COMSATS UNIVERSITY ISLAMABAD
Department of Computer Science
2nd Sessional Examination, Spring-2021
Solution

Course: Database Systems – I (CSC371) **Time Allowed (mins) / Total Marks: 60 / 15**
Instructors: Qasim Malik / Rashid Mehmood **Date: May 05, 2021**

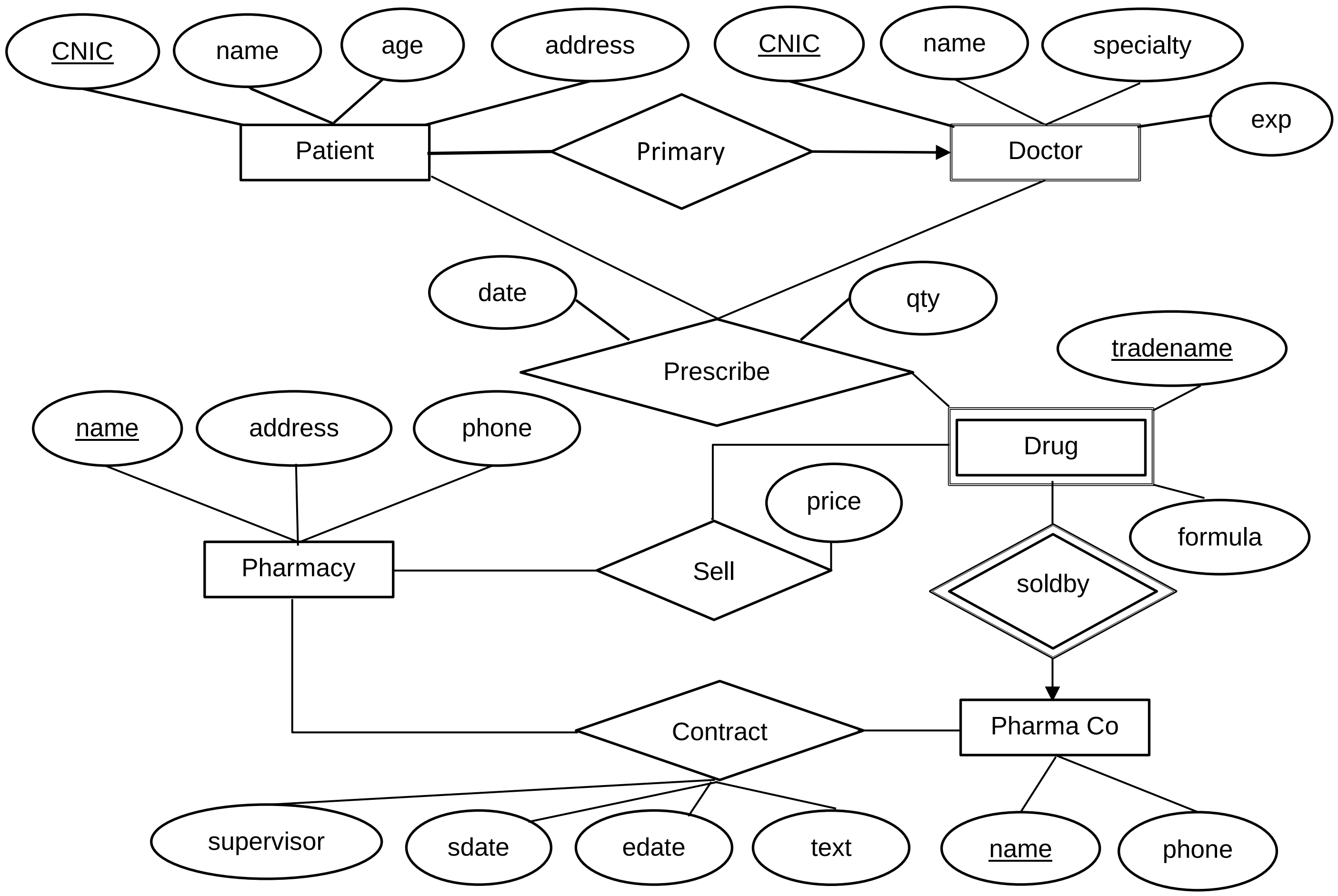
Student Name: ________________________ **Registration no: ____________________**

1- **[CLO-C3]** You are to design a normalized relational database for Shameen group of pharmacies. Here are the requirements provided to you:

- Patients are identified by their CNICs. For each patient, the name, address, and age must be recorded.
- Doctors are identified by their CNICs. For each doctor, the name, specialty, and years of experience must be recorded.
- Every patient has a primary doctor. Every doctor has at least one patient.
- Each pharmaceutical company is uniquely identified by name and has a phone number.
- For each drug, the trade name and formula must be recorded. Each drug is sold by a given pharmaceutical company, and the trade name identifies a drug uniquely from among the products of that company only and not across all the companies.
- Each pharmacy has a unique name, address, city, and phone number.
- Each pharmacy sells several drugs and has a price for each. A drug could be sold at several pharmacies, and the price could vary from one pharmacy to another.
- Doctors prescribe drugs for patients. A doctor could prescribe one or more drugs for several patients, and a patient could obtain prescriptions from several doctors. Each prescription is assigned an ID, has a date and, a quantity associated with it.
- Pharmaceutical companies have long-term contracts with pharmacies. A pharmaceutical company can have a contract with several pharmacies, and a pharmacy can have contracts with several pharmaceutical companies. For each contract, you have to store a start date, an end date, and the text of the contract. Pharmacies also appoint a supervisor for each contract and the supervisor's name must also be stored.

Part-I: Draw an Entity-Relationship Diagram (ERD) that captures the above information. **[4]**

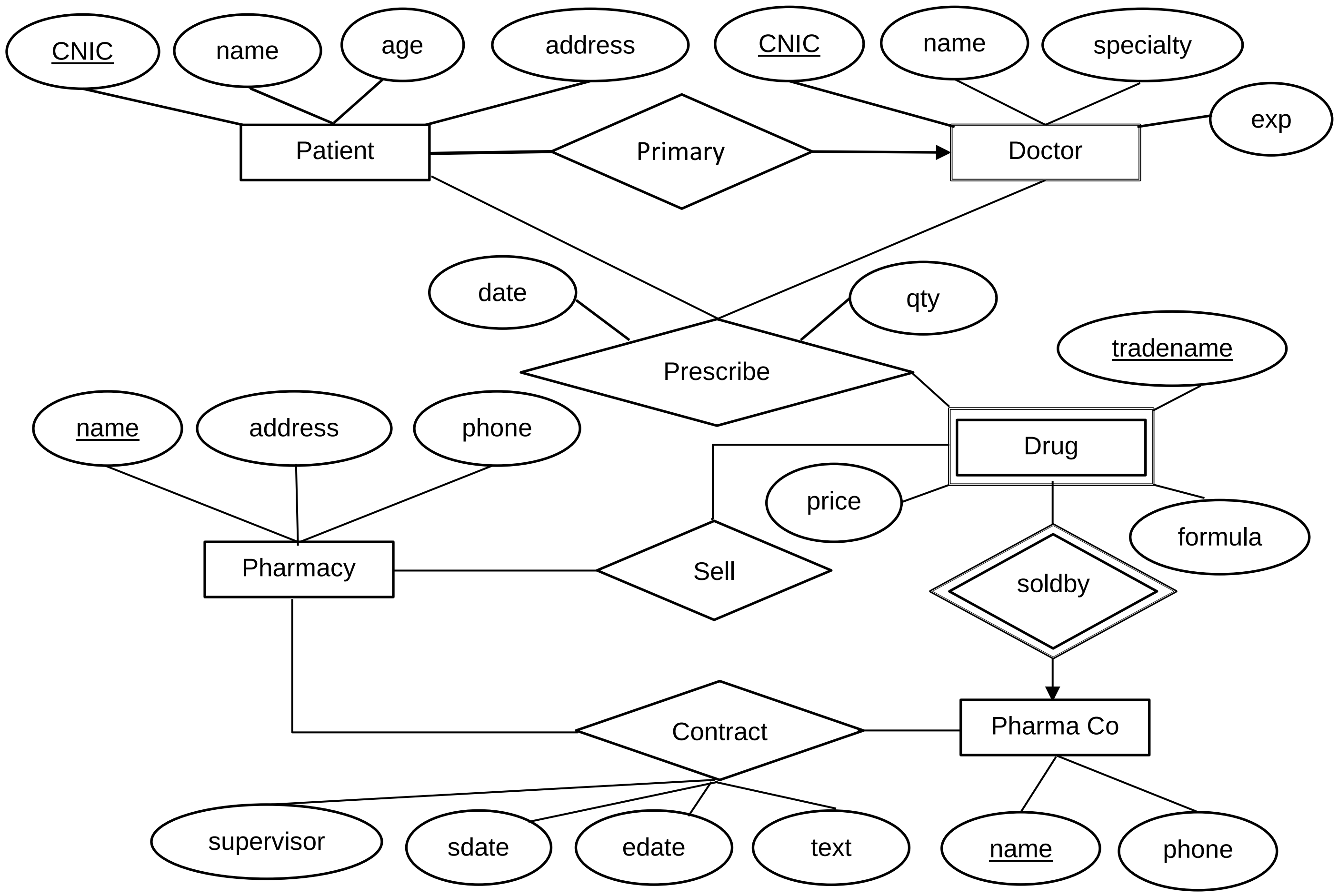
Solution:



Part-II: If we later on decide that each drug must be sold at a fixed price by all pharmacies, how would the design of your ERD change? **[1]**

Solution:

We will now make *price* an attribute of *Drug* entity type.



2- **[CLO-C3]** Map the ERD created in Question 1 (Part-I), to its corresponding relational schema. Specify all the primary key and foreign key constraints in the resultant schema. **[3]**

Solution:

Patient(<u>CNIC</u>, name, age, address, doctorCNIC [FK])

Doctor(<u>CNIC</u>, name specialty, exp)

Pharmacy(<u>name</u>, address, phone)

PharmaCo(<u>name</u>, phone)

Drug(<u>pharmaCo_name</u> [FK], <u>tradename</u>, formula)

Prescribe(<u>patienCNIC</u> [FK], <u>doctorCNIC</u> [FK], <u>pharmaCo_name</u> [FK], <u>tradename</u> [FK], date, qty)

Sell(<u>pharmacy_name</u> [FK], <u>pharmaCo_name</u> [FK], <u>tradename</u> [FK], price)

Contract(<u>pharmacy_name</u> [FK], <u>pharmaCo_name</u> [FK], supervisor, sdate, edate, text)

3- **[CLO-C3]** A small bank wants to redesign a database to manage the accounts of their clients. The bank has many local branches, each identified by a unique branch code. Accounts belong to a specific branch, and are identified by account numbers that are unique within that branch. An account can be shared by several clients, and the only other information necessary to store is the account balance.
For clients, the bank stores their CNIC, first and last names, as well as a contact address and phone number. At the moment the schema of its database is quite simple and have only one mega relation which looks like:

`Bank (CNIC, firstName, lastName, address, phoneNumber, branchCode, branchName, accountNumber, accountBalance)`

Part-I: This schema is not fully normalized, and thus suffers from a number of problems. Provide examples of how insertion, deletion, and update anomalies can occur in this table. **[2]**

Solution:

<u>Insertion:</u> If we want to insert only a new branch, it requires other insertions to be made in order to have consistent data (e.g., a new branch can be inserted only if it has a client having an account in it); such supplementary insertions are neither necessary nor desirable.
<u>Deletion:</u> If we want to delete the lone client in a branch, it will result in the deletion of the branch from the database too.
<u>Modification:</u> If we modify the accountBalance of a shared account, we will have to update it in several rows. If we forget it to propagate everywhere, the database will become inconsistent.

Part-II: Identify all functional dependencies that are expected to hold in this mega relation based on the given information. **[2]**

Solution:

FD1: CNIC → firstName, lastName, address, phoneNumber
FD2: branchCode → branchName
FD3: branchCode, accountNumber → accountBalance

Part-III: Bring the table to BCNF (you can do this both directly or by going through intermediate forms i.e., 2NF and 3NF). Specify the primary keys in all the resultant relations. **[3]**

Solution:

To start with, we have a mega relation: *Bank* (CNIC, firstName, lastName, address, phoneNumber, branchCode, branchName, accountNumber, accountBalance)

Let's first compute the keys for *Bank*.

Since {CNIC, branchCode, accountNumber}$^+$ = { CNIC, firstName, lastName, address, phoneNumber, branchCode, branchName, accountNumber, accountBalance }, therefore the key is { CNIC, branchCode, accountNumber }. It is the only candidate key in this relation.

Let's now apply the BCNF decomposition algorithm:

For *Bank* table to be in BCNF, all the applicable FD's must satisfy BCNF condition. Let's choose FD1 first. FD1 violates the BCNF condition which states that for a given FD: A → B, A (L.H.S) must be a key. Since CNIC alone is not a key for *Bank,* as a result of this violation, we will decompose the relation *Bank* into two following relations:

Client(<u>CNIC</u>, firstName, lastName, address, phoneNumber)
Other(CNIC, branchCode, branchName, accountNumber, accountBalance)

The *Client* relation now does not violate BCNF condition for any of the FDs that are applicable to it (only FD1 in this case). Hence, *Client* is in BCNF.

Now let's see if *Other* relation is in BCNF or not. For that let's first compute the keys for *Other* relation.

Since {CNIC, branchCode, accountNumber}$^+$ = { CNIC, branchCode, branchName, accountNumber, accountBalance }, therefore the key is { CNIC, branchCode, accountNumber }. It is the only candidate key in this relation.

Both FD2 and FD3 apply to *Other* relation. Let's choose FD2 first. FD2 violates the BCNF condition since {branchCode}$^+$ = { branchCode , branchName} ≠ All of the attributes of *Other* relation. As a result of this violation, we will decompose the relation *Other* into two following relations:

Branch(branchCode, branchName)
Account(branchCode, accountNumber, accountBalance)

The *Branch* relation now does not violate BCNF condition for any of the FDs that are applicable to it (only FD2 in this case). Hence, *Branch* is in BCNF.

Now let's see if *Account* relation is in BCNF or not. For that let's first compute the keys for *Account* relation.

Since {branchCode, accountNumber}$^+$ = { branchCode, accountNumber, accountBalance }, therefore the key is { branchCode, accountNumber }. It is the only candidate key in this relation.

The *Account* relation also does not violate BCNF condition for any of the FDs that are applicable to it (only FD3 in this case). Hence, *Account* is in BCNF.

Since all the decomposed relations are now in BCNF, the algorithm stops here. The resultant schema is therefore:

Client(CNIC, firstName, lastName, address, phoneNumber)
Branch(branchCode, branchName)
Account(branchCode, accountNumber, accountBalance)