CPTR 319 Review 1: Chapters 1-6

SQL

Directions: Below is a database design followed by different questions about the data. Write an SQL query for each question (1-9).

Students(ID, FirstName, LastName, Zip)

Courses(ClassID, ClassName, Description, DepartmentID)
(This is the top level of information like you would find in a Course catalog. ClassID might be “CPTR 319”

Teachers(ID, FirstName, LastName, DepartmentID, salary)

Classes(ClassID, Section, Year, Semester, TeacherID)

Enrollment(StudentID, ClassID)

Department(DepartmentID, DeptName, DeptChair)

1. List the names of all the classes offered.

SELECT Distinct ClassName
FROM Courses AS CO, Classes AS CL
WHERE CO.ClassID = CL.ClassID;

2. List all the student names with no duplicates.

SELECT DISTINCT FirstName, LastName
FROM Students;

3. List all the students in CPTR 319 in alphabetical order.

SELECT FirstName, LastName
FROM Students, Enrollment
WHERE ClassID = ‘CPTR 319’
ORDER BY LastName, FirstName;

4. List all teacher’s first and last names who are also students.

SELECT FirstName, LastName
FROM Teachers
INTERSECT
SELECT FirstName, LastName
FROM Students;

Assumption: Names are unique. I gave two other ways in class.

5. How many students total does the teacher with ID=045367 teach?

SELECT DISTINCT Count(E.StudentID)
FROM Classes AS C, Enrollment AS E
WHERE C.ClassID = E.ClassID AND C.TeacherID=’045367’;

6. List the number of students and ClassID in each class taught by the teacher with ID=045367, winter 2008. Order them by descending class size.

```
SELECT E.ClassID, Count(E.StudentID) AS EnrollmentCount
FROM Classes AS C, Enrollment AS E
WHERE C.ClassID = E.ClassID AND C.TeacherID='045367'
ORDER BY EnrollmentCount;
```

7. Find the total salaries paid by each department.

```
SELECT DepartmentID, Sum(Salary)
FROM Teachers
GROUP BY DepartmentID;
```

8. Find the maximum salary paid.

```
SELECT MAX(Salary)
FROM Teachers;
```

9. List all the teachers in the "Computing" department that are not the chair (assume "Computing" is the DepartmentID).

```
SELECT *
FROM Teachers
WHERE DepartmentID = 'Computing' AND
TeacherID NOT IN (
    SELECT DeptChair
    FROM Department
    WHERE DepartmentID = 'Computing');
```

10. List all classes with more than 30 students and order the classes by size from largest to smallest.

```
SELECT C.ClassID, C.Section, C.Year, C.Semester, Count(E.StudentID) AS CNT
FROM Classes AS C, Enrollment AS E
WHERE C.ClassID = E.ClassID
GROUP BY C.ClassID, C.Section, C.Year, C.Semester
HAVING CNT>30
ORDER BY CNT DESC;
```

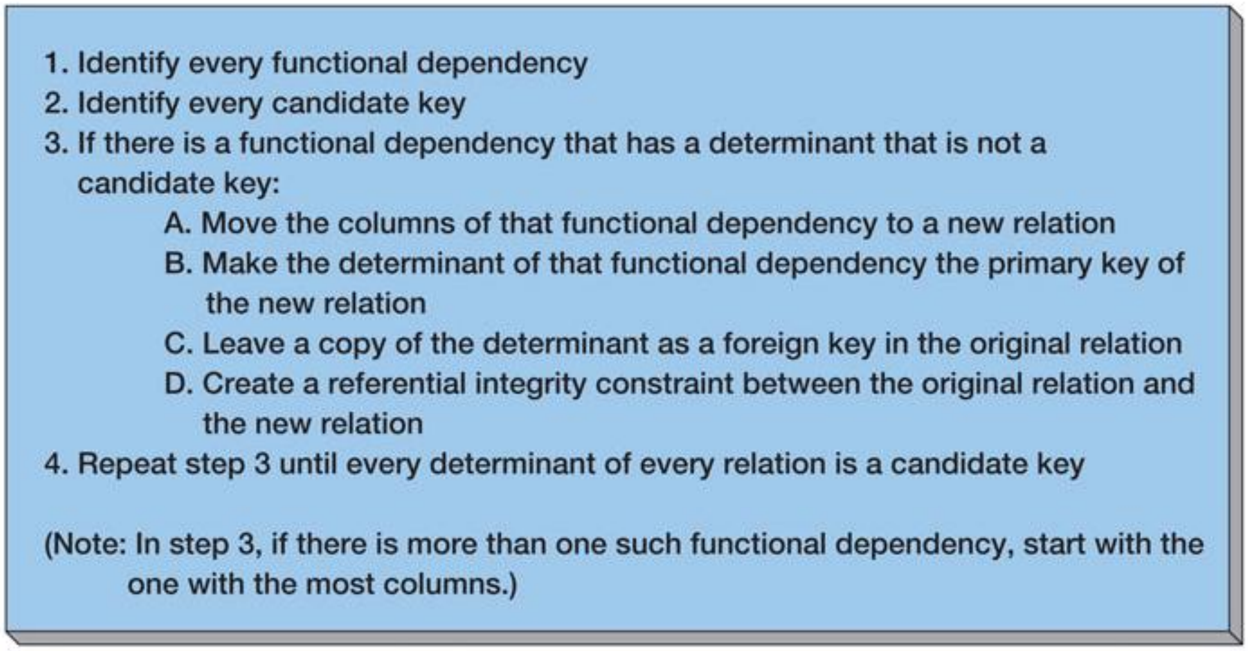
Principles of Database Design

Short Answer:

11. Define BCNF and 4th normal forms

    See Def in book

12. What are the steps to putting a database in BCNF and 4th normal form



1. Identify every functional dependency
2. Identify every candidate key
3. If there is a functional dependency that has a determinant that is not a candidate key:
    - A. Move the columns of that functional dependency to a new relation
    - B. Make the determinant of that functional dependency the primary key of the new relation
    - C. Leave a copy of the determinant as a foreign key in the original relation
    - D. Create a referential integrity constraint between the original relation and the new relation
4. Repeat step 3 until every determinant of every relation is a candidate key

(Note: In step 3, if there is more than one such functional dependency, start with the one with the most columns.)

13. Define candidate key **: See Book**

14. Define primary key**: See Book**

15. Define surrogate key**: See Book**

16. Define foreign key**: See Book**

Database Design

Suppose that we have the following description of hospital data:

Doctor: Name, age, specialty

Patient: Name height, weight, age

We track a patient's **ancestors** to determine hereditary diseases

We track patient's **visits** to doctors including the date and diagnosis and treatments.

Unfortunately Doctors have a list of Treatments that socialized medicine covers for a given diagnosis. So we have an approved **treatment** list from which doctors may pick one or more treatments to prescribe for each **diagnosis**.

Think carefully about the attributes of each entity when answering the questions below and designing the database.

17. List the entities and their attributes *using schema notation*.

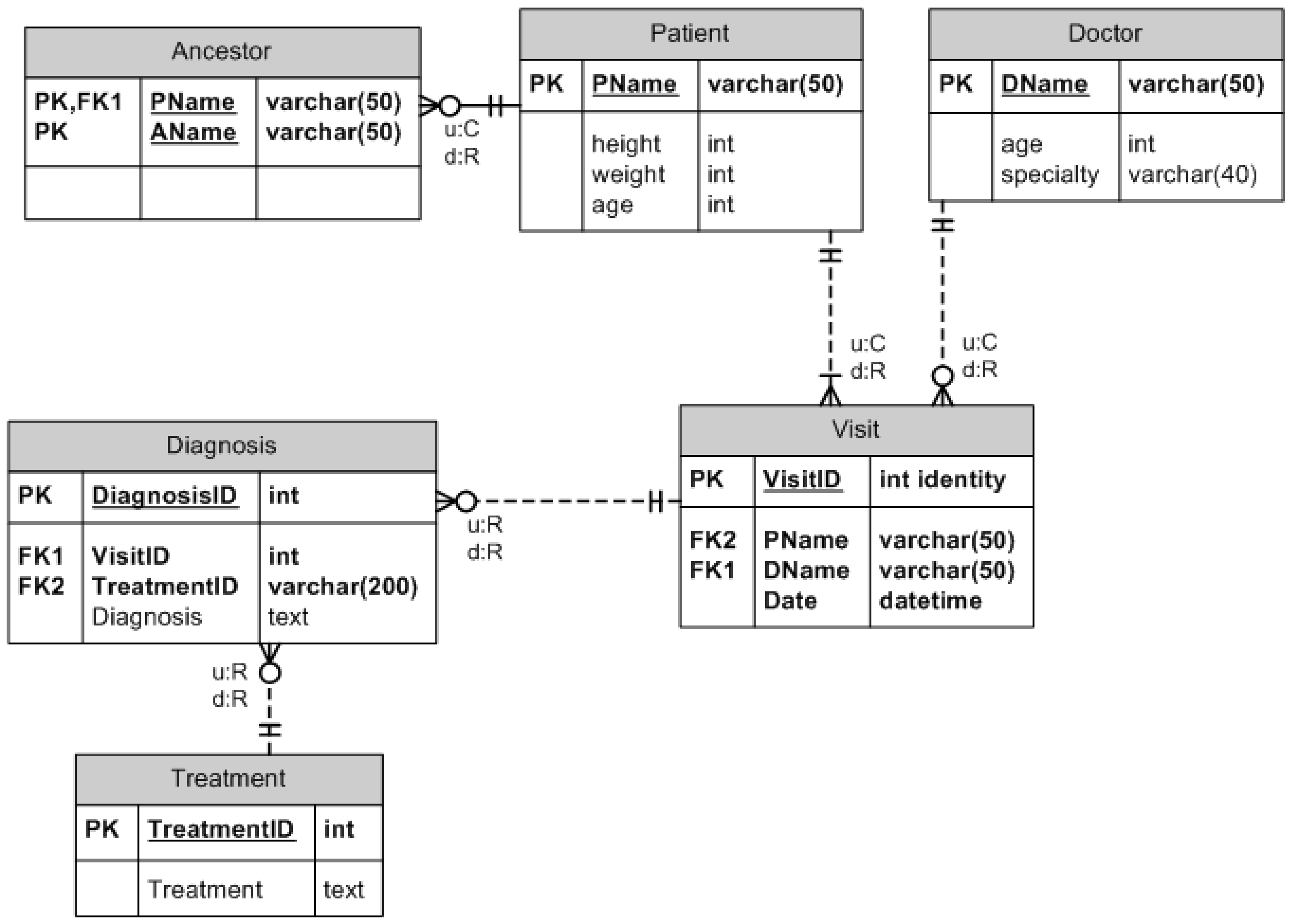
Doctor(<u>DName</u>, age, specialty) –Assumption: DNAme is unique
Patient(<u>PName</u>, height, weight, age) –Assumption: PName is unique
Ancestor(<u>*PName*</u>, <u>AName</u>) –Assumption: Ancestor Name may never have been a patient, so it's not a foreign key.
Visits(<u>VisitID</u>, *PName*, *Dname*, DateTime) – Assumption: Added Surrogate Key
Diagnosis(<u>DiagnosisID</u>, *VisitID*, Diagnosis, TreatmentID) –Assumption: Added Surrogate Key
Treatment(<u>TreatmentID</u>, Treatment)

18. Identify all functional and multiple dependencies for each entity
    Notice if you do 17 well, this becomes much easier!

Doctor:	DName → age, specialty
Patient:	PName → height, weight, age
Ancestor:	No functional dependencies
Visits:	VisitID → PName, DName, DateTime
Diagnosis:	DiagnosisID → VisitID, Diagnosis, TreatmentID
Treatment:	TreatmentID → Treatment

19. Draw an E-R diagram representing the information above. Include:
    a. Entities
    b. Relationships (Identifying, Non-Identifying)
    c. Maximum and Minimum Cardinality
    d. Make sure the design is in BCNF and 4th normal form
    e. Add surrogate keys as needed.
    f. Include constraints (e.g. U:R etc.).
    g. IF YOU FEEL THAT A VITAL PIECE OF INFORMATION IS MISSING, ASK.

Again: A good design above will simplify this immensely.



Ancestor
PK,FK1 PName varchar(50)
PK AName varchar(50)
u:C
d:R
Patient
PK PName varchar(50)
height int
weight int
age int
Doctor
PK DName varchar(50)
age int
specialty varchar(40)
u:C
d:R
u:C
d:R
Visit
PK VisitID int identity
FK2 PName varchar(50)
FK1 DName varchar(50)
Date datetime
Diagnosis
PK DiagnosisID int
FK1 VisitID int
FK2 TreatmentID varchar(200)
Diagnosis text
u:R
d:R
u:R
d:R
Treatment
PK TreatmentID int
Treatment text