SCHOOL OF COMPUTING

TEESSIDE UNIVERSITY

MIDDLESBROUGH

TS1 3BA

Medical Health History Database

Michael Watts

N3071956

BSc Computer Science

20.01.2017

Supervisor: Jim Longstaff

Keywords: SQL, medical, doctors, patients, database design, architecture, models, functionality, data

CONTENTS

1 DESCRIPTION

1.1.1 Summary of the application area

The medical record system is a database management system that uses database technologies to manipulate and maintain various data about a patient's medical history. The database can track and update all the information of registered patients in the medical centre or hospital during a time span.

The database is for staff members only to log patient details that will go no further than the hospital or medical centre. The purpose of storing patient information is to help understand the logical and well-being of a patients care. The purpose of digital storage is to help reduce paper resources and to speed up the process of data manipulation. Electronic storage means that there is less of a risk of loss but also each staff member has access to patient details at more than one time.

1.1.2 Description of the functionality

The medical record serves a variety of purposes and is essential to the proper functioning of the medical practice. The medical record is a key instrument used in planning, evaluating and coordinating patient care in to the patient from the doctor. The content of the medical record is essential for patient care.

A patient's medical record is registered on the first visit to the doctor, the patient can also make one or more appointments with one or more doctors. The doctor can accept appointments with more than one patient, however, each appointment is made with only one doctor, and each appointment references a single patient. This is when the visit to the doctor yields a diagnosis and if needed a possible treatment is recorded for the patient. Each visit to the doctor will be recorded by the doctor.

1.1.3 Data that will be stored

Information such as name, age, address, office and phone number is provided for each doctor that is assigned to each patient. Medicine and prescription data is stored also, such information as diagnosis id and category of the diagnosis are stored along with the medicine that is prescribed to a patient. Medical records are created when a patient receives treatment from a health professional. Records may include:

- Personal information
- Medical history
- Test results
- Medications that have been prescribed
- ***And if time*** - Results, reports and medical procedures

2 TASKS

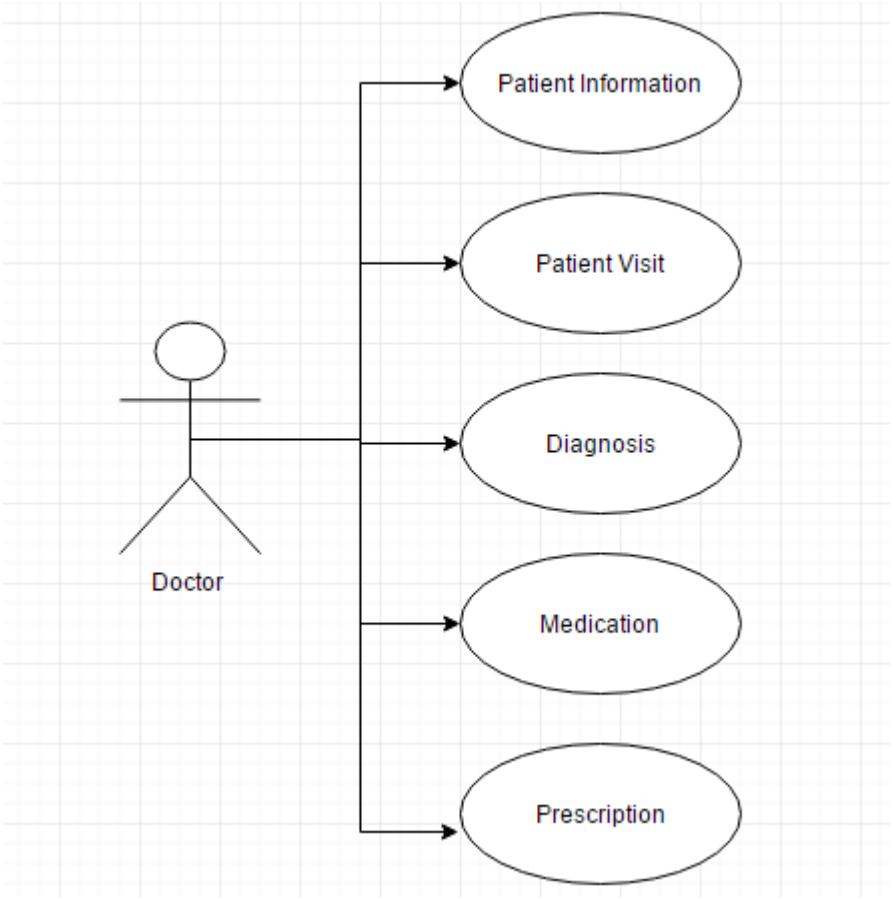
2.1 SQL Server Database Design

2.1.1 Brief description of my data architecture

Below is a concise and detailed approach of what my application is intended to do. Below is a normalised database design with documented ER models and Use Case Diagrams. An activity Diagram has been provided to show the state transition of the doctor's walkthrough on the database. Indexes/constraints with primary keys and foreign keys has also been provided.

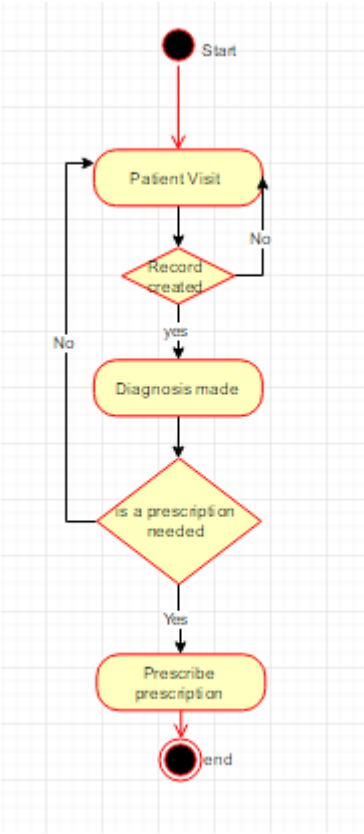
2.1.2 Use case diagram

The diagram below shows the system objectives for the doctor's requirements. The use case methodology is used to analyse the database system. The doctor can access patient information and records but also create them, they can also acquire medication and prescriptions for patients. The use case diagram states the dependency from one single actor, at this stage there is no need for the patient or another actor to be involved.



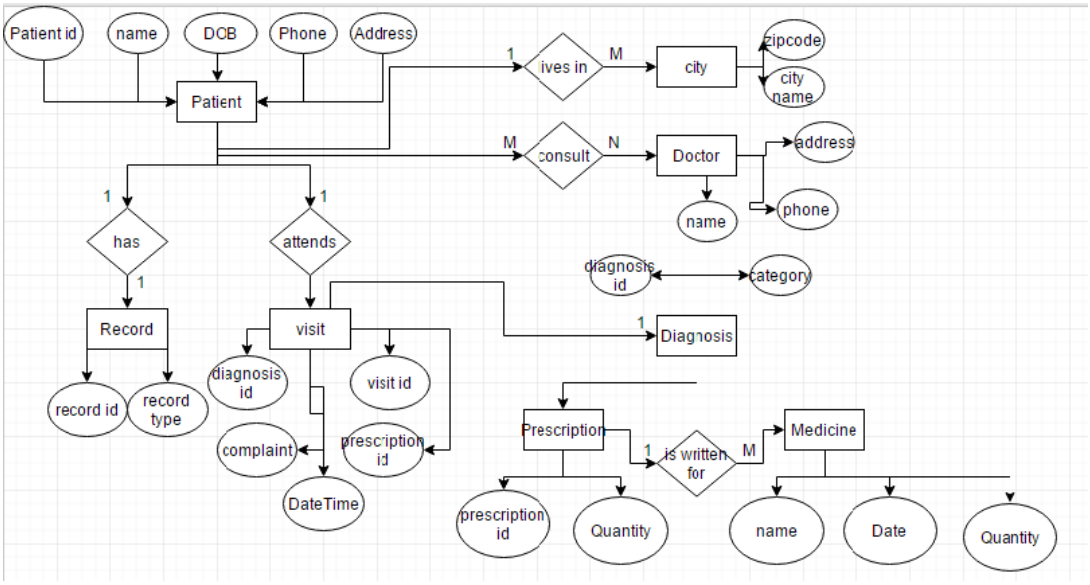
2.1.3 Activity Diagram

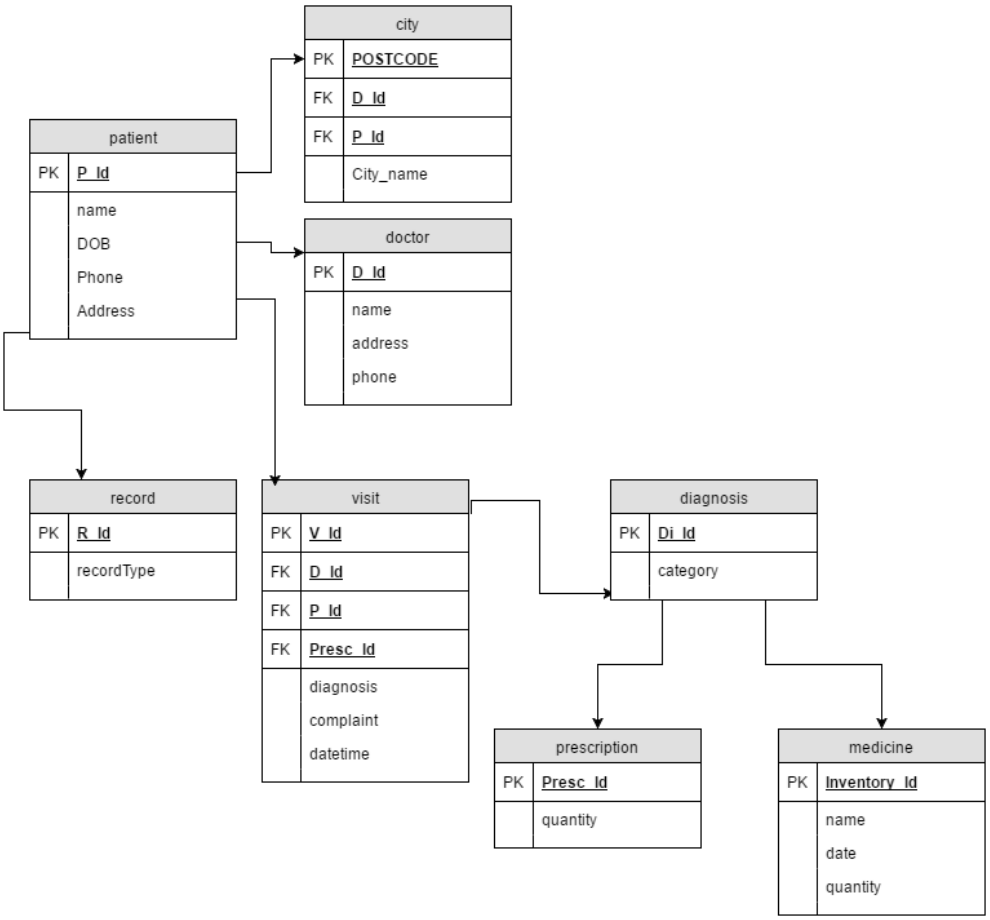
Below is an activity diagram, the diagram captures the dynamic behaviour of the system. The diagram is a flow chart to represent the flow form of one activity to another. The patient will visit the doctor, then a record is created for that patient with a diagnosis and medication prescribed.



2.1.4 Conceptual schema and ER models

The er model below defines the conceptual view of my database. At the database view level, the er model is considered a good option for designing the database context. The er model below shows the entities with attributes sharing similar values. For example, a visit table may contain the patient and doctor details. The diagrams below also show the relationships between the data.





The full list of tables and brief overview of their purpose is below.

- Doctor

  Stores the information about doctors that are assigned to a medical establishment.
- Patient

  Stores the relevant information needed by doctors to contact the patient.
- City

  A partial table that is defined by foreign keys, it is needed for both the doctor and patient.
- Record

  A record is created by the doctor from the patient.
- Visit

  This table will store the date, time and the complaint the patient had and what doctor they saw. It will also store the diagnosis and the prescription proscribed.
- Diagnosis

  A diagnosis is created by the doctor for the patient
- Prescription

  If a patient requires a prescription the doctor will have access to see the medicine quantity.
- Medicine

  The medicine table shows the manufacturer id, name, quantity that should be taken but also the expiry date.

Normalised Database structure diagram.

Entity	Relationship
Doctor	***(D_Id)*** name, age, phone, address
Patient	***(P_Id)*** name, dob, phone, address
City	***(zip)*** zip, city_name
Visit	date, time, complaint, diagnosis, prescription
Diagnosis	***(Di_Id)*** diagnosis_id, category
Prescription	***(Prescrip_Id)*** prescription_id, medicine_quantity
Medicine	***(MInventry_Id)*** name, price, quantity, date

2.1.5 Domain key normal form

Relation	Connections
DoctorCity	**Key:** (FK)POSTCODE (FK)D_Id **Domain:** POSTCODE: 7 (Varchar), D_Id: 10 (Varchar)
Patient - City	**Key:** (FK)POSTCODE (FK) P_Id **Domain:** POSTCODE: (Varchar), P_Id: (Varchar)
Visit	**Key:** (PK)V_Id (FK)Di_Id (FK)D_In (FK)Presc_Id **Domain:** V_Id: (Varchar), date: Date, time: DateTime, complaint: (Varchar) , Di_Id: (Varchar), D_Id: (Varchar), Prescr_Id: (Varchar)
History	**Key:** P_Id + V_Id - date, time, complaint , Di_Id, D_Id, Presc_Id **Domain:** P_Id: (Varchar) ,V_Id: (Varchar)
PatientRecord	**Key:** (FK)R_Id (FK) P_Id **Domain:** R_Id: (Varchar), P_Id: (Varchar)
Medicine	**Key:** (PK)Inventory_Id (FK)Manu_Id **Domain:** Inventory_Id: (Varchar), Manu_Id: (Varchar), M_name: (varchar), Quantity: (int), Exp_Date: Date
PrescriptionMedicine	**Key:** (FK) Inventory_Id (FK) presc_Id **Domain:** Inventory_Id: (Varchar) , presc_Id: (Varchar)
Doctor	**Key:** (PK) D_Id **Domain:**

	D_Id: (Varchar), name: (Varchar), Age: (int), Phone:(Varchar), address: (Varchar), Office: (Varchar)
Patient	**Key:** (PK)P_Id **Domain:** P_Id:(Varchar), name:(String), DOB: Date, Phone: (Varchar), address:(Varchar)
City	**Key:** (PK) POSTCODE **Domain:** postcode (Varchar) , city_name: (Varchar)
Diagnosis	**Key:** (PK)Di_Id **Domain:** Di_Id: (Varchar) , category: (Varchar)
Prescription	**Key:** (PK)Presc_Id **Domain:** Presc_Id: (Varchar), Quantity:(int)
Record	**Key:** (PK) R_Id **Domain:** R_Id: (Varchar), type: (Varchar)
Medicine	**Key:** (PK)Inventory_Id (FK)Manu_Id **Domain:** Inventory_Id: (Varchar), Manu_Id:(Varchar), name: (Varchar), Quantity:(int), Exp_Date: Date
Manufacturer	**Key:** (PK)Manu_Id **Domain:** Manu_Id:(Varchar), Manu_name: (Varchar)

2.1.6 Primary Keys

Relations	***(Primary Keys)***
Doctor	D_Id
Patient	P_Id
City	postcode
Diagnosis	Di_Id
Prescription	Presc_Id
Record	R_Id
Medicine	Inventory_Id
Manufacturer	Manu_Id
Visit	V_Id

2.1.7 Foreign Keys

Relations	***(Foreign Keys)***
DoctorCity	D_Id ref Doctor.D_Id POSTCODE ref City.POSTCODE
Patient - City	P_Id ref Patient.P_Id POSTCODE ref City.POSTCODE
Visit	Di_Id ref Diagnosis.Di_Id D_Id ref Doctor.D_Id Presc_Id ref Prescription. Presc_Id
History (for the use of the DW)	P_Id ref Patient.P_Id V_Id ref Visit.V_Id
PatientRecord	R_Id ref Record.R_Id P_Id ref Patient.P_Id
Medicine	Manu_Id ref Manufacturer.Manu_Id
PrescriptionMedicine	Inventory_Id ref Medicine.Inventory_Id Presc_Id ref Prescription.Presc_Id

2.2 SQL Server Implementation

2.2.1 Physical Tables with sample data

- *Doctor*

```
CREATE TABLE [dbo].[doctor](
	[D_Id] [int] NOT NULL,
	[name] [varchar](255) NOT NULL,
	[FirstName] [varchar](255) NULL,
	[Address] [varchar](255) NULL,
	[Office] [varchar](255) NULL,
	[Phone] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[D_Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

	D_Id	name	FirstName	Address	Office	Phone
▶	1	Dr watts	Michael	52 Pierremont ...	Orchard Court	1325
*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

The doctor table holds the information necessary to contact a specific doctor. Also, a primary Id is assigned to each doctor. This allows searches to be carried out when looking at patient information but to also know which doctor met or is dealing with which patients. The primary id (D_Id) is used throughout the database.

- *Patient*

```
CREATE TABLE [dbo].[patient](
	[P_Id] [int] NOT NULL,
	[Name] [varchar](255) NOT NULL,
	[DOB] [varchar](255) NULL,
	[Address] [varchar](255) NULL,
	[Phone] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[P_Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

	P_Id	Name	DOB	Address	Phone
▶	1	Sophie	24/01/1994	39 Lunedale Ro...	746094517
*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

The patient table hold personal information that the doctors have access too, the patient details are needed so that the doctors can contact patients and create the medical visit history information when the patient visits the doctor. The patient has a primary key, the primary key is needed about holding patient visit information and medical history.

- *City*

```
CREATE TABLE [dbo].[city](
	[postcode] [varchar](255) NOT NULL,
	[P_Id] [int] NOT NULL
) ON [PRIMARY]
```

	postcode	P_Id
▶	DL30GX	1
*	NULL	NULL

The city table gives the postcode of each patient inside the patient table. The primary key is the P_Id, and this is what shows the doctor where the patient lives and when they do a search on a patient.

- *Visit*

```
CREATE TABLE [dbo].[visit](
	[V_Id] [int] NOT NULL,
	[Date] [varchar](255) NOT NULL,
	[Time] [datetime] NULL,
	[Complaint] [varchar](255) NULL,
	[Di_Id] [int] NULL,
	[D_Id] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[V_Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

	V_Id	Date	Time	Complaint	Di_Id	D_Id
▶	1	24/08/2016	6:40pm	stomach ache	3	1
*	NULL	NULL	NULL	NULL	NULL	NULL

An entry to the visit table is created by the doctor, the table is used to show the date, time, complaint and which doctor that patient saw during the medical examination. The visit table also shows the diagnosis that the doctor gave to the patient, the diagnosis is a primary key that has values with medicine so that doctor can prescribe the correct prescription to the patient.

- *Diagnosis*

```
CREATE TABLE [dbo].[diagnosis](
	[Di_Id] [int] NOT NULL,
	[Category] [varchar](255) NOT NULL,
PRIMARY KEY CLUSTERED
(
	[Di_Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

	Di_Id	Category
▶	3	gasterix
*	NULL	NULL

The diagnosis is set as an id inside the visit table, the diagnosis shows a category for that diagnosis e.g. a patient has a diagnosis and that diagnosis shows a category of illness gastric.

- *Prescription*

```
CREATE TABLE [dbo].[prescription](
	[Prescrip_Id] [int] NOT NULL,
	[quantity] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[Prescrip_Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

	Prescrip_Id	quantity
▶	1	2
*	*NULL*	*NULL*

The prescription table is used by the doctor so they know how much of a type of medicine to prescribe to a patient. The prescription id is connected to the medicine table and shows the name of the medicine prescribed.

- *Medicine*

```
CREATE TABLE [dbo].[medicine](
	[Minventry_Id] [int] NOT NULL,
	[name] [varchar](255) NULL,
	[quantity] [int] NULL,
	[date] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[Minventry_Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

	Minventry_Id	name	quantity	date
▶	1	hydrochloride	2 daily	01/01/2017
*	*NULL*	*NULL*	*NULL*	*NULL*

The medicine table shows the amount of the medicine that is available in stock but also the name and the date of shelf life.

- *History*

```
CREATE TABLE history
(
V_Id int,
P_Id int,
FOREIGN KEY (V_Id) REFERENCES visit(V_Id),
FOREIGN KEY (P_Id) REFERENCES patient(P_Id)
)
```

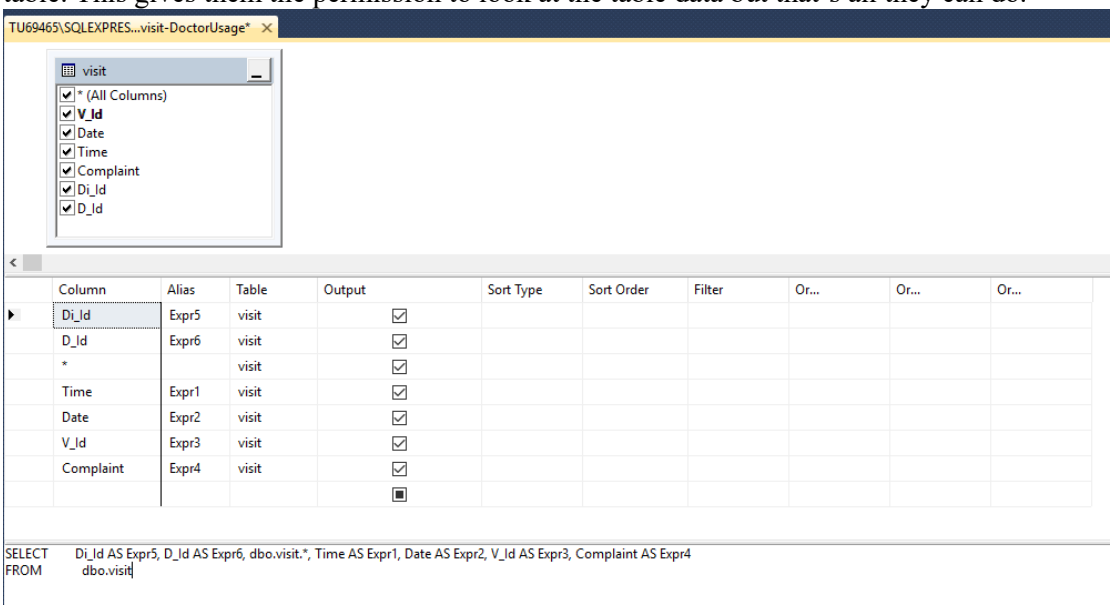
	V_Id	P_Id
▶	1	1
*	*NULL*	*NULL*

The history table is used to show the date, time of visit, complaint, diagnosis the doctor they saw but also the prescription prescribed.

2.2.2 Views

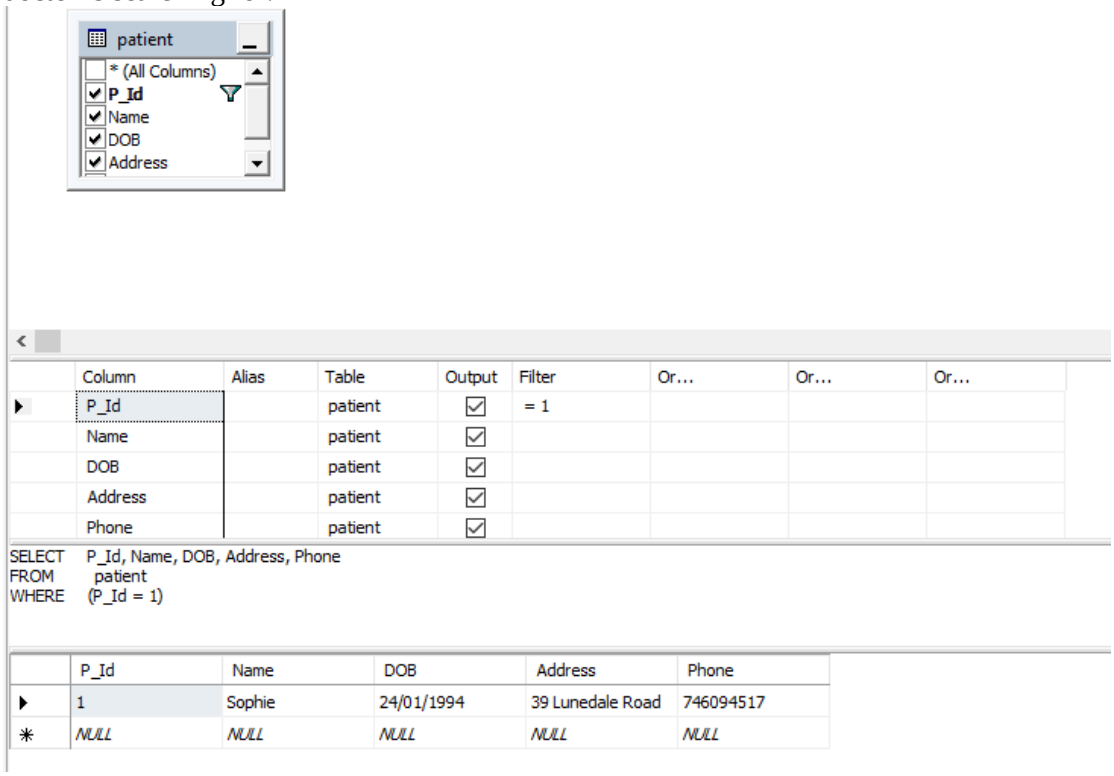
- *Doctor View*

The picture above shows the view for the final visit to the doctor. However, as a security mechanism I only allow the doctors to access this information through the view and not the table. This gives them the permission to look at the table data but that's all they can do.



- *Patient View*

This view shows the patient information where the patients identification is equal to the id the doctor is searching for.



2.2.3 Metrics – User and query conflicts

In my model, I will be using optimistic concurrency control, this assumes that multiple transactions can frequently complete without interfering with each other. While running, transactions can use data without acquiring locks on those resources. Before the query is committed, each transaction verifies that no other transaction has modified the data it has read.

The image below shows me using optimistic locking in sql. As you can see, implementing optimistic locking utilizes the row version datatype is an effective, low overhead way to prevent lost updates while still maintaining my application concurrency.

```
declare @rowid rowversion

select @rowid = D_Id
from [visit-DoctorUsage]
where D_Id = 1

-- wait 30 seconds to simulate
-- a user examining the row and
-- making a data modification
waitfor delay '00:00:30'


update [visit-DoctorUsage]
set sku = 'connection 1: updated'
where partid = 1
and D_Id = @rowid

if @@rowcount = 0
begin
      if not exists (select 1 from visit where D_Id = 1)
          print 'this row was deleted by another user'
      else
          print 'this row was updated by another user.'
end
go
```

2.2.4 Stored Procedures

- *Getting all the visits granted by a doctor*

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Michael>
-- Create date: <06/01/2017>
-- Description: <visitdetails>
-- =============================================
CREATE PROCEDURE [dbo].[medical_details]
    -- Add the parameters for the stored procedure here

    @doctor int = 0001

AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here

    SELECT visit.date, time, diagnosis.category, prescription.quantity
    FROM visit, diagnosis, prescription
    WHERE visit.D_Id = @doctor
    AND visit.Di_Id = diagnosis.Di_Id

END
GO
```

```
USE [medical-staff]
GO

DECLARE @return_value int

EXEC    @return_value = [dbo].[medical_details]

SELECT  'Return Value' = @return_value

GO
```

100 %

Results | Messages

	date	time	category	quantity
1	24/01/2016	6pm	gasterix	2

As you can see from the procedure above I am getting the visit granted by a doctor. I have given the main parameter of one doctor and it has displayed the details of the date and time the patient was seen, there medical category from the complaint they provided and the quantity of the prescription provided to that patient.

- ***Update Patient Details***

The update procedure allows the doctor to change certain values of a patient such as a patient change in address or contact number. This allows the doctor to easily update a patient details without dropping or recreating the patient's entry, this is done by using the patients identification number.

```
USE [medical-staff]
GO
/****** Object:  StoredProcedure [dbo].[UPDATE-PATIENT]
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Michaelwatts>
-- Create date: <27.01.2017>
-- Description: <update>
-- =============================================
ALTER PROCEDURE [dbo].[UPDATE-PATIENT]
    -- Add the parameters for the stored procedure here
     @id int = NULL,
    @name nvarchar(20) = NULL,
    @dob nvarchar(20) = NULL,
    @address nvarchar(50) = NULL,
    @phone int = null
AS
BEGIN
    SET NOCOUNT ON;
    UPDATE patient
    SET Name=ISNULL(@name, Name),
        DOB=ISNULL(@dob,DOB),
        Address=ISNULL(@address, Address),
        Phone=ISNULL(@phone, Phone)
    WHERE P_Id=@id
END
```

- ***Select Patient Details***

This allows the doctor to search for a patient using any of their credentials such as identification number or name. It is used mainly to contact the patient via letter to search for their home address.

```
USE [medical-staff]
GO
/****** Object:  StoredProcedure [dbo].[UPDATE-PATIENT]
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Michaelwatts>
-- Create date: <27.01.2017>
-- Description: <update>
-- =============================================
ALTER PROCEDURE [dbo].[UPDATE-PATIENT]
    -- Add the parameters for the stored procedure here
     @id int = NULL,
    @name nvarchar(20) = NULL,
    @dob nvarchar(20) = NULL,
    @address nvarchar(50) = NULL,
    @phone int = null
AS
BEGIN
    SET NOCOUNT ON;
    UPDATE patient
    SET Name=ISNULL(@name, Name),
        DOB=ISNULL(@dob,DOB),
        Address=ISNULL(@address, Address),
        Phone=ISNULL(@phone, Phone)
    WHERE P_Id=@id
END
```

- *Delete Patient Records*

The delete procedure allows the doctor to remove the patient from their administration. However, the doctor can't remove a patient unless given consent.

```
USE [medical-staff]
GO
/****** Object:  StoredProcedure [dbo].[DELETE-PATIENT]
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <michaelwatts>
-- Create date: <27.01.2016>
-- Description: <DeletePatient>
-- =============================================
ALTER PROCEDURE [dbo].[DELETE-PATIENT]
    -- Add the parameters for the stored procedure here
     @id int = NULL,
    @name nvarchar(20) = NULL,
    @dob nvarchar(20) = NULL,
    @address nvarchar(50) = NULL,
    @phone int = null
AS
BEGIN
    SET NOCOUNT ON;
    DELETE Name,
            DOB,
            Address,
            Phone
  FROM dbo.patient
 WHERE P_Id = @id
END
```

2.2.5 Basic Queries

- *Getting all the doctors*

```
select * from doctor
```

100 %

Results | Messages

	D_Id	name	FirstName	Address	Office	Phone
1	1	Dr watts	Michael	52 Pierremont Road	Orchard Court	1325

The basic query is self-explanatory, it gets all the doctors from the doctors table.

- *Getting patients details from their name*

```
SELECT DOB, Address, Phone
FROM patient
WHERE Name = 'Sophie'
```

100 %

Results | Messages

	DOB	Address	Phone
1	24/01/1994	39 Lunedale Road	746094517

This query allows me to get the patients details by doing a search on a patient.

- *Getting a certain doctor contact details*

```
SELECT FirstName, Address, Office, Phone
FROM doctor
WHERE Name = 'Dr Watts'
```

100 %

Results | Messages

	FirstName	Address	Office	Phone
1	Michael	52 Pierremont Road	Orchard Court	1325

This is showing the contact details for a provided doctor.

2.3 Advanced work Tasks

2.3.1 Data warehousing exploration

Research from my current medical history database has lead me to interacting and conforming plans to think about the use of a data warehouse. The data warehouse would consolidate patient records, medical history of the patients but also when the patient visited. The data warehouse would be an external database with a collection of multiple heterogeneous sources. The data stored would help the medical staff to support medical reports, and the use of decision making in the medical industry.

The medical database will undergo frequent changes daily because transactions i.e. patient to visit the doctors. Suppose a doctor wants to analyse a previous patient's medical history, then the executive will have no data available after a certain time or would have to go trawling for certain data that they don't have access to because the previous data has been updated due to more than one visit to the doctor's surgery. The data warehouse would provide generalized and consolidated data in a multidimensional view.

With the use of a data warehouse using the medical database data mining functions such as association, clustering and prediction can be integrated with online tools to help the interactive mining of knowledge at multiple level of abstractions. This is the main reason I think the data ware house would be an important abstraction and data analysis for my medical history database.

The main feature too using a separate database for my data ware house would be the non-volatile data. Non-volatile would mean that the medical data stored of a patient would not be erased when new data is added. Data held by OLTP (online transaction processing) systems can be inconsistent but also subject to change, therefore the operational data must be cleaned up before it can be used in the data warehouse. The choice of OLTP systems would help me to analyse historical patient information through raw data.

Unfortunately, due to time restrictions I did not get chance to implement a working data warehouse, therefore, I have showed strong suggestions to how a data warehouse could help and educate the doctors of my medical history database. The use of AdventureWorksDW was also not deployed and constructed to time efficiency.

2.3.2 Azure Exploration

Throughout the process of development of the medical history database, I have been doing research on deployment. I feel that the database I have created would work well with a front-end application. However, due to time restrictions I have not yet implemented a working front end application. The front-end application would be constructed in a way that could be deployed as a project on azure and would have the option to conform the database to be hosted also on azure. Due to credit restrictions on my account I have not been able to deploy the database onto azure for further testing purposes.

The reason for choosing the choice of azure is that they have a very reliable and platform that I can utilize the tools I already have alongside SQL. But not only is it reliable it is largely elastic in a way that being able to scale out with azure by adding instances as needed. Microsoft also provides an infrastructure of federations to make scaling out easier and less impactful, as well as tools like the sql azure data migration wizard to further automate this process.

3 CONCLUSION

When I first started the database project I felt confident in completing after undertaking the design and research phases of the intended database. When I came onto the phases of development I immediately ran into problems including, framework and version control problems with my home computer and the university computers. Therefore, my sole database was created on a university computer. This gave me restrictions and limitations to access, attach, and detach my database to work on it from home.

After 6 -7 weeks of development I have tackled those problems and designed and developed a database that would help doctors to create visit and history information from patients. I feel that in the future I would certainly work on the database more and make sure that it was completely solid and was corrective and working in the correct manner. Currently there are a few glitches that need to be ironed out. In the future I, would make sure that a data warehouse was implemented so that doctors could gain access to further history from patients e.g. I have created a history table so in the future I can pull out relevant information from the patients to be stored.

I personally feel that the relationships between tables are not yet stable, there are a few issues between primary and foreign keys that would need to be ironed out if the database was to be used with a front-end application. This has limited me to providing relevant test data and analysis conformed to get the doctors to input relevant information, and therefore has been inputted manually. The first main transaction that is run gets the doctor to input their details, then to create a patient and to also once the patient has been seen they can then create a successful visit record.

This project has been a big part to my initial studies, I felt that this project has made me a stronger sql developer and I strongly feel that I can tackle my final year project based on my personal performance from undergoing this project. In conclusion to the factors above, I feel that with more time to consolidate the work load and further thinking of the front-end application along with deployment I feel that I could really turn this project into a finely tuned database that would help in a real-world environment and help to promote productivity in the medical industry.