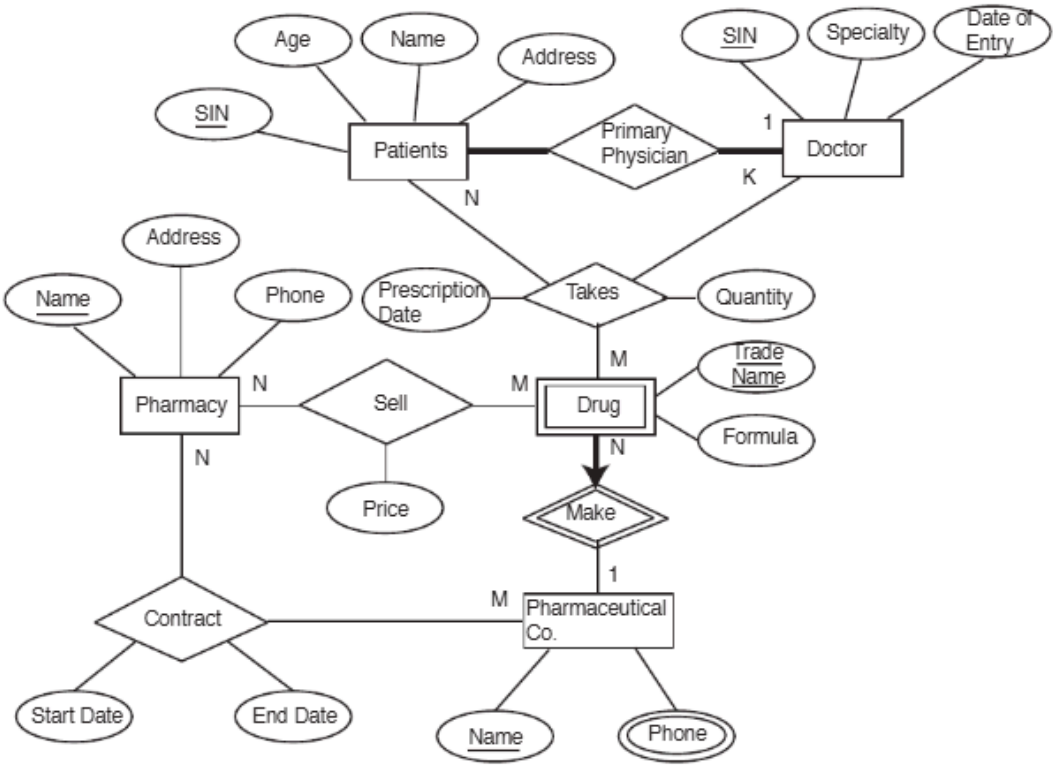

Figure 2: Solution to Question 4d

For each:
-0.5: missing attribute
-0.5: demarcation
-0.5: wrong/missing cardinality ratio/min-max (only cardinality ratio is really reqd)

Q2:

(a) (8 points) Complete the following SQL view definition to define the ToyList view as an appropriate query on the existing Toys, Client, and Request tables:

```
CREATE VIEW ToyList AS

SELECT r.client_id, c.name as client_name, c.age as client_age, r.toy_id

FROM Client c, Request r

WHERE c.id = r.client_id;
```

Note that the "as client_name" and "as client_age" are needed so that the view schema matches the original schema. If this was missed it was a minor deduction.

If correct logic:
-0.5 (each) for missing aliases
-0.5: (each) syntax

0: if incorrect logic

Question 3 (cont.) Space for your answer. Here is the original query using the view.

```
SELECT client_name, client_age
FROM ToyList
WHERE client_age > 2
ORDER BY client_name
```

Write the expanded version of the query below.

Step 1 – expand the query using the view definition to get a nested query.

```
SELECT v.client_name, v.client_age
FROM (SELECT r.client_id, c.name as client_name, c.age as client_age, r.toy_id
        FROM Client c, Request r
        WHERE c.id = r.client_id) as v
WHERE v.client_age > 2
ORDER BY v.client_name
```

Step 2 – unnest the query

```
SELECT c.name as client_name, c.age as client_age
FROM Client c, Request r
WHERE c.id = r.client_id and c.age > 2
ORDER BY c.name;
```

Queries for populating:

```
-- Create Toys table
CREATE TABLE Toys (
    id INT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    color VARCHAR(100) NOT NULL,
    min_age INT NOT NULL,
    weight INT NOT NULL,
    number_available INT NOT NULL
);

-- Create Client table
CREATE TABLE Client (
    id INT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    age INT NOT NULL,
```

```
    address VARCHAR(255) NOT NULL,
    city VARCHAR(100) NOT NULL
);

-- Create Request table
CREATE TABLE Request (
    client_id INT NOT NULL,
    toy_id INT NOT NULL,
    PRIMARY KEY (client_id, toy_id),
    FOREIGN KEY (client_id) REFERENCES Client(id),
    FOREIGN KEY (toy_id) REFERENCES Toys(id)
);

-- Insert data into Toys table
INSERT INTO Toys (id, name, color, min_age, weight, number_available) VALUES
(1, 'Teddy Bear', 'Brown', 3, 2, 50),
(2, 'Lego Set', 'Multicolor', 6, 5, 100),
(3, 'Toy Car', 'Red', 4, 1, 200),
(4, 'Doll', 'Pink', 3, 1, 150),
(5, 'Puzzle', 'Blue', 5, 3, 80);

-- Insert data into Client table
INSERT INTO Client (id, name, age, address, city) VALUES
(1, 'Alice', 7, '123 Candy Cane Lane', 'North Pole City'),
(2, 'Bob', 5, '456 Snowflake Street', 'Arctic Town'),
(3, 'Charlie', 6, '789 Reindeer Road', 'Holiday Village'),
(4, 'Daisy', 4, '321 Chimney Close', 'North Pole City'),
(5, 'Eve', 8, '654 Sleighbell Square', 'Arctic Town');

-- Insert data into Request table
INSERT INTO Request (client_id, toy_id) VALUES
(1, 2), -- Alice requests a Lego Set
(1, 5), -- Alice requests a Puzzle
(2, 1), -- Bob requests a Teddy Bear
(3, 3), -- Charlie requests a Toy Car
(4, 4), -- Daisy requests a Doll
(5, 2); -- Eve requests a Lego Set
```

a. 1.5 for expanded query

   If logic correct: (only if query mentioned)
   -1: if doesnt work(for missing ‘as v’ alias (unless they have handled referencing the inner table otherwise or it works)

   If logic incorrect: 0

b. 2.5 for unnested query
1.5: if query runs correctly and gives correct output (0.5 if wrong output)
0 if not

1 mark for logical query and explanation (only if query is mentioned)

Q3:
Ans: BC
The question must be attempted
0.5 for the correct answer for each option
i.e.
if A not marked +0.5, if A marked 0
if B not marked 0, if B marked +0.5
if C not marked 0, if C marked +0.5
if D not marked +0.5, if D marked 0

Q4:

```
CREATE TABLE Books (
    book_id varchar(100),
    book_name varchar(100),
    author varchar(100)
);

CREATE TABLE Readers (
    reader_id int,
    reader_name varchar(100),
    book_id varchar(100)
);

INSERT INTO Books (book_id, book_name, author) VALUES ("01_17", "Jane Eyre",
"Charlotte Bronte");
INSERT INTO Books (book_id, book_name, author) VALUES ("01_18", "Pride and
Prejudice", "Jane Austen");
INSERT INTO Books (book_id, book_name, author) VALUES ("02_10", "The BFG", "Roald
Dahl");
INSERT INTO Books (book_id, book_name, author) VALUES ("03_25", "The Da Vinci
Code", "Dan Brown");

INSERT INTO Readers (reader_id, reader_name, book_id) VALUES (2809, "Juliet",
"01_17");
```

INSERT INTO Readers (reader_id, reader_name, book_id) VALUES (2809, "Juliet", "01_18");
INSERT INTO Readers (reader_id, reader_name, book_id) VALUES (2513, "Blair", "03_25");
INSERT INTO Readers (reader_id, reader_name, book_id) VALUES (2290, "Henry", "01_18");

Answers

a - 1) Inner, Right, Natural and 2)Left, Right, Natural

2 marks if both
Depending on assumption/correct justification on order only 1 is correct, give marks accordingly

1 mark if one of the two is marked without assumption
0 if any incorrect option (since full can never give the correct answer)

b -
0 if join not mentioned
0 if query doesn't run/ syntax error

```
select reader_name, book_name, author
from Books
join Readers on Books.book_id = Readers.book_id
```

```
select reader_name, book_name, author
from Books
right join Readers on Books.book_id = Readers.book_id
```

```
select reader_name, book_name, author
from Books
natural join Readers
```

```
select reader_name, book_name, author
from Readers
left join Books on Books.book_id = Readers.book_id
```

Q3 -

+0.5 marks for each correct row
0 if missing row/more than one wrong row
-0.5: for extra columns

reader_name	book_name	author
Juliet	Jane Eyre	Charlotte Bronte
Henry	Pride and Prejudice	Jane Austen
Juliet	Pride and Prejudice	Jane Austen
Blair	The Da Vinci Code	Dan Brown

(Order may be different depending on type of join used)

Q5:

a. The query is given below. Its result is non-empty if and only if $B \rightarrow C$ **does not** hold on r.

```
SELECT B
FROM r
GROUP BY B
HAVING COUNT(DISTINCT C) > 1
```

```
CREATE TABLE r (
  A INT,
  B INT,
  C INT
);

INSERT INTO r (A, B, C) VALUES
(1, 1, 10),
(2, 2, 20),
(3, 3, 30),
(4, 4, 40),
(5, 5, 50);
```

for B->C dependency, the query should return empty set

Insert (2,2,25) into r, and see if the output is not null

Marking:

3 if runs and gives null

1.5 if correct output according to the reasoning provided

1 if only correct logic but query doesnt run

Give 2 if very minor syntax error

0 otherwise

(1.5 if they have a logic that doesnt use the null / empty set flow of logic)

Q6:

Solution:

a) **Candidate Key: AB**

As A -> A,
B->EBF
E -> D and D->C
Hence B->EBFDC
Hence AB -> ABCDEF

1 mark: if AB
0 marks otherwise

B->EBF is a partial dependence, Hence **only 1NF.**

1 mark: 1 NF
0 otherwise

b) i) B->A doesn't hold. Hence: **BC->ED**
ii) AB is the candidate key, hence it **AB->FCA holds**.
iii) E->D, D->C; hence, E->C, Hence **AE->C holds.**
iv) D->E does not hold (BD->E doesn't imply D->E). **Hence D->{}.**

1 mark each
0.5 for correct answer
0.5 for derivation

Q7: on the discretion of TAs:)
(4 marks for proof)
No partial marks for examples/intuitive explanations (like long paragraphs) only
Only if proof is rigorous and logical, then partial marks may be awarded
(2 marks for counter example)

Let $R = (A, B)$ be a schema with two attributes. Let F be a set of functional dependencies that hold on R.

We want to show:- R is in BCNF.

Suppose to the contrary that R is not in BCNF. That is, there exists $\alpha \rightarrow \beta$ in F^+ that is **not** trivial and α is **not** a superkey.

Since $\alpha \subseteq R$ and $\beta \subseteq R$, we know that, both $\alpha, \beta \in A, B, A, B$.

- **Case 1**: $\alpha = A$
  - **Sub-Case 1**: $\beta = A$
    This sub-case is impossible since, it would make the functional dependency trivial. And we have assumed that the functional dependency $\alpha \rightarrow \beta$ is not trivial.
  - **Sub-Case 2**: $\beta = B$
    Then we can write $\alpha \rightarrow \beta$ as $A \rightarrow B$. By **Reflexivity rule** followed by **Union rule** we know that this $A \rightarrow A, B$ holds. This means α is a superkey. But we have assumed α is not a superkey. Thus this sub-case is also impossible.
  - **Sub-Case 3**: $\beta = A, B$
    Also impossible, since this would make α a superkey. And we have assumed that α is not a superkey.
- **Case 2**: $\alpha = B$
  - **Sub-Case 1**: $\beta = A$
    Then we can write $\alpha \rightarrow \beta$ as $B \rightarrow A$. By **Reflexivity rule** followed by **Union rule** we know that this $B \rightarrow A, B$ holds. This means α is a superkey. But we have assumed α is not a superkey. Thus this sub-case is impossible.
  - **Sub-Case 2**: $\beta = B$
    This sub-case is impossible since, it would make the functional dependency trivial. And we have assumed that the functional dependency $\alpha \rightarrow \beta$ is not trivial.
  - **Sub-Case 3**: $\beta = A, B$
    Also impossible, since this would make α a superkey. And we have assumed that α is not a superkey.
- **Case 3**: $\alpha = A, B$
  - This case is impossible, since α is obviously a superkey.

Therefore R is in BCNF.