VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Database Systems (CO2014)

Assignment 2

Store Procedure & Application

Group 1

Professor: Phan Trọng Nhân
Group Members: Nguyễn Đình Khương Duy 1952207
Nguyễn Lê Nhật Dương 1952638
Nguyễn Văn Quốc Chương 1950004
Trương Đăng Quang 1952940
Trần Nguyễn Phước Nhân 1952893

HO CHI MINH CITY, November 2021

Contents

Member list & Workload

No.	Fullname	Student ID	Workload
1	Nguyền Đình Khương Duy	1952207	20%
2	Nguyễn Lê Nhật Dương	1952638	20%
3	Nguyễn Văn Quốc Chương	1950004	20%
4	Trương Đăng Quang	1952940	20%
5	Trần Nguyễn Phước Nhân	1952893	20%

1 Physical Database Design

1.1 Relational Data Schema

From the `Assignment 1` we have the following `Relational Schema`.

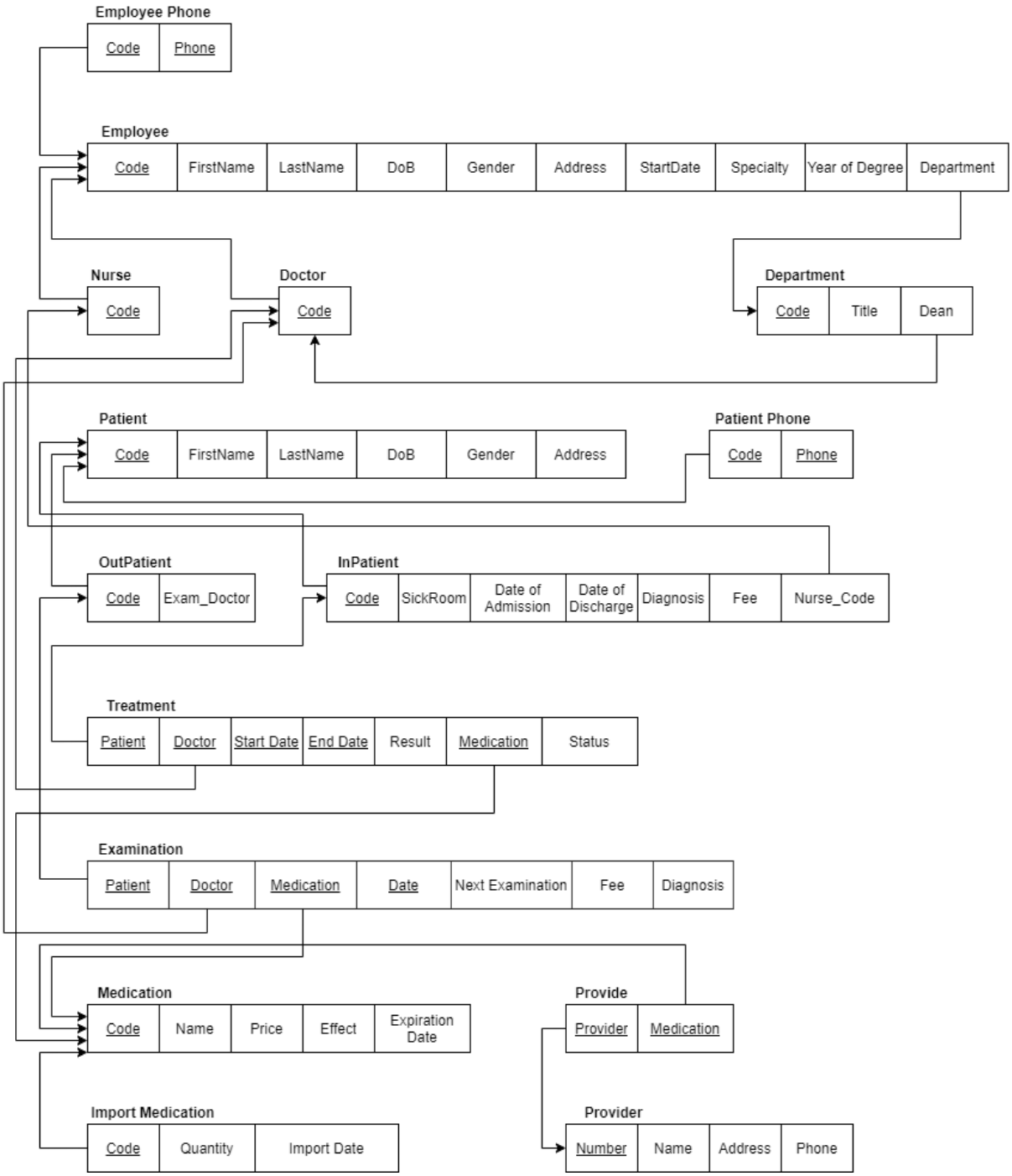

Figure 1: Relational Schema

1.2 Implement the database

From the Relational Schema we have already done from the first part of the the Assignment, we can put all of the schema into practice. Using SQL (Structured Query Language) as DDL(Data definition Language) and DML(Data manipulation language), we could translate these table into SQL code. All of these was implemented in Oracle DBMS.
First of all, we have got the first table: Employee,

```
CREATE TABLE EMPLOYEE(
Code                        VARCHAR(9),
FirstName                   VARCHAR(40) NOT NULL,
LastName                    VARCHAR(40) NOT NULL,
DoB                         DATE,
Gender                      CHAR,
Address                     VARCHAR(40),
StartDate                   DATE,
Specialty                   VARCHAR(20),
YearOfDeg                   INT,
PRIMARY KEY (Code)
);
```

Code data type is string, the length of the string is 9 from our group's definition, because it should be the same as the code of Patient's code. FirstName and LastName are also string, the length of these strings should be 40, because mostly first and last names do not have the length more than 40, so 40 is the optimal value for it, also they must not be NULL because everyone has a specific name. With Gender we use the CHAR data type to present the gender with 2 values Male(M) and Female(F). Addresses should be lengthy enough so we choose the length of 40 characters for this field. DoB and StartDate data types are DATE because it accurately describe the date with many date formats. Specialty should be lengthy enough to display most specialties in the Hospital, and we choose 20 for the most optimal value. YearOfDeg is just to show a year, so we use INT for perfectly fitted data type. Finally, we have to declare what is the PRIMARY KEY and here it is the Code attribute.
We move to the next table, which is Employee_Phone

```
CREATE TABLE Employee_Phone(
Code                        VARCHAR(9)      NOT NULL,
Phone_Num                   VARCHAR(15),
PRIMARY KEY (Code, Phone_Num),
FOREIGN KEY (Code) REFERENCES Employee(Code)
);
```

- Code is related to the Code we pre-defined from Employee table, so it has the length of 9
- Phone_Num should be stored inside a string for easier modification. The optimal length should be 15 because some numbers can contain the prefix code of certain areas.

- PRIMARY KEY here are Code and Phone_Num because phone numbers are unique for each individual, and the Code attribute is referenced from the Employee table.
- FOREIGN KEY is Code referenced from the Employee table.

The next table is Doctor:

```
CREATE TABLE Doctor(
Code                        VARCHAR(9),
PRIMARY KEY (Code),
FOREIGN KEY (Code)          REFERENCES Employee (Code)
);
```

- Code attribute is referenced from the Employee table, so it has the length of 9 as well.
- PRIMARY KEY is Code, because it is the only attribute.
- FOREIGN KEY is Code, referenced from Employee table.

The following table is Nurse

```
CREATE TABLE Nurse(
Code                        VARCHAR(9),
PRIMARY KEY (Code),
FOREIGN KEY (Code)          REFERENCES Employee (Code)
);
```

- Code attribute is referenced from the Employee table, so it has the length of 9 as well.
- PRIMARY KEY is Code, because it is the only attribute.
- FOREIGN KEY is Code, referenced from Employee table.

The following table is Department

```
CREATE TABLE Department(
Code                        VARCHAR(9),
Title                       VARCHAR(40)     NOT NULL,
Dean                        VARCHAR(9) ,
PRIMARY KEY (Code),
FOREIGN KEY (Dean)          REFERENCES Doctor(Code)
);
```

- Code here is the Department code which is the unique attributes so it should have the length of 9 and string data type.
- Title is the Department's title, so it should use the string data type to store this kind of information, the optimal length we choose is 40 and must not be NULL.

- Dean is the Doctor who is in charge of each Department, so it is related to the Doctor table, also it has the length of 9 and string data type.
- PRIMARY KEY is Code, because we can only differentiate Departments by unique codes with the length of 9.
- FOREIGN KEY is Code, referenced from Doctor table.

Next up is some SQL code for fixing some constraints that may create error when not declared.

```
ALTER TABLE Employee ADD Department VARCHAR(9) DEFAULT NULL;
ALTER TABLE Employee ADD FOREIGN KEY (department) REFERENCES Department(code);
```

Because the Employee table was pre-defined, but previously we did not define the attribute Department and the FOREIGN KEY Department. So now we use ALTER command in order to add the remaining constraints to the Employee table.
Next up is the Patient table,

```
CREATE TABLE Patient(
Code                        VARCHAR(11),
First_Name                  VARCHAR(40)     NOT NULL,
Last_Name                   VARCHAR(40)     NOT NULL,
DoB                         DATE,
Gender                      CHAR,
Address                     VARCHAR(40),
PRIMARY KEY (Code)
);
```

- Code is the unique code that is generated for each patient to the Hospital, so it has the data type of string and from the example code, it indicated that the length is 11, respectively.
- FirstName, LastName, DoB, Gender, Address are the same as the attributes of Employee table.
- PRIMARY KEY is defined as Code because it is the only attribute that differs patients to patients.

The following table is Out_Patient, mostly inherited from the Patient table

```
CREATE TABLE Out_Patient(
Code                        VARCHAR(11),
Exam_Doctor                 VARCHAR(9) DEFAULT NULL,
PRIMARY KEY (Code),
FOREIGN KEY (Exam_Doctor)   REFERENCES Doctor(Code),
FOREIGN KEY (Code)          REFERENCES Patient(Code)
);
```

- Code is the attribute specialized from the Patient table, so it will has the properties of Code attribute which is the string data type with length of 11.

- Exam_Doctor is the attribute inherited from Doctor table, so it is the same as the one in the Doctor table.
- PRIMARY KEY here is the Code because it differs the Patients by its Code.
- Exam_Doctor and Code are referenced from Doctor(Code) and Patient(Code) table respectively.

The following table is the In_Patient table

```
CREATE TABLE In_Patient(
Code                        VARCHAR(11),
SickRoom                    VARCHAR(9),
Date_of_Admission           DATE            NOT NULL,
Date_of_Discharge           DATE            DEFAULT NULL,
Diagnosis                   VARCHAR(40)     DEFAULT NULL,
Fee                         DECIMAL(10,2)   DEFAULT NULL,
Nurse_Code                  VARCHAR(9)      DEFAULT NULL,
PRIMARY KEY (Code),
FOREIGN KEY (Nurse_Code)    REFERENCES Nurse(Code),
FOREIGN KEY (Code)          REFERENCES Patient(Code)
);
```

- Code is the attribute specialized from the Patient table, so it will has the properties of Code attribute which is the string data type with length of 11.
- SickRoom is the string of length 9 to represent the room of the In_Patient.
- Date_Of_Admission and Date_Of_Discharge are the two date type that represent the date. Also they are defined with NOT NULL and NULL respectively.
- Diagnosis is the string with length of 40 to briefly describe the patient's symptoms, disease,.. and left DEFAULT NULL because some diagnosis may not find any diseases.
- Fee is represent by the decimal number with 2 digits after the decimal point, which accurately describe the cost, additionally, it is defined as DEFAULT NULL because some may not have any fees.
- Nurse_Code is the string literals that has the length of 9, referenced from the Nurse(Code) table.
- PRIMARY KEY here is the Code because it differs the Patients by its Code.
- Nurse_Code and Code are referenced from Nurse(Code) and Patient(Code) table respectively.

Following up is the Patient_Phone attribute,

```
CREATE TABLE Patient_Phone(
Code                        VARCHAR(11)     NOT NULL,
Phone_Num                   VARCHAR(15),
PRIMARY KEY (Code, Phone_Num),
FOREIGN KEY (Code)          REFERENCES Patient(Code)
);
```

- `Code` is the attribute specialized from the `Patient` table, so it will has the properties of `Code` attribute which is the string data type with length of 11.
- `PRIMARY KEY` here is the `Code` because it differs the Patients by its Code.
- `Code` is also referenced from the table `Patient(Code)`.

The next table is `Medication`,

```
CREATE TABLE Medication(
Code                        VARCHAR(9),
Name                        VARCHAR(20)     NOT NULL,
Price                       DECIMAL(5,2)    NOT NULL,
Effect                      VARCHAR(100)    NOT NULL,
Expiration_Date             DATE            NOT NULL,
PRIMARY KEY (Code)
);
```

- `Code` is determined uniquely by each type of medicine, so it has the data type of string, and the optimal value for the length is 9.
- `Name` is the name for each medication, so it is a string with length of 20, it must not be `NULL` because every medication has its name.
- `Price` is the price for each medication, so it is a decimal number ranging from 0.xx to 999.xx with maximum of 2 digits after the precision dot. Also, it must not be `NULL` because the price is set for every type of medication.
- `Effect` is the effect of each particular medication, so it is a string of length 100 to describe its effect in details. Also, the effect must not be `NULL` for doctors to read cautions.
- `Expiration_Date` is date data type to store date of expiration. Also, it must must not be `NULL` because the date has to be included with medication.
- `PRIMARY KEY` here is the `Code` because it differs the Medication by its Code.

The following table is for Import_Medication, which inherits from Medication,

```
CREATE TABLE Import_Medication(
Code                        VARCHAR(9),
Quantity                    INT,
Import_Date                 DATE            NOT NULL,
PRIMARY KEY(Code),
FOREIGN KEY (Code) REFERENCES Medication(Code)
);
```

- Code is from the table Medication so it has all properties of Medication(Code)
- Quantity is an integer represent the quantity.
- Import_Date is date data type to store import date and must declare NOT NULL to track the expiry date.
- PRIMARY KEY here is the Code because it differs the Medication by its Code.
- FOREIGN KEY is the Code that references from Medication table.

Next, we come up with Examination table.

```
CREATE TABLE Examination(
Patient VARCHAR(11),
Doctor VARCHAR(9) ,
Medication VARCHAR(9),
DateExam Date,
NextExam Date DEFAULT NULL,
Fee DECIMAL(10,2),
Diagnosis VARCHAR(40),
PRIMARY KEY (Patient, Doctor, Medication, DateExam),
FOREIGN KEY (Medication) REFERENCES Medication(Code),
FOREIGN KEY (Doctor) REFERENCES Doctor(Code),
FOREIGN KEY (Patient) REFERENCES Out_Patient(Code)
);
```

- Patient is from the table Out_Patient so it has all properties of Out_Patient(Code)
- Doctor is from the table Doctor so it has all properties of Doctor(Code)
- Medication is from the table Medication so it has all properties of Medication(Code)
- Fee is represent by the decimal number with 2 digits after the decimal point, which accurately describe the cost.
- DateExam is date data type to store date of the examination that can accurately describe the date.
- NextExam is date data type to store next examination date and defined as DEFAULT NULL because some next examinations are not scheduled.
- PRIMARY KEY here is the Patient, Doctor, Medication, DateExam because it differs the Examination by these attributes.
- Patient, Doctor and Medication are referenced from Out_Patient(Code), Doctor(Code) and Medication(Code) table respectively.

Following is the Treatment table.

```
CREATE TABLE Treatment (
Patient VARCHAR(11),
Doctor VARCHAR(9),
Medication VARCHAR(9),
StartDate Date,
EndDate Date,
Result VARCHAR(40),
Status VARCHAR(40),
PRIMARY KEY (Patient, Doctor, StartDate, EndDate, Medication),
FOREIGN KEY (Medication) REFERENCES Medication(Code),
FOREIGN KEY (Doctor) REFERENCES Doctor(Code),
FOREIGN KEY (Patient) REFERENCES In_Patient(Code)
);
```

- `Patient` is from the table `In_Patient` so it has all properties of `In_Patient(Code)`
- `Doctor` is from the table `Doctor` so it has all properties of `Doctor(Code)`
- `Medication` is from the table `Medication` so it has all properties of `Medication(Code)`
- `StartDate` and `EndDate` is date data type to store start and end date of the treatment.
- `Results` and `Status` are declared as `VARCHAR(40)` to store results of the treatment and status of patients.
- `PRIMARY KEY` here is the `Patient, Doctor, StartDate, EndDate, Medication` because it differs the Treatment by these attributes.
- `Patient`, `Doctor` and `Medication` are referenced from `In_Patient(Code)`, `Doctor(Code)` and `Medication(Code)` table respectively.

Next up is the `Provide` table.

```
CREATE TABLE PROVIDER (
Provider_Number INT ,
Provider_Name VARCHAR(40) NOT NULL,
Address VARCHAR(40),
Phone  VARCHAR(15) NOT NULL,
PRIMARY KEY (Provider_Number)
);
```

- `Provider_Number` is an unique code to differ Provider that is generated with `INT` type.
- `Provider_Name` is created to store name of Provider that can not be `NULL`.
- `Address` is stored address of `Provider`
- `Phone` should be stored inside a string for easier modification. The optimal length should be 15 because some numbers can contain the prefix code of certain areas.

- `PRIMARY KEY` here is the `Provider_number` because each Provider have a unique Provider_number.

The last table is `Provide`

```
Provider INT NOT NULL,
Medication VARCHAR(9)NOT NULL,
PRIMARY KEY (Provider, Medication),
FOREIGN KEY (Provider) REFERENCES Provider(Provider_Number),
FOREIGN KEY (Medication) REFERENCES Medication(Code)
```

- `Provider` is from the table `Provider` so it has all properties of `Provider(Provider_Number)` and it can't be `NULL` beacause if it's NULL, there will be no transaction or providing.
- `Medication` is from the table `Medication` so it has all properties of `Medication(Code)` and it can't be `NULL` for the same reason above.
- `PRIMARY KEY` here is the `Provider, Medication` because it differs each Provide by these attributes.
- `Provider` and `Medication` are referenced from `Provider(Provider_Number)` and `Medication(Code)` table respectively.

CONTRAINTS

Finally, we add some contraints to our implementation.

In the database description, there is a criteria for a doctor to be the dean of a department.

> The dean must hold a specific speciality and has had more than 5 years of experience since the date he or she was awarded the speciality degree.

We are able to solve this overhead by using the `CHECK` statement in `SQL`.

```
-- First, we add the year of the doctor to the department
ALTER TABLE DEPARTMENT ADD deanYear int;
-- We use ALTER statement to add a constraint
ALTER TABLE DEPARTMENT ADD CONSTRAINT checkYEAR CHECK (deanyear > 5);
```

Next, the hospital wish to manage the patient information easily.

> If one is an outpatient, the unique code for him or her starts with "OP," which is then followed by 9 digits. If one is an inpatient, the unique code for him or her starts with "IP," which is then followed by 9 digits.

We can circumvent this request by making use of the `LIKE` and `WILDCARD` statement in SQL. And similarly, we must add a `CHECK` statement to each table.

```
-- ADD constraints of patient
ALTER TABLE patient ADD CONSTRAINT checkID CHECK (code LIKE 'OP%' OR
            code LIKE 'IP%');
-- ADD constraint of Out_Patient
ALTER TABLE Out_Patient ADD CONSTRAINT checkID CHECK (code LIKE 'OP%');
-- ADD constraint of In_Patient
ALTER TABLE In_Patient ADD CONSTRAINT checkID CHECK (code LIKE 'IP%');
```

Last, we want to modify attribute `Status` in `Treatment` table if patient's status change.

```
ALTER TABLE Treatment ADD Status VARCHAR (40);
```

1.3 Insert Data

Physical Data for **EMPLOYEE** table:

Code	FirstName	LastName	DOB	Gender	Address	StartDate	Specialty	YearOfDegree	Department
1	John	Smith	1964-09-30	Male	731 Fondren, Houston, TX	1994-07-24	Dermatologists	27	1
2	Marry	Johnson	1981-02-05	Female	638 Voss, Houston, TX	2013-09-03	Ophthalmologists	9	1
3	Mason	Williams	1972-06-15	Male	3321 Castle, Spring, TX	2001-01-30	Cardiologists	20	5
4	Joe	Brown	1973-09-07	Male	291 Berry, Bellaire, TX	2003-08-27	Psychiatrists	15	3
5	Kevin	Jones	1973-01-20	Male	975 Fire Oak, Humble, TX	2000-02-29	Oncologists	15	2
6	Max	Garcia	1985-04-15	Male	5631 Rice, Houston, TX	2010-11-05	Endocrinologists	5	1
7	Clay	Miller	1988-08-18	Male	980 Dallas, Houston, TX	2010-05-13	Oncologists	3	2
8	Hanna	Davis	1983-03-19	Female	450 Stone, Houston, TX	2015-06-19	Cardiologists	6	5
9	Nguyen	Van A	1985-08-26	Male	41 Alton Street Tulare, CA	2020-10-23	Gastroenterologists	5	2
10	Li	Fei Fei	1980-09-04	Female	7 E. State St. Los Angeles, CA	2015-10-26	Neurologists	10	4
11	Kim	Jong Un	1985-02-02	Male	80 Railroad Court Palmetto, FL	2012-07-11	Allergists	3	3
12	Jenny	Moore	1976-02-11	Female	918 Manchester Ave. Hialeah, FL	1997-09-23	Endocrinologists	24	1

SQL code :

```
1 INSERT INTO employee VALUES ('1','John',
  ↪ 'Smith',TO_DATE('1964-09-30','YYYY-MM-DD'),'M', '731 Fondren, Houston,TX',TO_DATE
  ↪ ('1994-07-24','YYYY-MM-DD'),'Dermatologists',27,'1');
2 INSERT INTO employee VALUES ('2','Marry',
  ↪ 'Johnson',TO_DATE('1981-02-05','YYYY-MM-DD'),'F', '638 Voss, Houston,TX',TO_DATE(
  ↪ '2013-09-03','YYYY-MM-DD'),'Ophthalmologists',9,'1');
3 INSERT INTO employee VALUES ('3','Mason',
  ↪ 'Williams',TO_DATE('1972-06-15','YYYY-MM-DD'),'M', '3321 Castle, Spring,TX',TO_DA
  ↪ TE('2001-01-30','YYYY-MM-DD'),'Cardiologists',20,'5');
4 INSERT INTO employee VALUES ('4','Joe',
  ↪ 'Brown',TO_DATE('1973-09-07','YYYY-MM-DD'),'M', '291 Berry, Bellaire,TX',TO_DATE(
  ↪ '2003-08-27','YYYY-MM-DD'),'Psychiatrists',15,'3');
5 INSERT INTO employee VALUES ('5','Kevin',
  ↪ 'Jones',TO_DATE('1973-01-20','YYYY-MM-DD'),'M', '975 Fire Oak, Humble,TX',TO_DATE
  ↪ ('2000-02-29','YYYY-MM-DD'),'Oncologists',15,'2');
6 INSERT INTO employee VALUES ('6','Max',
  ↪ 'Gracia',TO_DATE('1985-04-15','YYYY-MM-DD'),'M', '5631 Rice, Houston,TX',TO_DATE(
  ↪ '2010-11-05','YYYY-MM-DD'),'Endocrinologists',5,'1');
7 INSERT INTO employee VALUES ('7','Clay',
  ↪ 'Miller',TO_DATE('1988-08-18','YYYY-MM-DD'),'M',
  ↪ '980 Dallas, Houston,TX',TO_DATE('2010-05-13','YYYY-MM-DD'),'Oncologists',3,'2');
```

```
INSERT INTO employee VALUES ('8','Hanna',
↪  'Davis',TO_DATE('1983-03-19','YYYY-MM-DD'),'F',
↪  '450 Stone, Houston,TX',TO_DATE('2015-06-19','YYYY-MM-DD'),'Cardiologists',6,'5');
INSERT INTO employee VALUES ('9','Nguyen',
↪  'Van A',TO_DATE('1985-08-26','YYYY-MM-DD'),'M', '41 Alton Street Tulare,CA',TO_DA
↪  TE('2020-10-23','YYYY-MM-DD'),'Gastroenterologists',5,'2');
INSERT INTO employee VALUES ('10','Li',
↪  'Fei Fei',TO_DATE('1980-09-04','YYYY-MM-DD'),'F', '7 E. State St.Los Angeles,CA',
↪  TO_DATE('2015-10-26','YYYY-MM-DD'),'Neurologists',10,'4');
INSERT INTO employee VALUES ('11','Kim',
↪  'Jong Un',TO_DATE('1985-02-02','YYYY-MM-DD'),'M', '80 Railroad Court Palmetto,FL'
↪  ,TO_DATE('2012-07-11','YYYY-MM-DD'),'Allergists',3,'3');
INSERT INTO employee VALUES ('12','Jenny',
↪  'Moore',TO_DATE('1976-02-11','YYYY-MM-DD'),'F', '918 Manchester Ave. Hialeah,FL',
↪  TO_DATE('1997-09-23','YYYY-MM-DD'),'Endocrinologists',24,'1');
```

Physical Data for **DEPARTMENT** table:

Code	Title	Dean
1	Medical	1
2	Nursing	5
3	Paramedical	4
4	Neurology	10
5	Cardiology	8

SQL code :

```
INSERT INTO DEPARTMENT(code, title) VALUES ('1', 'Medical');
INSERT INTO DEPARTMENT(code, title) VALUES ('2', 'Nursing');
INSERT INTO DEPARTMENT(code, title) VALUES ('3', 'Paramedical');
INSERT INTO DEPARTMENT(code, title) VALUES ('4', 'Neurology');
INSERT INTO DEPARTMENT(code, title) VALUES ('5', 'Cardiology');
```

Physical Data for **EMPLOYEE_PHONE**

Code	PhoneNum
1	210-200-8136
2	210-201-1414
3	239-200-5542
4	480-200-9620
4	480-201-6455
5	239-200-5542
6	239-200-5542
7	210-212-3291
8	202-201-2427
9	202-207-3655
10	210-202-9172
10	239-200-5542
11	480-220-1857
12	210-203-7035

SQL code for above table:

```sql
INSERT INTO employee_phone values ('1', '210-200-8136');
INSERT INTO employee_phone values ('2', '210-201-1414');
INSERT INTO employee_phone values ('3', '239-200-5542');
INSERT INTO employee_phone values ('4', '480-200-9620');
INSERT INTO employee_phone values ('4', '480-201-6455');
INSERT INTO employee_phone values ('5', '239-200-5542');
INSERT INTO employee_phone values ('6', '239-200-5542');
INSERT INTO employee_phone values ('7', '210-212-3291');
INSERT INTO employee_phone values ('8', '202-201-2427');
INSERT INTO employee_phone values ('9', '202-207-3655');
INSERT INTO employee_phone values ('10', '210-202-9172');
INSERT INTO employee_phone values ('10', '239-200-5542');
INSERT INTO employee_phone values ('11', '480-220-1857');
INSERT INTO employee_phone values ('12', '210-203-7035');
```

Physical Data of **DOCTOR** table:

Code
12
10
1
7
8
5
4
9

SQL code for table:

```
INSERT INTO Doctor values ('12');
INSERT INTO Doctor values ('10');
INSERT INTO Doctor values ('1');
INSERT INTO Doctor values ('7');
INSERT INTO Doctor values ('8');
INSERT INTO Doctor values ('5');
INSERT INTO Doctor values ('4');
INSERT INTO Doctor values ('9');
```

Physical Data of NURSE table:

Code
2
6
11
3

SQL code for inserting data into table:

```
INSERT INTO nurse values ('2');
INSERT INTO nurse values ('6');
INSERT INTO nurse values ('11');
INSERT INTO nurse values ('3');
```

Physical Data for **PATIENT** table:

Code	FirstName	LastName	DOB	Gender	Address
OP000000001	Zahra	Cooper	2012-02-20	Female	690 Brewery Court New York, NY
OP000000002	Shawn	Cobb	1986-07-23	Male	851 W. Vernon Ave. Arlington, TX
IP000000003	Lucie	Zhang	1978-04-17	Female	25 Bartholomew St Christmas, FL
OP000000004	Abraham	Ross	1959-08-16	Male	787 Andover St. Houston, TX
IP000000005	Paula	Nguyen	1951-02-18	Female	690 Brewery Court New York, NY
OP000000006	Mary	Wise	1947-02-23	Female	37 Gregory Lane Van Nuys, CA
OP000000007	Alice	Perkins	2004-05-03	Female	14 Shadow Brook Street Friendswood, TX
IP000000008	Rachael	Carter	1971-06-10	Female	85 Mayfair Dr. Salinas, CA
IP000000009	Todd	Chambers	1993-08-29	Male	299 Charles Drive Los Angeles, CA
OP000000010	Edwin	Ferguson	1960-03-18	Male	89 Trout St. Chino Hills, CA
IP000000011	Frank	Hamilton	1969-02-05	Male	331 Essex Ave. Riverside, CA
IP000000012	Tomas	Delay	1944-02-01	Male	84 Illinois Avenue El Paso, TX
IP000000013	Carlos	Moore	1972-12-21	Male	67 Church St. Oceanside, CA
IP000000014	Lukas	Foden	2008-01-28	Male	3 Newbridge Ave. Bronx, NY

SQL data implementation:

```
INSERT INTO patient values ('OP000000001', 'Zahra', 'Cooper',
↪ TO_DATE('2012-02-20','YYYY-MM-DD'),
'F','690 Brewery Court New York, NY');
INSERT INTO patient values ('OP000000002', 'Shawn', 'Cobb',
↪ TO_DATE('1986-07-23','YYYY-MM-DD'),
'M','851 W. Vernon Ave. Arlington, TX');
INSERT INTO patient values ('IP000000003', 'Lucie', 'Zhang',
↪ TO_DATE('1978-04-17','YYYY-MM-DD'),
'F','25 Bartholomew St Christmas, FL');
INSERT INTO patient values ('OP000000004', 'Abraham', 'Ross',
↪ TO_DATE('1959-08-16','YYYY-MM-DD'),
'M','787 Andover St. Houston, TX');
INSERT INTO patient values ('IP000000005', 'Paula', 'Nguyen',
↪ TO_DATE('1951-02-18','YYYY-MM-DD'),
'F','690 Brewery Court New York, NY');
```

```
INSERT INTO patient values ('OP000000006', 'Mary', 'Wise',
    TO_DATE('1947-02-23','YYYY-MM-DD'),
'F','37 Gregory Lane Van Nuys, CA');
INSERT INTO patient values ('OP000000007', 'Alice', 'Perkins',
    TO_DATE('2004-05-03','YYYY-MM-DD'),
'F','14 Shadow Brook Street Friendswood, TX');
INSERT INTO patient values ('IP000000008', 'Rachael', 'Carter',
    TO_DATE('1971-06-10','YYYY-MM-DD'),
'F','85 Mayfair Dr. Salinas, CA');
INSERT INTO patient values ('IP000000009', 'Todd', 'Chambers',
    TO_DATE('1993-08-29','YYYY-MM-DD'),
'M','299 Charles Drive Los Angeles, CA');
INSERT INTO patient values ('OP000000010', 'Edwin', 'Ferguson',
    TO_DATE('1960-03-18','YYYY-MM-DD'),
'M','89 Trout St. Chino Hills, CA ');
INSERT INTO patient values ('IP000000011', 'Tomas', 'Delay',
    TO_DATE('1944-02-01','YYYY-MM-DD'),
'M','84 Illinois Avenue El Paso, TX');
INSERT INTO patient values ('IP000000012', 'Frank', 'Hamilton',
    TO_DATE('1969-02-05','YYYY-MM-DD'),
'M','331 Essex Ave. Riverside, CA');
INSERT INTO patient values ('IP000000013', 'Carlos', 'Moore',
    TO_DATE('1972-12-21','YYYY-MM-DD'),
'M','67 Church St. Oceanside, CA');
INSERT INTO patient values ('IP000000014', 'Lukas', 'Foden',
    TO_DATE('2008-01-28','YYYY-MM-DD'),
'M','3 Newbridge Ave. Bronx, NY');
```

Physical Data of **PATIENT_PHONE** table:

Patient	Phone
IP000000005	210-200-3675
IP000000005	210-201-0574
IP000000014	209-200-4254
IP000000011	209-202-7537
IP000000009	210-204-8792

SQL data implementation:

```
INSERT INTO patient_phone VALUES ('IP000000005', '210-200-3675');
INSERT INTO patient_phone VALUES ('IP000000005', '210-201-0574');
INSERT INTO patient_phone VALUES ('IP000000014', '209-200-4254');
INSERT INTO patient_phone VALUES ('IP000000011', '209-202-7537');
INSERT INTO patient_phone VALUES ('IP000000009', '210-204-8792');
```

Physical Data of **OUT_PATIENT** table:

Code	Exam_doctor
OP000000001	9
OP000000002	5
OP000000004	12
OP000000006	10
OP000000007	7
OP000000010	1

SQL data implementation:

```
INSERT INTO out_patient VALUES ('OP000000001','9');
INSERT INTO out_patient VALUES ('OP000000002','5');
INSERT INTO out_patient VALUES ('OP000000004','12');
INSERT INTO out_patient VALUES ('OP000000006','10');
INSERT INTO out_patient VALUES ('OP000000007','7');
INSERT INTO out_patient VALUES ('OP000000010','1');
```

Physical Data of **IN_PATIENT** table:

Code	SickRoom	DateAdmisstion	DateDischarge	Diagnosis	Fee	Nurse_Code
IP000000003	201	2021-08-26	NULL	Liver cancer	1467.3	6
IP000000005	110	2018-01-23	2018-01-29	Tuberculosis	346	11
IP000000008	220	2020-12-26	2021-01-23	Blood infection	2436.4	2
IP000000009	210	2017-04-05	2017-05-02	Appendix operation	592	3
IP000000011	201	2019-11-15	NULL	Brain cancer	5689.2	3
IP000000012	309	2021-05-23	2021-06-07	Cataract	1263.4	11
IP000000013	411	2019-05-01	NULL	Blood cancer	6812.92	2
IP000000014	113	2021-10-11	NULL	Covid-19	45	6

SQL data implementation:

```
INSERT INTO in_patient VALUES ('IP000000003', '201',
↪  TO_DATE('2021-08-26','YYYY-MM-DD'),NULL, 'Liver cancer', 1467.3, '6');
INSERT INTO in_patient VALUES ('IP000000005', '110',
↪  TO_DATE('2018-01-23','YYYY-MM-DD'),TO_DATE('2018-01-29','YYYY-MM-DD'),
'Tuberculosis', 246, '11');
INSERT INTO in_patient VALUES ('IP000000008', '220',
↪  TO_DATE('2020-12-26','YYYY-MM-DD'), TO_DATE('2021-01-23','YYYY-MM-DD'),
'Blool infection', 2436.4, '2');
INSERT INTO in_patient VALUES ('IP000000009', '210',
↪  TO_DATE('2017-04-05','YYYY-MM-DD'),TO_DATE('2017-05-02','YYYY-MM-DD') ,
'Appendix operation', 592, '3');
INSERT INTO in_patient VALUES ('IP000000011', '201',
↪  TO_DATE('2019-11-15','YYYY-MM-DD'),NULL,
```

```
'Brain cancer', 5689.2, '3');
INSERT INTO in_patient VALUES ('IP0000000012', '309',
↪  TO_DATE('2021-05-23','YYYY-MM-DD'),TO_DATE('2021-06-07','YYYY-MM-DD'),
'Cataract', 1263.4, '11');
INSERT INTO in_patient VALUES ('IP0000000013', '411',
↪  TO_DATE('2019-05-01','YYYY-MM-DD'),NULL,
'Blood cancer', 5689.2, '2');
INSERT INTO in_patient VALUES ('IP0000000014', '113',
↪  TO_DATE('2021-10-11','YYYY-MM-DD'),NULL,
'Covid 19', 45, '6');
```

Physical Data of **TREATMENT** table:

Patient	Doctor	Medication	Startdate	Enddate	Result	Status
IP000000003	9	5	2021-08-26	2021-08-30	radiotherapy	Not recovered
IP000000005	12	7	2018-01-25	2012-01-29	Success	Recovered
IP000000008	10	2	2020-12-27	2021-01-20	Changed blood	Recovered
IP000000009	9	4	2017-04-06	2017-05-01	Success	Recovered
IP000000011	1	8	2019-11-16	2019-11-20	Chemotherapy	Not recovered
IP000000012	9	6	2021-05-24	2021-06-05	Success	Recovered
IP000000013	7	2	2019-05-02	2019-05-10	Changed blood	Not recovered
IP000000014	8	1	2021-10-13	2021-10-20	Positive	Not recovered
IP000000003	5	3	2021-08-10	2021-08-15	Chemotherapy	Not recovered
IP000000003	4	4	2021-08-10	2021-08-15	Chemotherapy	Not recovered
IP000000013	1	5	2019-05-20	2019-05-29	Hospitalization	Not recovered
IP000000011	12	7	2019-11-25	2019-12-01	Operation	Not recovered

SQL data implementation:

```
INSERT INTO TREATMENT VALUES ('IP0000000003', '9',
↪  '5',TO_DATE('2021-08-26','YYYY-MM-DD'),TO_DATE('2021-08-30','YYYY-MM-DD'),
'Radiotherapy', 'Not recovered');
INSERT INTO TREATMENT VALUES ('IP0000000005', '12',
↪  '7',TO_DATE('2018-01-25','YYYY-MM-DD'),TO_DATE('2018-01-29','YYYY-MM-DD'),
'Success', 'Recovered');
INSERT INTO TREATMENT VALUES ('IP0000000008', '10',
↪  '2',TO_DATE('2020-12-27','YYYY-MM-DD'),TO_DATE('2021-01-20','YYYY-MM-DD'),
'Changed blood', 'Recovered');
INSERT INTO TREATMENT VALUES ('IP0000000009', '9',
↪  '4',TO_DATE('2017-04-06','YYYY-MM-DD'),TO_DATE('2017-05-01','YYYY-MM-DD'),
'Success', 'Recovered');
INSERT INTO TREATMENT VALUES ('IP0000000011', '1',
↪  '8',TO_DATE('2019-11-16','YYYY-MM-DD'),TO_DATE('2019-11-20','YYYY-MM-DD'),
'Chemotherapy', 'Not recovered');
INSERT INTO TREATMENT VALUES ('IP0000000012', '9',
↪  '6',TO_DATE('2021-05-24','YYYY-MM-DD'),TO_DATE('2021-06-05','YYYY-MM-DD'),
'Success', 'Recovered');
```

```
INSERT INTO TREATMENT VALUES ('IP000000013', '7',
↪  '2',TO_DATE('2019-05-02','YYYY-MM-DD'),TO_DATE('2019-05-10','YYYY-MM-DD'),
'Changed blood', 'Not recovered');
INSERT INTO TREATMENT VALUES ('IP000000014', '8',
↪  '1',TO_DATE('2021-10-13','YYYY-MM-DD'),TO_DATE('2021-10-20','YYYY-MM-DD'),
'Positive', 'Not recovered');
INSERT INTO TREATMENT VALUES ('IP000000003', '5',
↪  '3',TO_DATE('2021-08-10','YYYY-MM-DD'),TO_DATE('2021-08-15','YYYY-MM-DD'),
'Chemotherapy', 'Not recovered');
INSERT INTO TREATMENT VALUES ('IP000000003', '4',
↪  '4',TO_DATE('2021-08-10','YYYY-MM-DD'),TO_DATE('2021-08-15','YYYY-MM-DD'),
'Chemotherapy', 'Not recovered');
INSERT INTO TREATMENT VALUES ('IP000000013', '1',
↪  '5',TO_DATE('2019-05-20','YYYY-MM-DD'),TO_DATE('2019-05-29','YYYY-MM-DD'),
'Hospitalization', 'Not recovered');
INSERT INTO TREATMENT VALUES ('IP000000011', '12',
↪  '7',TO_DATE('2019-11-25','YYYY-MM-DD'),TO_DATE('2019-12-01','YYYY-MM-DD'),
'Operation', 'Not recovered');
```

Physical Data of **MEDICATION** table:

Code	Name	Price	Effect	ExpirationDate
1	Losartan	6	Treat high blood pressure, reduce the risk of stroke	2023-09-29
2	Metformin	4	control blood sugar levels in type 2 diabetes	2022-06-12
3	Omeprazole	5.1	Reduce acid in the stomach	2021-12-15
4	Gabapentin	7.69	Treat seizures in adults with epilepsy and nerve pain	2023-02-09
5	Amlodipine	3.9	Treat high blood pressure and chest pain	2021-11-23
6	Synthroid	3.3	Treat thyroid hormone deficiency	2022-02-14
7	Atorvastatin	3.3	Treat high cholesterol	2024-05-07
8	Albuterol	19.05	prevent bronchospasm	2023-10-09

SQL data implementation:

```
INSERT INTO medication VALUES ('1','Losartan', 6,
↪  'Treat high blood pressure, reduce the risk of stroke',
TO_DATE('2023-09-29','YYYY-MM-DD'));
INSERT INTO medication VALUES ('2','Metformin', 4,
↪  'Control blood sugar levels in type 2 diabetes',
TO_DATE('2022-06-12','YYYY-MM-DD'));
INSERT INTO medication VALUES ('3','Omeprazole', 5.1, 'Reduce acid in the stomach',
TO_DATE('2021-12-15','YYYY-MM-DD'));
INSERT INTO medication VALUES ('4','Gabapentin', 7.69,
↪  'Treat seizures in adults with epilepsy and nerve pain',
TO_DATE('2023-02-09','YYYY-MM-DD'));
INSERT INTO medication VALUES ('5','Amlodipine', 3.9,
↪  'Treat high blood pressure and chest pain',
TO_DATE('2021-11-23','YYYY-MM-DD'));
```

```
INSERT INTO medication VALUES ('6','Synthroid', 3.3,
↪  'Treat thyroid hormone deficiency',
TO_DATE('2022-02-14','YYYY-MM-DD'));
INSERT INTO medication VALUES ('7','Atorvastatin', 3.3, 'Treat high cholesterol',
TO_DATE('2024-05-07','YYYY-MM-DD'));
INSERT INTO medication VALUES ('8','Albuterol ', 19.05, 'Prevent bronchospasm',
TO_DATE('2023-10-09','YYYY-MM-DD'));
```

Physical Data of **IMPORT_MEDICATION** table:

Code	Quantity	Imported Date
3	80	2020-12-03
5	101	2021-06-12
6	23	2019-11-01
7	267	2021-06-28
8	56	2020-10-23

SQL data implementation:

```
INSERT INTO import_medication VALUES ('3', 80, TO_DATE('2020-12-03','YYYY-MM-DD'));
INSERT INTO import_medication VALUES ('5', 101, TO_DATE('2021-06-12','YYYY-MM-DD'));
INSERT INTO import_medication VALUES ('6', 23, TO_DATE('2019-11-01','YYYY-MM-DD'));
INSERT INTO import_medication VALUES ('7', 267, TO_DATE('2021-06-28','YYYY-MM-DD'));
INSERT INTO import_medication VALUES ('8', 56, TO_DATE('2020-10-23','YYYY-MM-DD'));
```

Physical Data of **PROVIDER** table:

Number	Name	Address	Phone
1	Biogen	491 Calle Dia, Carpinteria, CA	(860) 742-1601
2	Par Pharmaceutical	Po Box 265 San Dimas, CA	(866) 462-2448
3	Radius Health	2102 Oakwood St, Haltom City, TX	(817) 834-4606
4	Sanofi	2723 W Briarcliffe Ln Peoria,IL	(309) 839-2462
5	Tagi Pharma	440 W Windsor St Montpelier, IN	(765) 728-3154

SQL data implementation:

```
INSERT INTO PROVIDER VALUES ('1','Biogen', '491 Calle Dia Carpinteria, CA',
↪  '(860) 742-1601');
INSERT INTO PROVIDER VALUES ('2','Par Pharmaceutical', 'Po Box 265 San Dimas, CA',
↪  '(866) 462-2448');
INSERT INTO PROVIDER VALUES ('3','Radius Health', '2102 Oakwood St Haltom City, TX',
↪  '(817) 834-4606');
INSERT INTO PROVIDER VALUES ('4','Sanofi', '2723 W Briarcliffe Ln Peoria,IL',
↪  '(309) 839-2462');
INSERT INTO PROVIDER VALUES ('5','Tagi Pharma', '440 W Windsor St Montpelier, IN',
↪  '(765) 728-3154');
```

Physical Data of **PROVIDE** table:

Number	Name	Address	Phone
1	Biogen	491 Calle Dia, Carpinteria, CA	(860) 742-1601
2	Par Pharmaceutical	Po Box 265 San Dimas, CA	(866) 462-2448
3	Radius Health	2102 Oakwood St, Haltom City, TX	(817) 834-4606
4	Sanofi	2723 W Briarcliffe Ln Peoria,IL	(309) 839-2462
5	Tagi Pharma	440 W Windsor St Montpelier, IN	(765) 728-3154

SQL data implementation:

```
INSERT INTO PROVIDE VALUES ('1','2');
INSERT INTO PROVIDE VALUES ('3','3');
INSERT INTO PROVIDE VALUES ('5','4');
INSERT INTO PROVIDE VALUES ('1','5');
INSERT INTO PROVIDE VALUES ('2','6');
INSERT INTO PROVIDE VALUES ('2','7');
INSERT INTO PROVIDE VALUES ('3','8');
INSERT INTO PROVIDE VALUES ('4','1');
```

Physical Data of **EXAMINATION** table:

Patient	Doctor	Medication	DateExam	NextExam	Fee	Diagnosis
OP000000001	9	8	2016-05-13	NULL	85.6	Common cold
OP000000002	12	5	2019-01-24	NULL	43.2	Influenza
OP000000004	10	7	2021-02-28	NULL	134.76	pneumonia
OP000000006	1	2	2020-12-03	2021-06-02	243.6	Hepatitis
OP000000007	7	4	2019-03-05	NULL	25.6	diarrhea
OP000000010	8	1	2020-07-28	NULL	30.8	constipation
OP000000006	5	6	2021-06-02	NULL	154.8	Hepatitis
OP000000004	10	3	2021-02-28	NULL	134.76	pneumonia

SQL data implementation:

```
INSERT INTO examination VALUES ('OP000000001', '9',
↪  '8',TO_DATE('2016-05-13','YYYY-MM-DD'), NULL,85.6, 'Common cold');
INSERT INTO examination VALUES ('OP000000002', '12',
↪  '5',TO_DATE('2019-01-24','YYYY-MM-DD'), NULL,43.2, 'Influenza');
INSERT INTO examination VALUES ('OP000000004', '10',
↪  '7',TO_DATE('2021-02-28','YYYY-MM-DD'), NULL,134.76, 'Pneumonia');
INSERT INTO examination VALUES ('OP000000006', '1',
↪  '2',TO_DATE('2020-12-03','YYYY-MM-DD'), TO_DATE('2021-06-02','YYYY-MM-DD'),243.6,
↪  'Hepatitis');
```

```
INSERT INTO examination VALUES ('OP000000007', '7', '4',TO_DATE('2019-03-05','YYYY-MM-DD'), NULL,25.6, 'Diarrhea');
INSERT INTO examination VALUES ('OP000000010', '8', '1',TO_DATE('2020-07-28','YYYY-MM-DD'), NULL,30.8, 'Constipation');
INSERT INTO examination VALUES ('OP000000006', '5', '6',TO_DATE('2021-06-02','YYYY-MM-DD'), NULL,154.8 ,'Hepatitis');
INSERT INTO examination VALUES ('OP000000004', '10', '3',TO_DATE('2021-02-28','YYYY-MM-DD'), NULL,134.76, 'pneumonia');
```

2 Store Procedure

2.1 Question a

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

For this task, we will come over 2 steps:

+Step 1: Retrieve the Inpatients whose date_of_admission is over 01/09/2020 and their date_of_discharge are NULL, which mean these Inpatients are not recovered yet.
+Step 2: We will update fee of these Inpatients for 110%.

```
UPDATE in_patient
SET fee = fee*1.1
WHERE date_of_admission > TO_DATE('2020-09-01', 'YYYY-MM-DD')
AND date_of_discharge IS NULL;
```

2.2 Question b

b) Select all the patients (outpatient and inpatient) of the doctor named 'Nguyen Van A'

For this task, we will go through two steps:
+Step 1: Retrieve the examinations and treatments that doctor Nguyen Van A is involved in.
+Step 2: We will get the patient information from all the examinations and treatment in step 1.

```
SELECT* from patient;
WITH  ALL_Patient(patient_ID) AS
(
SELECT treatment.patient
FROM (EMPLOYEE INNER JOIN TREATMENT ON EMPLOYEE.CODE = treatment.doctor)
where employee.firstname = 'Nguyen' AND employee.lastname = 'Van A'
UNION
SELECT examination.patient
```

```
FROM(EMPLOYEE INNER JOIN examination ON EMPLOYEE.CODE = examination.doctor)
where employee.firstname = 'Nguyen' AND employee.lastname = 'Van A'
)
SELECT *
FROM (ALL_patient INNER JOIN patient ON ALL_patient.patient_ID =
                                                    patient.code);
```

2.3 Question c

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

The SQL logic to solve this question is pretty straight forward, a simple SELECT statement should do the trick, however, the challenging part is how to incorporate that logic inside a SQL function and have that function return what we want, which in this case is a list of payment for each treatment or examination. Since a function in SQL can only return a predefined type, in order to get the function to return a list, we need to define a new type that contain all the attributes that we need.

```
CREATE OR REPLACE TYPE row_record AS OBJECT
(
PatientID VARCHAR(11),
DoctorID VARCHAR(9),
MedicationID VARCHAR(9),
MedicationName VARCHAR(20),
Price DECIMAL(5,2),
Start_date Date,
End_date Date
);
CREATE OR REPLACE TYPE record_table AS TABLE OF row_record;
```

In this question, we want to return a table that has the required attributes such as PatientID, DoctorID, MedicationID, MedicationName, Price, Start date and End date and in order to achieve this, we will create two new types with the *"CREATE OR REPLACE TYPE"* syntax. The first type is *"row record"*, which is created as an object to hold all the attributes; and the second one *"record table"*, which is a table that hold all the rows of the type *"row record"*. Even thought the input might be an outpatient who does not have a end date attribute, we still include it in our result table and the way we handle the situation will be explained here shortly.

```
CREATE OR REPLACE FUNCTION total_medication
(
PatientID IN patient.code%TYPE
)
RETURN record_table
IS
FinalResult record_table:=record_table();
```

```
CURSOR c_examination IS
(
SELECT Patient, doctor, medication,medication.name, price,
                                                    examination.dateexam
FROM examination JOIN medication ON examination.medication =
                                                        medication.code
WHERE examination.patient = PatientID;
)
CURSOR c_treatment IS
(
SELECT Patient, doctor, medication, medication.name ,price,
                                treatment.startdate, treatment.enddate
FROM Treatment JOIN medication ON treatment.medication =
                                                    medication.code
WHERE treatment.patient = PatientID;
)
```

The next step is to create the function, define the parameters, declare all the variables that is going to be used inside the function. We have declared three important variables in this code, including the *"FinalResult"* variable , which will be returned by the function, and the two cursors *"c examination"* and *"c treatment"*, which are placeholder for all the examination or treatment that the patient is involved in, respectively.

```
BEGIN
FOR r_examination  IN c_examination
    LOOP
        FinalResult.extend;
        FinalResult(FinalResult.COUNT):= row_record(r_examination.Patient,
        r_examination.doctor,
        r_examination.medication,r_examination.name,r_examination.price,
        r_examination.dateexam, NULL);
    END LOOP;
FOR r_treatment  IN c_treatment
    LOOP
        FinalResult.extend;
        FinalResult(FinalResult.COUNT):= row_record(r_treatment.Patient,
        r_treatment.doctor,
        r_treatment.medication,r_treatment.name,r_treatment.price,
        r_treatment.startdate, r_treatment.enddate);
    END LOOP;
    RETURN finalresult;
END;
```

Inside the function' body, we will invoke two for loop, the logic and function of which are identical. Each of them is used to transfer the data we got in the two above cursors to the *"FinalResult"* object to be returned by the function. The reason why there is two loops is because we so far, we have not identified the patient as in or out patient. so we need two loops, each for one of the two cases. If the patient is of the type inpatient, for

example, then the cursor corresponding to the outpatient type would be null and vice versa, there for, the for loop that use the null cursor will not be executed. In any case, only one of the two loops is run and it will copy the result from the corresponding cursor to *"FinalResult"* variable.

2.4 Question d

d) Write a procedure to sort the doctor in increasing number of patients he/she takes care in a period of time

The idea we use to solve this question is fairly simple. First, we retrieve all the examination and treatment that happened within that time interval; then, we use a single SELECT statement to join the result with the employee table, get the name of the doctors and sort them with regard to the number of patient that they are associated with. The implementation and detail explanation will be provided hereafter.

```
CREATE OR REPLACE PROCEDURE Number_Patients
( Lower_bound IN treatment.startdate%TYPE,
upper_bound IN treatment.enddate%TYPE
)
```

First, we create the procedure called *"Number Patients"* by using the "CREATE OR REPLACE PROCEDURE" syntax. Then, we define the parameters that this procedure will use, which is the start date and end date of the time interval.

```
IS
CURSOR total_work IS
 WITH Doctor_work
 AS
 (
    SELECT DISTINCT DOCTOR, Patient
    FROM examination
    WHERE examination.dateexam >= Lower_bound  AND examination.dateexam
                                                        < upper_bound
    UNION
    SELECT DISTINCT DOCTOR, Patient
    FROM Treatment
    WHERE TREATMENT.STARTDATE >= Lower_bound AND treatment.ENDdate
                                                        < upper_bound
 )
    SELECT FIRSTNAME, LASTNAME, COUNT(1) AS NUM_PATIENT
    FROM Doctor_work JOIN employee ON Doctor_work.doctor = employee.code
    GROUP BY FIRSTNAME, LASTNAME
    ORDER BY COUNT(1) ;
```

This part of the code is the implementation of the logic discussed above. In this part, we use the cursor "total work" to hold the result so that it could later on be printed out in the procedure body. In order to generate the result, we first need to create an intermediate

table "doctor work" by the union of two SELECT statement, with one statement used to get all the examination and the other for all the treatment that happened in the time interval. Notice that we use the "SELECT DISTINCT" instead of a normal SELECT, this is because there can be multiple treatments or examinations between one doctor and a patient, so by adding a DISTINCT key word, we can eliminate all the unwanted duplicate tuples, as we only care about the number of patient, not the number of treatment or examination.

After getting the intermediate result, we continue to join it with the employee table to get the name of the doctors, as well as sorting them in the correct order.

```
BEGIN
FOR r_total_work IN total_work
    LOOP
    DBMS_output.put_line(r_total_work.FIRSTNAME||' '||
    r_total_work.LASTNAME ||''||r_total_work.NUM_PATIENT);
    END LOOP;
END;
```

Inside the procedure' body, we simply run a for loop to iterate through the cursor and print out the result line by line using the DBMS output.put line command

One little notice is that in order for the $DBMS\_output.put\_line$ line to work, we need to include one line of code

```
SET SERVEROUTPUT ON;
```

This line of code only need to be invoked once, so we can include it at the start of the session and then the $DBMS\_output.put\_line$ command should work just fine.

3 Build Application

3.1 User Manual:

First we have to install the dependencies:

- Node.js
- ReactJS
- Git
- MySQL & MySQL Workbench

After installing all the dependencies, the first thing we need to do clone the repository to your local machine. The following link will redirect to our repository:

https://github.com/nhatduong01/Hospital-Management.git

To clone the remote repository to local machine, we issue the following command:

```
git clone https://github.com/nhatduong01/Hospital-Management.git
```

Unfortunately, we have two branches, and the default displaying branch is `main`, which is not the current directory for the web application, so we issue the next command:

```
git checkout master
```

We need to move to the web application directory, it should be located in the directory you issued the `clone` command. We change to the directory by:

```
cd Hospital-Management
```

Inside that folder, there is a Hospital websites folder, we will need to change our directory to that folder and run the two objects in it, which are my-hospital and server. These two folders store all the code to run our application but we will leave it there for now.

The next step is to create data in our MySQL workbench. This is necessary because our application only run on local host, so you will need to create data in your computer in order to use the application. To do that, first open MySQL workbench and create a new schema; but before that, we recommend that you set your root password to 123456 because we have predefined this root password in our code; however, you can modify that to be anything you want, just open the "index.js" file inside the the "server" folder, look for line 11 where the code is

```
password: "123456",
```

and change the 123456 to your password and things should be fine.
After creating the new schema, name it "hospital" and choose the "Schema" tab on the lower left corner of the screen. Then, run this statement

```
CREATE TABLE `hospital`.`userdata` (
  `username` VARCHAR(50) NOT NULL,
  `password` VARCHAR(45) NULL,
  PRIMARY KEY (`username`));
```

The code to create the tables and add all the data is rather lengthy, so we will put it in a github repo for convenience. You need to go to:

```
https://github.com/nhatduong01/Database-System-CO2014
```

Go to Assignment 2 and in there, you will find the 2 files "Create table for application.mysql" and "Insert data for application.mysql" that contain the code to create the database. Copy and paste to the workbench all code from the "Create table for application.mysql" first and then from the "Insert data for application.mysql"; after that just run

the script and all the data we need will be generated.

After creating the database, we comeback to the Hospital websites folder that we moved to before. You need to change the folder to server and if this is the first time that you run the project, you need to run the command
`npm install`

and then

`npm start`

and then create a new terminal and repeat these same commands on the my-hospital folder and the application should automatically pop up in your browser. After you are in the application, just follow the instruction on screen to perform the desired actions.

After the first time you run the application successfully, the following attempt is as easy as a simple

`npm start`

command in both the my-hospital and server folder.
Be aware that if you are using the latest version of Node (version 17.3.x) you might experience issues while trying to run the application, this is because of some of the default key size or algorithm changes in "OpenSSL 3.0" in the 17.3 version of Node. The error you might get while trying to run our code with Node version 17.3 is

"ERR_OSSL_EVP_UNSUPPORTED"

and if this is the case, our recommendation to fix the problem is to roll back to the long time support (LTS) version of Node, which is version 16.13.0.