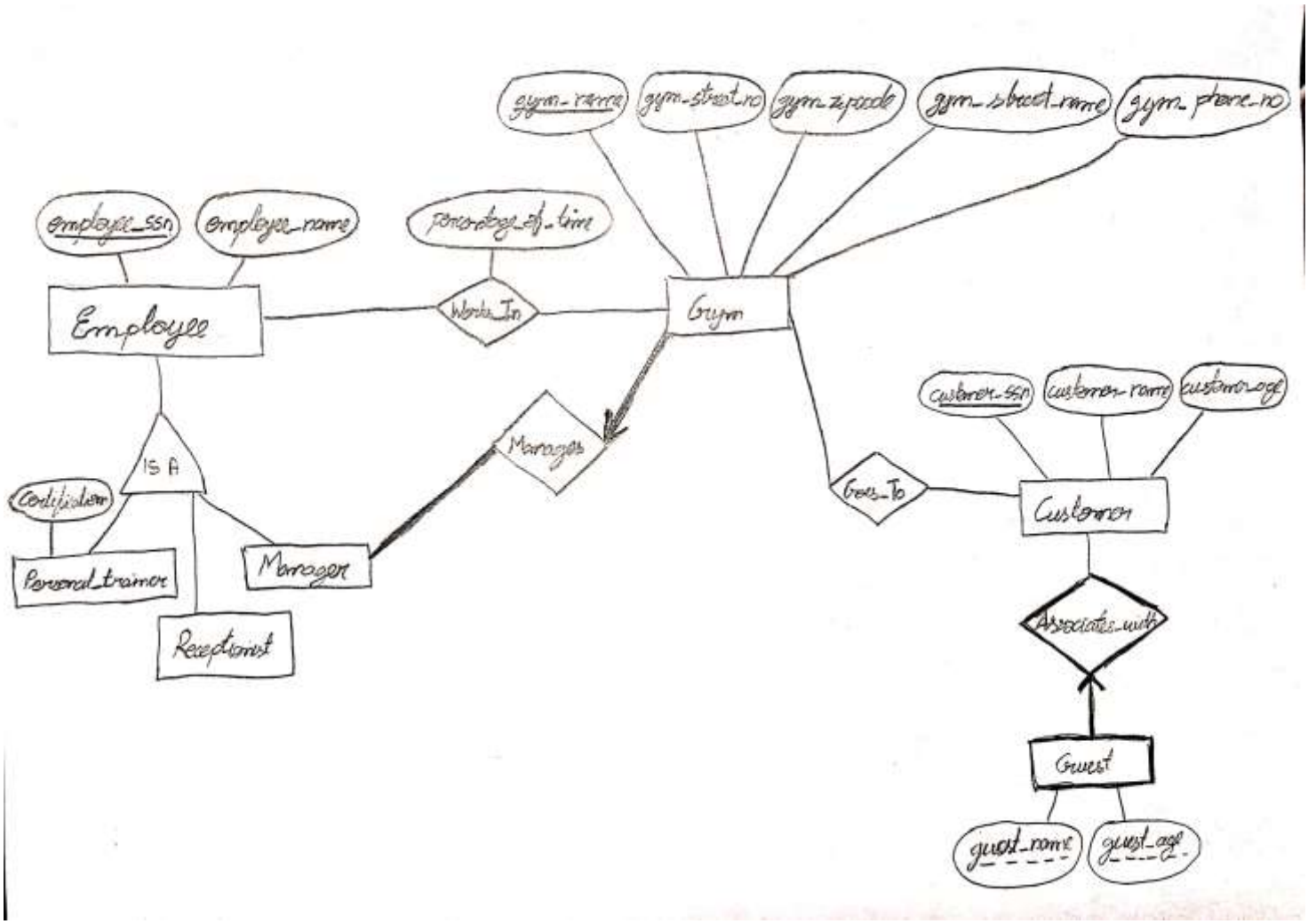
NAME- MAREESH KUMAR ISSAR
NETID- mi251

Ans 1 (1).



There is no overlapping constraint as an employee can only be one of (manager, receptionist or personal trainer) i.e. he/she can have zero or one specialisation.
There is a covering constraint
Manager AND Receptionist AND Personal_trainer COVER Employee

Ans 1 (2).

```
CREATE TABLE Employee (
                        employee_ssn CHAR(11),
                        employee_name CHAR(30),
                        specialisation CHAR(30),
```

```
                    PRIMARY KEY(employee_ssn)
                    )
CREATE TABLE Personal_trainer (
                    employee_ssn CHAR(11),
                    certification CHAR(30),
                    PRIMARY KEY(employee_ssn)
                    FOREIGN KEY (employee_ssn) REFERENCES Employee
                    )


CREATE TABLE Customer (
                    customer_ssn CHAR(11),
                    customer_name CHAR(30),
                    customer_age INTEGER,
                    PRIMARY KEY(customer_ssn)
                    )

CREATE TABLE Gym (
                    employee_ssn CHAR(11) NOT NULL,
                    gym_name CHAR(30),
                    gym_street_no INTEGER,
                    gym_street_name CHAR(30),
                    gym_zipcode INTEGER,
                    gym_phone_no INTEGER,
                    PRIMARY KEY(gym_name),
                    FOREIGN KEY (employee_ssn) REFERENCES Employee
                    )

CREATE TABLE Works_In (
                    employee_ssn CHAR(11),
                    gym_name CHAR(30),
                    percentage_of_time REAL,
                    PRIMARY KEY(employee_ssn, gym_name),
                    FOREIGN KEY (employee_ssn) REFERENCES Employee,
                    FOREIGN KEY (gym_name) REFERENCES Gym
                    )

CREATE TABLE Goes_To (
                    customer_ssn CHAR(11),
                    gym_name CHAR(30),
```

```
                    PRIMARY KEY(customer_ssn, gym_name),
                    FOREIGN KEY (customer_ssn) REFERENCES Customer,
                    FOREIGN KEY (gym_name) REFERENCES Gym
                    )


CREATE TABLE Guest (
                    customer_ssn CHAR(11),
                    guest_name CHAR(30),
                    guest_age INTEGER,
                    PRIMARY KEY(guest_name, guest_age, customer_ssn),
                    FOREIGN KEY (customer_ssn) REFERENCES Customer
                    )
```

Ans 2 (1).

```
SELECT S.sname
FROM Suppliers S
WHERE NOT EXISTS((SELECT P.pid
                  FROM Parts P)
                  EXCEPT
                  (SELECT C.pid
                   FROM Catalog C
                   WHERE S.sid=C.sid))
```

(2)

```
SELECT C.sid
FROM Catalog C
WHERE C.cost>(SELECT AVG(C1.cost)
              FROM Catalog C1
              WHERE C1.pid=C.pid
              )
```

(3)

```
SELECT S.sname, P.pname
FROM Suppliers S, Catalog C, Parts P
WHERE C.sid=S.sid AND P.pid=C.pid AND C.cost=(
```

```
                                        SELECT MAX (C1.cost)
                                        FROM Catalog C1
                                        WHERE C1.pid=P.pid
                                        )
```

(4)

```
SELECT C.sid
FROM Catalog C
WHERE NOT EXISTS (SELECT *
                  FROM Parts P
                  WHERE C.pid=P.pid AND P.color <> 'red')
```

(5)

```
SELECT C.sid
FROM Catalog C, Parts P
WHERE C.pid=P.pid AND P.color='red'
UNION
SELECT C.sid
FROM Catalog C, Parts P
WHERE C.pid=P.pid AND P.color='green'
```

(6)

```
SELECT S.sname, MAX(C.cost)
FROM Catalog C, Parts P, Suppliers S
WHERE C.pid=P.pid AND C.sid=S.sid AND (P.color='red' AND P.color='green')
GROUP BY S.sname
```

Ans 3 (1).

```
SELECT M.MovieID, M.MovieName
FROM Suppliers S, MovieSupplier MS, Movies M
WHERE (S.SupplierName='Ben's Video' OR S.SupplierName='Video Clubhouse') AND
      MS.MovieID=M.MovieID AND MS.SupplierID= S.SupplierID
```

(2)

```
SELECT M.MovieName
FROM Movies M, Rentals R, Inventory I
WHERE I.MovieID=M.MovieID AND R.TapeID=I.TapeID
      AND R.Duration=(SELECT MAX(R1.Duration)
                      FROM Rentals R1
```

)

(3)

```
SELECT S.SupplierName
FROM Suppliers S
WHERE NOT EXISTS((SELECT I.MovieID
                  FROM Inventory I)
                  EXCEPT
                  (SELECT MS.MovieID
                   FROM MovieSupplier MS
                   WHERE MS.SupplierID= S.SupplierID))
```

(4)

```
SELECT S.SupplierName, COUNT(DISTINCT MS.MovieID)
FROM MovieSupplier MS, Inventory I, Suppliers S
WHERE MS.MovieID=I.MovieID AND S.SupplierID=MS.SupplierID
GROUP BY S.SupplierName
```

(5)

```
SELECT M.MovieName
FROM Orders O, Movies M
WHERE O.MovieID=M.MovieID
GROUP BY M.MovieName
HAVING SUM(O.Copies)>4
```

(6)

```
SELECT C.FirstName, C.LastName
FROM Movies M, Rentals R, Inventory I, Customers C
WHERE M.MovieName='Kung Fu Panda' AND M.MovieID=I.MovieID AND I.TapeID=R.TapeID
      AND R.CustomerID=C.CustID
UNION
SELECT C.FirstName, C.LastName
FROM MovieSupplier MS, Rentals R, Inventory I, Customers C, Suppliers S
WHERE S.SupplierName='Palm Video' AND S.SupplierID=MS.SupplierID AND
MS.MovieID=I.MovieID AND AND I.TapeID=R.TapeID AND R.CustomerID=C.CustID
```

(7)

```
SELECT M.MovieName
FROM  Movies M
```

```
WHERE M.MovieID= (SELECT I1.MovieID
                  FROM Inventory I1
                  GROUP BY I1.MovieID
                  HAVING COUNT(*)>1
                 )
```

(8)

```
SELECT C.FirstName, C.LastName
FROM Customers C, Rental R
WHERE C.CustID=R.CustomerID AND R.Duration>=5
```

(9)

```
SELECT S.SupplierName
FROM Movies M, MovieSupplier MS, Suppliers S
WHERE MS.MovieName='Cinderella 2015' AND MS.MovieID=M.MovieID AND
MS.SupplierID=S.SupplierID AND MS.Price <= ALL (SELECT MIN(MS1.Price)
                                                FROM MovieSupplier MS1, Movies M1
                                                WHERE M1.MovieID=MS1.MovieID AND
                                                      MS1.MovieName='Cinderella 2015'
                                                )
```

(10)

```
SELECT M.MovieName
FROM Movies M
WHERE M.MovieID NOT IN (SELECT I.MovieID
                        FROM Inventory I
                       )
```

Ans 4 a.

Before an update is performed, the trigger event is activated. Since, here the test condition is satisfied, the trigger action is performed and the tuple of the Purchase table (111,4) is changed to (111,1.5). After this the original update action is performed and the tuple of the Purchase table (111,1.5) is changed to (111,3).

Therefore the end result is that the tuple of the Purchase table (111,4) is changed to (111,3).

b.

In this case the update action is performed first and the tuple of the Purchase table (111,4) is changed to (111,3). After the update is performed, the trigger event is activated. Since,

here the test condition is satisfied, the trigger action is performed and the tuple of the Purchase table (111,3) is changed to (111,1.5).

Therefore the end result is that the tuple of the Purchase table (111,4) is changed to (111,1.5).

c.
In this case instead of update, the trigger event is activated. Since, here the test condition is satisfied, the trigger action is performed and the tuple of the Purchase table (111,4) is changed to (111,1.5).

Therefore the end result is that the tuple of the Purchase table (111,4) is changed to (111,1.5).