Database System - CO2014

Assignment 2

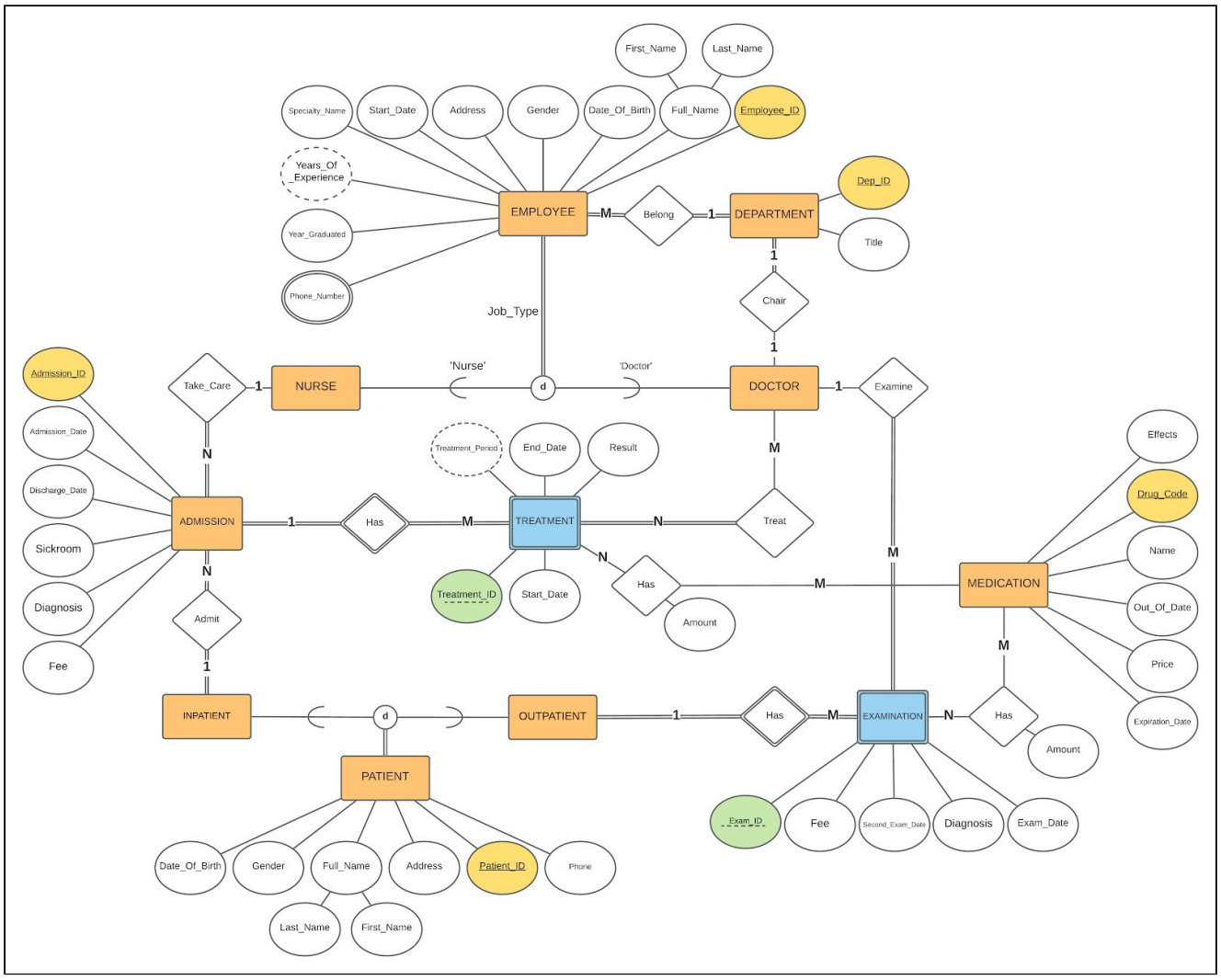
HOSPITAL DATABASE

Lecturer: Dr. Phan Trong Nhan

CC04 - Group 4

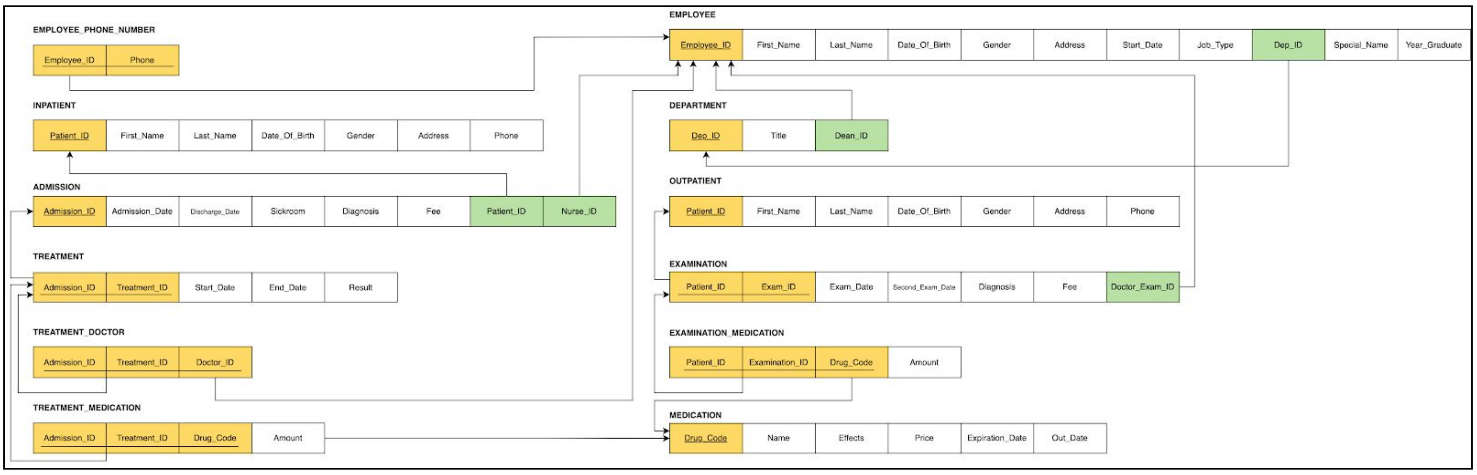
Name	Student ID
Trương Lê Vinh Khoa	1752298
Nguyễn Minh Đăng	1752170
Lê Nguyễn An Khương	1752305
Hồ Lê Thục Quyên	1752454
Lê Hiếu Phương	1752431

EER Diagram



For a better view of the image, please refer to this link: [ER_Model]

Relational Database Schema



For a better view of the image, please refer to this link: [Relational_Table]

Physical Implementation

1. Overall

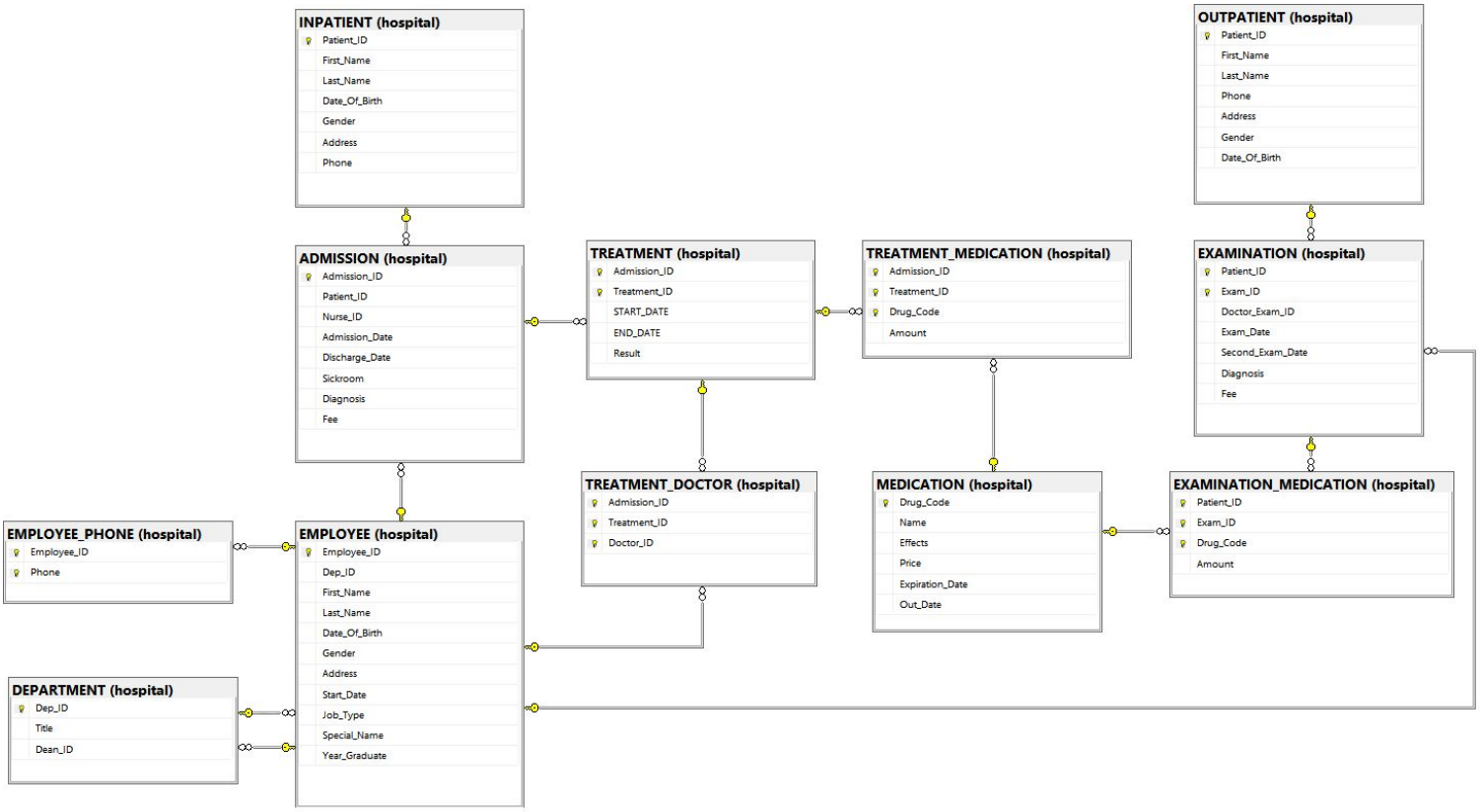
1.1. Database management system:

In this assignment, our group uses the **Microsoft SQL Server,** which is a relational database management system developed by Microsoft.

1.2. Database server:

To deploy our database, we choose **Azure SQL Database**, which is based on the latest stable version of the Microsoft SQL Server database engine.

2. Implementation



A. Tables

1/ MEDICATION

- **Create**

```
CREATE TABLE hospital.MEDICATION
(
   Drug_Code INTEGER IDENTITY(1,1),
```

```sql
    Name VARCHAR(20) NOT NULL,
    Effects VARCHAR(100) NOT NULL,
    Price NUMERIC(10,1) NOT NULL,
    Expiration_Date DATE NOT NULL,
    Out_Date CHAR(1) NOT NULL DEFAULT 0,
    CHECK(Out_Date IN ('0','1')),
    CONSTRAINT Empty_PK_MEDICATION
        PRIMARY KEY(Drug_Code)
);
```

- **Primary Key:** ***Drug_Code***
- **Foreign Key:** *None*
- **Trigger:**
  - Tri_Update_Drug_Out_Date: We create a trigger in Azure Server that will trigger the procedure Check_Outdate_Medication daily at 12:00AM

```sql
CREATE OR ALTER PROCEDURE hospital.Check_Outdate_Medication
AS
    UPDATE
        MEDICATION
    SET
        Out_Date = 1
    WHERE
        Expiration_Date < CONVERT(date, CURRENT_TIMESTAMP);
```

- **Constraint:**
  - Make sure Out_Date is ‘1’ or ‘0’. ‘1’ means the medication is outdated while ‘0’ means the medication is not outdated.
- **Data type:**
  - **VARCHAR(20)** is used for the drug name because it is a suitable data type and 20 characters are enough to show.
  - **VARCHAR(100)** is used for effects because it is a suitable string data type for “Effects” and 100 is long enough to briefly describe the effects of the drug.

- **CHAR(1)** is used for the flag "Out_Date" as it just only shows if the drug is out of date or not then 1 character is enough. '1' means outdated while '0' is not outdated
- **INTEGER** is used for **Drug_Code** instead of *smallint* since *smallint* limits the uniqueness of **Drug_Code** to about 32 000 while INTEGER up to 2,147,483,647. Therefore, **INTEGER** is suitable to make sure every employee gets a unique code
- **NUMERIC(10,1)** is used for the price of a drug because it is suitable to limit the price below 10,000,000,000 and keep one after decimal point in Vietnamese currency (VND).
- **DATE** is used to represent the date of expiration.

2/ EMPLOYEE

- **Create**

```sql
CREATE TABLE hospital.EMPLOYEE
(
   Employee_ID INTEGER IDENTITY(1,1),
   Dep_ID INTEGER NOT NULL,
   First_Name VARCHAR(7) NOT NULL,
   Last_Name VARCHAR(35) NOT NULL,
   Date_Of_Birth DATE NOT NULL,
   Gender CHAR(1) NOT NULL,
   Address VARCHAR(50) NOT NULL,
   Start_Date DATE NOT NULL,
   Job_Type CHAR(1) NOT NULL,
   Special_Name VARCHAR(40),
   Year_Graduate DATE,
   CONSTRAINT Empty_PK_EMPLOYEE
       PRIMARY KEY (Employee_ID),
   CHECK(Gender IN ('m','f')),
   CHECK(Job_Type IN ('d', 'n')),
   CHECK(DATEDIFF(year,Date_Of_Birth,GETDATE()) > 18)
);
-- Add foreign key deapartment id for employee
ALTER TABLE hospital.EMPLOYEE
ADD CONSTRAINT Empty_FK_Dep_ID_EMPLOYEE
   FOREIGN KEY (Dep_ID) REFERENCES hospital.DEPARTMENT(Dep_ID);
```

- **Primary Key:** ***Employee_ID***: auto-set by database
- **Foreign Key:** ***Dept_ID*** reference to attribute ***Dep_ID*** in table *hospital.DEPARTMENT*
- **Trigger:**
  - *CheckInsertEmployeeWithSpecialty*: make sure if the employee has a specialty, the employee must have the year graduation and vise versa.

```sql
CREATE OR ALTER TRIGGER hospital.CheckInsertEmployeeWithSpecialty
ON hospital.EMPLOYEE
FOR INSERT, UPDATE AS
   DECLARE @Year_Graduate DATE;
   DECLARE @Special_Name VARCHAR(40);

   SELECT @Year_Graduate = Year_Graduate, @Special_Name = Special_Name
   FROM inserted;
   IF (@Year_Graduate IS NOT NULL AND @Special_Name IS NULL)
       OR (@Year_Graduate IS NULL AND @Special_Name IS NOT NULL)
   BEGIN
       PRINT('The specialty of employee has error');
       ROLLBACK TRANSACTION;
       RETURN
   END;
```

- **Constraint:**
  - Gender must be Male or Female
  - Job type must be Doctor or Nurse
  - Age of employees must be greater or equal than 18.
- **Data types** :
  - **INTEGER** for “Employee_ID"and “Dep_ID” like for “Drug_Code” in *Medication* above.
  - **VARCHAR(7)** for “First_Name", **VARCHAR(35)** for “Last_Name”, **VARCHAR(50)** for “Address” and **VARCHAR(40)** for “Special_Name” because it is suitable for string fields.
  - **DATE** for “Date_Of_Birth”, “Start_Date” and “Year_Graduate” fields since it is suitable for date fields.

- **CHAR(1)** is used for both "Job_Type" and "Gender" since these ones just only need to show between two options, d - doctor and n - nurse for job, m - male and f - female for gender.

3/ DEPARTMENT

- **Create**

```
CREATE TABLE hospital.DEPARTMENT
(
   Dep_ID INTEGER IDENTITY(1,1),
   Title VARCHAR(20) NOT NULL,
   Dean_ID INTEGER NOT NULL,
   CONSTRAINT Empty_PK_DEPARTMENT
       PRIMARY KEY (Dep_ID),
   CONSTRAINT Empty_FK_Dean_ID_DEPARTMENT
       FOREIGN KEY (Dean_ID) REFERENCES hospital.EMPLOYEE(Employee_ID)
);
```

- **Primary Key: *Dep_ID***
- **Foreign Key: *Dean_ID*** reference to attribute ***Employee_ID*** in table *hospital.EMPLOYEE*
- **Trigger:**
  - *CheckDeanRequirements*: check conditions for a dean. They are not being dean of other departments, having more than 5 years experience, having a specialty and must be a doctor.

```
CREATE OR ALTER TRIGGER hospital.CheckDeanRequirements
ON hospital.DEPARTMENT
FOR INSERT, UPDATE AS
   DECLARE @Year_Graduate DATE;
   DECLARE @Special_Name VARCHAR(40);
   DECLARE @Job_Type CHAR(1);
   DECLARE @Year_Of_Experience INT;
   DECLARE @Inserted_Dean_ID INT;
   DECLARE @Inserted_Dept_ID INT;

   SELECT  @Inserted_Dean_ID = Dean_ID,
           @Inserted_Dept_ID = Dep_ID
   FROM inserted;
```

```
   IF EXISTS(SELECT * FROM hospital.DEPARTMENT WHERE Dean_ID = @Inserted_Dean_ID
AND Dep_ID != @Inserted_Dept_ID)
   BEGIN
       PRINT 'This dean is already a dean of another department'
       ROLLBACK TRANSACTION;
       RETURN
   END

   SELECT  @Year_Graduate = E.Year_Graduate,
           @Special_Name = E.Special_Name,
           @Job_Type = E.Job_Type
   FROM hospital.EMPLOYEE AS E
   WHERE E.Employee_ID = @Inserted_Dean_ID;

   SET @Year_Of_Experience = DATEDIFF(year,@Year_Graduate,GETDATE());

   IF @Year_Of_Experience <= 5
       OR @Special_Name IS NULL
       OR @Job_Type = 'n'
   BEGIN
       PRINT('The deans requirement error');
       ROLLBACK TRANSACTION;
       RETURN
   END;
```

- **Constraint:** None
- **Data types** :
  - **INTEGER** is used for “Dean_ID” and “Dep_ID” like in *Employee*.
  - **VARCHAR(20)** for “Title" because it is suitable for string fields and is long enough to indicate the name of the department.

4/ EMPLOYEE_PHONE

- **Create**

```
CREATE TABLE hospital.EMPLOYEE_PHONE
(
```

```
    Employee_ID INT NOT NULL,
    Phone VARCHAR(11) NOT NULL,
    CONSTRAINT Empty_PK_EMPLOYEE_PHONE
        PRIMARY KEY (Employee_ID, Phone),
    CONSTRAINT Empty_FK_Employee_ID_EMPLOYEE_PHONE
        FOREIGN KEY (Employee_ID) REFERENCES hospital.EMPLOYEE(Employee_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CHECK(LEN(Phone) IN (10,11))
);
```

- **Primary Key:** composite key from ***Employee_ID*** and ***Phone***
- **Foreign Key:**
  - ***Employee_ID*** reference to attribute ***Employee_ID*** in table *hospital.EMPLOYEE*
    - *ON DELETE CASCADE*
    - *ON UPDATE CASCADE*
- **Data type:**
  - **VARCHAR(11)** is used for Phone numbers with a maximum number of 10 or 11.
- **Trigger:** None
- **Constraint:**
  - Length of phone number is 10 or 11
- **Data types** :
  - **INTEGER** for "Employee_ID" like in *Employee*.
  - **VARCHAR(11)** is used for "Phone" because the length of phone number is 10 or 11, therefore 11 is used as the maximum for length.

5/ INPATIENT

- **Create**

```
CREATE TABLE hospital.INPATIENT
(
    Patient_ID CHAR(7) NOT NULL,
    First_Name VARCHAR(7) NOT NULL,
    Last_Name VARCHAR(35) NOT NULL,
    Date_Of_Birth DATE NOT NULL,
```

```sql
    Gender CHAR(1) NOT NULL,
    Address VARCHAR(50) NOT NULL,
    Phone VARCHAR(11) NOT NULL,
    CONSTRAINT Empty_PK_INPATIENT
        PRIMARY KEY (Patient_ID),
    CHECK(Gender IN ('m','f')),
    CHECK(LEN(Phone) IN (10,11)),
    CHECK(SUBSTRING(Patient_ID, 1,2) = 'IP')
);
```

- **Primary Key:** ***Patient_ID***
- **Foreign Key:** *None*
- **Data type:**
  - **CHAR(7)** is used for Patient_ID as the first 2 characters of patient is 'IP' so we have to choose CHAR to store. Then the reset 5 characters we just use [hospital].[InpatientSeq] to count up and CONCAT with 'IP'.
  - **VARCHAR(7)** is used for First Name as the maximum word in VietNamese does not exceed 6 characters. Moreover, we use varchar as it will fit to the word that we insert.
  - **VARCHAR(35)** is used for last name as we allow maximum 5 words in last name, each word has 7 characters. Then we have 35 characters
  - **CHAR(1)** is used to store gender of patients. '1': Male, '0': Femal.
  - **VARCHAR(11)** is used for Phone numbers with a maximum number of 10 or 11.
  - **DATE** is used to represent the date_of_birth fields since it is suitable for date fields.
  - **VARCHAR(50)** for "Address" because it is suitable for string fields.
- **Trigger:** *None*
- **Constraint:**
  - Length of phone number is 10 or 11
  - Gender must be Male or Female
  - First 2 character of Patient_ID is 'IP'

6/ ADMISSION

- **Create**

```sql
CREATE TABLE hospital.ADMISSION
(
   Admission_ID INTEGER NOT NULL,
   Patient_ID CHAR(7) NOT NULL,
   Nurse_ID INTEGER NOT NULL DEFAULT 0,
   Admission_Date DATE NOT NULL,
   Discharge_Date DATE,
   Sickroom CHAR(4) NOT NULL,
   Diagnosis VARCHAR(50) NOT NULL,
   Fee NUMERIC(10,1) NOT NULL DEFAULT 0,
   CONSTRAINT Empty_PK_ADMISSION
       PRIMARY KEY (Admission_ID),
   CONSTRAINT Empty_FK_Patient_ID_ADMISSION
       FOREIGN KEY (Patient_ID) REFERENCES hospital.INPATIENT(Patient_ID),
   CONSTRAINT Empty_FK_Nurse_ID_ADMISSION
       FOREIGN KEY (Nurse_ID) REFERENCES hospital.EMPLOYEE(Employee_ID)
       ON DELETE SET DEFAULT
       ON UPDATE CASCADE
);
```

- **Primary Key:** ***Admission_ID***
- **Foreign Key:**
  - ***Patient_ID*** reference to attribute ***Patient_ID*** in table *hospital.INPATIENT*
  - ***Nurse_ID*** reference to attribute ***Employee_ID*** in table *hospital.EMPLOYEE*
    - *ON DELETE SET DEFAULT*
    - *ON UPDATE CASCADE*
- **Data type:**
  - **INTEGER** is used for ***Admission_ID*** instead of *smallint* since *smallint* limits the uniqueness of ***Admission_ID*** to about 32 000 while INTEGER up to 2,147,483,647. Therefore, **INTEGER** is suitable to make sure every employee gets a unique code
  - **CHAR(4)** is used for sickroom such as 'A101'

- **DATE** is used to represent the "Discharge_Date", "Admission_Date" fields since it is suitable for date fields.
- **VARCHAR(50)** for "Diagnosis" because it is suitable for string fields.
- **NUMERIC(10,1)** is used for the price of a Admission because it is suitable to limit the price below 10,000,000,000 and keep one after the decimal point in Vietnamese currency (VND).

- **Trigger:**
  - *CheckAdmission:* make sure that the patient will not admiss if he/she has already admissed, and the nurse must be present.

```sql
CREATE OR ALTER TRIGGER hospital.CheckAdmission
ON hospital.ADMISSION
FOR INSERT AS
   DECLARE @is_outadmitted INT;
   DECLARE @inserted_Patient_ID CHAR(7);
   DECLARE @inserted_Nurse_ID INT;
   DECLARE @inserted_admisison INT;
   DECLARE @Job_Type CHAR(1);

   SELECT  @inserted_Nurse_ID = Nurse_ID,
           @inserted_Patient_ID = Patient_ID,
           @inserted_admisison = Admission_ID
   FROM inserted;

   SELECT @Job_Type = Job_Type
   FROM hospital.EMPLOYEE AS E
   WHERE @inserted_Nurse_ID = E.Employee_ID;

   SELECT @is_outadmitted = COUNT(*)
   FROM ADMISSION AS A
   WHERE   A.Patient_ID = @inserted_Patient_ID
       AND A.Admission_ID != @inserted_admisison
       AND Discharge_Date IS NULL;

   IF @is_outadmitted != 0
   BEGIN
       PRINT('The patient is already admitted');
```

```
        ROLLBACK TRANSACTION;
        RETURN
    END
    ELSE IF @Job_Type != 'n'
    BEGIN
        PRINT('There is no nurse');
        ROLLBACK TRANSACTION;
        RETURN
    END;
```

- **Constraint:** None

7/ TREATMENT

- **Create**

```
CREATE TABLE hospital.TREATMENT(
    Admission_ID INTEGER NOT NULL,
    Treatment_ID INTEGER NOT NULL,
    START_DATE DATE NOT NULL,
    END_DATE DATE NOT NULL,
    Result VARCHAR(50) NOT NULL,
    CONSTRAINT Empty_PK_TREATMENT
        PRIMARY KEY(Admission_ID, Treatment_ID),
    CONSTRAINT Empty_FK_Admission_ID_TREATMENT
        FOREIGN KEY (Admission_ID) REFERENCES hospital.ADMISSION(Admission_ID)
);
```

- **Primary Key:** *composite key of* ***Admission_ID*** *and* ***Treatment_ID***
  - *Treatment_ID:* auto-set by database
- **Foreign Key:** ***Admission_ID*** reference to attribute ***Admission_ID*** in table *hospital.ADMISSION*
- **Data type:**
  - **INTEGER** is used for ***Treatment_ID*** instead of *smallint* since *smallint* limits the uniqueness of ***Treatment_ID*** to about 32 000 while INTEGER up to 2,147,483,647. Therefore, **INTEGER** is suitable to make sure every employee gets a unique code

- **DATE** is used to represent the "START_DATE", "END_DATE" fields since it is suitable for date fields.
- **VARCHAR(50)** for "Result" because it is suitable for string fields.

- **Trigger:** None
- **Constraint:** None

8/ TREATMENT_DOCTOR

- **Create**

```
CREATE TABLE hospital.TREATMENT_DOCTOR
(
   Admission_ID INTEGER NOT NULL,
   Treatment_ID INTEGER NOT NULL,
   Doctor_ID INTEGER NOT NULL DEFAULT -1,
   CONSTRAINT Empty_PK_TREATMENT_DOCTOR
       PRIMARY KEY(Admission_ID, Treatment_ID, Doctor_ID),
   CONSTRAINT Empty_FK_Admission_ID_Treatment_ID_TREATMENT_DOCTOR
       FOREIGN KEY(Admission_ID,Treatment_ID) REFERENCES
       hospital.TREATMENT(Admission_ID,Treatment_ID),
   CONSTRAINT Empty_FK_Doctor_ID_TREATMENT_DOCTOR
       FOREIGN KEY(Doctor_ID) REFERENCES hospital.EMPLOYEE(Employee_ID)
       ON DELETE SET DEFAULT
       ON UPDATE CASCADE
);
```

- **Primary Key:** composite key of ***Admission_ID***, ***Treatment_ID*** *and* ***Doctor_ID***
- **Foreign Key:**
  - Composite foreign key ***Admission_ID***, ***Treatment_ID*** reference to attributes ***Admission_ID***, ***Treatment_ID*** in table *hospital.TREATMENT*
  - ***Doctor_ID*** reference to attribute ***Employee_ID*** in table *hospital.EMPLOYEE*
    - ON DELETE SET DEFAULT
    - ON UPDATE CASCADE
- **Trigger:**

- *CheckTreatmentDoctor*: make sure the treatment person is a doctor.

```sql
CREATE OR ALTER TRIGGER hospital.CheckTreatmentDoctor
ON hospital.TREATMENT_DOCTOR
FOR INSERT, UPDATE AS
   DECLARE @doctor_id INT;
   DECLARE @Job_Type CHAR(1);

   SELECT  @doctor_id = Doctor_ID
   FROM inserted;

   SELECT @Job_Type = Job_Type
   FROM hospital.EMPLOYEE AS E
   WHERE @doctor_id = E.Employee_ID;

   IF @Job_Type != 'd'
   BEGIN
       PRINT('There is no doctor');
       ROLLBACK TRANSACTION;
       RETURN
   END;
```

- **Constraint:** None

9/ TREATMENT_MEDICATION

- **Create**

```sql
CREATE TABLE hospital.TREATMENT_MEDICATION
(
   Admission_ID INTEGER NOT NULL,
   Treatment_ID INTEGER NOT NULL,
   Drug_Code INTEGER NOT NULL DEFAULT 0,
   Amount INT NOT NULL,
   CONSTRAINT Empty_PK_TREATMENT_MEDICATION
       PRIMARY KEY(Admission_ID,Treatment_ID, Drug_Code),
   CONSTRAINT Empty_FK_Admission_ID_Treatment_ID_TREATMENT_MEDICATION
       FOREIGN KEY(Admission_ID,Treatment_ID) REFERENCES
       hospital.TREATMENT(Admission_ID,Treatment_ID),
```

```sql
   CONSTRAINT Empty_FK_Drug_Code_TREATMENT_MEDICATION
       FOREIGN KEY(Drug_Code) REFERENCES hospital.MEDICATION(Drug_Code)
       ON DELETE SET DEFAULT
       ON UPDATE CASCADE
);
```

- **Primary Key:** composite key of ***Admission_ID***, ***Treatment_ID*** and ***Drug_Code***
- **Foreign Key:**
  - Composite foreign key ***Admission_ID***, ***Treatment_ID*** reference to attributes ***Admission_ID***, ***Treatment_ID*** in table *hospital.TREATMENT*
  - ***Drug_Code*** reference to attribute ***Drug_Code*** in table *hospital.MEDICATION*
    - ON DELETE SET DEFAULT
    - ON UPDATE CASCADE
- **Trigger:** None
- **Constraint:** None

10/ OUTPATIENT

- **Create**

```sql
CREATE TABLE hospital.OUTPATIENT
(
   Patient_ID CHAR(7) NOT NULL,
   First_Name VARCHAR(7) NOT NULL,
   Last_Name VARCHAR(35) NOT NULL,
   Phone VARCHAR(11) NOT NULL,
   Address VARCHAR(50) NOT NULL,
   Gender CHAR(1) NOT NULL,
   Date_Of_Birth DATE NOT NULL,
   CONSTRAINT Empty_PK_OUTPATIENT
       PRIMARY KEY (Patient_ID),
   CHECK(Gender IN ('m','f')),
   CHECK(LEN(Phone) IN (10,11)),
   CHECK(SUBSTRING(Patient_ID, 1,2) = 'OP')
);
```

- **Primary Key:** ***Patient_ID***
- **Foreign Key:** None
- **Data type:**
  - **CHAR(7)** is used for Patient_ID as the first 2 characters of patient is 'OP' so we have to choose CHAR to store. Then the reset 5 characters we just use [hospital].[OutpatientSeq] to count up and CONCAT with 'OP'.
  - **VARCHAR(7)** is used for First Name as the maximum word in VietNamese does not exceed 6 characters. Moreover, we use varchar as it will fit to the word that we insert.
  - **VARCHAR(35)** is used for last name as we allow maximum 5 words in last name, each word has 7 characters. Then we have 35 characters
  - **CHAR(1)** is used to store gender of patients. 'm': Male, 'f': Femal.
  - **VARCHAR(11)** is used for Phone numbers with a maximum number of 10 or 11.
  - **DATE** is used to represent the date_of_birth fields since it is suitable for date fields.
  - **VARCHAR(50)** for "Address" because it is suitable for string fields.
- **Trigger:** None
- **Constraint:**
  - Length of phone number is 10 or 11
  - Gender must be Male or Female
  - First 2 character of Patient_ID is 'IP'

11/ EXAMINATION

- **Create**

```
CREATE TABLE hospital.EXAMINATION
(
   Patient_ID CHAR(7) NOT NULL,
   Exam_ID INTEGER NOT NULL,
   Doctor_Exam_ID INTEGER NOT NULL DEFAULT -1,
   Exam_Date DATE NOT NULL,
```

```
    Second_Exam_Date DATE,
    Diagnosis VARCHAR(50) NOT NULL,
    Fee NUMERIC(10,1) NOT NULL,
    CONSTRAINT Empty_PK_EXAMINATION
        PRIMARY KEY(Patient_ID, Exam_ID),
    CONSTRAINT Empty_FK_Patient_ID_EXAMINATION
        FOREIGN KEY (Patient_ID) REFERENCES hospital.OUTPATIENT(Patient_ID),
    CONSTRAINT Empty_FK_Doctor_Exam_ID_EXAMINATION
        FOREIGN KEY (Doctor_Exam_ID) REFERENCES hospital.EMPLOYEE(Employee_ID)
        ON DELETE SET DEFAULT
        ON UPDATE CASCADE
);
```

- **Primary Key:** composite key of ***Patient_ID*** and ***Exam_ID***
- **Foreign Key:**
  - ***Patient_ID*** reference to attribute ***Patient_ID*** in table *hospital.OUTPATIENT*
  - ***Doctor_Exam_ID*** reference to attribute ***Employee_ID*** in table *hospital.EMPLOYEE*
    - ON DELETE SET DEFAULT
    - ON UPDATE CASCADE
- **Data type:**
  - **INTEGER** is used for ***Exam_ID*** instead of *smallint* since *smallint* limits the uniqueness of ***Exam_ID*** to about 32 000 while INTEGER up to 2,147,483,647. Therefore, **INTEGER** is suitable to make sure every employee gets a unique code
  - **CHAR(4)** is used for sickroom such as 'A101'
  - **DATE** is used to represent the "Exam_Date", "Second_Exam_Date" fields since it is suitable for date fields.
  - **VARCHAR(50)** for "Diagnosis" because it is suitable for string fields.
  - **NUMERIC(10,1)** is used for the price of a Examination because it is suitable to limit the price below 10,000,000,000 and keep one after the decimal point in Vietnamese currency (VND).
- **Trigger:**
  - *CheckExamination*: make sure the examiner is doctor

```sql
CREATE OR ALTER TRIGGER hospital.CheckEXAMINATION
ON hospital.EXAMINATION
FOR INSERT AS
   DECLARE @doctor_id INT;
   DECLARE @Job_Type INT;

   SELECT  @doctor_id = Doctor_Exam_ID
   FROM inserted;

   SELECT @Job_Type = Job_Type
   FROM hospital.EMPLOYEE AS E
   WHERE @doctor_id = E.Employee_ID;

   IF @Job_Type != 'd'
   BEGIN
       PRINT('There is no doctor');
       ROLLBACK TRANSACTION;
       RETURN
   END;
```

- **Constraint:** None

12/ EXAMINATION_MEDICATION

- **Create**

```sql
CREATE TABLE hospital.EXAMINATION_MEDICATION
(
   Patient_ID CHAR(7) NOT NULL,
   Exam_ID INTEGER NOT NULL,
   Drug_Code INTEGER NOT NULL DEFAULT 0,
   Amount INT NOT NULL,
   CONSTRAINT Empty_PK_EXAMINATION_MEDICATION
       PRIMARY KEY(Patient_ID,Exam_ID, Drug_Code),
   CONSTRAINT Empty_FK_Patient_ID_Exam_ID_EXAMINATION_MEDICATION
       FOREIGN KEY(Patient_ID,Exam_ID) REFERENCES
hospital.EXAMINATION(Patient_ID,Exam_ID),
   CONSTRAINT Empty_FK_Drug_Code_EXAMINATION_MEDICATION
       FOREIGN KEY(Drug_Code) REFERENCES hospital.MEDICATION(Drug_Code)
```

```
        ON DELETE SET DEFAULT
        ON UPDATE CASCADE
);
```

- **Primary Key:** Composite key of ***Patient_ID***, ***Exam_ID*** and ***Drug_Code***
- **Foreign Key:**
  - Composite foreign key of ***Patient_ID***, ***Exam_ID*** reference to attributes ***Patient_ID***, ***Exam_ID*** in table *hospital.EXAMINATION*
  - ***Drug_Code*** reference to attribute ***Drug_Code*** in table *hospital.MEDICATION*
    - *ON DELETE SET DEFAULT*
    - *ON UPDATE CASCADE*
- **Data type:**
  - CHAR(7) is used because it is a suitable string data type for `B_Name`, `B_No`, … and it takes the storage corresponding number to its length, e.g 50 is enough to represent the name.
  - *int* is used for `B_NumOfEmployee` instead of *smallint* because *smallint* only allows the number up to about 32,000.
- **Trigger:** None
- **Constraint:** None

B. Stored Procedure

a. Store procedure to add new department with new dean

```
CREATE OR ALTER PROCEDURE hospital.Create_New_Department_New_Dean
   @Department_Title VARCHAR(20),
   @Dean_First_Name VARCHAR(7),
   @Dean_Last_Name VARCHAR(35),
   @Dean_Address VARCHAR(50),
   @Dean_Start_Date DATE,
   @Dean_Date_Of_Birth DATE,
   @Dean_Gender CHAR(1),
   @Dean_Job_Type CHAR(1),
   @Dean_Special_Name VARCHAR(40),
   @Dean_Year_Graduate DATE,
```

```sql
    @Phone VARCHAR(11)
AS
    DECLARE @TransactionName VARCHAR(20) = 'CreateNewDepartmentNewDean';
    BEGIN TRANSACTION @TransactionName;
        BEGIN TRY
            DECLARE @department_id INT;
            DECLARE @dean_id INT;

            -- Create new employee with default department_id = 0
            INSERT INTO hospital.EMPLOYEE(
                First_Name, Last_Name, Date_Of_Birth,
                Gender, Address, Start_Date, Job_Type,
                Dep_ID, Special_Name, Year_Graduate
            )
            VALUES (
                @Dean_First_Name, @Dean_Last_Name, @Dean_Date_Of_Birth,
                @Dean_Gender, @Dean_Address, @Dean_Start_Date, @Dean_Job_Type,
                0, @Dean_Special_Name, @Dean_Year_Graduate
            );
            -- Get id of previous inserted employee
            SELECT @dean_id = MAX(Employee_ID)
            FROM hospital.EMPLOYEE;

            -- Insert data to EMPLOYEE_PHONE table
            INSERT INTO hospital.EMPLOYEE_PHONE(Employee_ID, Phone)
            VALUES(@dean_id, @Phone);

            -- Create new department with dean is @dean_id
            INSERT INTO hospital.DEPARTMENT(Title, Dean_ID)
            VALUES(@Department_Title, @dean_id);

            -- Get id of previous inserted department
            SELECT @department_id = MAX(Dep_ID)
            FROM hospital.DEPARTMENT;

            -- Update department id for dean with department id = @department_id
            UPDATE hospital.EMPLOYEE
            SET Dep_ID = @department_id
            WHERE Employee_ID = @dean_id;
```

```
            COMMIT TRANSACTION @TransactionName;
        END TRY
        BEGIN CATCH
            PRINT 'Error occurs when create new dep with new dean'
            ROLLBACK TRANSACTION @TransactionName;
        END CATCH
```

b. Store procedure to create new department with exist employee

```
CREATE OR ALTER PROCEDURE hospital.Create_New_Department_Exist_Dean
   @Dean_ID INT,
   @Department_Title VARCHAR(20)
AS
   DECLARE @department_id INT;
    -- Create new department
   INSERT INTO hospital.DEPARTMENT(Title, Dean_ID)
   VALUES(@Department_Title, @Dean_ID);

   -- Get id of previous inserted department
   SELECT @department_id = MAX(Dep_ID)
   FROM hospital.DEPARTMENT;

   -- Update department id for new dean
   UPDATE hospital.EMPLOYEE
   SET Dep_ID = @department_id
   WHERE Employee_ID = @Dean_ID;
```

c. Store procedure to change dean of department

```
CREATE OR ALTER PROCEDURE hospital.Change_Dean_Department
   @Replace_Dean_ID INTEGER,
   @Dep_ID INTEGER
AS
   DECLARE @TransactionName VARCHAR(20) = 'ChangeDeanDepartment';
   BEGIN TRANSACTION @TransactionName;
       BEGIN TRY
           UPDATE hospital.DEPARTMENT
           SET Dean_ID = @Replace_Dean_ID
```

```
            WHERE Dep_ID = @Dep_ID;
            COMMIT TRANSACTION @TransactionName;
        END TRY
        BEGIN CATCH
            PRINT 'Error occurs when change dean for deaprtment'
            ROLLBACK TRANSACTION @TransactionName;
        END CATCH
```

d. Store procedure to add new employee

```
CREATE OR ALTER PROCEDURE hospital.Add_Employee
    @Dep_ID INTEGER,
    @First_Name VARCHAR(7),
    @Last_Name VARCHAR(35),
    @Date_Of_Birth DATE,
    @Gender CHAR(1),
    @Address VARCHAR(50),
    @Start_Date DATE,
    @Job_Type CHAR(1),
    @Special_Name VARCHAR(40),
    @Year_Graduate DATE,
    @Phone VARCHAR(11)
AS
    DECLARE @Employee_ID INT;
    DECLARE @TransactionName VARCHAR(20) = 'AddEmployee';
    BEGIN TRANSACTION @TransactionName;
        BEGIN TRY
            INSERT INTO hospital.EMPLOYEE(
                Dep_ID, First_Name, Last_Name, Date_Of_Birth, Gender, Address,
                Start_Date, Job_Type, Special_Name, Year_Graduate
            )
            VALUES(
                @Dep_ID,@First_Name,@Last_Name,@Date_Of_Birth,@Gender,@Address,
                @Start_Date,@Job_Type, @Special_Name, @Year_Graduate
            );

            SELECT @Employee_id = MAX(Employee_ID)
            FROM hospital.EMPLOYEE;
```

```
        INSERT INTO hospital.EMPLOYEE_PHONE(Employee_ID, Phone)
        VALUES(@Employee_id, @Phone);
        COMMIT TRANSACTION @TransactionName;
    END TRY
    BEGIN CATCH
        PRINT 'Error occurs when create employee'
        ROLLBACK TRANSACTION @TransactionName;
    END CATCH
```

e. Store procedure to add new inpatient

```
CREATE OR ALTER PROCEDURE hospital.NewInpatient
   @First_Name VARCHAR(7),
   @Last_Name VARCHAR(35),
   @Date_Of_Birth DATE,
   @Gender CHAR(1),
   @Address VARCHAR(50),
   @Phone VARCHAR(11),
   @Nurse_ID INTEGER,
   @Admission_Date DATE,
   @Sickroom CHAR(4),
   @Diagnosis VARCHAR(30),
   @Fee NUMERIC(10,1),
   @Doctor_ID INTEGER
AS
   DECLARE @TransactionName VARCHAR(20) = 'AddNewInpatient';
   DECLARE @patient_id CHAR(7);
   DECLARE @admission_id INTEGER;
   BEGIN TRANSACTION @TransactionName;
       BEGIN TRY

       SELECT @patient_id = CONCAT('IP',format(NEXT VALUE FOR
                            hospital.InpatientSeq, '00000'));
       SELECT @admission_id = NEXT VALUE FOR hospital.AdmissionSeq;

       INSERT INTO hospital.INPATIENT(
           Patient_ID, First_Name, Last_Name, Date_Of_Birth, Gender,
```

```
            Address, Phone
        )
        VALUES(
            @patient_id, @First_Name, @Last_Name, @Date_Of_Birth, @Gender,
            @Address, @Phone
        );

        INSERT INTO hospital.ADMISSION(
            Admission_ID, Patient_ID, Nurse_ID, Admission_Date, Discharge_Date,
            Sickroom, Diagnosis, Fee
        )
        VALUES(
            @admission_id, @patient_id, @Nurse_ID, @Admission_Date, NULL,
            @Sickroom, @Diagnosis, @Fee
        );

        INSERT INTO hospital.TREATMENT(Admission_ID, Treatment_ID, START_DATE,
            END_DATE, Result)
        VALUES(@admission_id, 1, @Admission_Date, @Admission_Date, 'nhap vien');

        INSERT INTO hospital.TREATMENT_DOCTOR(Admission_ID, Treatment_ID, Doctor_ID)
        VALUES(@admission_id, 1, @Doctor_ID)

        COMMIT TRANSACTION @TransactionName;
        END TRY
        BEGIN CATCH
        ROLLBACK TRANSACTION @TransactionName;
        END CATCH
```

f. Store procedure to add new admission

```
CREATE OR ALTER PROCEDURE hospital.NewAdmission
   @patient_id CHAR(7),
   @Nurse_ID INTEGER,
   @Admission_Date DATE,
   @Sickroom CHAR(4),
   @Diagnosis VARCHAR(30),
   @Fee NUMERIC(10,1),
```

```
    @Doctor_ID INTEGER
AS
    DECLARE @TransactionName VARCHAR(20) = 'AddAdmission';
    DECLARE @admission_id INTEGER;
    BEGIN TRANSACTION @TransactionName;
        BEGIN TRY

        SELECT @admission_id = NEXT VALUE FOR hospital.AdmissionSeq;

        INSERT INTO hospital.ADMISSION(
            Admission_ID, Patient_ID, Nurse_ID, Admission_Date, Discharge_Date,
            Sickroom, Diagnosis, Fee)
        VALUES(
            @admission_id, @patient_id, @Nurse_ID, @Admission_Date, NULL,
            @Sickroom, @Diagnosis, @Fee);

        INSERT INTO hospital.TREATMENT(
            Admission_ID, Treatment_ID, START_DATE, END_DATE, Result)
        VALUES(
            @admission_id, 1, @Admission_Date, @Admission_Date, 'nhap vien');

        INSERT INTO hospital.TREATMENT_DOCTOR(Admission_ID, Treatment_ID, Doctor_ID)
        VALUES(@admission_id, 1, @Doctor_ID)

        COMMIT TRANSACTION @TransactionName;
        END TRY
        BEGIN CATCH
        ROLLBACK TRANSACTION @TransactionName;
        END CATCH
```

g. Store procedure to add new treatment

```
CREATE OR ALTER PROCEDURE hospital.NewTreatment
    @Admission_ID INTEGER,
    @START_DATE DATE,
    @END_DATE DATE,
    @Result VARCHAR(30),
    @Doctor_ID INTEGER
```

```sql
AS
   DECLARE @TransactionName VARCHAR(20) = 'NewTreatment';
   DECLARE @treatment_id INTEGER;
   BEGIN TRANSACTION @TransactionName;
       BEGIN TRY

       SELECT @treatment_id = MAX(Treatment_ID) + 1
       FROM hospital.TREATMENT
       WHERE Admission_ID = @Admission_ID;

       INSERT INTO hospital.TREATMENT(
           Admission_ID, Treatment_ID, START_DATE, END_DATE, Result)
       VALUES(
           @Admission_ID, @treatment_id, @START_DATE, @END_DATE, @Result);

       INSERT INTO hospital.TREATMENT_DOCTOR(Admission_ID, Treatment_ID, Doctor_ID)
       VALUES(@Admission_ID, @treatment_id, @Doctor_ID);

       COMMIT TRANSACTION @TransactionName;
       END TRY
       BEGIN CATCH
       ROLLBACK TRANSACTION @TransactionName;
       END CATCH
```

h. Store procedure to add new treatment medication

```sql
CREATE OR ALTER PROCEDURE hospital.NewTreatmentMedication
   @Admission_ID INTEGER,
   @Treatment_ID INTEGER,
   @Drug_Code INTEGER,
   @Amount INTEGER
AS
   DECLARE @TransactionName VARCHAR(20) = 'NewTreatmentMedication';
   BEGIN TRANSACTION @TransactionName;
       BEGIN TRY

       INSERT INTO hospital.TREATMENT_MEDICATION(
           Admission_ID, Treatment_ID,Drug_Code, Amount)
```

```
        VALUES(
            @Admission_ID, @Treatment_ID, @Drug_Code, @Amount);

        COMMIT TRANSACTION @TransactionName;
        END TRY
        BEGIN CATCH
        ROLLBACK TRANSACTION @TransactionName;
        END CATCH
```

i. Store procedure to outadmiss inpatient

```
CREATE OR ALTER PROCEDURE hospital.OutAdmission
   @Patient_ID CHAR(7),
   @Discharge_Date DATE
AS
   DECLARE @TransactionName VARCHAR(20) = 'OutAdmission';
   BEGIN TRANSACTION @TransactionName;
       BEGIN TRY

       UPDATE
           hospital.ADMISSION
       SET
           Discharge_Date = @Discharge_Date
       WHERE
           Patient_ID = @Patient_ID
           AND Discharge_Date IS NULL;

       COMMIT TRANSACTION @TransactionName;
       END TRY
       BEGIN CATCH
       ROLLBACK TRANSACTION @TransactionName;
       END CATCH
```

j. Store procedure to add new outpatient

```
CREATE OR ALTER PROCEDURE hospital.NewOutPatient
   @First_Name VARCHAR(7),
```

```sql
    @Last_Name VARCHAR(35),
    @Phone VARCHAR(11),
    @Address VARCHAR(50),
    @Gender CHAR(1),
    @Date_Of_Birth DATE,
    @Doctor_Exam_ID INTEGER,
    @Exam_Date DATE,
    @Second_Exam_Date DATE,
    @Diagnosis VARCHAR(30),
    @Fee NUMERIC(10,1),
    @Drug_Code INTEGER
AS
    DECLARE @TransactionName VARCHAR(20) = 'Create new outpatient';
    DECLARE @patient_id CHAR(7);
    BEGIN TRANSACTION @TransactionName;
        BEGIN TRY

        SELECT @patient_id = CONCAT('OP',format(NEXT VALUE FOR
                                hospital.InpatientSeq, '00000'));

        INSERT INTO hospital.OUTPATIENT(
            Patient_id, First_Name, Last_Name, Phone, Address, Gender,
            Date_Of_Birth)
        VALUES(
            @patient_id, @First_Name, @Last_Name, @Phone, @Address, @Gender,
            @Date_Of_Birth);

        INSERT INTO hospital.EXAMINATION(
            Patient_ID, Exam_ID, Doctor_Exam_ID, Exam_Date, Second_Exam_Date,
            Diagnosis, Fee)
        VALUES(
            @patient_id, 1, @Doctor_Exam_ID, @Exam_Date, @Second_Exam_Date,
            @Diagnosis, @Fee);

        INSERT into hospital.EXAMINATION_MEDICATION(Patient_ID, Exam_ID, Drug_Code)
        VALUES(@patient_id, 1, @Drug_Code)

        COMMIT TRANSACTION @TransactionName;
        END TRY
```

```
        BEGIN CATCH
        ROLLBACK TRANSACTION @TransactionName;
        END CATCH;
```

k. Store procedure to add new examination

```
CREATE OR ALTER PROCEDURE hospital.NewExamination
   @Patient_ID CHAR(7),
   @Doctor_Exam_ID INTEGER,
   @Exam_Date DATE,
   @Second_Exam_Date DATE,
   @Diagnosis VARCHAR(30),
   @Fee NUMERIC(10,1)
AS
   DECLARE @TransactionName VARCHAR(20) = 'Create new examination';
   DECLARE @examination_id INTEGER;
   BEGIN TRANSACTION @TransactionName;
       BEGIN TRY

       SELECT @examination_id =  MAX(Exam_ID) + 1
       FROM hospital.EXAMINATION
       WHERE Patient_ID = @Patient_ID;

       INSERT INTO hospital.EXAMINATION(
           Patient_ID, Exam_ID, Doctor_Exam_ID, Exam_Date,
           Second_Exam_Date, Diagnosis, Fee)
       VALUES(
           @Patient_ID, @examination_id, @Doctor_Exam_ID, @Exam_Date,
           @Second_Exam_Date, @Diagnosis, @Fee);

       COMMIT TRANSACTION @TransactionName;
       END TRY
       BEGIN CATCH
       ROLLBACK TRANSACTION @TransactionName;
       END CATCH;
```

l. Store procedure to add new examination medication

```sql
CREATE OR ALTER PROCEDURE hospital.NewTExaminationMedication
   @Patient_ID CHAR(7),
   @Exam_ID INTEGER,
   @Drug_Code INTEGER,
   @Amount INTEGER
AS
   DECLARE @TransactionName VARCHAR(20) = 'NewTExaminationMedication';
   BEGIN TRANSACTION @TransactionName;
       BEGIN TRY

       INSERT INTO hospital.EXAMINATION_MEDICATION(
           Patient_ID, Exam_ID,Drug_Code, Amount)
       VALUES(
           @Patient_ID, @Exam_ID, @Drug_Code, @Amount);

       COMMIT TRANSACTION @TransactionName;
       END TRY
       BEGIN CATCH
       ROLLBACK TRANSACTION @TransactionName;
       END CATCH
```

- Store procedure to calculate and update fee of all examinations

```sql
CREATE OR ALTER PROCEDURE hospital.CalculateFeeExam
AS
   DECLARE @pre_patient_id CHAR(7);
   DECLARE @pre_exam_id INT;
   DECLARE @patient_id CHAR(7);
   DECLARE @exam_id INT;
   DECLARE @pirce NUMERIC(10,1);
   DECLARE @amount INT;
   DECLARE @total_fee NUMERIC(10,1);

   DECLARE fee_exam CURSOR FOR
       SELECT
           EM.Patient_ID,
           EM.Exam_ID,
```

```
        M.Price,
        EM.Amount
    FROM
        hospital.EXAMINATION_MEDICATION AS EM
    JOIN
        hospital.MEDICATION AS M
        ON M.Drug_Code = EM.Drug_Code;

OPEN fee_exam

FETCH NEXT FROM fee_exam
INTO @patient_id, @exam_id, @pirce, @amount;

SET @pre_exam_id = @exam_id;
SET @pre_patient_id = @patient_id;
SET @total_fee = 0;

WHILE @@FETCH_STATUS = 0
BEGIN

    IF @pre_exam_id != @exam_id OR @pre_patient_id != @patient_id
    BEGIN
        UPDATE
            hospital.EXAMINATION
        SET
            fee = @total_fee
        WHERE
            Patient_ID = @pre_patient_id
            AND Exam_ID = @pre_exam_id;

        SET @pre_exam_id = @exam_id;
        SET @pre_patient_id = @patient_id;
        SET @total_fee = 0;
    END

    SET @total_fee = @total_fee + @pirce*@amount

    FETCH NEXT FROM fee_exam
    INTO @patient_id, @exam_id, @pirce, @amount;
```

```
    END

    UPDATE
        hospital.EXAMINATION
    SET
        fee = @total_fee
    WHERE
        Patient_ID = @pre_patient_id
        AND Exam_ID = @pre_exam_id;

    CLOSE fee_exam;
    DEALLOCATE fee_exam;
    RETURN
```

p. Store procedure to calculate and update fee of all admissions

```
CREATE OR ALTER PROCEDURE hospital.CalculateFeeAdmission
AS
    DECLARE @pre_admission_id INT;
    DECLARE @admission_id INT;
    DECLARE @pirce NUMERIC(10,1);
    DECLARE @amount INT;
    DECLARE @total_fee NUMERIC(10,1);

    DECLARE fee_admission CURSOR FOR
        SELECT
            TM.Admission_ID, M.Price, TM.Amount
        FROM
            hospital.TREATMENT_MEDICATION AS TM
        JOIN
            hospital.MEDICATION AS M
            ON M.Drug_Code = TM.Drug_Code;

    OPEN fee_admission

    FETCH NEXT FROM fee_admission
    INTO @admission_id, @pirce, @amount;
```

```sql
SET @pre_admission_id = @admission_id;
SET @total_fee = 0;

WHILE @@FETCH_STATUS = 0
BEGIN
    if  @pre_admission_id != @admission_id
    BEGIN
        PRINT RTRIM(@pre_admission_id) + N' ' + RTRIM(@total_fee)
        UPDATE
            hospital.ADMISSION
        SET
            fee = @total_fee
        WHERE
            Admission_ID = @pre_admission_id;

        SET @pre_admission_id = @admission_id;
        SET @total_fee = 0;
    END

    SET @total_fee = @total_fee + @pirce*@amount

    FETCH NEXT FROM fee_admission
    INTO @admission_id, @pirce, @amount;
END

UPDATE
    hospital.ADMISSION
SET
    fee = @total_fee
WHERE
    Admission_ID = @pre_admission_id;

CLOSE fee_admission;
DEALLOCATE fee_admission;
RETURN
```

C. Views

a. View of current admitted inpatient in hospital

```
CREATE OR ALTER VIEW hospital.CURRENT_ADMITTED_INPATIENT
AS
   SELECT
       AD.Patient_ID, AD.Admission_ID
   FROM
       hospital.ADMISSION as AD
   INNER JOIN
   (
       SELECT
           A.Patient_ID, MAX(A.Admission_ID) AS Ad_ID
       FROM
           hospital.ADMISSION AS A
       WHERE
           A.Discharge_Date IS NULL
       GROUP BY
           A.Patient_ID
   ) AS TEMP
   ON AD.Admission_ID = TEMP.Ad_ID;
```

D. Sequences

a. Sequence for auto increasing Admission ID

```
CREATE SEQUENCE hospital.AdmissionSeq
   AS INT
   START WITH 1
   INCREMENT BY 1;
```

b. Sequence for auto increasing Inpatient ID

```
CREATE SEQUENCE InpatientSeq
   AS INT
   START WITH 1
   INCREMENT BY 1;
```

c. Sequence for auto increasing Outpatient ID

```
CREATE SEQUENCE hospital.OutpatientSeq
   AS INT
   START WITH 1
   INCREMENT BY 1;
```

3. Stored Procedure / Function / SQL

This section concerns the stored procedure and functions that are specified in the requirement in the Assignment 2 task pdf file.

1/ Increase Inpatient Fee to 10% for all the inpatients who are admitted to hospital from 01/09/2020.

```
CREATE OR ALTER PROCEDURE hospital.Increase_Fee
AS
BEGIN
   UPDATE
       hospital.ADMISSION
   SET
       Fee = Fee * 1.1
   WHERE
       Admission_Date >= '2020-09-01';
END;
```

- The above Procedure is used to increase the fee of an admission which has Admission_Date greater or equal than '2020-09-01'.

```
CREATE OR ALTER TRIGGER hospital.TRIG_INCREASE_FEE
ON hospital.ADMISSION
for INSERT, UPDATE
AS
BEGIN
   DECLARE @fee NUMERIC(10,1);
   DECLARE @admission_id INT;
   DECLARE @admission_date DATE;
```

```
    SELECT
        @fee = Fee,
        @admission_id = Admission_ID,
        @admission_date = Admission_Date
    FROM
        inserted;

    UPDATE hospital.ADMISSION
    SET Fee = @fee * 1.1
    WHERE Admission_ID = @admission_id
      And @admission_date>='2020-09-01';

    RETURN
END;
```

- The above Trigger is used to increase the fee when inserting or updating the fee of an admission which has Admission_Date greater or equal than '2020-09-01'.

2/ Select all the patients (outpatient & inpatient) of the doctor named 'Nguyen Van A'.

```
CREATE OR ALTER PROCEDURE hospital.Get_Patient_Of_NguyenVanA
AS BEGIN
    SELECT
        OUTPATIENT.Patient_ID,
        First_Name,
        Last_Name,
        Date_Of_Birth,
        Gender,
        Address,
        Phone
    FROM EXAMINATION
    JOIN
    (
        SELECT EMPLOYEE_ID
        FROM EMPLOYEE
```

```sql
        WHERE EMPLOYEE.job_type='d'
            AND CONCAT(EMPLOYEE.last_name, ' ', EMPLOYEE.first_name)
                LIKE '%Nguyen Van A%'
    ) AS DOCTOR_NVA
        ON EXAMINATION.Doctor_Exam_ID = DOCTOR_NVA.EMPLOYEE_ID
    JOIN OUTPATIENT
        ON EXAMINATION.Patient_ID=OUTPATIENT.Patient_ID
    UNION ALL

    SELECT
        INPATIENT.Patient_ID,
        First_Name,
        Last_Name,
        Date_Of_Birth,
        Gender,
        Address,
        Phone
    FROM TREATMENT_DOCTOR
    JOIN
    (
        SELECT EMPLOYEE_ID
        FROM EMPLOYEE
        WHERE EMPLOYEE.job_type='d'
            AND CONCAT(EMPLOYEE.last_name, ' ', EMPLOYEE.first_name)
                LIKE '%Nguyen Van A%'
    ) AS DOCTOR_NVA
        ON DOCTOR_NVA.EMPLOYEE_ID=TREATMENT_DOCTOR.DOCTOR_ID
    JOIN TREATMENT
        ON TREATMENT_DOCTOR.Admission_ID=TREATMENT.Admission_ID
        AND TREATMENT_DOCTOR.Treatment_ID=TREATMENT.Treatment_ID
    JOIN ADMISSION
        ON TREATMENT.Admission_ID=ADMISSION.Admission_ID
    JOIN INPATIENT
        ON ADMISSION.Patient_ID=INPATIENT.Patient_ID
END;
```

- We separately select inpatients and outpatients then union two results to produce a full list of patients belonging to a doctor named 'Nguyen Van A'.

3/ Write a function to calculate the total medication price a patient has to pay for each treatment or examination

Input: Patient ID

Output: A list of balance per treatment or examination

```
CREATE OR ALTER FUNCTION hospital.Calculate_Fee
   (@inputID AS CHAR(8))
   RETURNS @totalFee TABLE (Total INT)
AS BEGIN
   IF
       SUBSTRING(@inputID, 1, 1) = 'I'
       INSERT INTO
           @totalFee
       SELECT
           sum(m.price*tm.Amount) Total
       FROM
           ADMISSION a
       LEFT JOIN
           TREATMENT_MEDICATION tm
           ON a.Admission_ID = tm.Admission_ID
       LEFT JOIN
           MEDICATION m
           ON tm.Drug_Code = m.Drug_Code
       WHERE
           a.Patient_ID = @inputID
       GROUP BY
           a.Admission_ID,
           tm.Treatment_ID
   ELSE
       INSERT INTO
           @totalFee
       SELECT
           SUM(m.Price*em.Amount) Total
       FROM
```

```
            EXAMINATION_MEDICATION em
        LEFT JOIN
            MEDICATION m
            ON em.Drug_Code = m.Drug_Code
        WHERE
            em.Patient_ID = @inputId
        GROUP BY
            em.Exam_ID
    RETURN
END
```

- In this function, we will return the table with the price for each treatment (or examination) depending on the input ID (can both be Inpatient or Outpatient).

4/ Write a procedure to sort the doctor in an increasing number of patients he/she takes care of in a period of time.

Input: Start date, End date

Output: A list of sorting doctors.

```
CREATE OR ALTER PROCEDURE hospital.Sort_Doctors_In_A_Period_Of_Time
    @start_date AS DATE,
    @end_date AS DATE
AS BEGIN
    SELECT
        EMPLOYEE.Employee_ID,
        -- CONCAT(EMPLOYEE.Last_Name, ' ', EMPLOYEE.First_Name) AS Employee_Name,
        COUNT(*) AS NUM_PATIENTS
    FROM
        EMPLOYEE
    JOIN
        TREATMENT_DOCTOR
        ON EMPLOYEE.Employee_ID=TREATMENT_DOCTOR.Doctor_ID
    JOIN
        TREATMENT
```

```
        ON TREATMENT_DOCTOR.Admission_ID=TREATMENT.Admission_ID
        AND TREATMENT_DOCTOR.Treatment_ID=TREATMENT.Treatment_ID
    WHERE
         TREATMENT.START_DATE <= @end_date
         AND @start_date <= TREATMENT.END_DATE
    GROUP BY
        EMPLOYEE.Employee_ID
    ORDER BY
        NUM_PATIENTS;
END;
```

- Given two intervals $I = [i_1, i_2]$ and $J = [j_1, j_2]$, I and J overlap when there exists some number C belong to both intervals:

$$i_1 \leq C \leq i_2 \text{ and } j_1 \leq C \leq j_2$$

  therefore, it is sufficient to prove that two intervals overlap when:

$$i_1 \leq j_2 \text{ and } j_1 \leq i_2$$

- We use this condition to count the number of patients within the given period for each doctor, then sort the result by using ORDER BY statement.

BUILDING APPLICATIONS

In this assignment, we built a web-based application:

- Programming environment: Web application.
- Programming language: HTML/CSS, JavaScript for Front-end and PHP for Back-end.
- Database Server: **Azure SQL Database**, which is based on the latest stable version of the Microsoft SQL Server database engine.

1. Create user

We create a login and the corresponding user for the hospital manager to access the database. This user has the permission to **read/write** the database as well as **execute** functions/stored procedures.

```sql
-- Create Login
create login manager_1 WITH PASSWORD='group5@123';


-- Create account for read and write as well as execute functions/stored procedures
create user manager_1 for LOGIN manager_1 with DEFAULT_schema=[hospital];
EXEC sp_addrolemember 'db_datareader', 'manager_1';
EXEC sp_addrolemember 'db_datawriter', 'manager_1';
GRANT EXECUTE on SCHEMA ::hospital to manager_1;
```

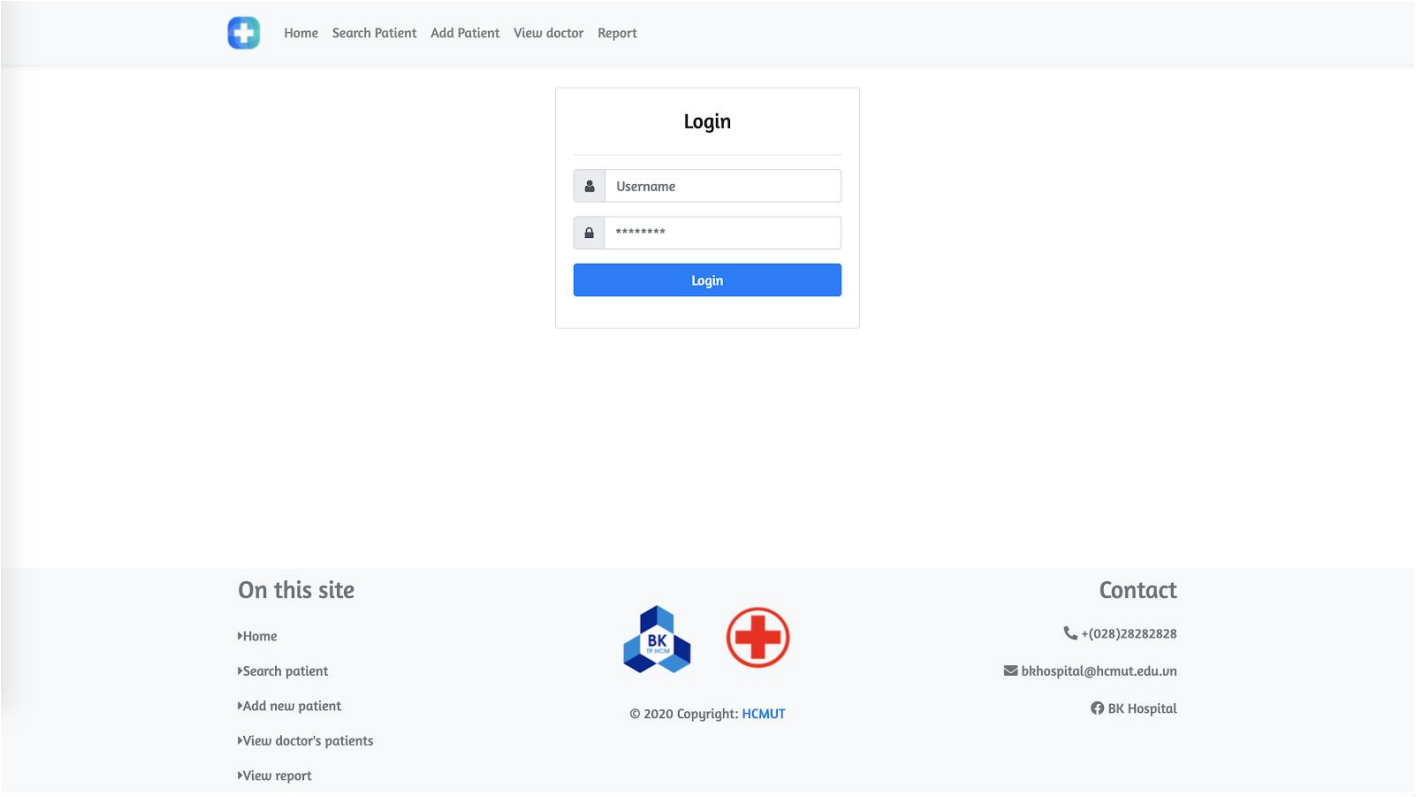
2. Requirement function

- **Login as user manager**
  - Log in, log out (enter the user name/password for Manager account to log in/out).
    + Using function connect() of class DbModel for connecting to database and login request.
    + First, we create a connection to the database server with the login account and then check if this account has the permission as manager or not.

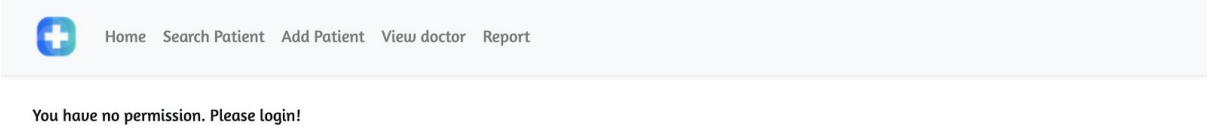
```php
class DbModel {
       private $servername = "group5-database.database.windows.net";
       private $db = "HospitalDB";

       public function connect($username, $password) {
           $servername = $this->servername;
           $connectionOptions = array(
               "Database" => $this->db,
               "UID" => $username,
               "PWD" => $password
           );

           $conn = sqlsrv_connect($servername, $connectionOptions);

           $sql = "IF (IS_MEMBER('db_datareader') = 1
                   and IS_MEMBER('db_datawriter') = 1)
                    SELECT 1
```

```
                ELSE
                 SELECT 0;";
        $getResults = sqlsrv_query($conn, $sql);
        if ($getResults == FALSE) {
            return false;
        }
        while ($row = sqlsrv_fetch_array($getResults, SQLSRV_FETCH_ASSOC)) {
            if (implode($row) == 1) {
                return $conn;
            }
        }
        return false;
    }
}
```

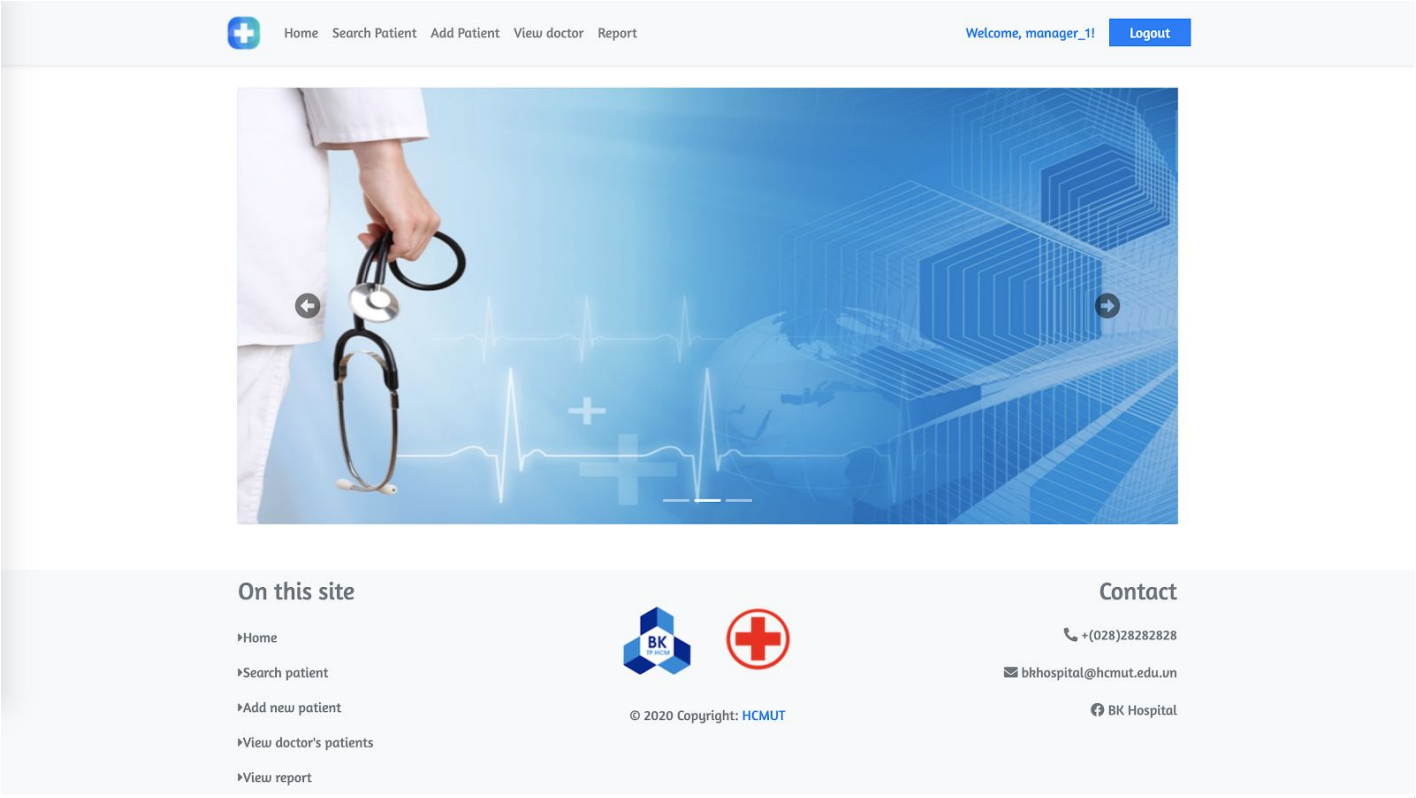
+ Login UI of our web application.



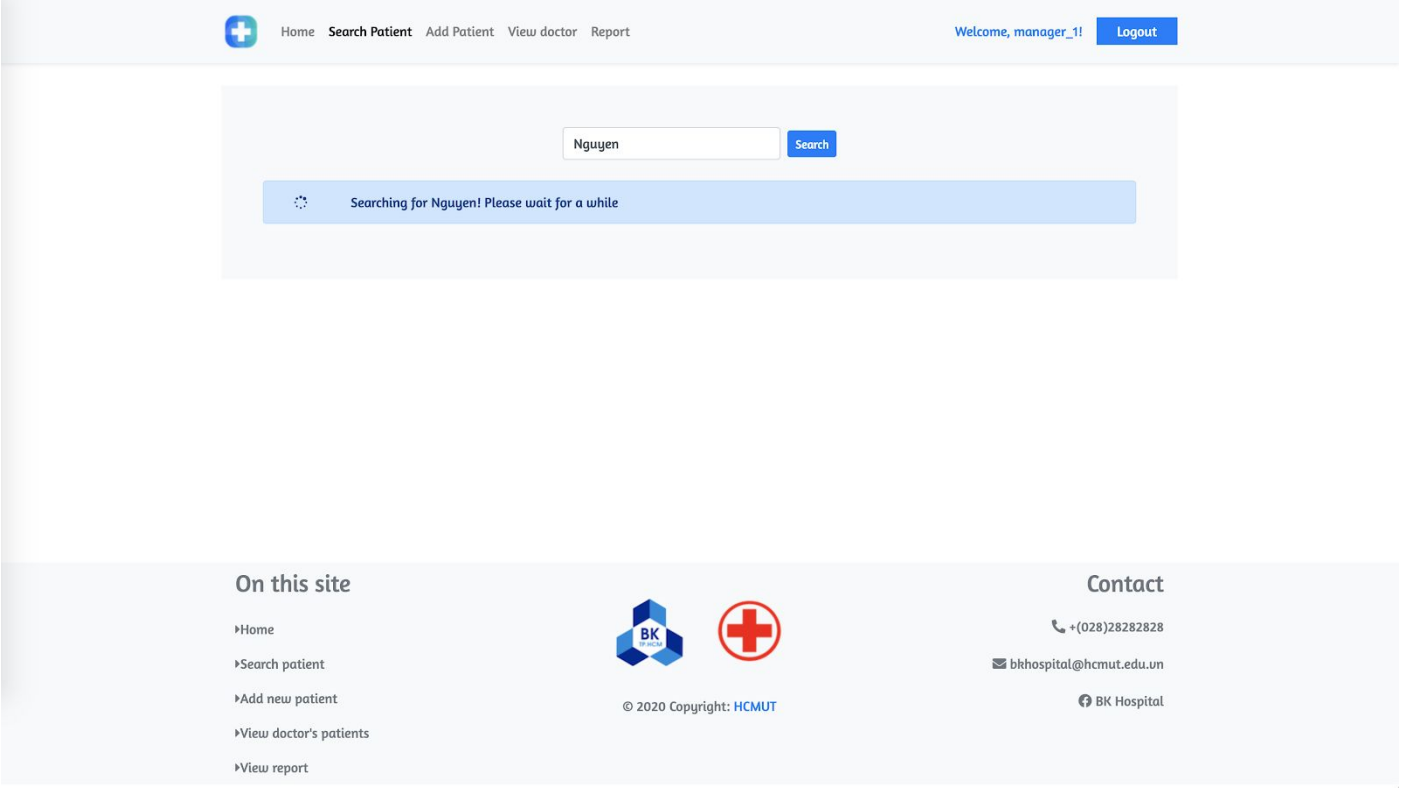
+ When login fails, our application will stay at the login page and the user can be able to access and use our functions.



+ When login success with manager_1 account, our application will display. Here, you can search all patient's information, add information for a new inpatient or outpatient, list details of all patients which are treated by a specific doctor, and make a report that provides full information about the payment for each treatment or examination of a patient.



- **Search patient information:**



Search results include the name, phone number and information about the treatments and visits of the patient.

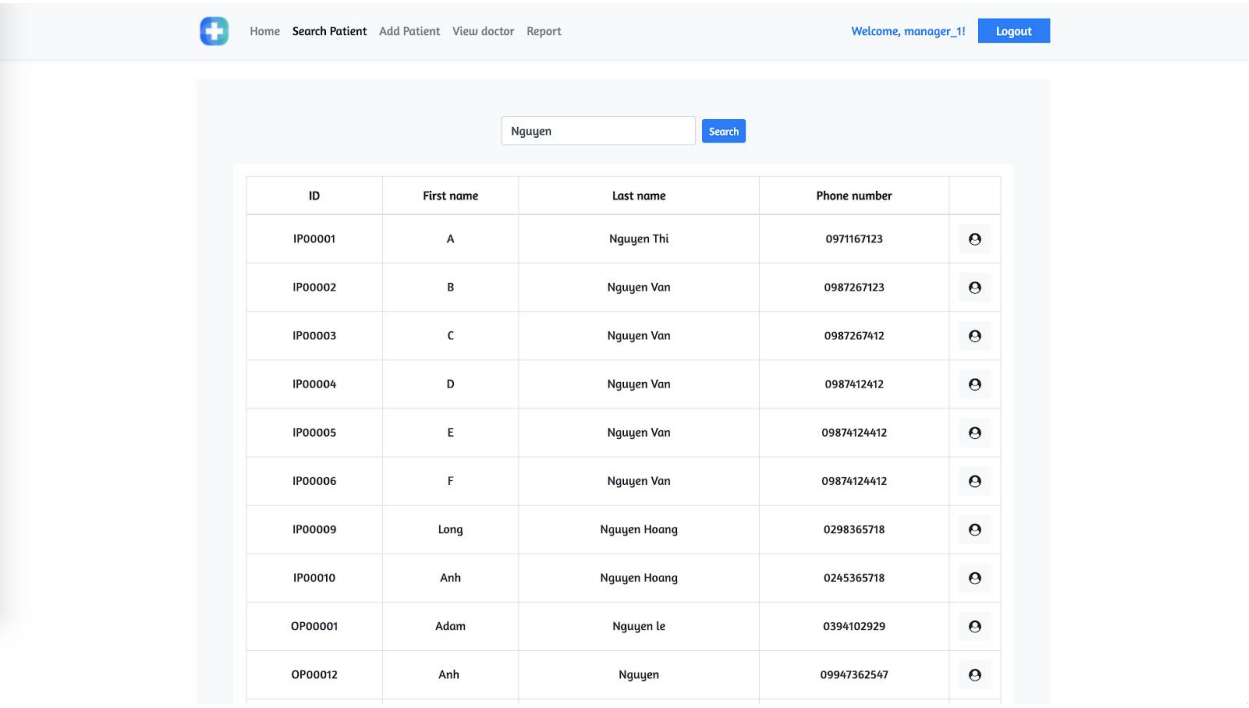
- Code query SQL for Inpatient:

```
$sql = "SELECT
            P.Patient_ID, P.First_Name, P.Last_Name, P.Phone,
            A.Admission_ID, A.Admission_Date, A.Discharge_Date, T.START_DATE,
            T.END_DATE, T.Result,
            CONCAT(N.Last_Name, ' ', N.First_Name) as Nurse_Name,
            CONCAT(e.Last_Name, ' ', E.First_Name) AS Doctor_Name
        FROM
            hospital.INPATIENT AS P
        JOIN
            hospital.ADMISSION AS A ON A.Patient_ID = P.Patient_ID
        JOIN
            hospital.TREATMENT AS T ON T.Admission_ID = A.Admission_ID
        JOIN
            hospital.TREATMENT_DOCTOR AS TD
            ON TD.Treatment_ID = T.Treatment_ID
            AND A.Admission_ID = TD.Admission_ID
        JOIN
            hospital.EMPLOYEE AS N
            ON N.Employee_ID = A.Nurse_ID
        JOIN
            hospital.EMPLOYEE AS E
            ON E.Employee_ID = TD.Doctor_ID
        WHERE
            P.First_Name = '$searchStr'
            OR P.Last_Name LIKE '%$searchStr'
            OR P.Last_Name LIKE '$searchStr%'
            OR CONCAT(P.Last_Name, ' ', P.First_Name) LIKE '$searchStr%'
            OR CONCAT(P.First_Name, ' ', P.Last_Name) LIKE '$searchStr%'
            OR P.Patient_ID LIKE '$searchStr%'
        ORDER BY
            P.Patient_ID";
```

○ Code query SQL for Outpatient:

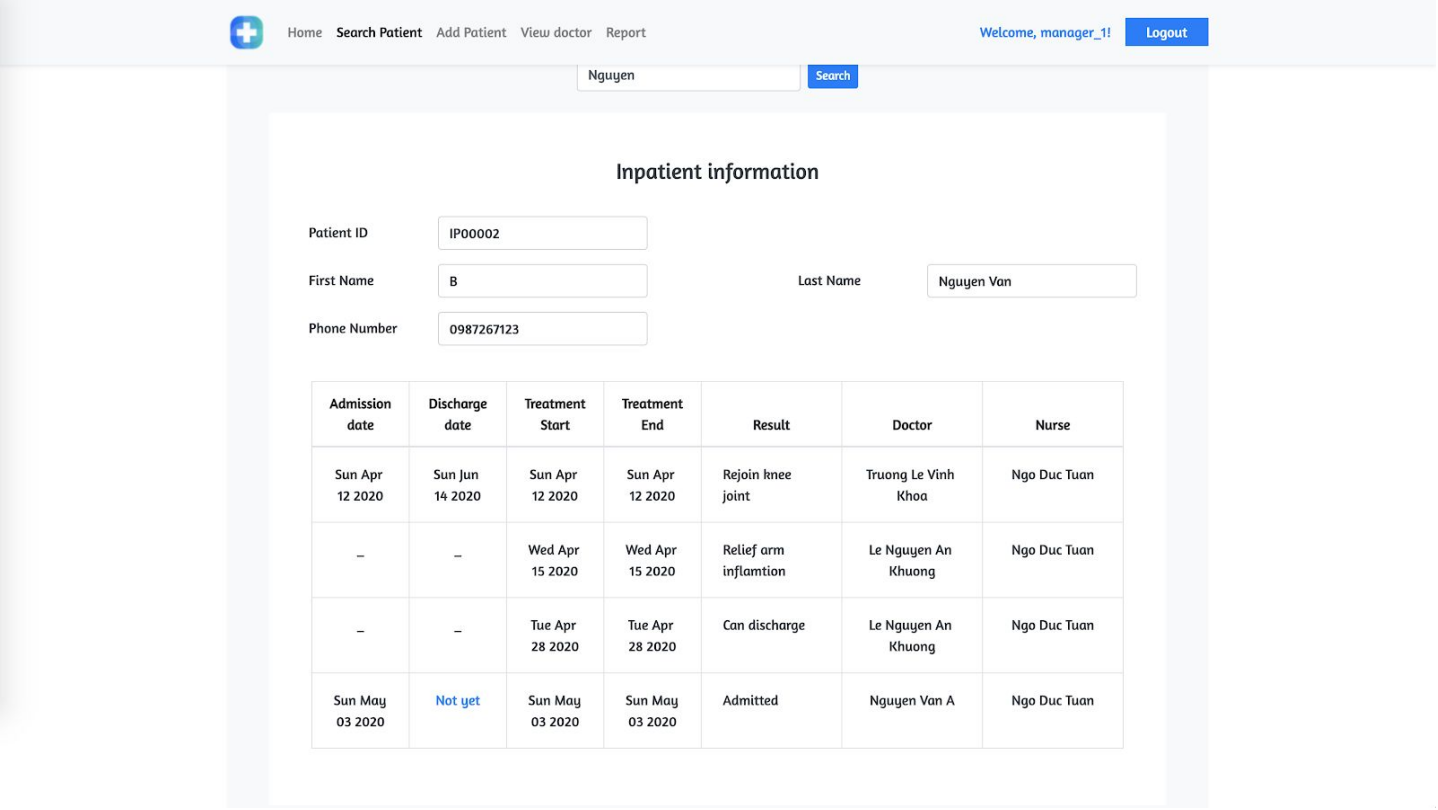
```
$sql = "SELECT
            P.Patient_ID, P.Last_Name, P.First_Name, P.Phone, EX.Exam_Date,
EX.Second_Exam_Date,
            CONCAT(E.Last_Name, ' ', E.First_Name) AS Doctor_Name, EX.Diagnosis
        FROM
            hospital.OUTPATIENT AS P
        JOIN
            hospital.EXAMINATION AS EX ON EX.Patient_ID = P.Patient_ID
        JOIN
            hospital.EMPLOYEE AS E ON E.Employee_ID = EX.Doctor_Exam_ID
        WHERE
            P.First_Name = '$searchStr'
            OR P.Last_Name LIKE '%$searchStr'
            OR P.Last_Name LIKE '$searchStr%'
            OR CONCAT(P.Last_Name, ' ', P.First_Name) LIKE '$searchStr%'
            OR CONCAT(P.First_Name, ' ', P.Last_Name) LIKE '$searchStr%'
            OR P.Patient_ID LIKE '$searchStr%'";
```

+ Results of search functions for patients' name which includes Nguyen in both First and Last Name

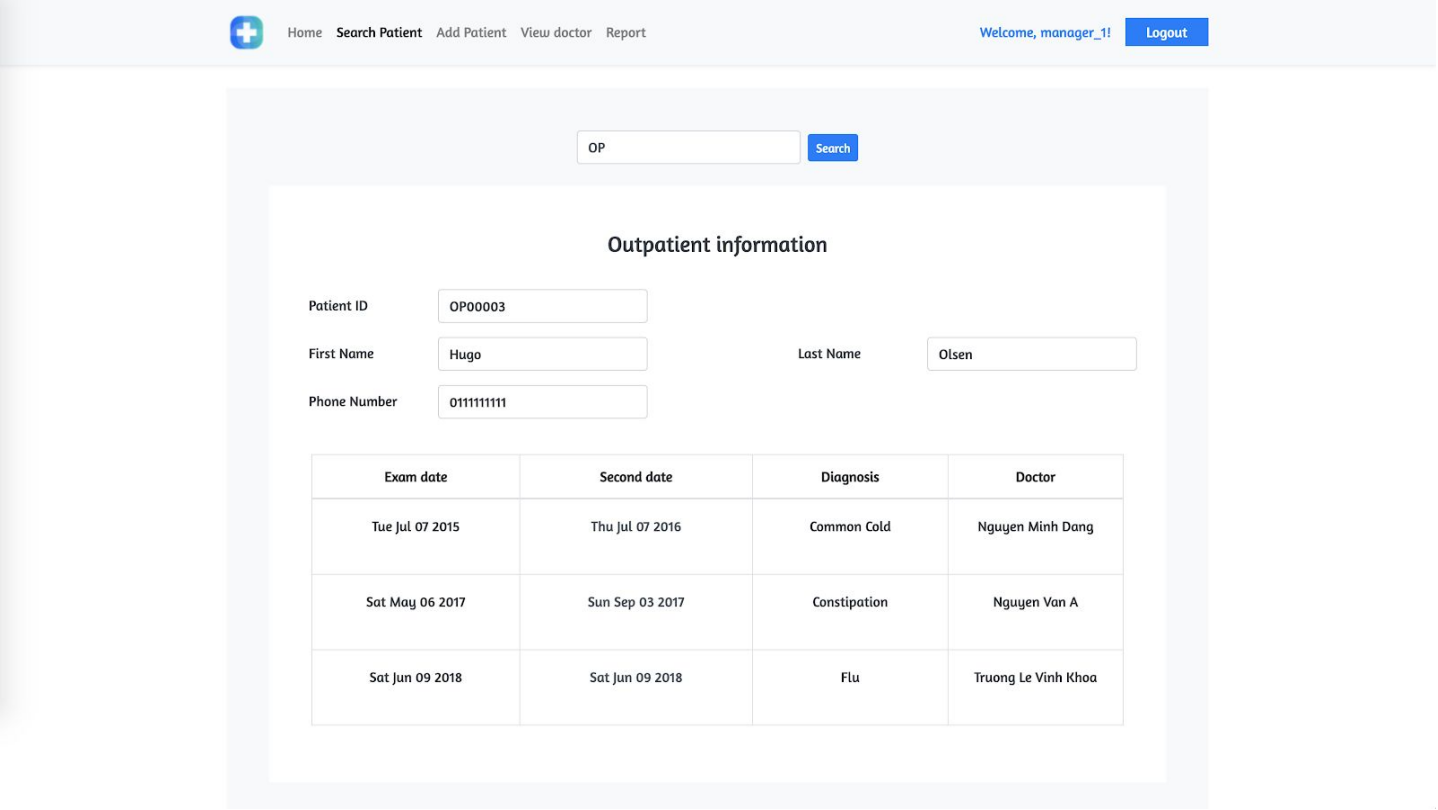


ID	First name	Last name	Phone number	
IP00001	A	Nguyen Thi	0971167123	
IP00002	B	Nguyen Van	0987267123	
IP00003	C	Nguyen Van	0987267412	
IP00004	D	Nguyen Van	0987412412	
IP00005	E	Nguyen Van	09874124412	
IP00006	F	Nguyen Van	09874124412	
IP00009	Long	Nguyen Hoang	0298365718	
IP00010	Anh	Nguyen Hoang	0245365718	
OP00001	Adam	Nguyen le	0394102929	
OP00012	Anh	Nguyen	09947362547	

+ Information of inpatients including their information and each admission.
+ Search inpatient with ID or name and return the patient's admission date. Each admission will have one or several treatments (start time and end time of the treatment). Admission that has no discharge date means the patients are still in the treatment process and have not discharged yet



+ Information of outpatients including their information and each examinations that they visited

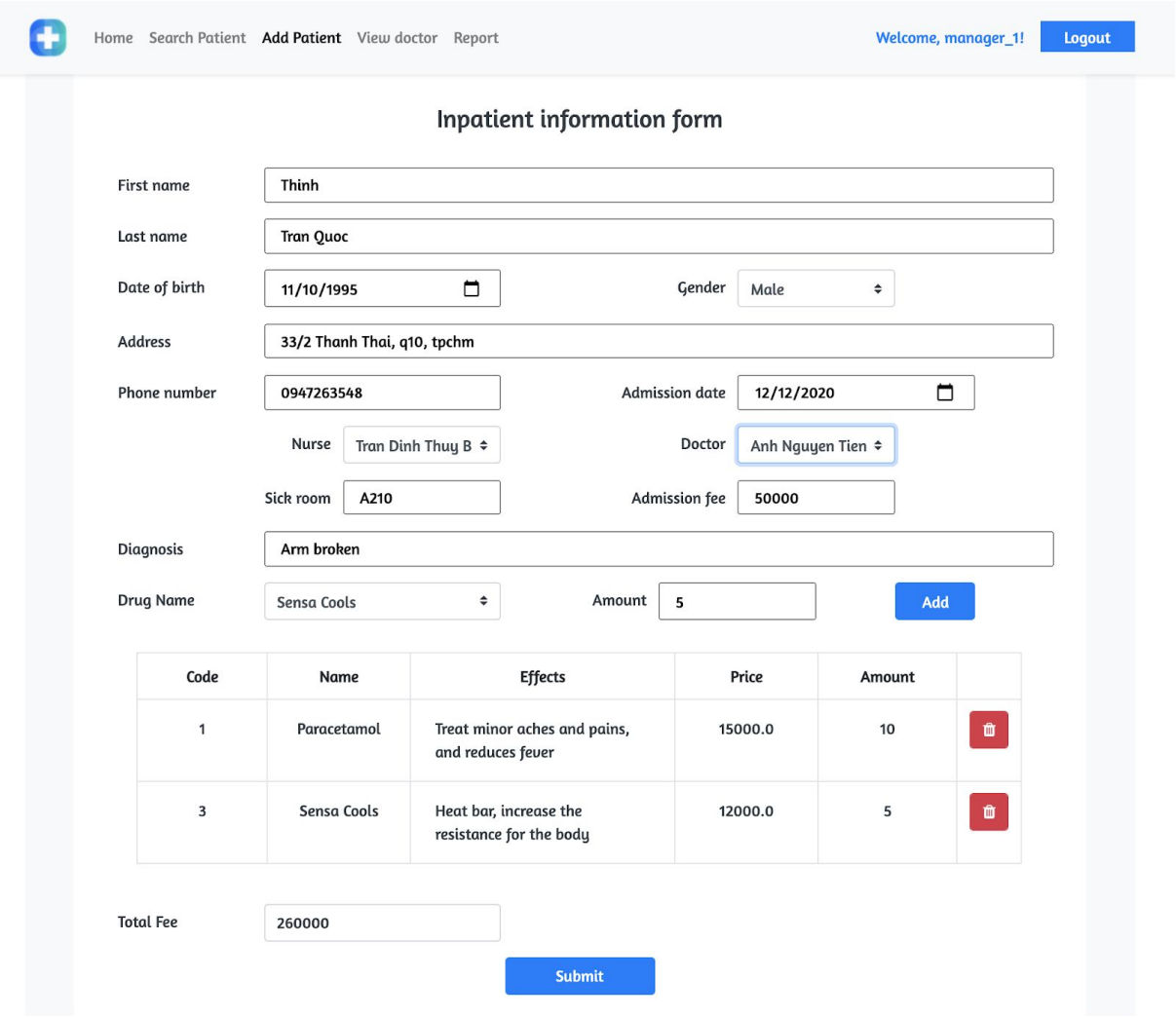


- **Add information for a new patient.**
  Inpatient
  - Query SQL Code for adding new Inpatient:

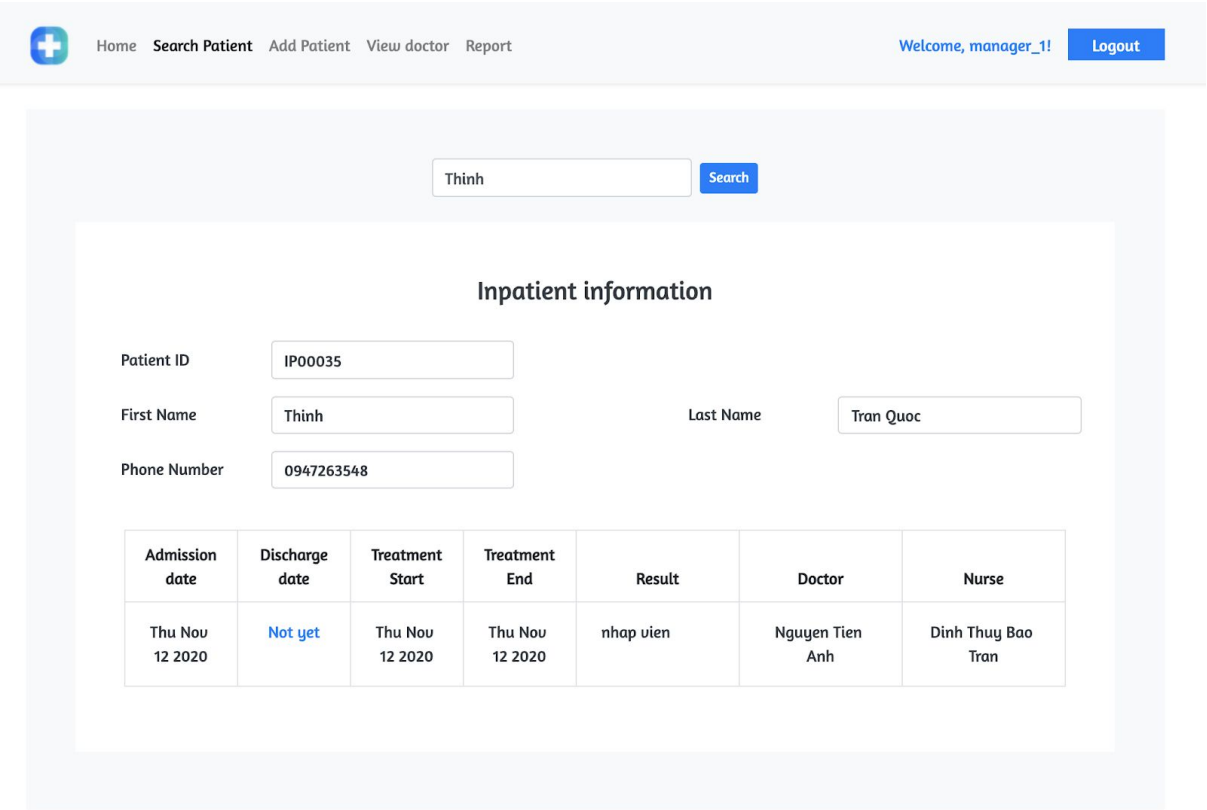
```
-- This procedure will add inpatient information and their first treatment
$sql= "EXEC hospital.NewInpatient
       @First_Name = '$fName',
       @Last_Name = '$lName',
       @Date_Of_Birth ='$dob',
       @Gender ='$gender',
       @Address = '$addr',
       @Phone = '$phone',
       @Nurse_ID = '$nurseId',
       @Admission_Date = '$date',
       @Sickroom = '$room',
       @Diagnosis = '$diagnosis',
       @Fee = '$fee',
       @Doctor_ID = '$doctorId'";
```

```
-- We then loop our table to add medication for this treatment
$sql = "EXEC hospital.NewTreatmentMedication
       @Admission_ID = $aId,
       @Treatment_ID = $tId,
       @Drug_Code = '$code',
       @Amount = '$amount'";
```

- We add information for new inpatient, which includes the first treatment as 'nhập viện' and medication for that treatment.
- To add the drug information for a new inpatient, we in turn add new treatment with a corresponding drug on the table.



- Then search again to check whether a new inpatient is added to the database or not.



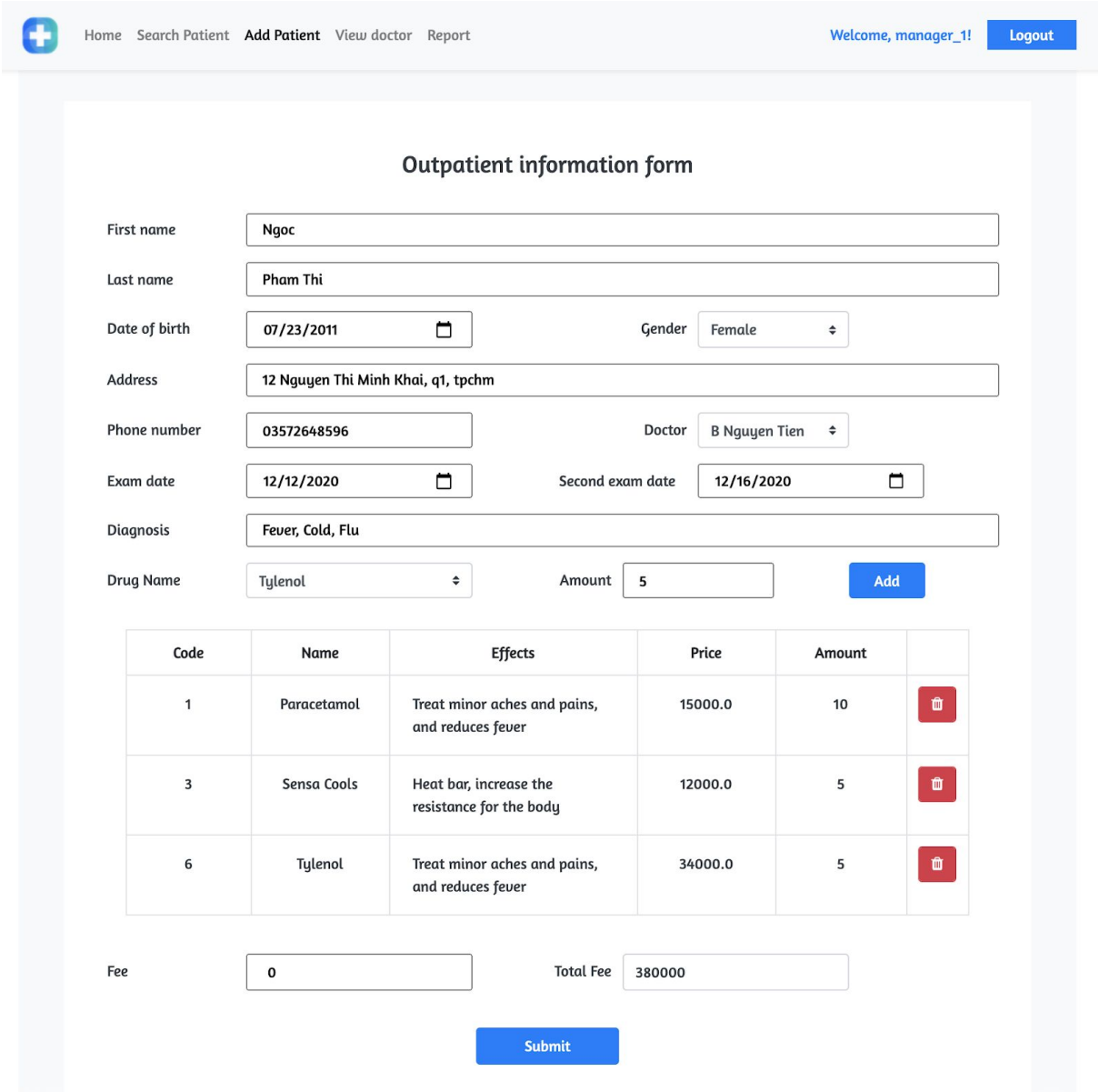
Outpatient

- Query SQL Code for adding new Outpatient:

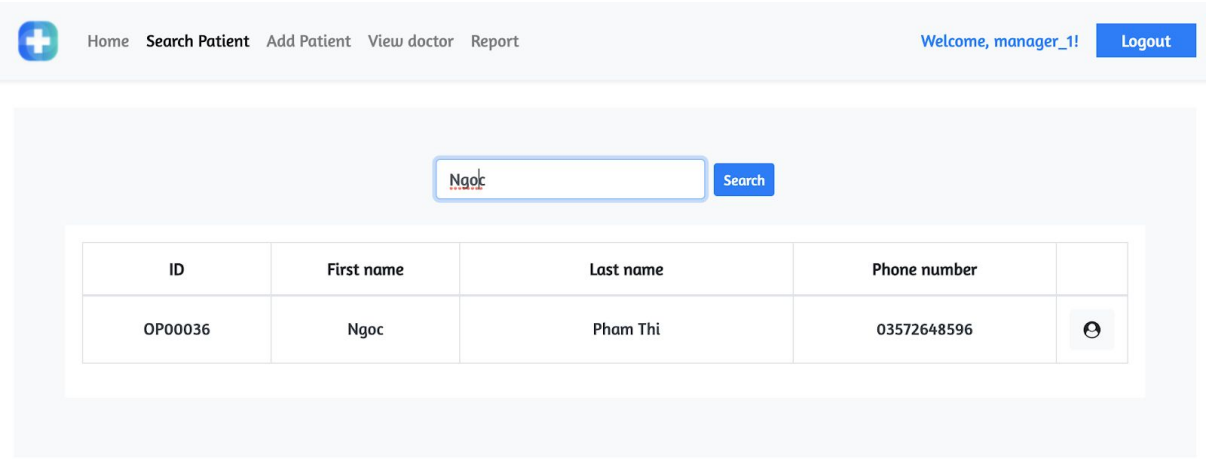
```
-- This procedure will add outpatient information and their first examination
$sql= "EXEC hospital.NewOutPatient
       @First_Name = '$fName'
       ,@Last_Name = '$lName'
       ,@Phone = '$phone'
       ,@Address = '$addr'
       ,@Gender = '$gender'
       ,@Date_Of_Birth = '$dob'
       ,@Doctor_Exam_ID = '$doctorId'
       ,@Exam_Date = '$examDate'
       ,@Second_Exam_Date = '$secondDate'
       ,@Diagnosis = '$diagnosis'
```

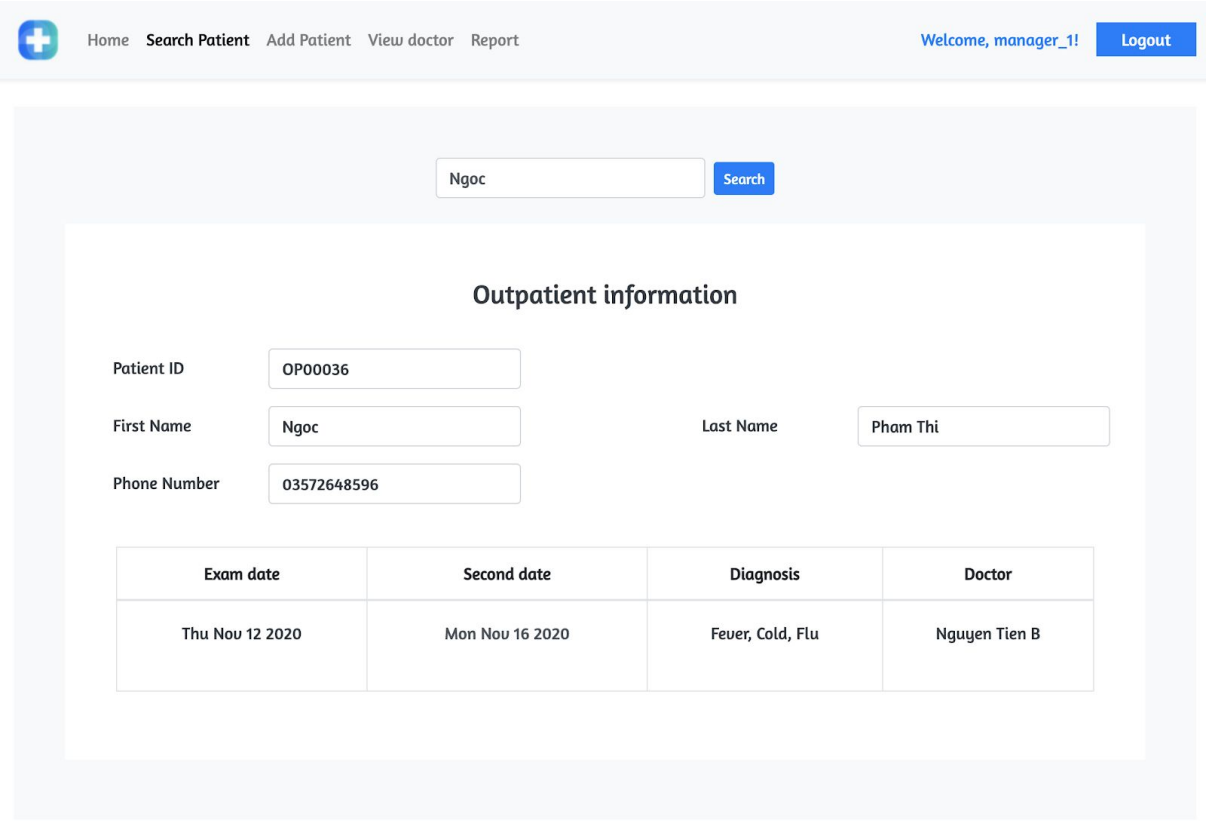
```
        ,@Fee = '$fee'";

-- We then loop our table to add medication for this examination
$sql = "EXEC hospital.NewTExaminationMedication
        @Patient_ID = '$pId'
        ,@Exam_ID = '$eId'
        ,@Drug_Code = '$code'
        ,@Amount = '$amount'";
```

- We add information for new outpatients.



- Then search again to check whether a new outpatient is added to the database or not.





- **List details of all patients which are treated by a doctor.**
  - Query SQL Code for Inpatient:

```
$sql =" SELECT
            P.Patient_ID, P.First_Name, P.Last_Name, P.Phone, P.Address,
            P.Date_Of_Birth, P.Gender, A.Admission_ID, A.Admission_Date,
            A.Discharge_Date, T.Treatment_ID, T.START_DATE, T.END_DATE,
            T.Result, M.Name, TM.Amount, M.Price, M.Expiration_Date
        FROM
            hospital.INPATIENT AS P
        JOIN
            hospital.ADMISSION AS A
            ON A.Patient_ID = P.Patient_ID
        JOIN
            hospital.TREATMENT AS T
            ON T.Admission_ID = A.Admission_ID
        JOIN
            hospital.TREATMENT_DOCTOR AS TD
            ON TD.Treatment_ID = T.Treatment_ID AND A.Admission_ID = TD.Admission_ID
        JOIN
            hospital.EMPLOYEE AS E
            ON E.Employee_ID = TD.Doctor_ID
        JOIN
            hospital.TREATMENT_MEDICATION AS TM
            ON TM.Admission_ID = A.Admission_ID
            AND TM.Treatment_ID = T.Treatment_ID
        JOIN
            hospital.MEDICATION AS M
            ON M.Drug_Code = TM.Drug_Code
        WHERE
            TD.Doctor_ID = $dId
        ORDER BY
            P.Patient_ID, A.Admission_ID, T.Treatment_ID";
```

- Query SQL Code for Outpatient:

```
$sql ="SELECT
            P.Patient_ID, P.Last_Name, P.First_Name, P.Phone, P.Address,
            P.Date_Of_Birth, P.Gender, EX.Exam_ID, EX.Exam_Date, EX.Diagnosis,
            EX.Fee, EX.Second_Exam_Date, M.Name, EM.Amount, M.Price,
            M.Expiration_Date
        FROM
            hospital.OUTPATIENT AS P
        JOIN
            hospital.EXAMINATION AS EX
            ON EX.Patient_ID = P.Patient_ID
        JOIN
            hospital.EXAMINATION_MEDICATION AS EM
            ON EX.Exam_ID = EM.Exam_ID AND P.Patient_ID = EM.Patient_ID
        JOIN
            hospital.MEDICATION AS M
            ON M.Drug_Code = EM.Drug_Code
        JOIN
            hospital.EMPLOYEE AS E
            ON E.Employee_ID = EX.Doctor_Exam_ID
        WHERE
            EX.Doctor_Exam_ID = $dId
        ORDER BY
            P.Patient_ID, EX.Exam_ID";
```

- List details of all patients which are treated by a doctor Le Nguyen An Khuong. We just took photos of two patients (one inpatient and one outpatient) as a demonstration since the list of patients is quite long.

Home Search Patient Add Patient View doctor Report Welcome, manager_1! Logout

Doctor: Khuong Le Nguyen An

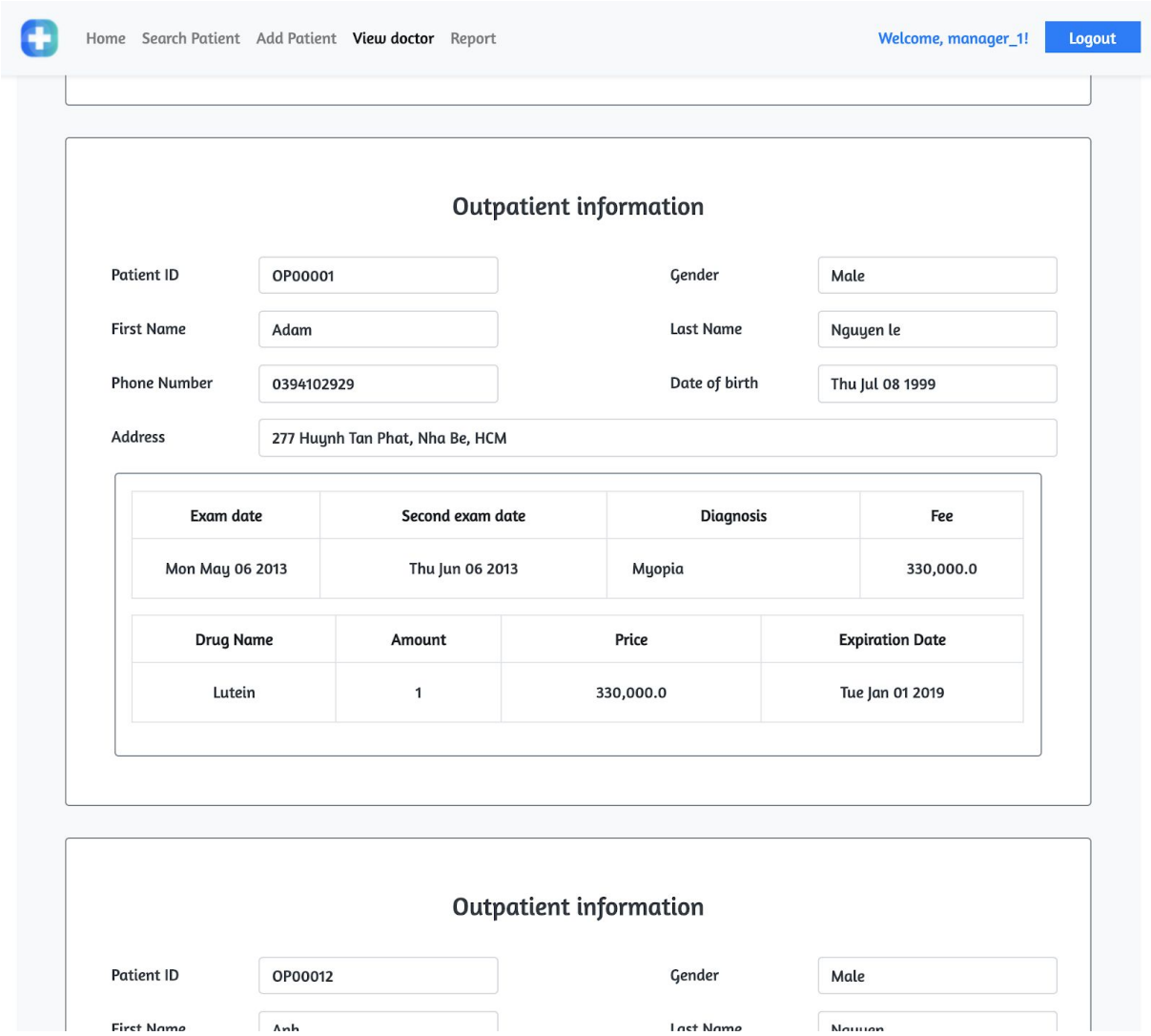
Inpatient information

Patient ID	IP00001	Gender	Male
First Name	A	Last Name	Nguyen Thi
Phone Number	0971167123	Date of birth	Thu Feb 12 1970
Address	412 Lang Cha Ca, Quan 1		

Amission Date: Sat Feb 01 2020 Discharge Date: Sat Mar 14 2020

Start date	End date	Result
Sun Feb 02 2020	Sun Feb 02 2020	Less diarrhea

Drug Name	Amount	Price	Expiration Date
Paracetamol	8	15,000.0	Sat Jan 01 2022
Sensa Cools	9	12,000.0	Sat Jan 01 2022
UPSA-C	4	26,000.0	Sat Jan 01 2022



- **Make a report that provides full information about the payment for each treatment or examination of a patient.**
  - Query SQL Code for inpatients' report:

```
$sql = "SELECT
            P.Patient_ID, P.First_Name, P.Last_Name, P.Phone, P.Address, P.Gender,
            P.Date_Of_Birth, A.Admission_ID, A.Fee, A.Admission_Date,
            A.Discharge_Date, T.Treatment_ID, T.START_DATE, T.END_DATE, T.Result,
            M.Name, TM.Amount, M.Price,
            CONCAT(E.Last_Name, ' ', E.First_Name) AS Doctor_Name
        FROM
            hospital.INPATIENT AS P
        JOIN
            hospital.ADMISSION AS A
```

```
            ON A.Patient_ID = P.Patient_ID
        JOIN
            hospital.TREATMENT AS T
            ON T.Admission_ID = A.Admission_ID
        JOIN
            hospital.TREATMENT_DOCTOR AS TD
            ON TD.Treatment_ID = T.Treatment_ID
            AND A.Admission_ID = TD.Admission_ID
        JOIN
            hospital.EMPLOYEE AS E
            ON E.Employee_ID = TD.Doctor_ID
        JOIN
            hospital.TREATMENT_MEDICATION AS TM
            ON TM.Admission_ID = A.Admission_ID
            AND TM.Treatment_ID = T.Treatment_ID
        JOIN
            hospital.MEDICATION AS M
            ON M.Drug_Code = TM.Drug_Code
        WHERE
            P.Patient_ID = '$pId'
        ORDER BY
            A.Admission_ID, T.Treatment_ID";
```

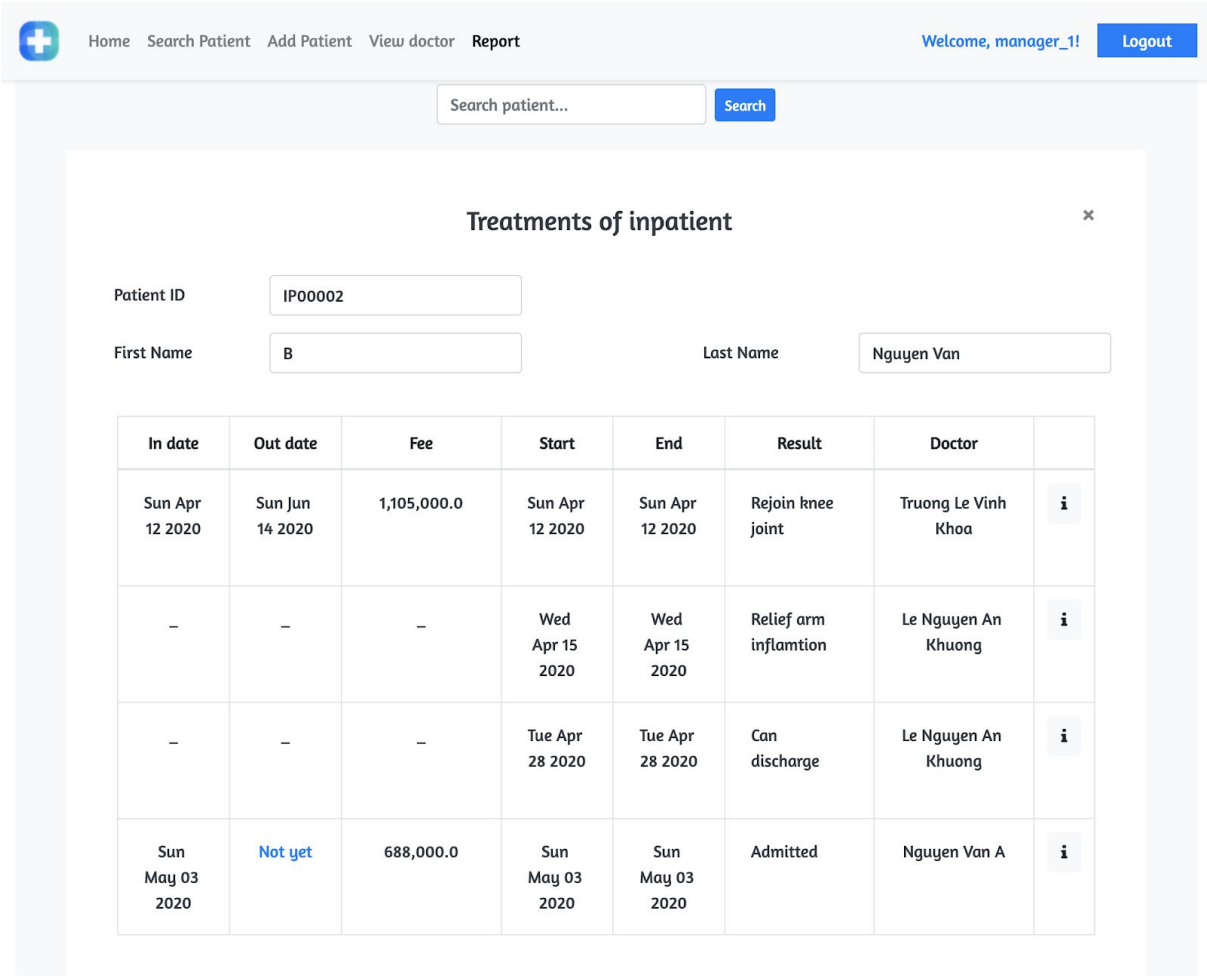
- Query SQL Code for outpatients' report:

```
$sql = "SELECT
            P.Patient_ID, P.Last_Name, P.First_Name, P.Phone, P.Address,
            P.Date_Of_Birth, P.Gender, EX.Exam_Date, EX.Second_Exam_Date,
            EX.Diagnosis, EX.Exam_ID, EX.Fee, M.Name, EM.Amount, M.Price,
            CONCAT(E.Last_Name, ' ', E.First_Name) AS Doctor_Name
        FROM
            hospital.OUTPATIENT AS P
        JOIN
            hospital.EXAMINATION AS EX
            ON EX.Patient_ID = P.Patient_ID
        JOIN
            hospital.EXAMINATION_MEDICATION AS EM
            ON EX.Exam_ID = EM.Exam_ID AND P.Patient_ID = EM.Patient_ID
        JOIN
```

```
        hospital.MEDICATION AS M
        ON M.Drug_Code = EM.Drug_Code
    JOIN
        hospital.EMPLOYEE AS E
        ON E.Employee_ID = EX.Doctor_Exam_ID
    WHERE
        P.Patient_ID = '$pId'
    ORDER BY
        EX.Exam_ID";
```

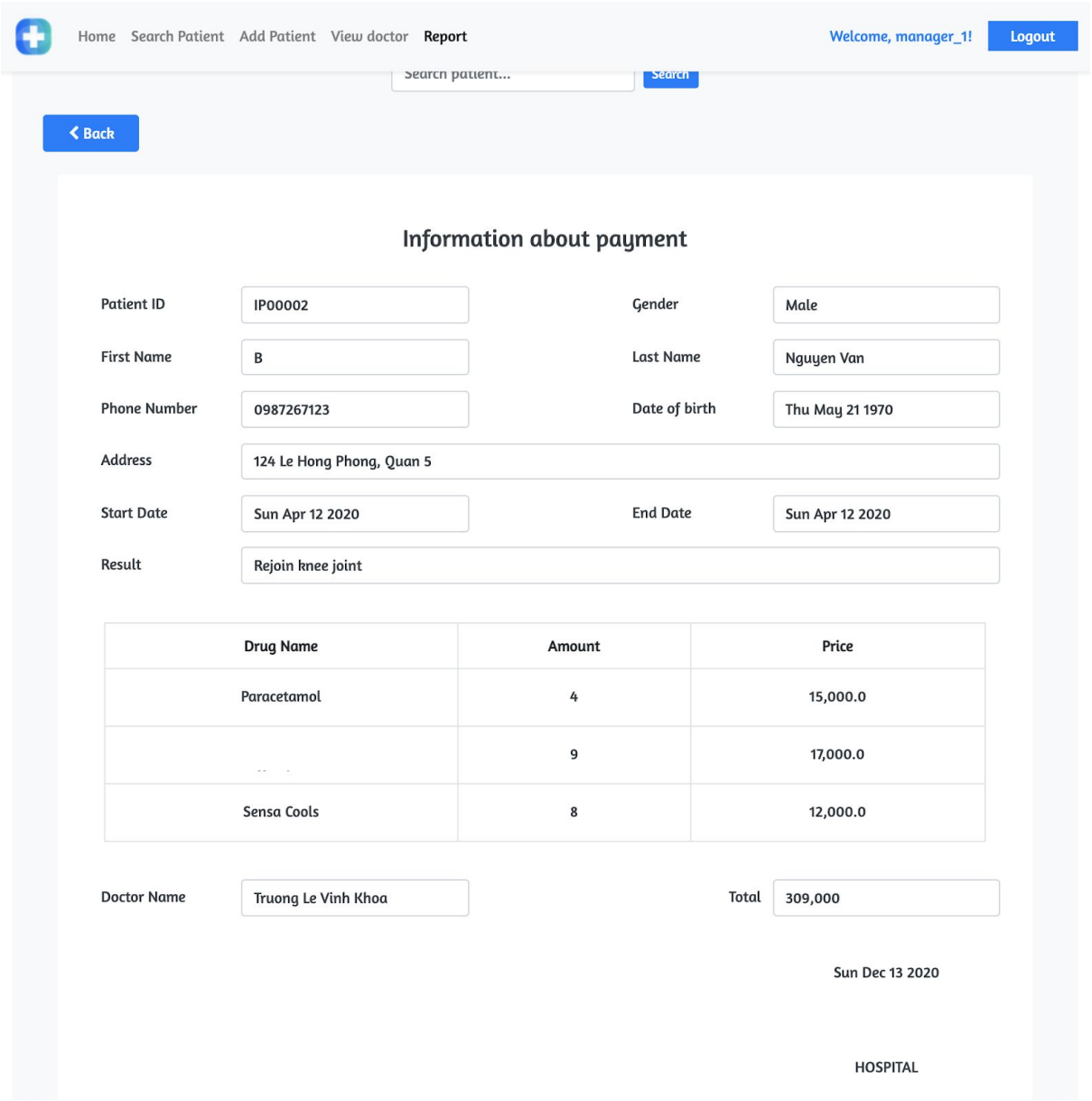
Inpatient

- First we list all treatments that belong to specific admissions of inpatients. With each admission, we have a total fee including the fee of all treatment in that admission.



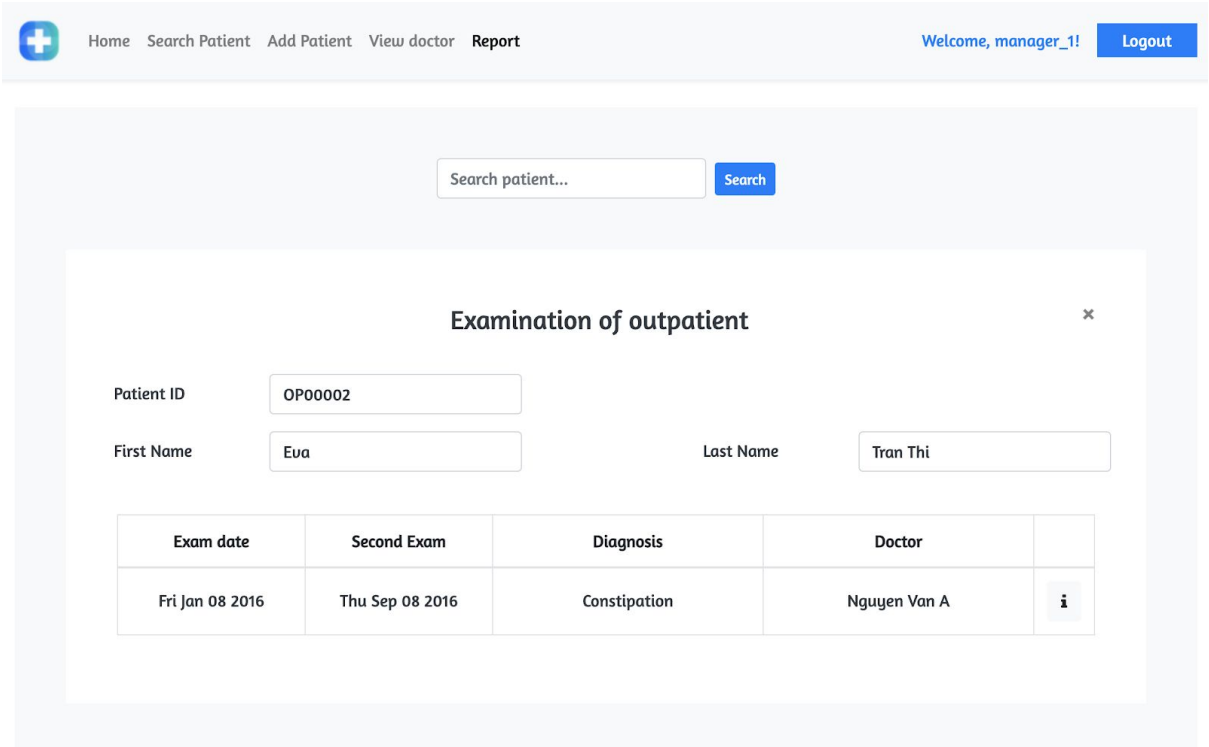
- To view the payment information of each treatment in detail, we click on the information button (! button) at the last column of the

corresponding row. The detail information of payment will be displayed as below.

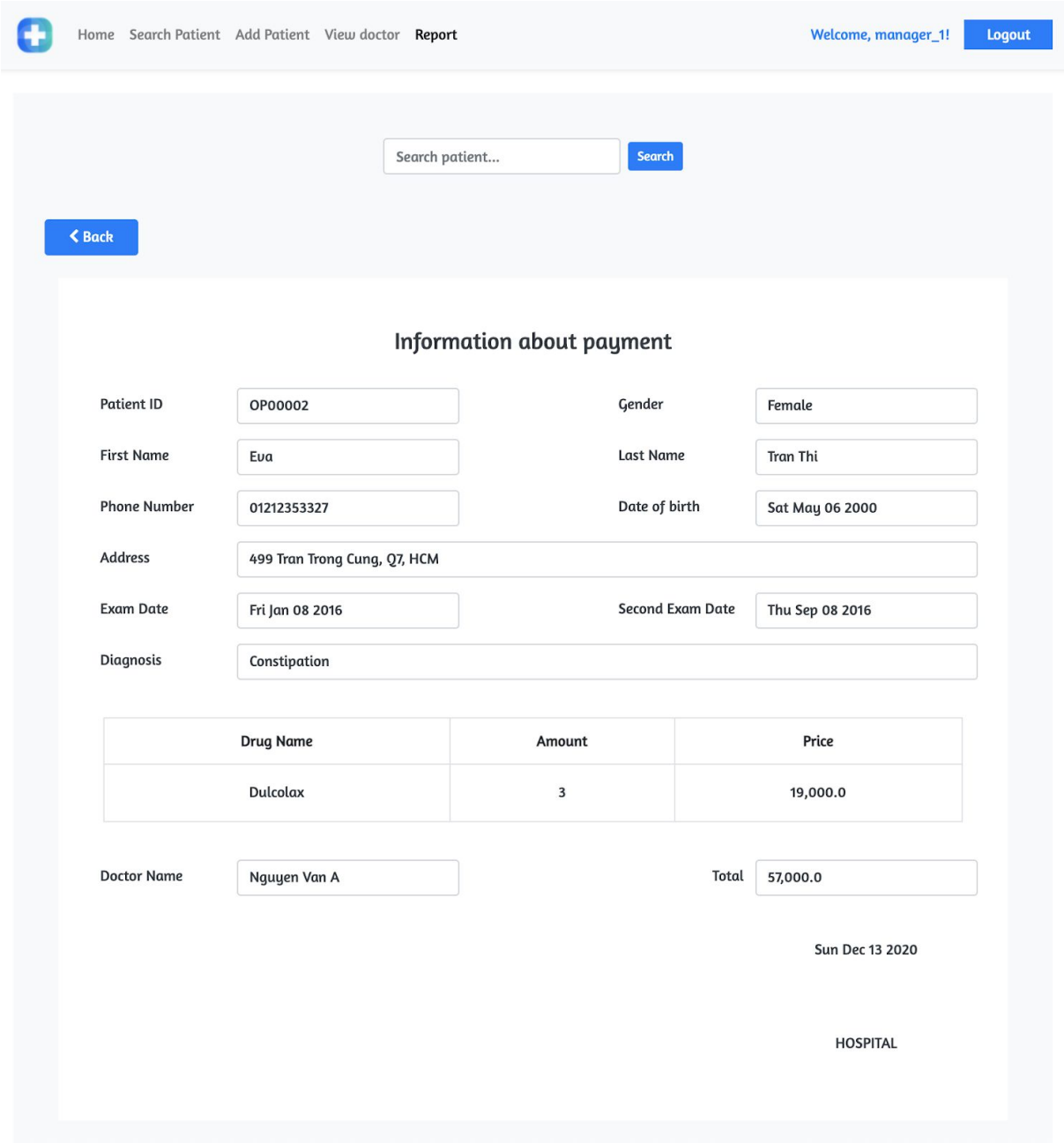


Outpatient

- First, we list all the examinations of outpatients.



- To view the payment information of each examination in detail, we click on the information button (! button) at the last column of the corresponding row. The detailed information of payment will be displayed as above.



Github

The source code of this web application can be found at Github.