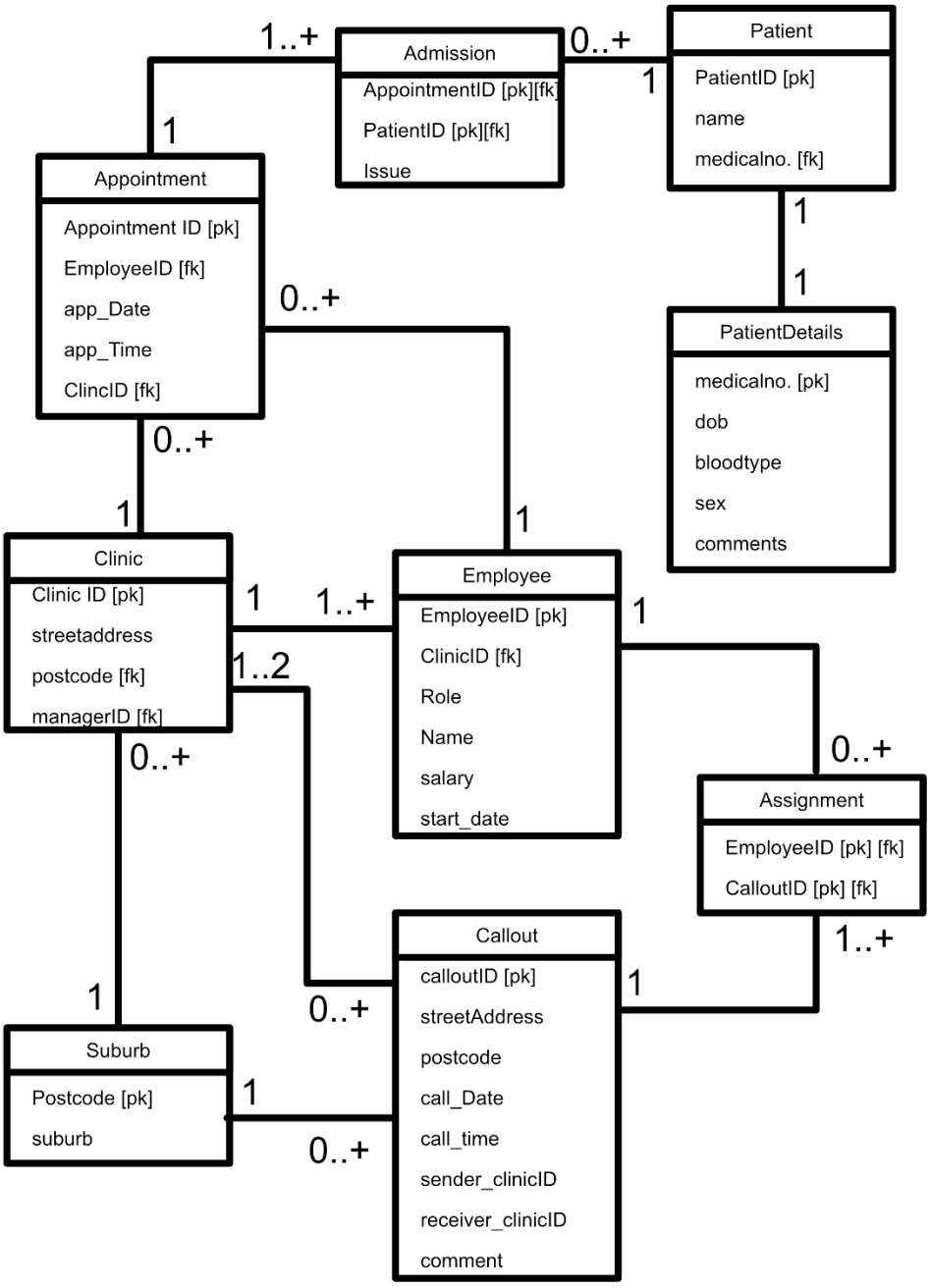
The Datastore

Datastore Requirements

The medical service is a small Victoria wide business that needs a database for tracking its patients and their appointments as well as employees and clinics. Furthermore, the medical service would like the database to be able to handle online bookings for patients and should be able to manage appointments both for individuals and families. The database should also be able to integrate with the government medical datastore that uses medical numbers to hold medical history, so that the medical service can copy the medical data necessary for serving their patients. The medical service has recently invested in a new callout system for doctors and nurses to treat patients at home, the database should also record these events and their resolutions of whether the patient had to go back to a clinic.

Data Solution

UML Illustration



Entity Descriptions

Appointment				
appointmentID	employeeID	app_Date	app_Time	clinicID
[pk]	[fk]			[fk]
Int Unsigned	Int unsigned	Date	Time	Int unsigned
Not null, auto-incmrement	Not null	Not null	Not null	Not null
This table stores information about appointments, the data and time they are scheduled, at which clinic and the doctor that is attending them.				

Admission <<Weak>>		
appointmentID	patientID	issue
[pk][fk]	[pk][fk]	
Int unsigned	Int unsigned	Varchar(200)
Not null	Not null	Default 'General CheckUp'
This weak entity allows a patient to have many appointments and an appointment can have many patients, such as a family appointment, this links the patientID to an appointmentID, as well as holding the issue, for the reason why the patient has booked an appointment.		

Patients		
patientID	name	medical No.
[pk]		[fk]
Int unsigned	Varchar(40)	Numeric(11,1)
Not null, auto-increment	Not null	Not null, no duplicates
The patient table holds all patients that have had an appointment, this holds the medicalno which also would be able to uniquely identify rows as there shouldn't be duplicates. The reason medicalno and patientID are separate columns is because in this hypothetical system, patientID is local to the services database, whereas the medicalno holds medical history and is assigned to individuals by the government healthcare system. By using their own id system to identify patients the medical service is able to separate changes by the government system from effecting their database, this increases the administrative ease of the system.		

PatientDetails				
medicalNo.	Dob	Bloodtype	sex	comments
[pk]				
Numeric(11,1)	Date	Char(3)	Char(1)	Varchar(200)
Not null		Check for A,B,AB,O (-/+)	Check for M or F	
PatientDetails holds the medical history of a patient. This is effectively a copy of information of what would be held on the government database, as medical history is functionally dependant on the medical no.				

Employee					
employeeID	clinicID	role	name	salary	start_Date
[pk]	[fk]				
Unsigned int	Unsigned int	Char(10)	Varchar(40)	Unsigned Int	date
Not null, auto-increment	Not null	Not null	Not null	Not null	Not null
Employee is a record of all employees that work at the medical service as well as the clinic that they are employed at.					

Clinic			
clinicID	streetAdd	postcode	managerID
[pk]		[fk]	[fk]
Unsigned int	Var char(40)	Numeric(4)	Int unsigned
Not null	Not null	Not null, Check starts with 3 or 8	
Clinic holds records of all opened clinics and their location, as well as a foreign key to the employee that is the manager of the clinic			

Suburb	
Postcode	suburb
[pk]	
Numeric(4)	Varchar(20)
Not null, check starts with 3 or 8	Not null
This table exists to eliminate the transitive dependency between postcode and suburbs to reduce contradictory entries that could arise otherwise.	

Callout							
calloutID	street add	postcode	call_Date	call_Time	sender_ClinicID	receiver_ClinicID	comment
[pk]		[fk]			[fk]	[fk]	
Unsinged int	Varchar(40)	Numeric(4)	Date	Time	Unsigned int	Unsigned int	Varchar(200)
Not null, auto-incremnt	Not null	Not null	not null	Not null	Not null	Can be null if can be resolved on site	
This table holds the records of all callouts that occur, it includes the location of the callout as well as any comments about what transpired. Every callout has to be attended by a clinic, which is the sender_clinic of the doctors. If the callout cannot be solved on site, then the patient may be taken back to the same or a different clinic which is the receiver clinic.							

Assignment <<Weak>>	
employeeID	calloutID
Unsigned int	Unsinged int
[pk] [fk]	[pk] [fk]
Not null	Not null
This is a weak entity that links the employees that are sent to attend a callout.	

Typical Use Cases for The Database.

Use cases for the database involve Patients booking appointments, the assignment of employees to appointments and callouts, considering where new branches should be built, as well as patients requesting their medical data.

1. A patient wishes to request their medical data.

This first part of the query joins the patient and patient details tables together so that patient name, and history can be given to the requestee. This could be a common operation for both patients and doctors requesting a patients data and so a view has been created.

```
create view PatientData as
select name, medicalno, dob,bloodtype, sex, comments
from patient natural join patientdetails;
```

Then a specific patient or employee can request the data of a specific tuple using the medicalno

```
set @mno = 1023786932.2;
select *
from PatientData
where medicalno = @mno;
```

name	medicalno	dob	bloodtype	sex	comments
James Leroy	1023786932.2	2002-05-31	AB+	M	NULL

2. The Medical Service needs to know how many appointments and callouts each clinic has served to plan for expansion of amenities.

This query is constructed of 3 sub queries, the first counts the number of appointments that have been served, the second the number of callouts sent, and the third finds the number of callouts received. Outer joins are used to ensure that clinics that have had no work are still shown in the table. The output is then ordered by the total amount of work in descending order to show the clinic that may be most in need of expansion at the top. This one is also made into a view as it is used in following queries.

```
create view ClinicWorkLoad as
select *, Appointments + CalloutsSent + CalloutsRecieved as total
from (select clinicID, count(appointmentID) as Appointments from clinic natural join appointment group by clinicID) as a
natural join (select clinicID, count(calloutID) as CalloutsSent from clinic left join callout on clinic.clinicID = callout.sender_clinicID group by clinicID) as cs
natural join (select clinicID, count(calloutID) as CalloutsRecieved from clinic left join callout on clinic.clinicID = callout.reciever_clinicID group by clinicID) as cr
order by (total) desc;
```

clinicID	Appointments	CalloutsSent	CalloutsRecieved	total
3001	3	1	1	5
1001	3	1	0	4
1002	2	0	1	3
2001	1	1	0	2

3. What suburbs have had callouts but do not have clinics to consider where new branches should be built

This query creates a list of all suburbs that do not have clinics, using the where not exists condition. This query then automatically excludes suburbs with no callouts because it uses an inner join which excludes suburbs that don't have a matching tuple in the callout table.

```
select postcode, suburb, Count(calloutID) as Callouts
from suburb natural join callout
where not exists (select clinicID from clinic where suburb.postcode = clinic.postcode)
group by postcode
order by (Callouts) desc;
```

postcode	suburb	Callouts
3136	Croydon	1
3162	Caufield	1

4. How many appointments has a patient attended to check billing information?

This query joins the patient and admissions tables to retrieve patientID, name and a count of the admissions is then grouped by patient to return a count of the number of admissions that the patient has had, which is equivalent to the number of appointments.

```
select patientID, name, count(*) as Admissions
from patient natural join admission
group by patientID
order by (Admissions) asc;
```

patientID	name	Admissions
6	Marilyn Leroy	1
18	Pheobie Larston	1
11	Mary Mudle	1
5	Holly Smirn	1

5. How much has work has each doctor done?

This query uses two sub queries to count how many appointments each doctor has had and how many assignments to callouts they have had. A total of the workload is then calculated. IN order to only return doctor employees a where clause filters for employees with role 'doctor'. A View is made from this as this could be a common query and forms the basis of another query.

```
create view EmployeeWork as
select employeeID, name, eAppoint.AppCount + eAssign.AssCount as WorkLoad
from employee natural join
(select employeeID, Count(appointment.employeeID) as AppCount from employee natural left join appointment group by employee.employeeId) as eAppoint
natural join
(select employeeID, Count(assignment.employeeID) as AssCount from employee natural left join assignment group by employee.employeeID) as eAssign
where role = 'doctor'
order by WorkLoad asc;
```

employeeID	name	WorkLoad
19	Anna Bartel	0
10	Elizebeth Draxton	1
24	Karen Yert	1
17	Derek Mandel	1

6. How much work has a doctor done at a particular clinic?

This query uses the previous query, additionally filtering based on a chosen clinic.

```
set @clinic = 1001;
select employeeID, name, WorkLoad
from EmployeeWork natural join employee join Clinic on Clinic.clinicID = Employee.clinicID
where Clinic.clinicID = @clinic;
```

employeeID	name	WorkLoad
10	Elizebeth Draxton	1
9	Danton Kold	2
8	Melissa Frandern	2

7. How many employees work at each clinic comapred to its workload, to determine employment strategies.

This Query uses the clinic Workload view, to get the workload for each clinic it then joins that with the clinic and employee tables to return a count of the employees at each clinic.

```
select clinicID, Count(EmployeeID) as Employees, total as WorkLoad
from ClinicWorkLoad natural join clinic natural join employee
group by clinicID;
```

clinicID	Employees	WorkLoad
3001	4	5
1001	6	4
1002	3	3
2001	6	2

8. Book an Appointment?

This would retrieve data entered from the user in a website to book an appointment. This transaction is composed of insert statements combined with various queries to book appointments.

```
set @medicalno = 1048724870.3;
set @name = 'Mary Hammond';
set @appointmentDate = '2021-08-13';
set @appointmentTime = '18:30:00';
set @clinic = 1002; #this would be s
```

The first stage is two inserts that insert new tuples into patient and patient details if the patient is new to the service. The insert ignore means that if it is a repeating user the statement is ignored so that duplicate tuples are not made and an error is not thrown.

```
insert ignore into patientdetails(medicalno)
values(@medicalno);
insert ignore into patient(name, medicalNo)
values(@name, @medicalno);
```

The second Stage uses two select statements, the first retrieves the patient ID using medicalno. The second uses the view EmployeeWork to find a doctor that had the least amount of work at the clinic so that they can be assigned to the appointment.

```
set @patientID = (select patientID from patient where medicalno = @medicalno limit 1);
set @employeeID = (select employeeID
from EmployeeWork natural join employee join Clinic on Clinic.clinicID = Employee.clinicID
where Clinic.clinicID = @clinic
limit 1);
```

The final stage then inserts tuples into the appointment and admission tables to finish the transaction

```
insert into Appointment(employeeID, app_date, app_time, clinicID)
values(@employeeID, @appointmentDate,@appointmentTime,@clinic);
insert into admission(patientID, appointmentID)
values(@patientID, last_insert_id());
```

Database Concurrency and Performance

The medical Service Database can be accessed by a variety of individuals, such as:

1. Patients accessing their data or booking an appointment
2. Clinic Managers requesting information about employee workload
3. Central Management Analysing where expansion is required

This can cause issues with performance and concurrency as multiple sessions can result in phantom rows, lost updates, and dirty reads, similarly many sessions open can negatively affect the databases Performance.

Performance

To increase performance when using the database, views have been created for queries that are common or form the basis of other queries, this is seen in queries 1,2 and 5. Views help performance as it allows the DBMS to develop efficient query in advance rather than on the fly during execution.

To further increase performance the use of varchar has been limited to only when necessary, such as for comments and for columns that have data that vary in length but remain relatively static, such as street address. This prevents issues caused when a tuple has a varchar column updated with more bits than previous, this can result in the tuple needing to be relocated physically in storage if there is not enough room.

In order to increase the efficiency of searching through the patient table, the medicalno is set to be unique this means it is indexed which dramatically increases search speed this can be seen in the explain output of query 1. The Type of eq_ref means that the search was able to use the same index from the previous table in the join operation which is more efficient than sequentially searching through tuples.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	patientdetails	ALL	PRIMARY	NULL	NULL	NULL	19	
1	SIMPLE	patient	eq_ref	medicalNo	medicalNo	6	medicalservice.patientdetails.medicalNo	1	

Concurrency

Concurrency is important in a medical database as it can impact whether appointments are served, whether billing is properly covered or whether employees are quickly assigned to callouts. Any issues in concurrency particularly a lost update could result in important assistance never being delivered. Hence an isolation level of read committed should be set, this strikes a balance between performance but prevents the crucial concurrency issues of lost updates and dirty reads.

A possible dirty read could happen in Query 8 of booking an appointment, if transaction 1 created a patient, and transaction 2 was booking an appointment for the same patient a dirty read could occur, as transaction 2 would not insert into patient details and patient because those tuples exist due to transaction 1, then transaction 2 will create appointments. If transaction 1 aborts, then appointments will have been created without any corresponding patient data. Read committed prevents this from happening.

Scripts

The scripts are included separately: There are scripts for:

- Database Creation
- Insertion of patients and patient details
- Insertion of Employees, clinics and suburbs
- Insertion of Appointments and Admissions
- Insertion Of Callouts and Assignments
- Queries

Design Process

Normalization

Initial Requirements

Appointment				
PatientID	DateTime	EmployeeID	ClinicID	Location
This table is for appointments which have a patient at time and date, it also has the clinic and location of appointment.				

Employee					
EmployeeID	Name	ClinicID	Role	MangerID	Salary
This table is for the employees of the clinic how much they earn, where they work and their job title.					

Patient						
PatientID [pk]	MedicalNo.	Name	Bloodtype	Dob	sex	comment
This holds the patient's data, patientID and medical no. both uniquely identify the tuples however according to the brief one is for the internal database while the other is government mandated hence both should uniquely identify a row. However patient ID will be the primary key						

Callout

EmployeeIDs	Location	DateTime	Sender_Clinic_ID	Receiver_Clinic_ID	comments
Call out holds information about the location of the destination of the callout, who attended the callout and where the ambulance originated from and the clinic they may have gone to					

1st Normal Form

Appointment							
PatientID [pk]	EmployeeID	Date [pk]	Time [pk]	ClinicID [pk]	streetAdd	Suburb	Postcode
The location and DateTime columns meant that the entry had multiple parts of information, so this has been split up into sperate columns to achieve first normal form.							

Employee				
EmployeeID	ClinicID	Role	name	managerID
No changes				

Patient						
PatientID [pk]	medicalNo	Name	Bloodtype	Dob	sex	comments
No changes						

Callout								
EmployeeID [pk]	Street add [pk]	Suburb	Postcode [pk]	Date [pk]	Time [pk]	Sender_ClinicID	Receiver_ClinicID	comment
Location and DateTime have been split as well as EmployeeID only holds one employee and rows would have to be duplicated for each employee in order to conform to 1st normal form.								

2nd Normal Form

Appointment				
patientID	EmployeeID	Date [pk]	Time [pk]	ClinicID [pk]
There were two functional dependencies in this table, as EmployeeID and, date and time were dependant on the patientID while the streetAdd, suburb and postcode were dependant on the ClinicID part of the primary key. Hence clinic has been given its own table.				

Employee			
EmployeeID [pk]	ClinicID	Role	Name
No changes for normal form, however with creation of clinics, the managerID has been moved to the clinic table as the manager is the manager of a clinic, and it would be more appropriate this way.			

Patient						
PatientID [pk]	medicalNo	Name	Bloodtype	Dob	sex	comments
No changes with this table.						

Callout								
EmployeeID [pk]	Street add [pk]	Suburb	Postcode [pk]	Date [pk]	Time [pk]	Sender_ClinicID	Receiver_ClinicID	comment
Surburb is functionally dependent on postcode in this table and transitively depend anent in other tables so this will be sovled in the 3rd normal form segment.								

Clinic			
ClinicID [pk]	streetAdd	Suburb	postcode
This is the new clinic table which holds the locations of the clinics			

3rd Normal Form

Appointment				
patientID	EmployeeID	Date [pk]	Time [pk]	ClinicID [pk]
No changes				

Employee			
EmployeeID [pk]	ClinicID	Role	Name
No changes			

Patients		
PatientID [pk]	Name	Medical No.
Medical history is associated with the medical no. and hence this was a transitive dependency and has been given its own table.		

Clinic		
ClinicID [pk]	streetAdd	postcode
Suburb and potocode are a transitive dependency and so have been given their own table		

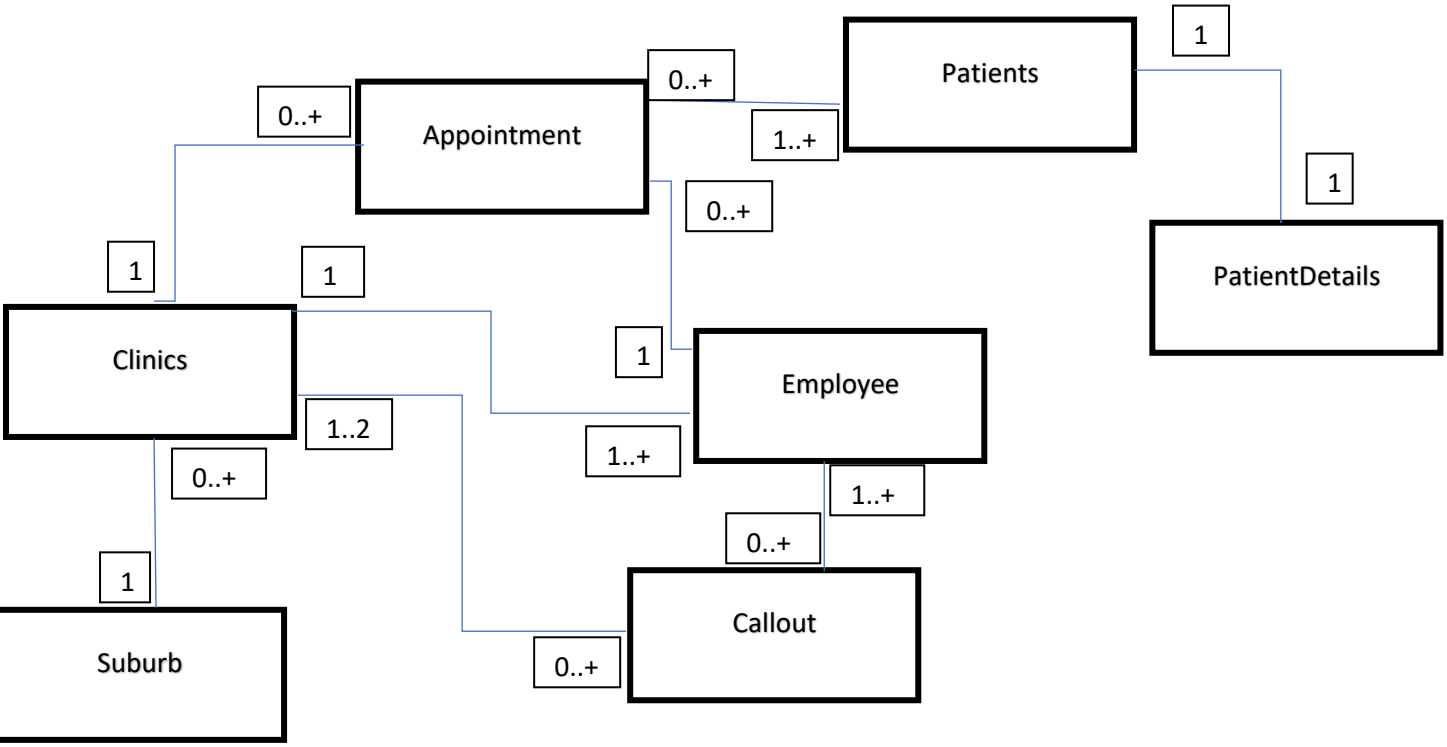
PatientDetails				
medicalNo. [pk]	Dob	Bloodtype	sex	comments
This holds the medical services records of the medical history retrieved from a hypothetical government database.				

Suburb	
Postcode [pk]	name
Mapping of postcode to suburb	

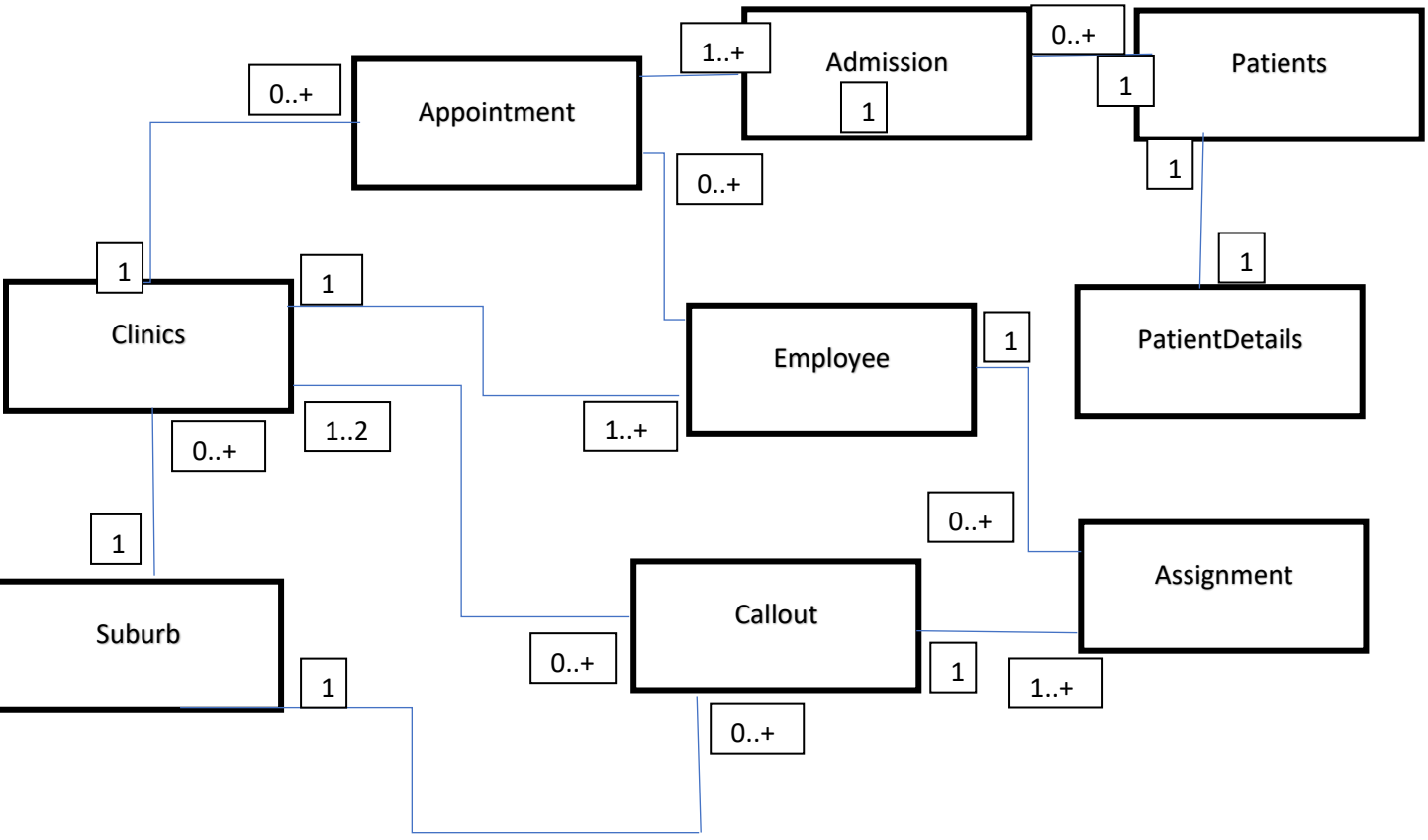
Callout							
EmployeeID [pk]	Street add [pk]	Postcode [pk]	Date [pk]	Time [pk]	Sender_ClinicID	Receiver_ClinicID	comment
Suburb has been moved							

ER Modelling

Initial requirements



Eliminating many to many relationships



Final Entity Changes

In the entity diagrams the ability for multiple patients to attend an appointment has not been reflected. The necessary changes have been made, as well as adding the assignments and admissions table.

Appointment				
AppointmentID [pk]	EmployeeID	Date	Time	ClinicID

Admission	
AppointmentID	PatientID

Callout							
CalloutID [pk]	Street add	Postcode	Date	Time	Sender_ClinicID	Receiver_ClinicID	comment

Assignment	
EmployeeID	CalloutID