CS 443

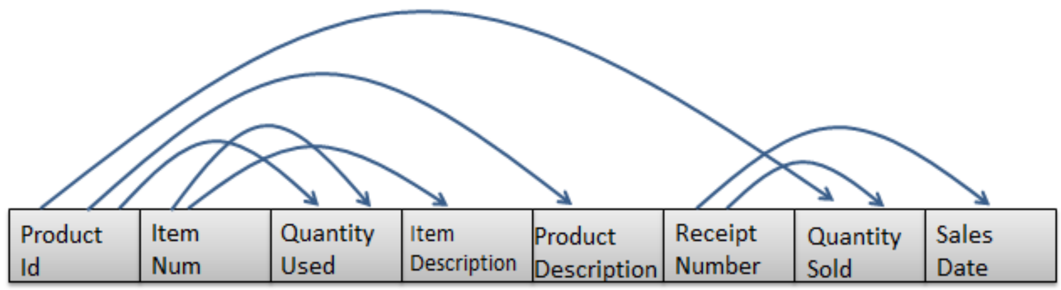
ASSIGNMENT 1

Malialosa Taupule

March 9, 2017

Question #1

Consider the following data. Arrows show the functional dependency.



a) Write the tables.

Step 1: Determine the functional dependencies.

Format: Key attribute → (determines) Non-key attribute

ReceiptNumber → SalesDate
ProductID, ReceiptNumber → QuantitySold
ProductID → ProductDescription
ItemNum →ItemDescription
ProductID, ItemNum → QuantityUsed

Step 2: Create separate tables for each A → B relationship (determined from above).

T1: (ReceiptNumber, SalesDate)
T2: (ProductID*, ReceiptNumber*, QuantitySold)
T3: (ProductID, ProductDescription)
T4: (ItemNum, ItemDescription)
T5: (ProductID*, ItemNum*, QuantityUsed)

b) Place the tables in 3rd normal form (if necessary).

Step 3: Place tables in 3rd form (if necessary).

Fact 1: Tables are assumed to be in first normal form because data for the tables are not given (just their attributes). Hence,

✓ **Each table that was created in Step 2 are in first normal form.**

Fact 2: T1, T3, and T4 are in second normal form because they only have one primary key.
Fact 3: Although T2 and T5 have composite primary keys, there is no existing partial functional dependency within these two tables qualifying each as in second normal form.
Fact 4: Each functional dependency was removed in Step 1. Therefore,

✓ **Each table that was created in Step 2 are in second normal form.**

Assumptions:

- By looking at each attribute, I have concluded (and have assumed) that there are no derived dependencies because there are no obvious calculations that could be used to derive some attributes from others (like QuantitySold x SalePrice = TotalRevenue).
- I have also concluded that there are no transitive dependencies, which was based on what I know of each table attribute. Since there was no given description of each attribute, from common knowledge, I know that:
  - SalesDate cannot determine the ReceiptNumber because there could be many transactions in a day.
  - QuantitySold cannot determine ReceiptNumber because there could be many customers that buy the same amount of the same product.
  - QuantitySold cannot determine ProductID because there could be many products that have the same amount sold.
  - ProductDescription cannot determine ProductID because descriptions have the potential to be unspecific; therefore, making its values very ambiguous and unimportant. Also, competitor products have the potential of being very similar to one another.
  - QuantityUsed cannot determine ProductID for the same reason that QuantitySold cannot determine ProductID: there could be many products that have the same amounts used.
  - QuantityUsed cannot determine ItemNumber because there could be many items that have the same amounts used.

Fact 5: Based on my assumptions outlined above,

✓ **Each table that was created in Step 2 is in third normal form.**

c) Create ERD based on the normalized tables

Step 4: Determine cardinalities between entities.

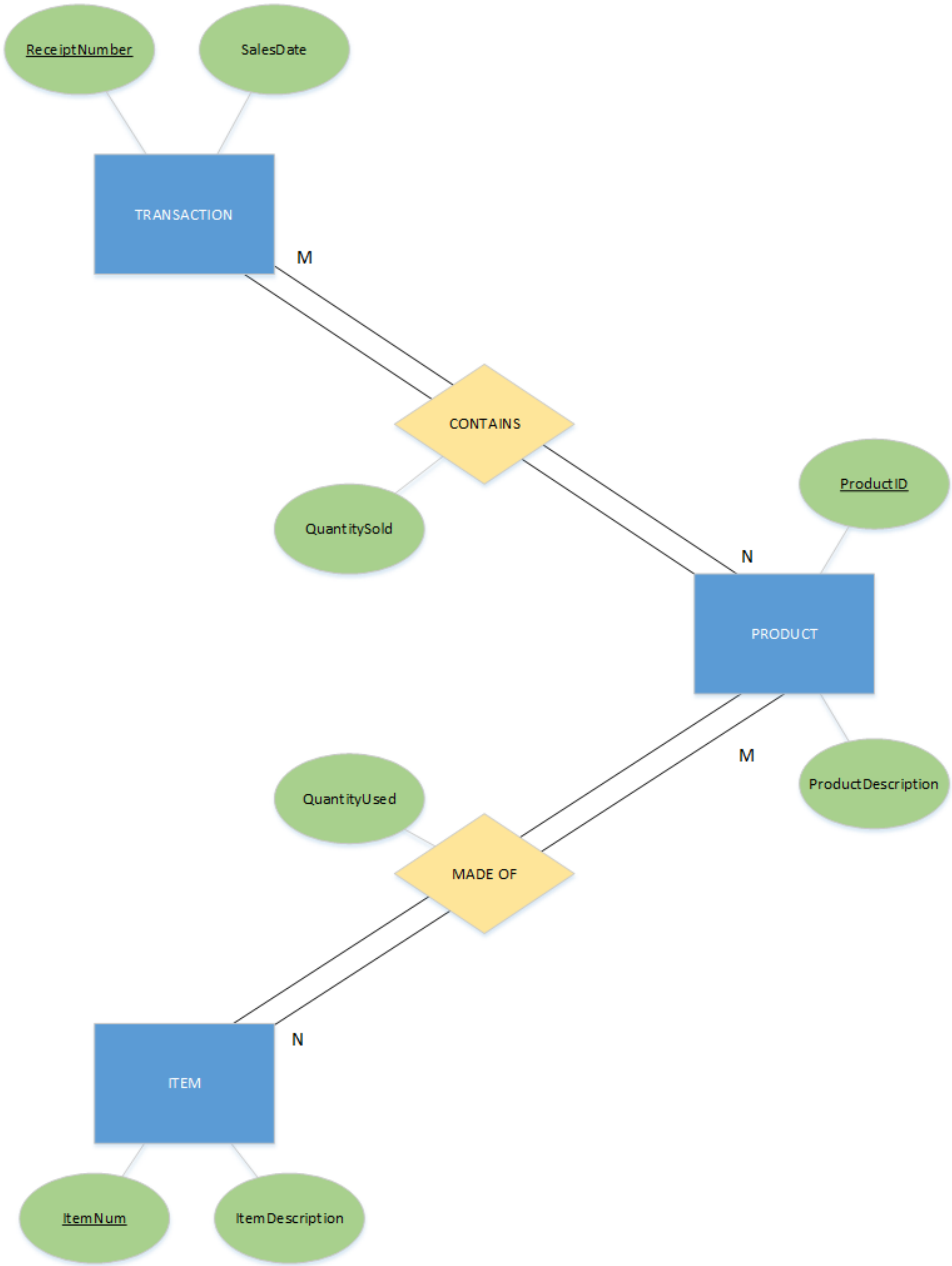
Based on the attributes of some tables, we can name immediately name:

T1: TRANSACTION
T3: PRODUCT
T4: ITEM

Due to T2 and T5 having composite keys, in which each are foreign keys, we know (from Lecture 2: Entity Relational Diagram Modeling) that these tables are a result from a **M:N relationship**. Based on T2's foreign keys, we can conclude that T2 is a M:N relationship between TRANSACTION and PRODUCT. Based on T5's foreign keys, we can conclude that T5 is a M:N relationship between PRODUCT and ITEM. Thus, we can name each:

T2: CONTAINS
T5: MADE OF

Step 5: **Create ERD from tables including each cardinality**.



d) Write a script to create a database. Your script should create the tables and ensures that all constraints are set properly.

Query:

Edit Q1_Queries

In emacs:
```
CREATE TABLE Transaction
(ReceiptNumber NUMBER,
SalesDate DATE,
CONSTRAINT TransactionPK PRIMARY KEY(ReceiptNumber));
```

Result:

```
SQL> @Q1_Queries

Table created.

SQL> desc Transaction;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 RECEIPTNUMBER                             NOT NULL NUMBER
 SALESDATE                                          DATE
```

Query: //before running @Q1_Queries, I dropped table Transaction

In emacs:
```
CREATE TABLE Product
(ProductId NUMBER,
ProductDescription VARCHAR(200),
CONSTRAINT ProductPK PRIMARY KEY(ProductID));
```

Result:

```
SQL> @Q1_Queries

Table created.


Table created.

SQL> desc Product;
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 PRODUCTID                                 NOT NULL NUMBER
 PRODUCTDESCRIPTION                                 VARCHAR2(200)
```

Query: //before running @Q1_Queries, I dropped table Transaction and table Product

In emacs:
```
CREATE TABLE Item
(ItemNum NUMBER,
ItemDescription VARCHAR(200),
CONSTRAINT ItemPK PRIMARY KEY(ItemNumber));
```

Result:

```
SQL> @Q1_Queries

Table created.


Table created.


Table created.

SQL> desc Item;
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 ITEMNUM                                   NOT NULL NUMBER
 ITEMDESCRIPTION                                    VARCHAR2(200)
```

Query: //before running @Q1_Queries, I dropped table Transaction, Item, and Product

In emacs:
```
CREATE TABLE Contains
(QuantitySold NUMBER,
ReceiptNumber NUMBER,
ProductID NUMBER,
CONSTRAINT ContainsPK PRIMARY KEY(ReceiptNumber, ProductID),
CONSTRAINT CheckQuantSold CHECK(QuantitySold >= 0)
CONSTRAINT ContainsFK_ReceiptNum
       FOREIGN KEY(ReceiptNumber)
       REFERENCES Transaction(ReceiptNumber),
CONSTRAINT ContainsFK_ProdID
       FOREIGN KEY(ProductID)
       REFERENCES Product(ProductID));
```

Result:

```
SQL> @Q1_Queries

Table created.


Table created.


Table created.


Table created.

SQL> desc Contains;
 Name                                      Null?    Type
 ----------------------------------------- -------- --------
 QUANTITYSOLD                                       NUMBER
 RECEIPTNUMBER                             NOT NULL NUMBER
 PRODUCTID                                 NOT NULL NUMBER
```

Query: //before running @Q1_Queries, I dropped table Transaction, Product, Item, and Contains

In emacs:

```
CREATE TABLE MadeOf
(ProductID NUMBER,
ItemNum NUMBER,
QuantityUsed NUMBER,
CONSTRAINT MadeOfPK PRIMARY KEY(ProductID, ItemNumber),
CONSTRAINT CheckQuantUsed CHECK(QuantityUsed >= 0),
CONSTRAINT MadeOfFK_ProdID
       FOREIGN KEY(ProductID)
       REFERENCES Product(ProductID),
CONSTRAINT MadeOfFK_ItemNum
       FOREIGN KEY(ItemNum)
       REFERENCES Item(ItemNum));
```

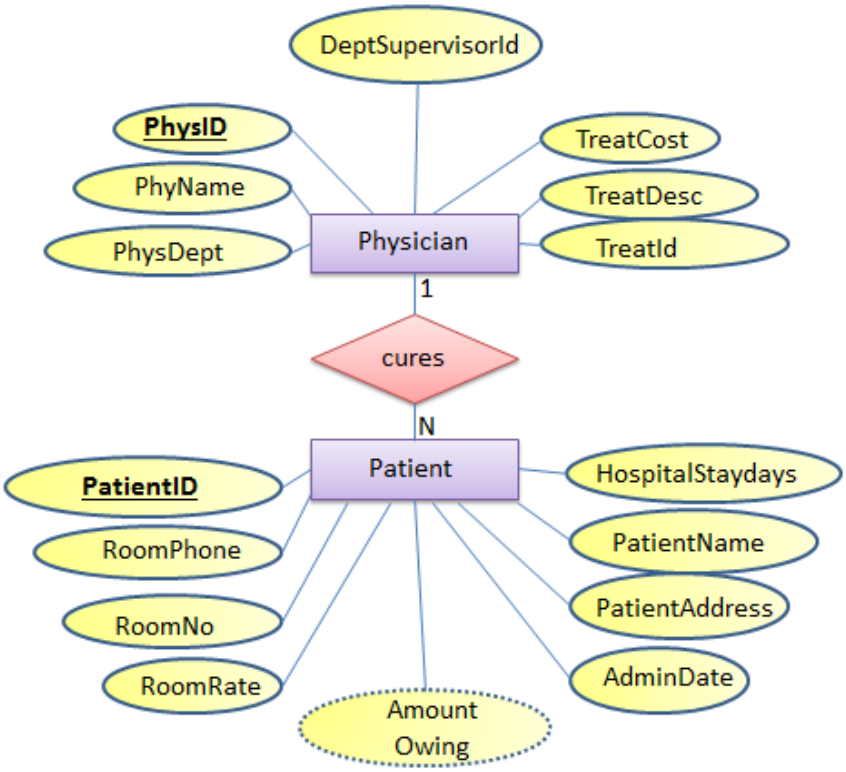
Result:

```
SQL> @Q1_Queries

Table created.


Table created.


Table created.


Table created.


Table created.

SQL> desc madeof;
 Name                                      Null?    Type
 ----------------------------------------- -------- --------
 PRODUCTID                                 NOT NULL NUMBER
 ITEMNUM                                   NOT NULL NUMBER
 QUANTITYUSED                                       NUMBER
```

```
CREATE TABLE Transaction
(ReceiptNumber NUMBER,
SalesDate DATE,
CONSTRAINT TransactionPK PRIMARY KEY(ReceiptNumber));

CREATE TABLE Contains
(QuantitySold NUMBER,
ReceiptNumber NUMBER,
ProductID NUMBER,
CONSTRAINT ContainsPK PRIMARY KEY(ReceiptNumber, ProductID),
CONSTRAINT CheckQuantSold CHECK(QuantitySold >= 0)
C0NSTRAINT ContainsFK_ReceiptNum
       FOREIGN KEY(ReceiptNumber)
       REFERENCES Transaction(ReceiptNumber),
CONSTRAINT ContainsFK_ProdID
       FOREIGN KEY(ProductID)
       REFERENCES Product(ProductID));

CREATE TABLE Product
(ProductId NUMBER,
ProductDescription VARCHAR(200),
CONSTRAINT ProductPK PRIMARY KEY(ProductID));

CREATE TABLE Item
(ItemNum NUMBER,
ItemDescription VARCHAR(200),
CONSTRAINT ItemPK PRIMARY KEY(ItemNumber));

CREATE TABLE MadeOf
(ProductID NUMBER,
ItemNum NUMBER,
QuantityUsed NUMBER,
CONSTRAINT MadeOfPK PRIMARY KEY(ProductID, ItemNumber),
CONSTRAINT CheckQuantUsed CHECK(QuantityUsed >= 0),
CONSTRAINT MadeOfFK_ProdID
       FOREIGN KEY(ProductID)
       REFERENCES Product(ProductID),
CONSTRAINT MadeOfFK_ItemNum
       FOREIGN KEY(ItemNum)
       REFERENCES Item(ItemNum));
```

Question #2

Consider the following ERD



a) Change the ERD to tables

Step 1: Determine the functional dependencies of each entity.

PHYSICIAN
PhysID → PhysDept, PhyName
PhysDept → DeptSupervisorID
TreatID → TreatDesc, TreatCost

PATIENT
PatientID → RoomNo, AmtOwing, AdminDate, PatientAddress, PatientName, HospitalStayDays
RoomNo → RoomPhone, RoomRate

Step 2: Create separate tables for each A → B relationship (determined from above).

T1: (<u>PhysID</u>, PhysDept*, PhyName)
T2: (<u>PhysDept</u>, DeptSupervisorID*)
T3: (<u>TreatID</u>, TreatDesc, TreatCost)
T4: (<u>PatientID</u>, PatientName, PatientAddress, RoomNo*, AdminDate, HospitalStayDays, AmtOwing, TreatID*, PhysID*)

- TreatID and PhysID added as foreign keys based on relationship description given in question description

T5: (<u>RoomNo</u>, RoomPhone, RoomRate)

b) Place the tables in 3rd normal form (if necessary)

Step 3: Place tables into 3rd normal form (if necessary).

<u>Fact 1</u>: Tables are assumed to be in first normal form because data for the tables are not given (just their attributes). Hence,

✓ **Each table that was created in Step 2 are in first normal form.**

<u>Fact 2</u>: T1, T2, T3, and T4 are in second normal form because they only have one primary key.

✓ **Each table that was created in Step 2 are in second normal form.**

<u>Assumptions</u>:

- I have also concluded that there are no transitive dependencies, which was based on what I know of each table attribute. Since there was no given description of each attribute, from common knowledge, I know that:
  - PhyName cannot determine PhysID because there could be more than one physician with the same exact name.
  - DeptSupervisorID is a candidate key and therefore could determine the PhysDept; however it's unnecessary to have a composite primary key with both attributes. Thus, PhysDept will suffice.
  - Neither TreatCost nor TreatDesc can determine the treatment because there could be more than one treatment with the same cost and treatment description has the potential of being ambiguous.
  - RoomRate cannot determine RoomNo because there could be more than one room with the same rate.
  - RoomPhone is a candidate key; however, is unnecessary to be used as a primary key since RoomNo is unique.

<u>Fact 3</u>: With my assumptions in mind:

✓ T1, T2, T3, and T5 are in 3rd normal form.

<u>Fact 4</u>: To normalize T4, I removed AmtOwing because it is derived from HospitalStayDays x RoomRate. Every other attribute is cleared since PatientID is all that's needed to determine the others. Thus, the new normalized T4 is:

T4: (<u>PatientID</u>, PatientName, PatientAddress, RoomNo*, AdminDate, HospitalStayDays, TreatID*, PhysID*)

<u>Fact 5</u>: Based on these facts,

✓ **Each table is now in third normal form.**

c) Revise the given ERD based on the normalized tables (if necessary)

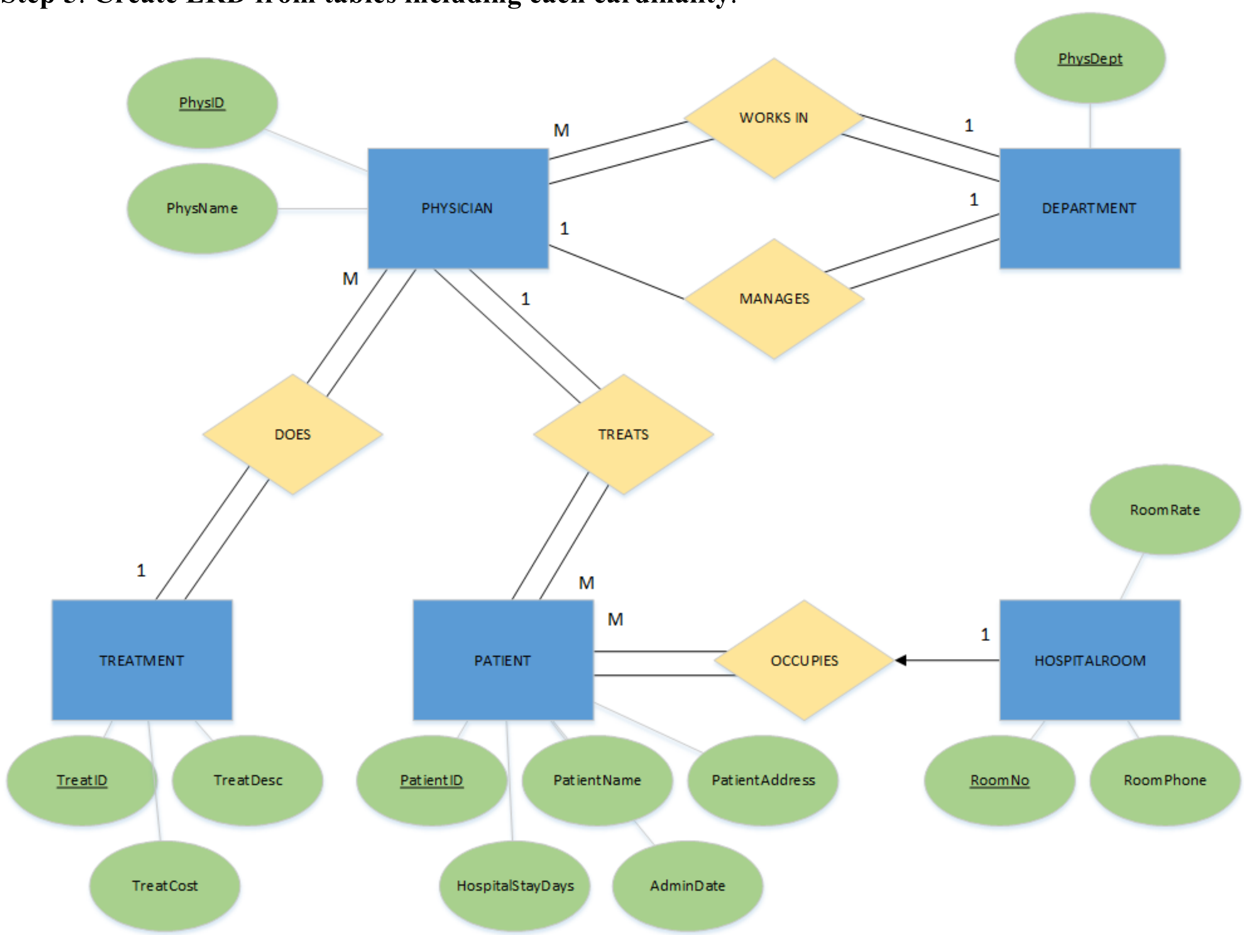
Step 4: Determine cardinalities between entities.

Based on the attributes of some tables, we can name immediately name:

T1: PHYSICIAN
T2: DEPARTMENT
T3: TREATMENT
T4: PATIENT
T5: HOSPITALROOM

Cardinalities are described in question outline.

Step 5: **Create ERD from tables including each cardinality**.



d) Write a script to create a database. Your script should create the tables and ensures that all constraints are set properly.

Query:

Edit Q2_Queries

In Emacs:

```
CREATE TABLE Physician
(PhysID       NUMBER,
PhysDept      NUMBER,
PhyName       VARCHAR(50)
       CONSTRAINT PhyName_NotNull NOT NULL,
CONSTRAINT PhysicianPK PRIMARY KEY(PhysID));
```

Result:

```
SQL> @Q2_Queries

Table created.

SQL> desc physician;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 PHYSID                                    NOT NULL NUMBER
 PHYSDEPT                                           NUMBER
 PHYNAME                                   NOT NULL VARCHAR2(50)
```

Query:

In emacs:

```
CREATE TABLE Department

(PhysDept     NUMBER,
DeptSupervisorID NUMBER,
CONSTRAINT DeptPK PRIMARY KEY(PhysDept),
CONSTRAINT DeptFK_SuperID
       FOREIGN KEY(DeptSupervisorID)
       REFERENCES Physician(PhysID));
```

Result:

```
SQL> @Q2_Queries

Table created.


Table created.

SQL> desc department
 Name                                      Null?    Type
 ----------------------------------------- -------- -------
 PHYSDEPT                                  NOT NULL NUMBER
 DEPTSUPERVISORID                                   NUMBER
```

Query:

In emacs:
```
ALTER TABLE Physician
ADD CONSTRAINT PhysFK_Dept
      FOREIGN KEY(PhysDept)
      REFERENCES Department(PhysDept);
```

Result:

```
SQL> @Q2_Queries

Table created.


Table created.


Table altered.
```

Query: //Drops all foreign key constraints & previously created first

In emacs:
```
CREATE TABLE Treatment
(TreatID      NUMBER,
TreatDesc     VARCHAR(200),
TreatCost     NUMBER(10,2),
CONSTRAINT TreatPK PRIMARY KEY(TreatID),
CONSTRAINT CheckTreatCost CHECK(TreatCost>=50.00));
```

Result:

```
SQL> @Q2_Queries

Table created.


Table created.


Table altered.


Table created.

SQL> desc Treatment;
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 TREATID                                   NOT NULL NUMBER
 TREATDESC                                          VARCHAR2(200)
 TREATCOST                                          NUMBER(10,2)
```

Query: //Drops all foreign key constraints & previously created first

In emacs:

```
CREATE TABLE HospitalRoom
(RoomNo NUMBER,
RoomPhone VARCHAR(8),
RoomRate NUMBER(10,2),
CONSTRAINT RmPK PRIMARY KEY(RoomNo),
CONSTRAINT Rm_RoomNoCheck CHECK(RoomNo>=100 AND RoomNo<=999),
CONSTRAINT RmRateCheck CHECK(RoomRate>=50.00));
```

Result:

```
SQL> @Q2_Queries

Table created.


Table created.


Table altered.


Table created.


Table created.

SQL> desc HospitalRoom;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 ROOMNO                                    NOT NULL NUMBER
 ROOMPHONE                                          VARCHAR2(8)
 ROOMRATE                                           NUMBER(10,2)
```

Query: //Drops all foreign key constraints & previously created first

In emacs:

```
CREATE TABLE Patient
(PatientID NUMBER,
PatientName VARCHAR(50)
CONSTRAINT PAddr_NotNull NOT NULL,
PatientAddress VARCHAR(200),
      CONSTRAINT PName_NotNull NOT NULL,
RoomNo NUMBER,
TreatID NUMBER,
HospitalStayDays NUMBER,
AdminDate DATE,
PhysID NUMBER,
CONSTRAINT PatientPK PRIMARY KEY(PatientID),
CONSTRAINT Patient_RoomNoCheck CHECK(RoomNo>=100 AND RoomNo<=999),
CONSTRAINT StayDaysCheck CHECK(HospitalStayDays>=0),
CONSTRAINT PatientFK_RmNo
      FOREIGN KEY(RoomNo)
      REFERENCES HospitalRoom(RoomNo),
CONSTRAINT PatientFK_TreatID
      FOREIGN KEY(TreatID)
      REFERENCES Treatment(TreatID),
CONSTRAINT PatientFK_PhysID
      FOREIGN KEY(PhysID)
      REFERENCES Physician(PhysID));
```

Result:

```
Table dropped.

SQL> @Q2_Queries

Table created.


Table created.


Table altered.


Table created.


Table created.


Table created.

SQL> desc Patient;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PATIENTID                                 NOT NULL NUMBER
 PATIENTNAME                               NOT NULL VARCHAR2(50)
 PATIENTADDRESS                            NOT NULL VARCHAR2(200)
 ROOMNO                                             NUMBER
 TREATID                                            NUMBER
 HOSPITALSTAYDAYS                                   NUMBER
 ADMINDATE                                          DATE
 PHYSID                                             NUMBER
```

```
CREATE TABLE Physician
(PhysID NUMBER,
PhysDept       NUMBER,
PhyName VARCHAR(50)
        CONSTRAINT PhyName_NotNull NOT NULL,
CONSTRAINT PhysicianPK PRIMARY KEY(PhysID));

CREATE TABLE Department
(PhysDept      NUMBER,
DeptSupervisorID NUMBER,
CONSTRAINT DeptPK PRIMARY KEY(PhysDept),
CONSTRAINT DeptFK_SuperID
        FOREIGN KEY(DeptSupervisorID)
        REFERENCES Physician(PhysID));

ALTER TABLE Physician
        ADD CONSTRAINT PhysFK_Dept
                FOREIGN KEY(PhysDept)
                REFERENCES Department(PhysDept);

CREATE TABLE Treatment
(TreatID       NUMBER,
TreatDesc      VARCHAR(200),
TreatCost      NUMBER(10,2),
CONSTRAINT TreatPK PRIMARY KEY(TreatID),
CONSTRAINT CheckTreatCost CHECK(TreatCost>=50.00));

CREATE TABLE HospitalRoom
(RoomNo NUMBER,
RoomPhone VARCHAR(8),
RoomRate NUMBER(10,2),
CONSTRAINT RmPK PRIMARY KEY(RoomNo),
CONSTRAINT Rm_RoomNoCheck CHECK(RoomNo>=100 AND RoomNo<=999),
CONSTRAINT RmRateCheck CHECK(RoomRate>=50.00));

CREATE TABLE Patient
(PatientID NUMBER,
PatientName VARCHAR(50)
        CONSTRAINT PName_NotNull NOT NULL,
PatientAddress VARCHAR(200)
        CONSTRAINT PAddr_NotNull NOT NULL,
RoomNo NUMBER,
TreatID NUMBER,
HospitalStayDays NUMBER,
AdminDate DATE,
PhysID NUMBER,
CONSTRAINT PatientPK PRIMARY KEY(PatientID),
CONSTRAINT Patient_RoomNoCheck CHECK(RoomNo>=100 AND RoomNo<=999),
CONSTRAINT StayDaysCheck CHECK(HospitalStayDays>=0)
CONSTRAINT PatientFK_RmNo
        FOREIGN KEY(RoomNo)
        REFERENCES HospitalRoom(RoomNo),
CONSTRAINT PatientFK_TreatID
        FOREIGN KEY(TreatID)
        REFERENCES Treatment(TreatID),
CONSTRAINT PatientFK_PhysID
        FOREIGN KEY(PhysID)
        REFERENCES Physician(PhysID));
```

Question #3

Step 1: **For each strong entity create a table with its simple attributes**.

A(A1, A2
B(B1, B2
C(C1, C2
D(D1, D5, D2, D3, D4

Step 2: **Handle all one-to-many relationships.**

A(A1, A2)
B(B1, B2, A1*, C1*)
C(C1, C2)
D(D1, D5, D2, D3, D4)

Step 3: **Handle all one-to-many relationships**.

R3(C1*, (D1, D5)*, AttrOfR3)

Step 4: **Handle all weak entities and their relationships**.

E(E1*, (D1, D5)*, E2, AttrOfR4)
F(F1, F2, (E1, D1, D5)*, F3, F4)

Answer:

A(A1, A2)
B(B1, B2, A1*, C1*)
C(C1, C2)
D(D1, D5, D2, D3, D4)
R3(C1*, (D1, D5)*, AttrOfR3)
E(E1*, (D1, D5)*, E2, AttrOfR4)
F(F1, F2, (E1, D1, D5)*, F3, F4)

Question #4

Step 1: **For each strong entity create a table with its simple attributes**.

BANK(Code, Name, Addr)
ACCOUNT(AcctNo, Balance, Type
LOAN(LoanNo, Amount, Type
CUSTOMER(SSN, Phone, Name, Addr

Step 2: **Handle all one-to-many relationships**.

A-C(AcctNo*, SSN*)
L-C(SSN*, LoanNo*)

Step 4: **Handle all weak entities and their relationships**.

BANK-BRANCH(BranchCode, BankCode*, Addr)
ACCOUNT(AcctNo, Balance, Type, (BranchCode, BankCode)*)
LOAN(LoanNo, Amount, Type, (BranchCode, BankCode)*)
CUSTOMER(SSN, Phone, Name, Addr, (BranchCode, BankCode)*)

Answer:

BANK(Code, Name, Addr)
A-C(AcctNo*, SSN*)
L-C(SSN*, LoanNo*)
BANK-BRANCH(BranchCode, BankCode*, Addr)
ACCOUNT(AcctNo, Balance, Type, (BranchCode, BankCode)*)
LOAN(LoanNo, Amount, Type, (BranchCode, BankCode)*)
CUSTOMER(SSN, Phone, Name, Addr, (BranchCode, BankCode)*)

Question #5

Step 1: **Determine the functional dependencies of each entity.**

BOOK
BookID → BookName, PublisherName

PUBLISHER
Name → Address, Phone

BOOK_AUTHORS
BookID, AuthorName

BOOK_LOANS
BookID, BranchID, Card_No → DateOut, DueDate

LIBRARYBRANCH
BranchID → BranchName, Address

BORROWER
CardNo → Name, Address, Phone

BOOK_COPIES
BookID, BranchID → NoOfCopies

Step 2: **Create separate tables for each A → B relationship (determined from above).**

BOOK(BookID, BookName, PublisherName*)
PUBLISHER(Name, Address, Phone)
BOOK_AUTHORS(BookID*, AuthorName)
BOOK_LOANS(BookID*, BranchID*, Card_No*, DateOut, DueDate)
LIBRARYBRANCH(BranchID, BranchName, Address)
BORROWER(CardNo, Name, Address, Phone)
BOOK_COPIES(BookID*, BranchID*, NoOfCopies)

Step 3: Place tables into 3rd normal form (if necessary).

Fact 1: Tables are assumed to be in first normal form because data for the tables are not given (just their attributes). Hence,

✓ **Each table that was created in Step 2 are in first normal form.**

Fact 2: BOOK, PUBLISHER, LIBRARYBRANCH, and BORROWER are all in 2nd normal form because each has only 1 primary key.

Fact 3: Functional dependencies were removed in step 2, thus the remaining tables are in 2nd normal form.

✓ **Each table that was created in Step 2 are in second normal form.**

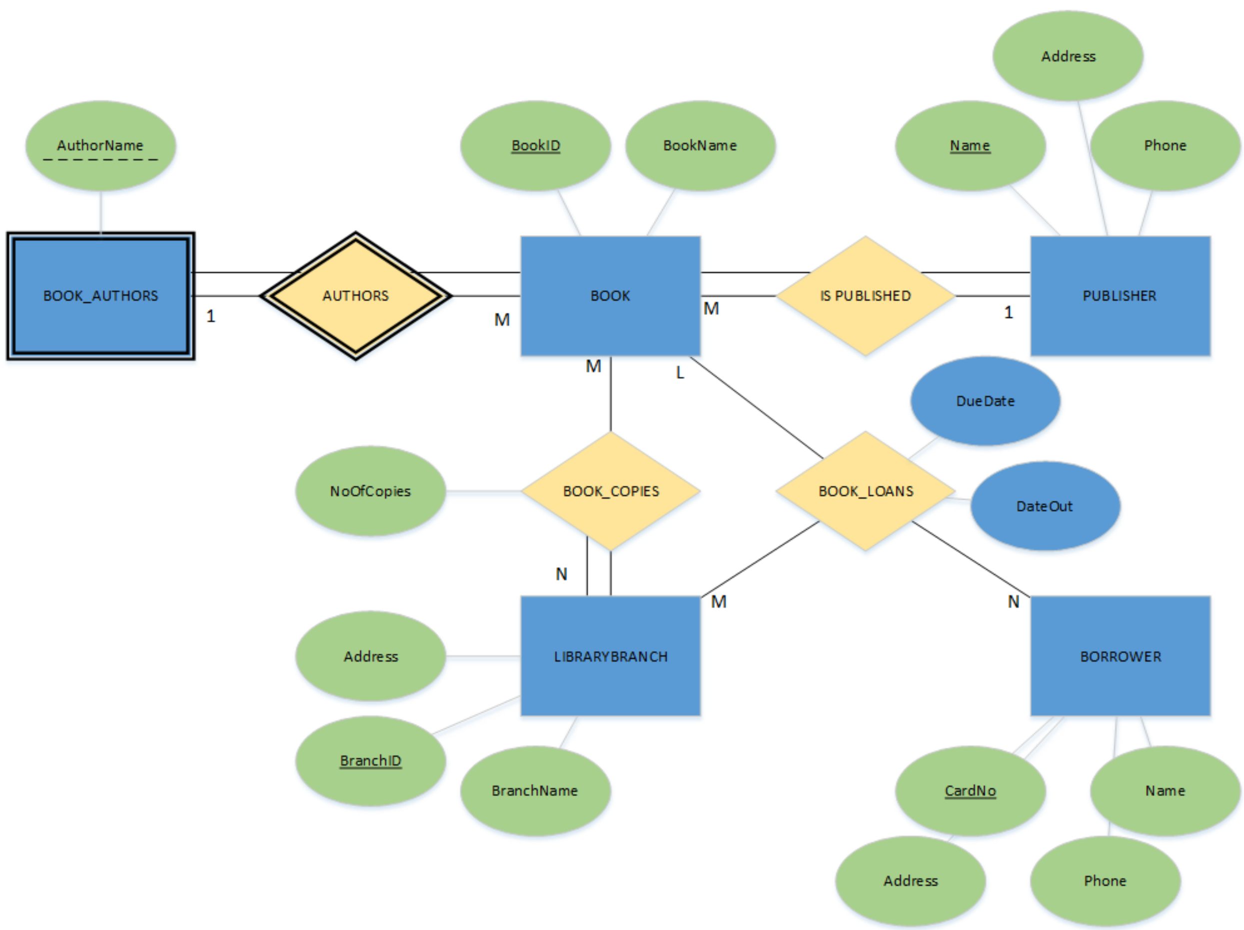
Assumptions:

- By looking at each attribute, I have concluded (and have assumed) that there are no derived dependencies because there are no obvious calculations that could be used to derive some attributes from others (like QuantitySold x SalePrice = TotalRevenue).
- I have also concluded that there are no transitive dependencies, which was based on what I know of each table attribute. Since there was no given description of each attribute, from common knowledge, I know that:
    - DateOut or DueDate cannot determine a book loan because there could be multiple books on loan that have either have the same due date, the same date out, or both.
    - NoOfCopies cannot determine book copies available in a library because there could be multiple books with the same amount of copies.
    - The rest of the tables primary keys determine the rest of the attributes.

Fact 4: Based on these assumptions,

✓ **Each table that was created in Step 2 are in second normal form.**

Step 4: Create ERD including each cardinality.



Question #6

Step 1: **Determine the functional dependencies of each entity.**

EMPLOYEE
SSN → FName, MInit, LName, BDate, Address, Sex, Salary, SuperSSN, DNo

DEPARTMENT
DNumber → DName, MgrSSN, MgrStartDate

DEPT_LOCATIONS
DNumber, DLocation

PROJECT
PNumber, → PName, PLocation, DNum

WORKS_ON
ESSN, PNo →Hours

DEPENDENT
ESSN, Dependent_Name → Sex, BDate, Relationship

Step 2: **Create separate tables for each A → B relationship (determined from above).**

EMPLOYEE(SSN, FName, MInit, LName, BDate, Address, Sex, Salary, SuperSSN*, DNo*)
DEPARTMENT(DNumber, DName, MgrSSN*, MgrStartDate)
DEPT_LOCATIONS(DNumber*, DLocation)
PROJECT(PNumber, PName, PLocation, DNum*)
WORKS_ON(ESSN*, PNo*, Hours)
DEPENDENT(ESSN*, Dependent_Name, Sex, BDate, Relationship)

Step 3: Place tables into 3rd normal form (if necessary).

Fact 1: Tables are assumed to be in first normal form because data for the tables are not given (just their attributes). Hence,

✓ **Each table that was created in Step 2 are in first normal form.**

Fact 2: EMPLOYEE, DEPARTMENT, and PROJECT are all in 2nd normal form because each has only 1 primary key.

Fact 3: Functional dependencies were removed in step 2, thus the remaining tables are in 2nd normal form.

✓ **Each table that was created in Step 2 are in second normal form.**

Assumptions:

- By looking at each attribute, I have concluded (and have assumed) that there are no derived dependencies because there are no obvious calculations that could be used to derive some attributes from others (like QuantitySold x SalePrice = TotalRevenue).

– I have also concluded that there are no transitive dependencies, which was based on what I know of each table attribute. Since there was no given description of each attribute, from common knowledge.

Fact 4: Based on these assumptions,

✓ **Each table that was created in Step 2 are in second normal form.**

Step 4: Create ERD including each cardinality.