CSE 414 - Databases HW 1

Abdullah Gelik
171044002

a- User Requirements

1- Users need to be able to register patients with personal information, medical history and insurance details.

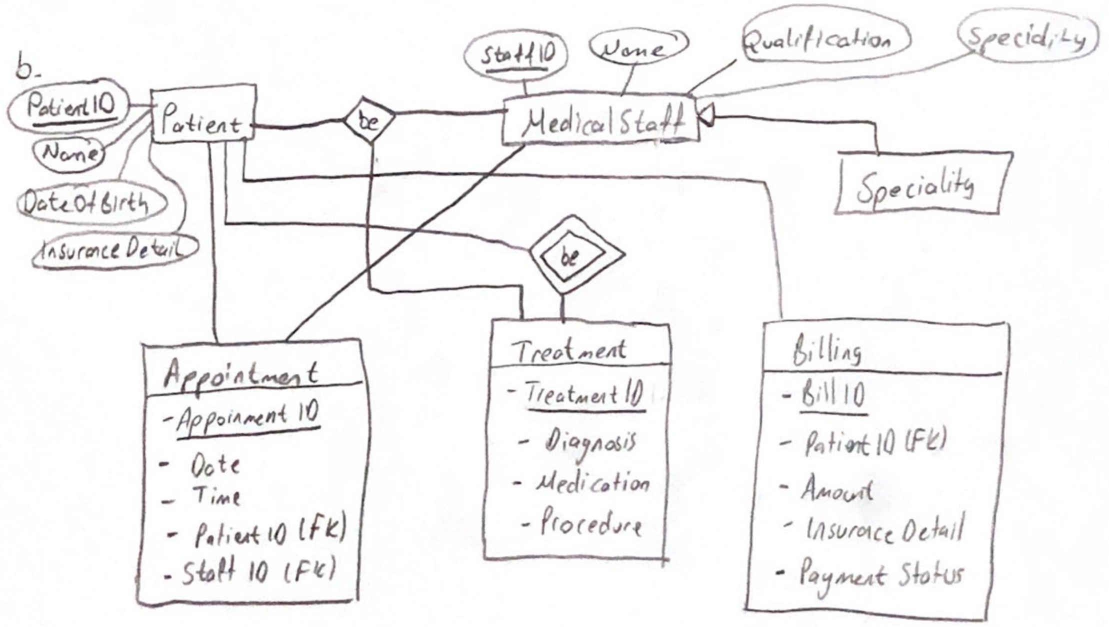
2- Doctors and nurses need to have profiles with their specialties, qualifications and schedules.

3- Patients need to be able to book and cancel appoinments with available doctors.

4- System should track treatment history including diagnoses, prescribed medications, performed procedures and hospitalizations.

5- Billing and payment information should be recorded and linked to insurance providers if applicable.

b-



c- Functional dependencies

1- 'Patient ID' → 'Name', 'Date Of Birth', 'Insurance Detail'

2- 'StaffID' → 'Name', 'Qualification', 'Speciality'

3- 'Appoinment ID' → 'Date', 'Time', 'Patient ID', 'Staff ID'

4- 'Treatment ID', 'Patient ID' → 'Diagnosis', 'Medication', 'Procedure'

5- 'Bill ID' → 'Patient ID', 'Amount', 'Insurance Detail', 'Paymet Status'

d. Normalization proofs:

1- 'Patient' table: 'Patient ID' (PK), 'Name', 'Date Of Birth', 'Insurance Detail'

- 3NF: Each non-key attribute is fully functionally dependent on 'Patient ID'
- BCNF: As there is only one candidate key, 'PatientID', and each non-key attribute is fully functionally dependent on it, the table is also in BCNF.

2- 'Appoinment' table: 'Appoinment ID' (PK), 'Date', 'Time', 'Patient ID' (FK), 'StaffID' (FK)

- 3NF and BCNF: Each non-key attribute is fully functionally dependent on 'Appoinment ID'

3. 'Billing' table: 'Bill ID' (PK), 'Patient ID' (FK), 'Amount', 'Insurance Detail', 'Payment Status'

- 3NF and BCNF: Each non-key attribute is fully functionally dependent on 'Bill ID'.

e. SQL functions:

1- Table SQL Function:

```
CREATE FUNCTION get-patient-treatments ()
RETURNS TABLE (Patient ID INT, Treatment ID INT, Diagnosis VARCHAR (255),
     Medication VARCHAR (255), Procedure VARCHAR (255) AS

BEGIN
   RETURN QUERY SELECT * FROM Treatment
END
```

2- Function with a for loop:

```
CREATE FUNCTION count-patients() RETURNS INT AS $$
DECLARE
        count INT := 0;
        patient RECORD;
BEGIN
    FOR patient IN SELECT * FROM Patient LOOP
        count := count +1;
    END LOOP
    RETURN count
END
```

3- Function with input variable, temporary variable and output variable:

```
CREATE FUNCTION get-patient-age (patient-id INT) RETURNS INT AS $$
DECLARE
    birth-date DATE;
    age INT;
BEGIN
    SELECT DateOfBirth INTO birth-date FROM Patient WHERE PatientID = patient-id;
    age := EXTRACT (YEAR FROM AGE (birth-date));
    RETURN age;
END
```

f. Triggers:

1- Trigger using "referencing old row as" and "referencing new row as":

```
CREATE OR REPLACE FUNCTION log-patient-update() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO PatientHistory (old-data, new-data, updated-data)
    VALUES (OLD.*, NEW.*, Now());
    RETURN NEW;
END

CREATE TRIGGER update-patient-history
AFTER UPDATE ON Patient
FOR EACH ROW
EXECUTE PROCEDURE log-patient-update();
```

2. Trigger with "WHEN" or "IF":

```
CREATE OR REPLACE FUNCTION verify_adult_patient() RETURNS TRIGGER
    AS $$
BEGIN
  IF (NEW.DateOfBirth > NOW() - INTERVAL '18 YEAR') THEN
     RAISE EXCEPTION 'Patient must be an adult.';
  END IF
  RETURN NEW
END

CREATE TRIGGER check_patient_age
BEFORE INSERT OR UPDATE ON Patient
FOR EACH ROW
EXECUTE PROCEDURE verify_adult_patient();
```

3. Trigger for each row:

```
CREATE OR REPLACE FUNCTION log_treatment_update() RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO TreatmentHistory (TreatmentID, PatientID, updated_at)
  VALUES (NEW.TreatmentID, NEW.PatientID, Now());
  RETURN NEW
END

CREATE TRIGGER update_treatment_history
AFTER UPDATE ON Treatment
FOR EACH ROW
EXECUTE PROCEDURE log_treatment_update();
```

4. Trigger with for each statement:

```
CREATE OR REPLACE TRIGGER total_treatment_count
AFTER INSERT ON Treatment
DECLARE
   total_count NUMBER;
BEGIN
  SELECT COUNT(*) INTO total_count FROM Treatment;
  DBMS_OUTPUT.PUT_LINE('Total treatments: ' || total_count);
END
```

5- Drop a trigger:

```
DROP TRIGGER IF EXISTS update_patient_history ON Patient;
```

Show triggers:

```
SELECT * FROM information_schema.triggers WHERE trigger_schema
    NOT LIKE 'pg%';
```

9- Atomic transactions:

1- Registering a new patient and booking their first appainment:

```
BEGIN
INSERT INTO Patient (Name, DateOfBirth, InsuranceDetail) VALUES
    ('Abdullah', '2000-01-01', 'Insurance Co.')
INSERT INTO Appoinment (Date, Time, PatientID, StaffID) VALUES
    ('2023-14-05', '12:00', (SELECT PatientID FROM Patient WHERE
     Name = 'Abdullah'), 1)
COMMIT
```

2- Recording a treatment and updating the bill:

```
BEGIN
INSERT INTO Treatment (Diagnosis, Medication, Procedure, PatientID)
        VALUES ('Flu', 'Antiviral medication', 'None', 1)
UPDATE Billing SET Amount = Amount + 100 WHERE PatientID = 1
COMMIT
```