Iteration 5

Reusable, Transaction-Oriented Store Procedure:

The 3rd use case for *HealthCare* is the registration and appointment use case I added in iteration 3 listed below.

Registration and Appointment Use Case (NEW)

1. Patient should register from the receptionist and file insurance form or claims and the medial reports.
2. The patient should identify if they are in-patient or out-patients.
3. The receptionist will schedule patient hospital admission, later than the appointment.
4. The patient will be assigned the doctor or nurse in an appropriate ward to go over the exam.

For this use case, I'll implement a transaction for those who are out-patient. The following is the screenshot of how my procedure simply looks like.

```
CREATE PROCEDURE AddOutPatient @PatientID DECIMAL(12), @AccountID DECIMAL(12),
  @ReceptionistID DECIMAL(12), @DoctorID DECIMAL(12), @FirstName VARCHAR(225), @LastName VARCHAR(225),
  @PatientType CHAR(1)
AS
BEGIN
  INSERT INTO Patient(PatientID, AccountID, ReceptionistID, DoctorID, FristName, LastName, PatientType)
  VALUES(@PatientID, @AccountID, @ReceptionistID, @DoctorID, @FristName, @LastName, @PatientType,
  GETDATE(), "F");
  INSERT INTO OutPatient(PatientID)
  VALUES(@PatientID);
END;
GO
```

I name the stored procedure "AddOutPatient", and give it parameters that correspond to the Patient and OutPatient tables. Since this procedure is always for a free account, I do not use a parameter for PatientType, but hardcode the character "F". Inside the stored procedure,

there are two insert statements to insert into the two respective tables. Here is a screenshot of my stored procedure execution.

```
14  BEGIN TRANSACTION AddOutPatient;
15  EXECUTE AddOutPatient 1, '243OP', 'xyzname', 'receplee', 'doctorchuang', 'xy', 'zhang', 'OutPatient';
16  COMMIT TRANSACTION AddOutPatient;
```

I made up the name for patient with first name xy and last name zhang, for receptionist with name ‘receplee’, and the doctor name for ‘chuang’, along with selecting patient type with Out-Patient.

The next procedure is going to maintain the history tables with the triggers. History tables are important tool for tracking important changes over time, so that we are able to solve the value we are concerned about. Also, maintaining with its trigger brings the benefits since trigger will record the change and in the process of triggering, we could know how the previous table and DBMS physical ERD exist in a right way.

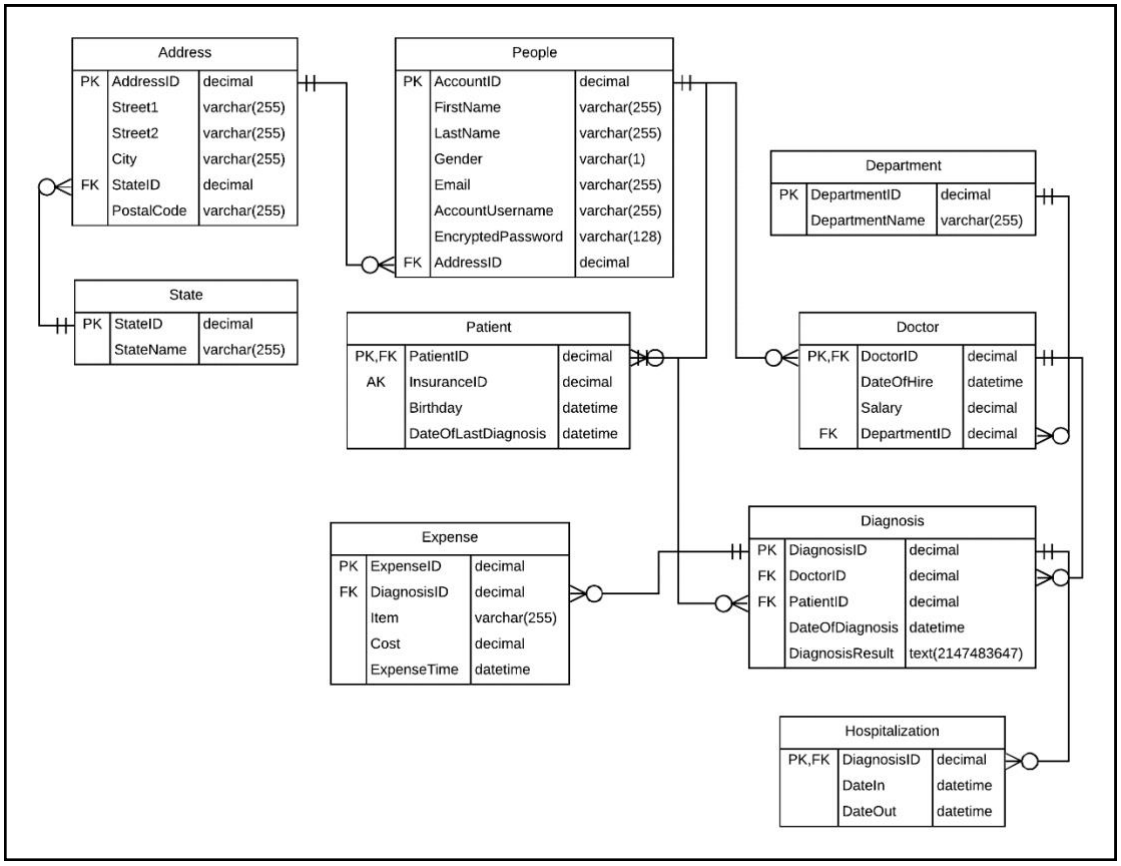
History Table:

With reviewing my DBMS physical ERD, I added BillingInfo entity and related to In-Patient table to calculate separately comparing with Out-patients. My latest structural database rule is: Each in-patient will have one or more bills to pay; one or more bills are for each in-patient.

However, in this phase we have to create trigger and record the changes of the table. Then, I found out that it’s not quite appropriate to trigger the BillingAddress for In-Patient,

which it is a qualitative metrics instead of a quantitative value being calculated and changed.

Hence, I rectified my following DBMS physical ERD in few directions:



Here is the created table for each entity:

```
create table [State](
  StateID decimal(12) not null primary key,
  StateName varchar(255) not null
);

create table Address(
  AddressID decimal(12) not null primary key,
  Street1 varchar(255),
  Street2 varchar(255),
  City varchar(255),
  StateID decimal(12) references [State](StateID),
  PostalCode varchar(255)
);

create table People(
  AccountID decimal(12) not null primary key,
  FirstName varchar(255) not null,
  LastName varchar(255) not null,
  Gender varchar(1) check(Gender in ('M', 'F')) default('M'),
  Email varchar(255) not null,
```

```
  AccountUsername varchar(255) not null,
  EncryptedPassword varchar(128) not null,
  AddressID decimal(12) references Address(AddressID)
);



create table Patient(
  PatientID decimal(12) not null primary key references
People(AccountID),
  InsuranceID decimal(12) not null unique,
  Birthday datetime,
  DateOfLastDiagnosis datetime
);

create table Department(
  DepartmentID decimal(12) not null primary key,
  DepartmentName varchar(255) not null
);

create table Doctor(
  DoctorID decimal(12) not null primary key references People(AccountID),
  DateOfHire datetime,
  Salary decimal(10,2),
  DepartmentID decimal(12) not null references Department(DepartmentID)
);

create table Diagnosis(
  DiagnosisID decimal(12) not null primary key,
  DoctorID decimal(12) not null references Doctor(DoctorID),
  PatientID decimal(12) not null references Patient(PatientID),
  DateOfDiagnosis datetime not null default(getdate()),
  DiagnosisResult text not null
);

create table Hospitalization(
  DiagnosisID decimal(12) not null primary key references
Diagnosis(DiagnosisID),
  DateIn datetime not null default(getdate()),
  DateOut datetime
);

create table Expense(
  ExpenseID decimal(12) not null primary key,
  DiagnosisID decimal(12) not null references Diagnosis(DiagnosisID),
  Item varchar(255) not null,
  Cost decimal(10,2),
  ExpenseTime datetime default(getdate())
);
```

After creating the table, it should go to insert data under each entity:

```
insert into [State] values
  (1, 'state1'),
  (2, 'state2'),
  (3, 'state3'),
```

```
  (4, 'state4'),
  (5, 'state5');

insert into Address values
  (1, 'street11', 'street21', 'city1', 1, '123451'),
  (2, 'street12', 'street22', 'city2', 3, '223452'),
  (3, 'street13', 'street23', 'city3', 2, '323453'),
  (4, 'street14', 'street24', 'city4', 3, '423454'),
  (5, 'street15', 'street25', 'city5', 4, '523455'),
  (6, 'street16', 'street26', 'city6', 5, '623456'),
  (7, 'street17', 'street27', 'city7', 1, '723457'),
  (8, 'street18', 'street28', 'city8', 4, '523455'),
  (9, 'street19', 'street29', 'city9', 5, '623456'),
  (10, 'street110', 'street210', 'city10', 1, '723457');

insert into People values
  (1, 'firstname1', 'lastname1', 'M', 'abc@123.com', 'username1',
'xxxyyy', 1),
  (2, 'firstname2', 'lastname2', 'M', 'abc@123.com', 'username2',
'xxxyyy', 2),
  (3, 'firstname3', 'lastname3', 'M', 'abc@123.com', 'username3',
'xxxyyy', 3),
  (4, 'firstname4', 'lastname4', 'M', 'abc@123.com', 'username4',
'xxxyyy', 4),
  (5, 'firstname5', 'lastname5', 'M', 'abc@123.com', 'username5',
'xxxyyy', 5),
  (6, 'firstname6', 'lastname6', 'M', 'abc@123.com', 'username6',
'xxxyyy', 6),
  (7, 'firstname7', 'lastname7', 'M', 'abc@123.com', 'username7',
'xxxyyy', 7),
  (8, 'firstname8', 'lastname8', 'M', 'abc@123.com', 'username8',
'xxxyyy', 8),
  (9, 'firstname9', 'lastname9', 'M', 'abc@123.com', 'username9',
'xxxyyy', 9),
  (10, 'firstname10', 'lastname10', 'M', 'abc@123.com', 'username10',
'xxxyyy', 10);

insert into Patient values
  (1, 123456, '1990-1-1', '2019-12-9'),
  (2, 223456, '1980-1-1', '2019-11-2'),
  (3, 323456, '1995-1-1', '2019-12-3'),
  (4, 423456, '1999-1-1', '2019-12-7'),
  (5, 523456, '2010-1-1', '2019-11-9');

insert into Department values
  (1, 'department1'),
  (2, 'department2'),
  (3, 'department3'),
  (4, 'department4'),
  (5, 'department5');

insert into Doctor values
  (6, '2001-1-1', 7000, 1),
  (7, '2010-1-1', 6000, 2),
  (8, '1997-1-1', 8000, 3),
  (9, '2015-1-1', 5000, 1),
  (10,'2019-1-1', 4000, 2);
```

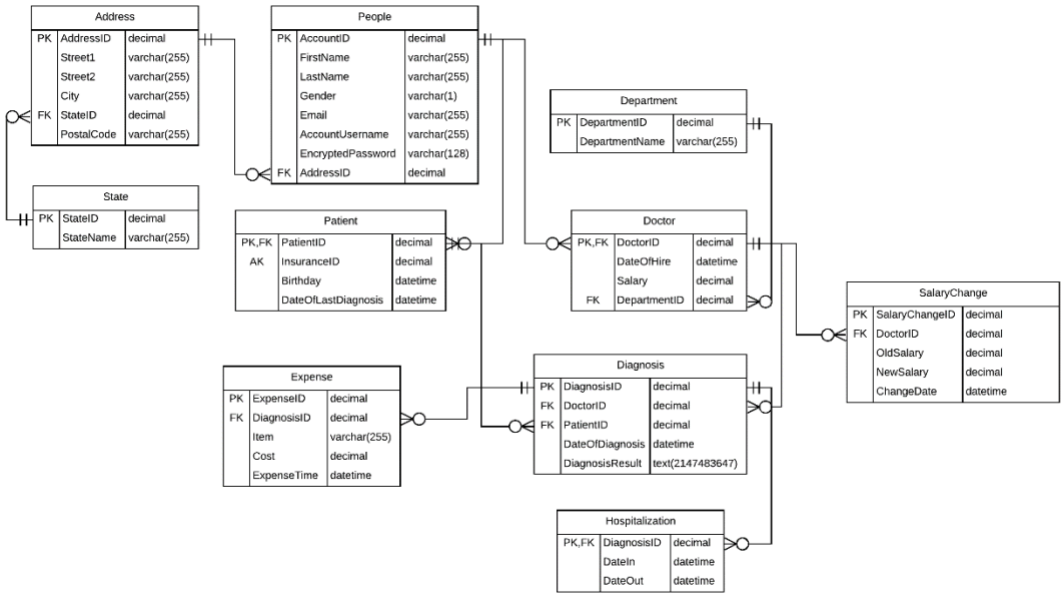
```
insert into Diagnosis values
  (1, 6, 1, '2019-10-1', 'need hospital treatment'),
  (2, 6, 2, '2019-10-3', 'need hospital treatment'),
  (3, 6, 3, '2019-10-6', 'need hospital treatment'),
  (4, 7, 1, '2019-11-1', 'need hospital treatment'),
  (5, 7, 2, '2019-11-2', 'need hospital treatment'),
  (6, 8, 5, '2019-11-9', 'need hospital treatment'),
  (7, 9, 1, '2019-12-1', 'need hospital treatment'),
  (8, 9, 3, '2019-12-3', 'need hospital treatment'),
  (9, 9, 4, '2019-12-7', 'need hospital treatment'),
  (10, 6, 1, '2019-12-8', 'need hospital treatment');

insert into Hospitalization values
  (1, '2019-10-1', '2019-10-3'),
  (2, '2019-10-3', '2019-10-6'),
  (3, '2019-10-6', '2019-10-18'),
  (4, '2019-11-1', '2019-11-10'),
  (5, '2019-11-2', '2019-11-4'),
  (6, '2019-11-9', '2019-11-13'),
  (7, '2019-12-1', '2019-12-5'),
  (8, '2019-12-3', null),
  (9, '2019-12-7', null),
  (10,'2019-12-8', null);

insert into Expense values
  (1, 1, 'basic', 10, '2019-10-1 13:00'),
  (2, 1, 'medicine', 20, '2019-10-1 15:00'),
  (3, 1, 'medicine', 30, '2019-10-2 20:00'),
  (4, 2, 'basic', 10, '2019-10-3 13:00'),
  (5, 2, 'medicine', 5, '2019-10-4 15:00'),
  (6, 3, 'basic', 10, '2019-10-6 13:00'),
  (7, 3, 'medicine', 25, '2019-10-6 20:00'),
  (8, 4, 'basic', 10, '2019-11-1 13:00'),
  (9, 4, 'medicine', 10, '2019-11-1 15:00'),
  (10, 4, 'medicine', 40, '2019-11-2 20:00'),
  (11, 4, 'medicine', 50, '2019-11-3 18:00'),
  (12, 5, 'basic', 10, '2019-11-2 13:00'),
  (13, 6, 'basic', 10, '2019-11-9 13:00'),
  (14, 7, 'basic', 10, '2019-12-1 13:00'),
  (15, 8, 'basic', 10, '2019-12-3 13:00');
```

For my trigger creation, for here I'd like to say that it's important to record the change of salary of any doctors, which has all of the same attributes and datatypes as indicated in the DBMS physical ERD.

```
create table SalaryChange(
  SalaryChangeID decimal(12) not null primary key,
  OldSalary decimal(10,2) not null,
  NewSalary decimal(10,2) not null,
  DoctorID decimal(12) not null references Doctor(DoctorID),
```

```
  ChangeDate datetime not null default(getdate())
);
```

Therefore, the new ERD with adding SalaryChange to any doctors has been updated and created:



The trigger will automatically record the change of salary for any doctor, and the trigger creation shows below which will maintain the SalaryChange table.

```
CREATE TRIGGER SalaryChangeTrigger
ON Doctor AFTER UPDATE
AS
BEGIN
  DECLARE @OldSalary DECIMAL(10,2) = (SELECT Salary FROM DELETED);
  DECLARE @NewSalary DECIMAL(10,2) = (SELECT Salary FROM INSERTED);

  IF (@OldSalary <> @NewSalary)
        INSERT INTO SalaryChange(SalaryChangeID, OldSalary, NewSalary,
DoctorID, ChangeDate)
        VALUES(ISNULL((SELECT MAX(SalaryChangeID)+1 FROM SalaryChange),
1),
              @OldSalary,
              @NewSalary,
              (SELECT DoctorID FROM INSERTED),
              GETDATE());
END
```

Code	Explanation

CREATE TRIGGER SalaryChangeTrigger ON Doctor AFTER UPDATE	This starts the definition of the trigger and names it "SalaryChangeTrigger". The trigger is linked to the Doctor table, and is executed after any updated to that table.
AS BEGIN	This is part of the syntax starting the trigger block.
DECLARE @OldSalary DECIMAL(10,2) = (SELECT Salary FROM DELETED); DECLARE @NewSalary DECIMAL(10,2) = (SELECT Salary FROM INSERTED);	This saves the old and new Salaries by referencing the DELETED and INSERTED pseudo tables, respectively.
IF (@OldSalary <> @NewSalary)	This action is only taken if the balance has been updated.
INSERT INTO SalaryChange(SalaryChangeID, OldSalary, NewSalary, DoctorID, ChangeDate) VALUES(ISNULL((SELECT MAX(SalaryChangeID)+1 FROM SalaryChange), 1), @OldSalary, @NewSalary, (SELECT DoctorID FROM INSERTED), GETDATE());	This inserts the record into the SalaryChange table. The primary key is set by obtaining one over the next highest. The old and new salaries are used from the variables. The DoctorID is obtained from the INSERTED table. The date of the change is obtained by using the built-in GETDATE function.
END	This ends the trigger definition.

To do test on the trigger creation, I used an update script to update the salary of doctor whose ID is 1.

```
update Doctor set Salary = Salary*1.1 where DoctorID = 6;
```

And use a select script to view data of table SalaryChange:

```
select * from SalaryChange;
```

	SalaryChangeID	OldSalary	NewSalary	DoctorID	ChangeDate
1	1	7000.00	7700.00	6	2019-12-08 19:30:04.050

That is, I verified that the SalaryChange table has a record of those changes. I could notify that there is one row for the change from $7000 to $7700. The old and new changes arenow tracked with a trigger and a history of table.

Question and Queries:

In this step, I will create 3 useful questions to my application along with writing the queries to address my questions.

1) What is the total cost of each patient?

```
select FirstName + ' ' + LastName as PatientName, TotalCost
from People, (
    select PatientID, sum(Cost) as TotalCost
    from Expense, Diagnosis
    where Expense.DiagnosisID = Diagnosis.DiagnosisID
    group by PatientID
) t1
where t1.PatientID = People.AccountID;
```

	PatientName	TotalCost
1	firstname1 lastname1	180.00
2	firstname2 lastname2	25.00
3	firstname3 lastname3	45.00
4	firstname5 lastname5	10.00

Explain:

This question will count approximately the costs for different patient, which will help me to determine which medical treats the most patients have come in and took. Then, it will adjust the medical staffs around the hospital. To get results, I used group by of PatientID table. Upon the monitoring, we can see that there are 4 rows coming out and with the most total cost $180 with PatientName “firstname1 last name1”; and, “firstname5 last name5” has the least amount of cost in $10. From this, it can be conjectured that “firstname1 last name1” took more examinations comparing to “firstname5 last name5” does; in addition, “firstname1 last name1” might be an out-patient and “firstname5 last name5” might be the in-patient. Then, the patient from different department should have different access to the hospital system by applying *HealthCare* to provide their insurance information/medical histories.

2) What are the numbers of doctors of each department?

```
select DepartmentName,
    sum(case when DoctorID is null then 0 else 1 end) as
NumberOfDoctors
from Department left join Doctor
    on Department.DepartmentID = Doctor.DepartmentID
group by DepartmentName;
```

	DepartmentName	NumberOfDoctors
1	department1	2
2	department2	2
3	department3	1
4	department4	0
5	department5	0

Explain:

Connecting with the previous question I came up with, I notified the adjustment of doctors in various departments. This question also help the organization to see the balance of staff allocation. To get the results, I utilized left join the Department to the Doctor table, limit the results to those with essential department; and, I group by the department name to categorize the # of each department into “1,2,3,4,5”. It shows department 1 & 2 both have 2 doctors, but the department 4 and 5 don’t maintain any doctor or the doctor in department 4 & 5 don't schedule visits too much due to the low demand in those type of medical treats from the patients.

3) Who are the patients that hasn't gone out of hospital yet?

```
select People.*
from Hospitalization, Diagnosis, People
where DateOut is null
    and Hospitalization.DiagnosisID = Diagnosis.DiagnosisID
    and Diagnosis.PatientID = People.AccountID;
```

	AccountID	FirstName	LastName	Gender	Email	AccountUsername	EncryptedPassword	AddressID
1	3	firstname3	lastname3	M	abc@123.com	username3	xxxyyy	3
2	4	firstname4	lastname4	M	abc@123.com	username4	xxxyyy	4
3	1	firstname1	lastname1	M	abc@123.com	username1	xxxyyy	1

Explain:

The listed results show the patient with specific ID and detailed information who are still in for the treatments at hospital. This question is useful because I think it's efficient and effective for hospital to make any records from the patient who completed the treatments. Since, each patient uses the HealthCare application to show their insurance info and medical histories, their information will be sent to hospital's system. Once each of them finished the treatment, they will turned the medical histories back to the receptionist who will make records. Later than, the bill from hospital will be saved in the patients' account under insurance info section and then the bills and invoices will be sent from insurance company to those patients.

Summary and Reflection:

My database is to track patient's insurance info from hospital system. Simply, the patients don't have to file the information and provide medical history every time they stop by the hospital as the information from *HealthCare* application is directly linked to hospital system, so that it will speed up the flow for scheduling the appointment and treatment time between each doctor and patient. That is, the database must support a patient entering reception, filing info, and even analyzing their info data under the system.

The structural database rules and ERD for my database design contain the important entities of Patient, Doctor, Department, Address etc, as well as the relationship between them. The ERD design contains a hierarchy of Patient/In-Patient and Patient/Out-Patient to reflect the two primary ways the patient take the treatment in the hospital. The DBMS physical ERD contains the same entities and relationships, and uses the best practice of synthetic keys. In the

last rectified ERD, the SalaryChange has been added to the table of Doctor to see the change from old salary to the new values.

The SQL script that creates all tables follows the specification from the DBMS physical ERD exactly. Stored procedures and maintaining histories of salary changes have been created and it executed and implemented successfully to elevated my database with data, by creating query which tracks changes to salary for a specific doctor with detailed ID and name. The trigger creation is important in the database design as it helps to realize the function on record the change of values and make in combination with table and trigger. Furthermore, some useful questions for *HealthCare* have been created and implemented with SQL queries.

To sum up, I've realized a big accomplishments in this database design project. Indeed, it has undertaken a long way and big workload; however, it's terrific to see that I completed a real database by coming up with my own mind to a network application *HealthCare,* which it has been already implemented with SQL queries, unless currently only with fictitious numbers and elements. Moreover, I can tell there is still more function and methods to develop in my database, for instance, the question of patient costs, it should come up with more table such as department and the date/time. The more solid foundation I provide, the longer way and more solid implementation the project will be.