CSE 414 Databases

Assignment 1

a. User requirements:

- Patient management: The system should provide features to efficiently manage patient information, including personal details, medical history, allergies, current medications, and ony relevant documents
- Appointment Scheduling: Users should be able to schedule appointments for patients. including selecting a preffered healthcare provider, specifying the reason for the visit, and managing the appointment calender to avoid conflicts.
- Medical Records: The system should enable authorised users to access and update comprehensive medical records for each patient, including diagnoses, treatments, laboratory results, imaging reports, and other relevant medical information.
- Billing and Payment: The database should incorporate billing and payment functionalities. to manage patient billing, insurence claims, and payment processing. It should handle financial transactions, generate invoices, and provide clear documentation for accounting.
- Inventory and Resource Management: The system should track and manage the inventory of medical supplies, medications, and equipment, ensuring their availability and efficient usage.
- Staff Management: The database should support the management of staff information, including employee details, qualifications, work schedules, and performance records.

b. Entities:

- Patient: This entity represents individual patients, and may include attributes such as patient ID, name, contact information, date of birth, gender, and insurance details.
- Healthcare provider: This entity represents healthcare professionals, such as doctors, nurses, and specialists. It may include attributes like provider ID, name, contact information, specialty, and qualifications.
- Appointment: This entity represents appointments scheduled between patients and healthcare providers. It may have attributes such as appointment ID, date, time, duration, status, and any additional notes.
- Medical record: This entity represents the medical records associated with each patient. It may include attributes like record ID, patient ID (foreign key), date of visit, diagnosis, treatment, prescribed medications, and any relevant test results.
- Billing: This entity represents the billing information for patients. It may include attributes like billing ID, patient ID (foreign key), invoice number, date, amount, payment status, and any additional billing details.
- Inventory item: This entity represents items in the inventory, such as medical supplies, medications, and equipment. It may include attributes like item ID, name, description, quantity, supplier details, and reorder level.

Relationships:

- Patient - Appointment relationship: A patient can have multiple appointments, and an appointment is associated with a single patient. This is a one-to-many relationship.
- Patient - Medical Record relationship: A patient can have multiple medical records, and each medical record is associated with a single patient. This is one-to-many relationship.
- Healthcare Provider - Appointment relationship: A healthcare provider can have multiple appointments, and an appointment is associated with a single healthcare provider. This is a one-to-many relationship.
- Patient - Billing relationship: A patient can have multiple billing records, and each billing record is associated with a single patient. This is one-to-many relationship.
- Inventory item - Supplier relationship: An inventory item can have a single supplier, and a supplier can supply multiple inventory items. This is a one-to-many relationship.

Weak Entity:

- Prescription: This entity represents the prescription provided to a patient by a healthcare provider. It depends on the medical report entity for identification since it is associated with a specific medical visit and it is not meaningful without it. The prescription entity may include attributes such as prescription ID, medical record ID (foreign key), date, prescribed medication details, dosage instructions, and duration.

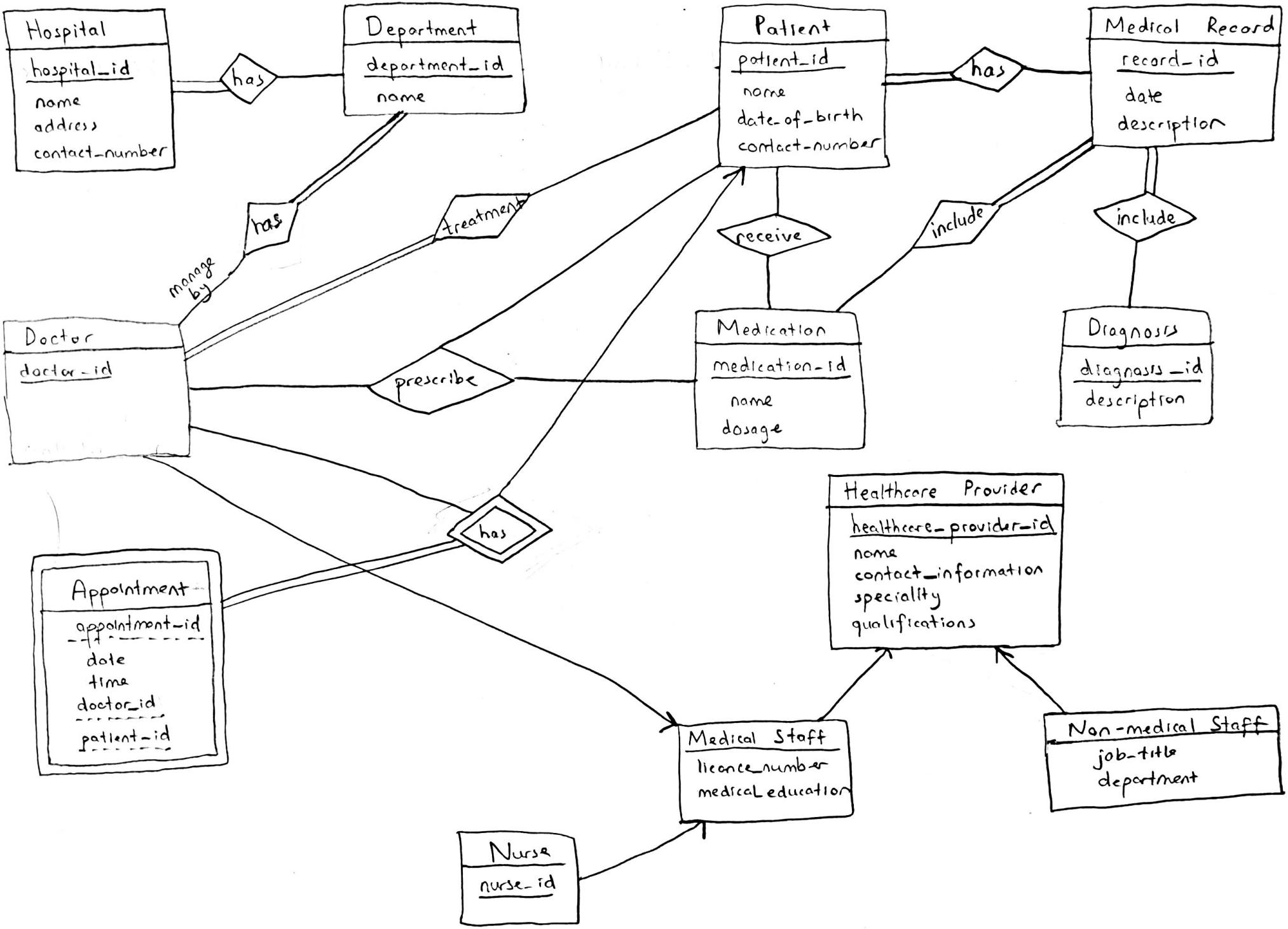
In this case the medical record entity would be the identifying or owner entity. The relationship between the medical record and prescription entities would be a one-to-many relationship, as one medical record can have multiple prescriptions but each prescription is associated with a single medical record.

Total Completeness Constraint:

A total completeness constraint means that every entity in the superclass must be associated with at least one entity in the subclass.

Partial Completeness Constraint:

A partial completeness constraint means that entities in the superclass may or may not have associations with entities in the subclass.


Hospital
hospital_id
name
address
contact_number
has
Department
department_id
name
Patient
patient_id
name
date_of_birth
contact_number
has
Medical Record
record_id
date
description
has
manage by
Doctor
doctor_id
treatment
receive
include
include
Medication
medication_id
name
dosage
Diagnosis
diagnosis_id
description
prescribe
has
Appointment
appointment_id
date
time
doctor_id
patient_id
Healthcare Provider
healthcare_provider_id
name
contact_information
speciality
qualifications
Medical Staff
licence_number
medical education
Non-medical Staff
job_title
department
Nurse
nurse_id

c. Hospital:

- hospital_id → name, address, contact_number

Department:

- department_id → name

Doctor:

- doctor_id → name, specialization, contact_number
- department_id → doctor_id (role relationship: a Doctor manages a Department)

Patient:

- patient_id → name, date-of-birth, contact_number

Medical Record:

- record_id → date, description
- patient_id → record_id (a Patient has a Medical Record)

Diagnosis:

- diagnosis_id → description
- record_id → diagnosis_id (a Medical Record includes Diagnoses)

Medication:

- medication_id → name, dosage
- record_id → medication_id (a Medical Record includes Medications)
- doctor_id, patient_id → medication_id (Doctor prescribes Medications to Patients)

Appointment (weak entity)

- appointment_id → date, time
- doctor_id, patient_id → appointment_id

Nurse (inheritance from Medical Staff)

- nurse_id → name, specilization, contact_number, additional nurse attributes

d. Table 1: Doctor

Candidate key = doctor_id

To check if it is in BCNF:

The only functional dependency in this table is doctor_id → name, specialization, contact_number. The determinant (doctor_id) is a candidate key. Therefore it satisfies the condition for being in BCNF.

To check if it is in 3NF:

Since it is in BCNF, it is also in 3NF.

Conclusion: The Doctor table is in both BCNF and 3NF.

Table 2: Medical Record

Candidate key: record_id

For same reasons with Doctor table, Medical Record table is in both BCNF and 3NF.

Table 3: Appointment

Candidate key: appointment_id

For the same reasons with Doctor and Medical Record tables, Appointment is in both BCNF and 3NF.

e. a. Table SQL Function

```
CREATE FUNCTION GetDoctorsBySpecialization (@specialization
RETURNS TABLE                                VARCHAR(50))
AS
RETURN (
    SELECT doctor_id, name, specialization, contact_number
    FROM Dactor
    WHERE specialization = @specialization
);
```

b. Function with a for loop

```
CREATE FUNCTION CalculateTotalMedicationDuration (@patientId INT)
RETURNS INT
AS
BEGIN
  DECLARE @totalDuration INT = 0;
  DECLARE @medicationId INT;
  DECLARE @duration INT;
  DECLARE medication_cursor CURSOR FOR
  SELECT m.medication_id, m.duration
  FROM Medication m
  INNER JOIN MedicalRecord mr ON m.record_id = mr.record_id
  WHERE mr.patient_id = @patientId;

  OPEN medication_cursor;
  FETCH NEXT FROM medication_cursor INTO @medicationId, @duration;

  WHILE @@FETCH_STATUS = 0
  BEGIN
    SET @totalDuration = @totalDuration + @duration;
    FETCH NEXT FROM medication_cursor INTO @medicationId, @duration;
  END;

  CLOSE medication_cursor;
  DEALLOCATE medication_cursor;

  RETURN @totalDuration;

END;
```

c. SQL Function with Variable Input, Temporary Variable, and Output

```
CREATE FUNCTION CalculatePatientAge (@dateOfBirth DATE)
RETURNS INT
AS
BEGIN
    DECLARE @currentDate DATE = GETDATE();
    DECLARE @age INT;

    SET @age = DATEDIFF (YEAR, @dateOfBirth, @currentDate);

    RETURN @age;
END;
```

f. a.

```
CREATE TRIGGER UpdateLastModified
ON Doctor
FOR INSERT, UPDATE
AS
BEGIN
    UPDATE d
    SET last_modified = GETDATE()
    FROM Doctor d
    INNER JOIN inserted i ON d.doctor_id = i.doctor_id;
END;
```

b.

```
CREATE TRIGGER AssignToDepartment
ON Doctor
AFTER INSERT
AS
BEGIN
    IF (SELECT specialization FROM inserted) = 'Cardiology'
    BEGIN
        INSERT INTO Department (name) VALUES ('Cardiology');
    END;
    ELSE IF (SELECT specialization FROM inserted) = 'Neurology'
    BEGIN
        INSERT INTO Department (name) VALUES ('Neurology');
    END;
      :
      :
END;
```

c.
```
CREATE TRIGGER UpdatePatientAge
AFTER UPDATE ON Patient
FOR EACH ROW
AS
BEGIN
  IF UPDATE (date_of_birth)
  BEGIN
    UPDATE Patient
    SET age = DATEDIFF (YEAR, inserted.date_of_birth, GETDATE())
    WHERE patient_id = inserted.patient_id;
  END;
END;
```

d.
```
CREATE TRIGGER CheckMedicalRecord
AFTER INSERT ON MedicalRecord
FOR EACH CASE
WHEN (SELECT COUNT(*) FROM Diagnosis WHERE record_id =
inserted.record_id) > 3
BEGIN
-- Perform actions when the number of diagnoses for a record exceeds 3
PRINT 'Warning: Excessive diagnoses for Medical Record ' +
    CAST(inserted.record_id AS VARCHAR(10));
-- Additional actions can be added
END;
```

e. To drop a trigger:

```
DROP TRIGGER trigger_name;
```

To show the triggers in a database;

```
SHOW TRIGGERS;
```

(MySQL)

9. 1.st

```
BEGIN TRANSACTION TransferPatient;

UPDATE MedicalRecord
SET department_id = @newDepartmentId
WHERE patient_id = @patientId;

UPDATE Doctor
SET department_id = @newDepartmentId
WHERE doctor_id = @doctorId;

COMMIT TRANSACTION TransferPatient;
```

2.nd

```
BEGIN TRANSACTION DischargePatient;

DELETE FROM MedicalRecord
WHERE patient_id = @patientId;

UPDATE Doctor
SET patient_count = patient_count - 1
WHERE doctor_id = @doctorId;

DELETE FROM Diagnosis
WHERE patient_id = @patientId;

DELETE FROM Medication
WHERE patient_id = @patientId;

DELETE FROM Appointment
WHERE patient_id = @patientId;

DELETE FROM Patient
WHERE patient_id = @patientId;

COMMIT TRANSACTION DischargePatient;
```