Technical University of Denmark

Database Systems

Project

Hospital Database System

By:
Hsuan Liang, s166126
Vibeke Hvidegaard Petersen, s143881
Pierre-William Breau, s162016
Jonathan Sim, s166121

Lecturer:
Flemming Schmidt

06-04-2017

Technical University of Denmark

Contents

1 Statement of Requirements

The database in this project is a model of a hospital. The hospital is organised in different departments. Each of the departments has rooms and offices and employs general staff and one or more doctors. Each office has a room and belongs a building. Each room has a maximum capacity on the number of patients. The staff are named and have a job title and a salary. The doctors are named and has a specialization and a salary. The patients have a name, gender and birth date. The hospital has a hospitalization record keeping track of the hospitalization of each patient, their admission and release date, illness, which room and doctor they were or are assigned to. The patients has appointments with the assigned doctor.

The purpose of the database is to keep track of patients in the hospital making sure that each of them is assigned to a room and a doctor. The database serves as a tool for planning the hospitalizations of patients. The users of the database are the staff of the hospital. They can be grouped into staff with the authority of hospitalizing patients, mainly doctors, and staff without this authority.

2 Entity-Relationship Diagram

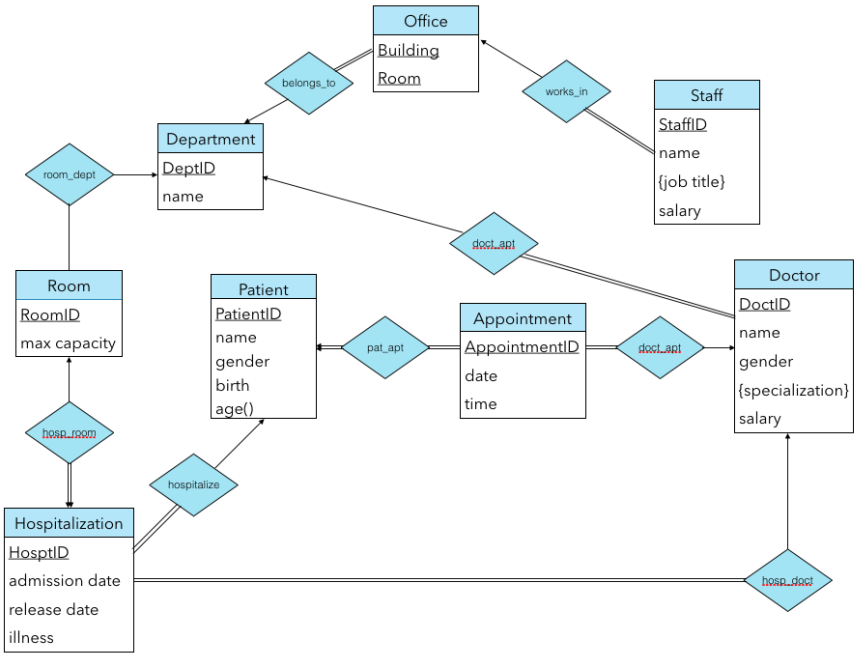


Figure 1: Entity-Relationship (E-R) Diagram.

Figure 1 illustrates the E-R Diagram for Hospital Database Schema. The underscored attributes in each entity are the primary keys. There are no weak entities in the schema since each entity has its own primary key. In entity *Patient*, attribute **age()** is defined as a derived attribute as it can be calculated from **birth**. In entity *Doctor*, attribute **{specialization}** is defined as a multivalued attribute since one doctor can have multiple specializations. Now, we shall look at the relationships defined.

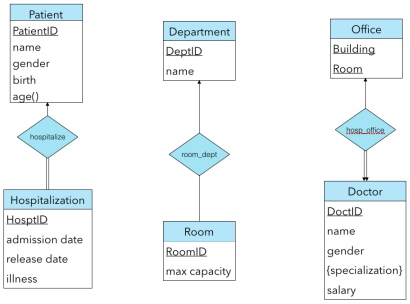


Figure 2: One-to-many relationship (left, mid) and one-to-one relationship.

The most common relationship in our schema is the *One-to-many relationship* with *total participation* on the many side (Figure 2, left). To interpret this relationship, we take the relationship of **Patient** and **Hospitalization** for example:

> A patient can zero or many hospitalization records.
>
> Every hospitalization record must belong to one patient.

In Figure 2 (mid), we see the both **Department** and **Room** is *partially participating* the relationship, which can be interpreted as:

> A patient can zero or many hospitalization records.
>
> Every hospitalization record must belong to one patient.

If we assign an office to each doctor, we would have the relationship as shown on the right of Figure 2. It illustrates an *One-to-one relationship* between **Doctor** and **Office**, and it can be interpreted as:

> One department can manage zero or many rooms.
>
> Not all departments has a room.
>
> Not all rooms belong to a department.

Finally, we look at the relationship between patients and doctors.

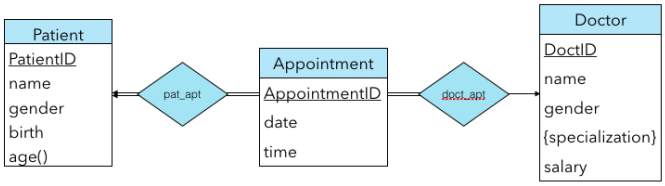


Figure 3: Relationship between Patient and Doctor.

Figure 3 illustrates the *Many-to-many relationship* between **Patient** and **Doctor**, which is break down into two *One-to-many relationships* and one intermediate entity, **Appointment**.

The relationship can be interpreted as:

> Every patient have zero or many appointments with one or many doctors.
> Every appointment must relate one patient and one doctor.
> Not all doctors have appointments.

3 MySQL Workbench Model

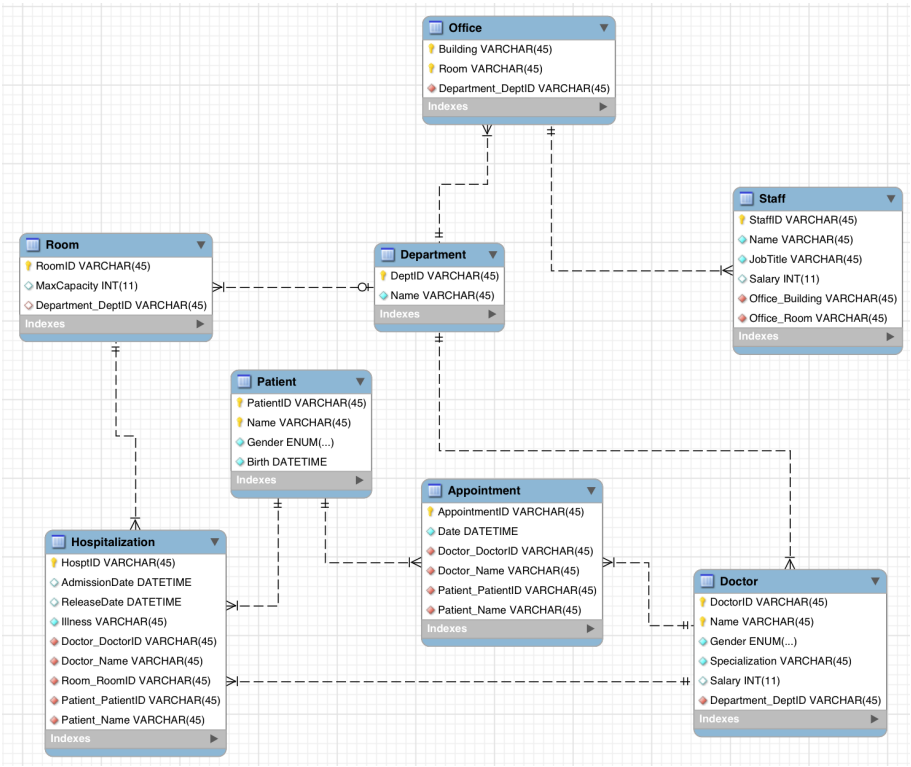


Figure 4: MySQL workbench model of the Hospital schema.

3.1 Relationships

Since there are no weak entities in our database schema, all relationships are *non-identifying relationships*, and in particular, *one-to-many relationship* as discussed in the previous section. The crow's foot end of a relationship marks that the cardinality of the corresponding side is "many". The foreign key of **Room** to **Department** can be NULL as a room can belong to no department. Other foreign keys are set to NOT NULL. **Office** is derived from **Staff** after normalization to 3NF. **Appointment** is created to solve the *many-to-many relationship* between **Patient** and **Doctor**, and thus besides the primary key and foreign keys, it only contain the *Date* attribute to specify the date and time of appointment. **Hospitalization** is also created to define another relationship between **Patient** and **Doctor**.

3.2 Attributes

All tables contain a primary key, named "— ID", to make sure all records have a unique key. *Gender* attribute of doctors and patients of *ENUM('Male', 'Female')* type, in order to maintain consistency in the entries and to avoid typos while entering new rows. It is also a fixed field for most people, and thus, choosing an ENUM type over a character string would reduce the storage as well.

4 Database Instance

In this section the data for all the different tables are listed along with a short description of the keys. Each of the tables are acquired by the SELECT query. For instance the Office table can be acquired by the following statement:

SELECT* FROM Office;

Building	Room	Department_Dept...
Main	1147	001
Main	1153	001
Doherty Institute	104	002
Main	1148	002
Main	1149	002
Doherty Institute	209	003
African Centre for Gene Technologies	104	004
African Centre for Gene Technologies	105	004
NorthWestern Surgical Center	323	005
NorthWestern Surgical Center	326	005
NULL	NULL	NULL

Figure 5: Data for the Office Table

Figure 5 shows the office table. The primary key of the office table is 'Building' and 'Room', while 'Department_Dept' is a foreign key which relates the office table to the department table.

DeptID	Name
001	Cardiology
002	Pediatric
003	Neonatal
004	Oncology
005	Orthopaedics

Figure 6: Data for the Department Table

In Figure 6 the department table is shown. 'DeptID' is the primary key.

StaffID	Name	JobTitle	Salary	Office_Building	Office_Room
s13114	Jack Dorsey	Executive Assistant	50000	African Centre for Gene Technologies	104
s13157	James Gorden	Floor Tech	45000	Main	1147
s13622	Kieth Hsieh	Research Compliance Analyst	60000	NorthWestern Surgical Center	323
s13714	Chris Manning	Research Compliance Analyst	58000	NorthWestern Surgical Center	323
s13904	Reed Hastings	Executive Assistant	54500	NorthWestern Surgical Center	326
s14156	Greg Jenson	Junior Assistant	40000	Main	1153
s14426	Meg Whitman	Nurse Manager	45000	African Centre for Gene Technologies	105
s14483	Floyd Gorden	Nurse Manager	47000	African Centre for Gene Technologies	105
s14523	Lloyd Blankfein	Floor Tech	45000	NorthWestern Surgical Center	326
s14546	Mark Parker	Research Compliance Analyst	70000	Main	1148
s14549	Michael Perry	Research Compliance Analyst	70000	Main	1149
s14690	Mary T. Barra	Executive Assistant	49500	Main	1148
s14934	Denise Morris...	Junior Assistant	40000	Doherty Institute	209
s15047	David Chen	Senior Assistant	51000	Doherty Institute	104
s15190	George Peter...	Junior Assistant	42000	Doherty Institute	104
NULL	NULL	NULL	NULL	NULL	NULL

Figure 7: Data for the Staff Table

The staff table, shown in Figure 7, has 'StaffID' as primary key, while both 'Office_Building' and 'Office_Room' are foreign keys. The two foreign keys relates the staff table to the office table.

RoomID	MaxCapacity	Department_Dept...
22A	4	002
23B	8	003
24A	4	001
24C	16	004
25A	4	001
25B	8	005
NULL	NULL	NULL

Figure 8: Data for the Room Table

In Figure 8 the room table is seen. 'RoomID' is primary key, while 'Department_DeptID' is foreign key. 'Department_DeptID' relates the room table to the department table.

PatientID	Name	Gender	Birth
P001	Sarah Wu	Female	1992-04-05 00:00:00
P002	Joe Rosberg	Male	1999-07-15 00:00:00
P003	John Woodgate	Male	1979-03-21 00:00:00
P004	Alina Ott	Female	1985-11-28 00:00:00
P005	Jasmine Gomez	Female	1952-06-15 00:00:00
P006	Toby Verthongen	Male	1967-04-25 00:00:00
NULL	NULL	NULL	NULL

Figure 9: Data for the Patient Table

Figure 9 shows the patient table. The primary keys are 'PatientID' and 'Name'.

DoctorID	Name	Gender	Specialization	Salary	Department_Dept...
d124567	Lauren Holmes	Female	Cardiology	90000	001
d125683	Charles Winston	Male	Neonatal	100000	003
d138954	Paul Wright	Male	Pediatric	80000	002
d143267	Carrie Miller	Female	Oncology	90000	004
d162198	Andrew Lowe	Male	Orthopaedics	80000	005
NULL	NULL	NULL	NULL	NULL	NULL

Figure 10: Data for the Doctor Table

Figure 10 shows the doctor table, where the primary keys are 'DoctorID' and 'Name'. The foreign key 'Department_DeptID' relates the doctor table to the department table.

Appointment...	Date	Doctor_Doctor...	Doctor_Name	Patient_PatientID	Patient_Name
A001	2007-04-05 00:00:00	d124567	Lauren Holmes	P005	Jasmine Gomez
A002	2013-06-01 00:00:00	d162198	Andrew Lowe	P001	Sarah Wu
A003	2011-03-01 00:00:00	d143267	Carrie Miller	P001	Sarah Wu
A004	2004-04-11 00:00:00	d124567	Lauren Holmes	P006	Toby Verthongen
A005	2002-02-02 00:00:00	d143267	Carrie Miller	P004	Alina Ott
A006	2000-10-09 00:00:00	d162198	Andrew Lowe	P003	John Woodgate
NULL	NULL	NULL	NULL	NULL	NULL

Figure 11: Data for the Appointment Table

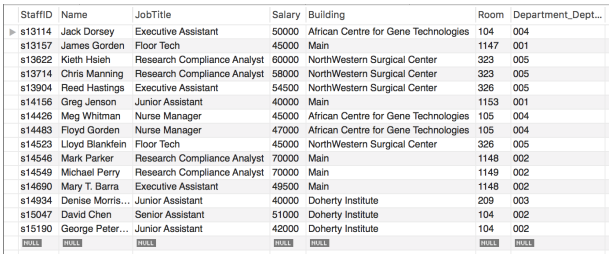
In Figure 11 the appointment table is shown. The primary key is 'AppointmentID', while there are four foreign keys: 'Doctor_DoctorID','Doctor_Name','Patient_PatientID' and 'Patient_Name'. 'Doctor_DoctorID' and 'Doctor_Name' relates the appointment table to the doctor table, while 'Patient_PatientID' and 'Patient_Name' relates the appointment table to the Patient table.

HosptID	AdmissionDate	ReleaseDate	Illness	Doctor_Doctor...	Doctor_Name	Room_RoomID	Patient_PatientID	Patient_Name
H001	2005-04-05 00:00:00	2005-04-10 00:00:00	Pneumonia	d138954	Paul Wright	22A	P002	Joe Rosberg
H002	2011-06-01 00:00:00	2011-06-05 00:00:00	Broken Collarbone	d162198	Andrew Lowe	25B	P001	Sarah Wu
H003	2006-01-24 00:00:00	2006-04-05 00:00:00	Leukemia	d143267	Carrie Miller	24C	P001	Sarah Wu
H004	2004-04-11 00:00:00	2004-04-26 00:00:00	Cardiac Arrest	d124567	Lauren Holmes	25A	P006	Toby Verthongen
H005	2002-07-02 00:00:00	2002-07-05 00:00:00	Childbirth	d125683	Charles Winston	23B	P004	Alina Ott
H006	2008-10-09 00:00:00	2008-10-12 00:00:00	Fractured Skull	d162198	Andrew Lowe	25B	P003	John Woodgate
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 12: Data for the Hospitalization Table

The hospitalization table is shown in Figure 12. The primary key is 'HosptID'. The foreign keys are 'Doctor_DoctorID', 'Doctor_Name', 'Room_RoomID', 'Patient_PatientID' and 'Patient_Name'. 'Doctor_DoctorID' and 'Doctor_Name' relates the hospitalization table with the doctor table. 'Room_RoomID' relates it to the room table. Finally, 'Patient_PatientID' and 'Patient_Name' relates the hospitalization table to the patient table.

5 Normalization



StaffID	Name	JobTitle	Salary	Building	Room	Department_Dept...
s13114	Jack Dorsey	Executive Assistant	50000	African Centre for Gene Technologies	104	004
s13157	James Gorden	Floor Tech	45000	Main	1147	001
s13622	Kieth Hsieh	Research Compliance Analyst	60000	NorthWestern Surgical Center	323	005
s13714	Chris Manning	Research Compliance Analyst	58000	NorthWestern Surgical Center	323	005
s13904	Reed Hastings	Executive Assistant	54500	NorthWestern Surgical Center	326	005
s14156	Greg Jenson	Junior Assistant	40000	Main	1153	001
s14426	Meg Whitman	Nurse Manager	45000	African Centre for Gene Technologies	105	004
s14483	Floyd Gorden	Nurse Manager	47000	African Centre for Gene Technologies	105	004
s14523	Lloyd Blankfein	Floor Tech	45000	NorthWestern Surgical Center	326	005
s14546	Mark Parker	Research Compliance Analyst	70000	Main	1148	002
s14549	Michael Perry	Research Compliance Analyst	70000	Main	1149	002
s14690	Mary T. Barra	Executive Assistant	49500	Main	1148	002
s14934	Denise Morris...	Junior Assistant	40000	Doherty Institute	209	003
s15047	David Chen	Senior Assistant	51000	Doherty Institute	104	002
s15190	George Peter...	Junior Assistant	42000	Doherty Institute	104	002
NULL	NULL	NULL	NULL	NULL	NULL	NULL

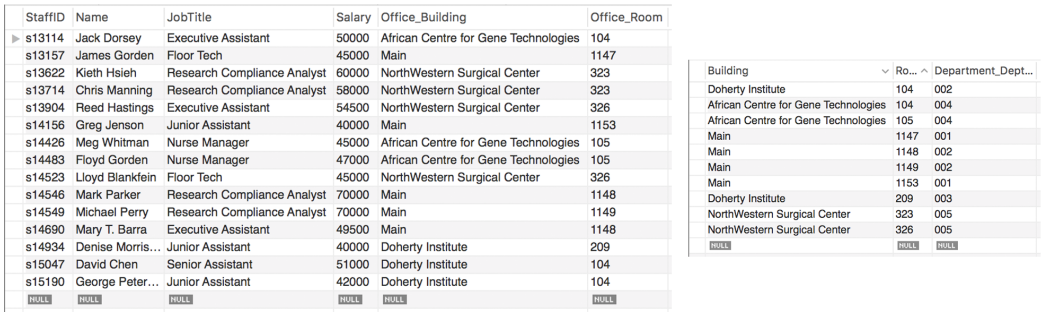
Figure 13: Staff table in 2NF.

As shown in Figure 13, the office of each staff is dependant on their staffID, and each office belongs to only one department. Yet, a department may have offices in different buildings, e.g., department 002 () have offices in the "Main Building" and "Doherty Institute". That said, **Department_DeptID** in Figure 13 is dependant on the primary key, **StaffID**, but also depends transitively via **(Building, Room)**. Now, we normalize the table by the projection operations:

$$Staff' \equiv \prod_{StaffID,\, Name,\, JobTitle,\, Salary,\, Building,\, Room}(Staff)$$

$$Office \equiv \prod_{Building,\, Room,\, Department\_DeptID}(Staff)$$

where $Staff'$ denotes the new $Staff$ table. Note that we rename **Building** and **Room** as **Office_Building** and **Office_Room** respectively in the new **Staff** table to specify that they are foreign keys referencing to the **Office** table.



StaffID	Name	JobTitle	Salary	Office_Building	Office_Room
s13114	Jack Dorsey	Executive Assistant	50000	African Centre for Gene Technologies	104
s13157	James Gorden	Floor Tech	45000	Main	1147
s13622	Kieth Hsieh	Research Compliance Analyst	60000	NorthWestern Surgical Center	323
s13714	Chris Manning	Research Compliance Analyst	58000	NorthWestern Surgical Center	323
s13904	Reed Hastings	Executive Assistant	54500	NorthWestern Surgical Center	326
s14156	Greg Jenson	Junior Assistant	40000	Main	1153
s14426	Meg Whitman	Nurse Manager	45000	African Centre for Gene Technologies	105
s14483	Floyd Gorden	Nurse Manager	47000	African Centre for Gene Technologies	105
s14523	Lloyd Blankfein	Floor Tech	45000	NorthWestern Surgical Center	326
s14546	Mark Parker	Research Compliance Analyst	70000	Main	1148
s14549	Michael Perry	Research Compliance Analyst	70000	Main	1149
s14690	Mary T. Barra	Executive Assistant	49500	Main	1148
s14934	Denise Morris...	Junior Assistant	40000	Doherty Institute	209
s15047	David Chen	Senior Assistant	51000	Doherty Institute	104
s15190	George Peter...	Junior Assistant	42000	Doherty Institute	104
NULL	NULL	NULL	NULL	NULL	NULL

Building	Ro...	Department_Dept...
Doherty Institute	104	002
African Centre for Gene Technologies	104	004
African Centre for Gene Technologies	105	004
Main	1147	001
Main	1148	002
Main	1149	002
Main	1153	001
Doherty Institute	209	003
NorthWestern Surgical Center	323	005
NorthWestern Surgical Center	326	005
NULL	NULL	NULL

Figure 14: Normalized Staff and Office table in 3NF.

Figure 14 shows the normalized schema, with the following properties:

Staff(StaffID, Name, JobTitle, Salary, Office_Building, Office_Room)
Foreign key(Office_Building, Office_Room) referencing Office(Building, Room)

and

Office(Building, Room, Department_DeptID)
Foreign key(Department_DeptID) referencing Department(DeptID)

To show the relation schemas have the valid primary keys, we list the functional dependencies and apply the **Armstrong's Rules**.

Functional dependencies :

StaffID → Name, JobTitle, Salary, Building, Room, Department_DeptID

{Building, Room} → Department_DeptID

- StaffID → StaffID — Self-Determination
- StaffID → Name — Decomposition
- StaffID → JobTitle — Decomposition
- StaffID → Salary — Decomposition
- StaffID → Building — Decomposition
- StaffID → Room — Decomposition
- StaffID → Name, JobTitle, Salary, Building, Room — Union
- {Building, Room} → Building, Room — Self-Determination
- {Building, Room} → Building, Room, Departmet_DeptID — Union

The above derivation gives

Staff'(<u>StaffID</u>, Name, JobTitle, Salary, Building, Room)

Office(<u>Building</u>, <u>Room</u>, Department_DeptID)

which is identical to the results above. In the original function dependencies, we see

StaffID → Building, Room

{Building, Room} → Department_DeptID

$\implies$ StaffID → Department_DeptID by **Transitivity**

and thus, we show the original table is in 2NF. Finally, though trivial, Armstrong's Rules validate our choice of primary keys in both normalized tables in 3NF.

6 SQL Table Modifications

PatientID	Name	Gender	Birth
P001	Sarah Wu	Female	1992-04-05 00:00:00
P002	Joe Rosberg	Male	1999-07-15 00:00:00
P003	John Woodgate	Male	1979-03-21 00:00:00
P004	Alina Ott	Female	1985-11-28 00:00:00
P005	Jasmine Gomez	Female	1952-06-15 00:00:00
P006	Toby Verthongen	Male	1967-04-25 00:00:00
NULL	NULL	NULL	NULL

Figure 15: SELECT * FROM Patient.

Figure 15 shows the original patient records. We will now modify this table using the following queries.

6.1 INSERT

A sample insert can be done by

```
INSERT Patient
VALUES ('P007','Steven Eriksen','Male',STR_TO_DATE('18/03/1992', '%d/%m/%Y'));
```

or

```
INSERT Patient (PatientID, Name, Gender, Birth)
VALUES ('P007','Steven Eriksen','Male',STR_TO_DATE('18/03/1992', '%d/%m/%Y'));
```

PatientID	Name	Gender	Birth
P001	Sarah Wu	Female	1992-04-05 00:00:00
P002	Joe Rosberg	Male	1999-07-15 00:00:00
P003	John Woodgate	Male	1979-03-21 00:00:00
P004	Alina Ott	Female	1985-11-28 00:00:00
P005	Jasmine Gomez	Female	1952-06-15 00:00:00
P006	Toby Verthongen	Male	1967-04-25 00:00:00
P007	Steven Eriksen	Male	1992-03-18 00:00:00
NULL	NULL	NULL	NULL

Figure 16: Insert single row into *Patient*.

Now, we could also insert multiple records at a time:

```
INSERT Patient (PatientID, Name, Gender, Birth) VALUES
('P008','John Liu','Male',STR_TO_DATE('11/03/1990', '%d/%m/%Y'))
('P009','Jenna Svensson','Female',STR_TO_DATE('04/07/1984', '%d/%m/%Y'))
('P010','Viktor Christiansen','Male',STR_TO_DATE('11/08/1975', '%d/%m/%Y'));
```

PatientID	Name	Gender	Birth
P001	Sarah Wu	Female	1992-04-05 00:00:00
P002	Joe Rosberg	Male	1999-07-15 00:00:00
P003	John Woodgate	Male	1979-03-21 00:00:00
P004	Alina Ott	Female	1985-11-28 00:00:00
P005	Jasmine Gomez	Female	1952-06-15 00:00:00
P006	Toby Verthongen	Male	1967-04-25 00:00:00
P007	Steven Eriksen	Male	1992-03-18 00:00:00
P008	John Liu	Male	1990-03-11 00:00:00
P009	Jenna Svensson	Female	1984-07-04 00:00:00
P010	Viktor Christiansen	Male	1975-08-11 00:00:00
NULL	NULL	NULL	NULL

Figure 17: Insert multiple rows into *Patient*.

6.2 UPDATE

Suppose we have an error entry in the birth year of patient 'P008', we could update the record by:

```
UPDATE Patient
SET Birth = STR_TO_DATE('04/07/1977', '%d/%m/%Y')
WHERE PatientID = 'P009';
```

PatientID	Name	Gender	Birth
P001	Sarah Wu	Female	1992-04-05 00:00:00
P002	Joe Rosberg	Male	1999-07-15 00:00:00
P003	John Woodgate	Male	1979-03-21 00:00:00
P004	Alina Ott	Female	1985-11-28 00:00:00
P005	Jasmine Gomez	Female	1952-06-15 00:00:00
P006	Toby Verthongen	Male	1967-04-25 00:00:00
P007	Steven Eriksen	Male	1992-03-18 00:00:00
P008	John Liu	Male	1990-03-11 00:00:00
▶ P009	Jenna Svensson	Female	1977-07-04 00:00:00
P010	Viktor Christiansen	Male	1975-08-11 00:00:00
NULL	NULL	NULL	NULL

Figure 18: Update the birth year of patient P008.

Now, we need a shorter representation for the *Gender* column simply by replacing **Male** with **M** and **Female** with **F**. We could perform two update statements as follows,

```
UPDATE Patient SET Gender = 'F' WHERE Gender = 'Female';
UPDATE Patient SET Gender = 'M' WHERE Gender = 'Male';
```

However, we could also use the **CASE** keyword to handle conditioning in a single statement. The one-line update statement is as follows

```
UPDATE Patient
SET Gender = CASE WHEN Gender = 'Male' THEN 'M' ELSE 'F' END;
```

PatientID	Name	Gender	Birth
▶ P001	Sarah Wu	F	1992-04-05 00:00:00
P002	Joe Rosberg	M	1999-07-15 00:00:00
P003	John Woodgate	M	1979-03-21 00:00:00
P004	Alina Ott	F	1985-11-28 00:00:00
P005	Jasmine Gomez	F	1952-06-15 00:00:00
P006	Toby Verthongen	M	1967-04-25 00:00:00
P007	Steven Eriksen	M	1992-03-18 00:00:00
P008	John Liu	M	1990-03-11 00:00:00
P009	Jenna Svensson	F	1977-07-04 00:00:00
P010	Viktor Christiansen	M	1975-08-11 00:00:00
NULL	NULL	NULL	NULL

Figure 19: Insert multiple rows into *Patient*.

The resulting table of both approaches is shown in Figure 19. Note that the attribute **Gender** is of **ENUM('Male', 'Female')** type, and thus we should alter the table before and after the update statements. The code for altering the table is:

```
ALTER TABLE Patient
CHANGE Gender Gender ENUM ('Male', 'Female', 'M', 'F');
UPDATE ...;
ALTER TABLE Patient
CHANGE Gender Gender ENUM ('M', 'F');
```

6.3 DELETE

To delete a particular patient record, we could query his or her *PatientID*:

```
DELETE FROM Patient WHERE PatientID='P008';
```

PatientID	Name	Gender	Birth
P001	Sarah Wu	F	1992-04-05 00:00:00
P002	Joe Rosberg	M	1999-07-15 00:00:00
P003	John Woodgate	M	1979-03-21 00:00:00
P004	Alina Ott	F	1985-11-28 00:00:00
P005	Jasmine Gomez	F	1952-06-15 00:00:00
P006	Toby Verthongen	M	1967-04-25 00:00:00
P007	Steven Eriksen	M	1992-03-18 00:00:00
P009	Jenna Svensson	F	1977-07-04 00:00:00
P010	Viktor Christiansen	M	1975-08-11 00:00:00
NULL	NULL	NULL	NULL

Figure 20: Delete patient P008.

Now, suppose we would like to remove all patients' records whose birth year is before 1980, the delete statement is:

```
DELETE FROM Patient WHERE Birth < '1980-01-01';
```

PatientID	Name	Gender	Birth
P001	Sarah Wu	F	1992-04-05 00:00:00
P002	Joe Rosberg	M	1999-07-15 00:00:00
P004	Alina Ott	F	1985-11-28 00:00:00
P007	Steven Eriksen	M	1992-03-18 00:00:00
NULL	NULL	NULL	NULL

Figure 21: Delete all patients born before 1980.

7 SQL Data Queries

7.1 SELECT with ORDER BY

Use of ORDER BY to show all patients ordered by date of birth.

```
SELECT Name, PatientID as 'Patient ID', Birth as 'Date of Birth', Gender
  FROM Patient
  ORDER BY Birth;
```

Name	Patient ID	Date of Birth	Gender
Jasmine Gomez	005	1952-06-15 00:00:00	Female
Toby Verthongen	006	1967-04-25 00:00:00	Male
John Woodgate	003	1979-03-21 00:00:00	Male
Alina Ott	004	1985-11-28 00:00:00	Female
Sarah Wu	001	1992-04-05 00:00:00	Female
Joe Rosberg	002	1999-07-15 00:00:00	Male

Figure 22: SELECT statement with ORDER BY.

7.2 SELECT with GROUP BY

Use of GROUP BY to count how many hospitalizations have taken place in each room.

```
Select Room_RoomID as 'Room #', Count(Patient_PatientID) as 'Number of hospitalizations in room'
  From Hospitalization
  Group by Room_RoomID;
```

Room #	Number of hospitalizations in room
22A	1
23B	1
24C	1
25A	1
25B	2

Figure 23: SELECT statement with GROUP BY.

7.3 Joins

Use of INNER JOIN on Patient and Hospitalization to show all known patient illnesses and their year of occurence

```
SELECT Patient.Name, Patient.PatientID AS 'Patient ID', Hospitalization.Illness AS 'Illness',
  Year(Hospitalization.ReleaseDate) AS 'Year of occurence', Hospitalization.Doctor_Name AS 'Assigned Doctor'
  FROM Patient
  INNER JOIN Hospitalization
  ON Hospitalization.Patient_PatientID=Patient.PatientID;
```

Name	Patient ID	Illness	Year of occurence	Assigned Doctor
Joe Rosberg	P002	Pneumonia	2005	Paul Wright
Sarah Wu	P001	Broken Collarbone	2011	Andrew Lowe
Sarah Wu	P001	Leukemia	2006	Carrie Miller
Toby Verthongen	P006	Cardiac Arrest	2004	Lauren Holmes
Alina Ott	P004	Childbirth	2002	Charles Winston
John Woodgate	P003	Fractured Skull	2008	Andrew Lowe

Figure 24: SELECT statement with JOIN.

USE of INNER JOIN on Department and Office to show all offices owned by the Pediatrics department

```
SELECT Name as 'Department Name', Office.Building as 'Building', Office.Room as 'Office'
  FROM Department
  INNER JOIN Office
  ON DeptID = Office.Department_DeptID
  WHERE Name = 'Pediatric';
```

Department Na...	Building	Office
Pediatric	Doherty Institute	104
Pediatric	Main	1148
Pediatric	Main	1149

Figure 25: SELECT statement with JOIN.

8 Data Queries in Formal Languages

In this section, we will show the **Relational Algebra**, **Tuple Calculus** and **Domain Calculus** for the following SELECT statement:

```
SELECT S.StaffID, S.Name, S.JobTitle, D.Name as Department, O.Building
        FROM Staff S INNER JOIN Office O ON S.Office_Building=O.Building AND S.Office_Room=O.Room
        INNER JOIN Department D ON O.Department_DeptID=D.DeptID
        WHERE O.Building='Main' OR O.Building='Doherty Institute';
```

which results in

StaffID	Name	JobTitle	Department	Building
s13157	James Gorden	Floor Tech	Cardiology	Main
s14156	Greg Jenson	Junior Assistant	Cardiology	Main
s15047	David Chen	Senior Assistant	Pediatric	Doherty Institute
s15190	George Peterson	Junior Assistant	Pediatric	Doherty Institute
s14546	Mark Parker	Research Compliance Analyst	Pediatric	Main
s14690	Mary T. Barra	Executive Assistant	Pediatric	Main
s14549	Michael Perry	Research Compliance Analyst	Pediatric	Main
s14934	Denise Morrison	Junior Assistant	Neonatal	Doherty Institute

Figure 26: SELECT statement joining **Staff**, **Office** and **Department**.

This query provides us with an overview of the basic information of staffs from the *Main building* and *Doherty Institute*. Since there two columns, *Staff.Name* and *Department.Name*, has the same name, inner joins are used instead to prevent ambiguity during natural joins.

8.1 Relational Algebra

The statement is equivalent to query in the natural join of *Staff*, *Office* and *Department*. Thus, we have

$$\prod_{S.StaffID,\ S.Name,\ S.JobTitle,\ D.Name,\ O.Building} (\sigma_{O.Building=\text{'Main'} \vee O.Building=\text{'Doherty Institute'}}(\rho_S(Staff) | \times | \rho_O(Office) | \times | \rho_D(Department)))$$

Note that all tables are renamed for simplicity.

8.2 Tuple Calculus

The statement is equivalent to query in the natural join of *Staff*, *Office* and *Department*. Thus, we have

$$\begin{aligned}\{t \mid \exists s \in &\text{Staff} \quad \exists o \in \text{Office} \quad \exists d \in \text{Department}(\\ & t[StaffID] = s[StaffID] \\ & \wedge t[Name] = s[Name] \\ & \wedge t[JobTitle] = s[JobTitle] \\ & \wedge t[Department] = d[Name] \\ & \wedge t[Building] = o[Building] \\ & \wedge s[Office\_Building] = o[Building] \\ & \wedge s[Office\_Room] = o[Room] \\ & \wedge o[Department\_DeptID] = d[DeptID] \\ & \wedge (o[Building] = \text{'Main'} \vee o[Building] = \text{'Doherty Institute'}))\}\end{aligned}$$

The first five clauses in the predicate specify the resulting free variable t, and the last one specify the condition we made to screen out staffs not in the *Main building* or *Doherty Institute*. The clauses in between are the columns we join on.

8.3 Domain Calculus

The statement is equivalent to query in the natural join of *Staff*, *Office* and *Department*. Thus, we have

$$\begin{aligned}\{< sid,\ name,\ job,\ dept,\ b > \mid & \\ & \exists r(< sid,\ name,\ job,\ b,\ r > \in \text{Staff} \\ & \wedge \exists deptid(< b,\ r,\ deptid > \in \text{Office} \\ & \wedge < deptid,\ dept > \in \text{Department}) \\ & \wedge (b = \text{'Main'} \vee b = \text{'Doherty Institute'}))\}\end{aligned}$$

Similar to tuple calculus, we join the tables using b, r, *deptid* and specify the condition ($b =$ 'Main' $\vee\ b =$ 'Doherty Institute').

9 SQL Programming

9.1 Function

To find if a room if vacant for a new patient, we can simply define a function as follows:

```
DROP FUNCTION IF EXISTS RemainCapacity;
DELIMITER //
CREATE FUNCTION RemainCapacity(rID VARCHAR(45))
    RETURNS INT
    BEGIN
    DECLARE max INT;
    DECLARE cur INT;
    SELECT MaxCapacity INTO @max FROM Room
    WHERE RoomID=rID;
    SELECT COUNT(*) INTO @cur FROM Hospitalization
    WHERE Room_RoomID=rID
    AND CURTIME() BETWEEN AdmissionDate AND  ReleaseDate;
    RETURN @max - @cur;
```

```
    END; //
DELIMITER ;
```

Using the two variables *max* and *cur*, we get the remaining capacity at present time, and to use the function, simply run:

```
SELECT RemainCapacity('23B');
```

which returns an integer.

9.2 Procedure

Now that we are easily accessible to the current capacity of rooms for patients, we can create a procedure that automatically assigns rooms to new patients and inserts new hospitalization records accordingly.

```
DROP PROCEDURE IF EXISTS FindVacantRoom;
DELIMITER //
CREATE PROCEDURE FindVacantRoom(IN patID VARCHAR(45), IN doctID VARCHAR(45), IN
    illness VARCHAR(45))
  BEGIN
  DECLARE room VARCHAR(45);
  DECLARE dept VARCHAR(45);
  DECLARE N_hospt INT;
  SELECT Department_DeptID INTO dept FROM Doctor
  WHERE DoctorID=doctID LIMIT 1;
  SELECT RoomID INTO room FROM Room
  WHERE Department_DeptID=dept
  AND RemainCapacity(RoomID) > 0 LIMIT 1;
  SELECT COUNT(*) INTO N_hospt FROM Hospitalization;
  INSERT INTO Hospitalization
    (HosptID, AdmissionDate, Illness,
      Doctor_DoctorID, Doctor_Name, Room_RoomID, Patient_PatientID, Patient_Name)
     VALUES
    (CONCAT('H', LPAD(N_hospt + 1, 3, '0')), CURTIME(), illness,
    doctID, (SELECT Name FROM Doctor WHERE DoctorID=doctID),
    room, patID, (SELECT Name FROM Patient WHERE PatientID=patID));
END; //
DELIMITER ;
```

Assume *Toby Verthongen*, with patient ID *P006* is assigned to doctor *Andrew Lowe (d162198)*, we could do

```
CALL FindVacantRoom('P006', 'd162198', 'Hip Fracture');
```

HosptID	AdmissionDate	ReleaseDate	Illness	Doctor_Doctor...	Doctor_Name	Room_RoomID	Patient_PatientID	Patient_Name
H001	2005-04-05 00:00:00	2005-04-10 00:00:00	Pneumonia	d138954	Paul Wright	22A	P002	Joe Rosberg
H002	2011-06-01 00:00:00	2011-06-05 00:00:00	Broken Collarbone	d162198	Andrew Lowe	25B	P001	Sarah Wu
H003	2006-01-24 00:00:00	2006-04-05 00:00:00	Leukemia	d143267	Carrie Miller	24C	P001	Sarah Wu
H004	2004-04-11 00:00:00	2004-04-26 00:00:00	Cardiac Arrest	d124567	Lauren Holmes	25A	P006	Toby Verthongen
H005	2002-07-02 00:00:00	2002-07-05 00:00:00	Childbirth	d125683	Charles Winston	23B	P004	Alina Ott
H006	2008-10-09 00:00:00	2008-10-12 00:00:00	Fractured Skull	d162198	Andrew Lowe	25B	P003	John Woodgate
H007	2017-04-07 11:22:12	NULL	Fever	d124567	Lauren Holmes	24A	P001	Sarah Wu
H008	2017-04-07 11:46:49	NULL	Hip Fracture	d162198	Andrew Lowe	25B	P006	Toby Verthongen
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 27: Result of FindVacantRoom('P006', 'd162198', 'Hip Fracture').

to quickly find an empty bed for him (see Figure 27). The release date is set to NULL, and is ready to be updated once the patient leaves the hospital.

9.3 Transaction

The Commit command is used to save changes invoked by a transaction to the database.It saves all transactions to the database since the last Commit

```
DELETE FROM patients
WHERE AGE = 20;
COMMIT;
```

The Rollback command works like an undo function to unsaved transactions since the previous Commit or rollback command.

```
DELETE FROM patients
WHERE AGE = 20;
ROLLBACK;
```

From this code, the DELETE operation will not have any impact on the data due to the ROLLBACK command

The Savepoint command is a point during a transaction where you can rollback to a certain point of a transaction without rolling back all the way to the the start.

```
SAVEPOINT SP1;
DELETE FROM Patients WHERE PatientID = P001;
SAVEPOINT SP2;
DELETE FROM Patients WHERE PatientID = P002;
```

From this code, we are able to rollback to the specific savepoints SP1 or SP2 and the last DELETE operation after it will be undone.

9.4 Trigger

In order to make statistics of treatment of patients it could be in the hospitals interest to have a log table keeping track of when patients are added to the hospitals database. Therefore a trigger is created which add a time stamp along with the patient information in a log table. This is done by first creating a log table with the same attributes as the patient table: The table is then altered with an additional attribute, namely a LogTime which is the time stamp:

```
CREATE TABLE PatientLog LIKE Patient;
ALTER TABLE PatientLog ADD LogTime TIMESTAMP(0);
```

Now the trigger is programmed:

```
DELIMITER //
CREATE TRIGGER Patient\_After\_Insert
AFTER INSERT ON Patient FOR EACH ROW
BEGIN
INSERT PatientLog VALUES
(New.PatientID, New.Name, New.Gender, New.Birth, NOW(0));
END; //
DELIMITER ;
```

Before testing the trigger we check the new patient log table by this query

```
SELECT* FROM PatientLog;\\
```

which gives the output seen in figure (28). It is seen that the table is completely empty.

PatientID	Name	Gender	Birth	LogTime
NULL	NULL	NULL	NULL	NULL

Figure 28: Newly added patient log table without data

Now, the trigger is tested by adding a patient to the patient table:
INSERT Patient VALUES
('P010','Felix Henderson', 'Male', '1962-06-24');
First, it is checked that the new patient is in fact added to the patient table:

```
SELECT * FROM Patient;
```

The result is shown in figure (29). It is clear that the new patient is correctly added in the last row.

PatientID	Name	Gender	Birth
P001	Sarah Wu	Female	1992-04-05 00:00:00
P002	Joe Rosberg	Male	1999-07-15 00:00:00
P003	John Woodgate	Male	1979-03-21 00:00:00
P004	Alina Ott	Female	1985-11-28 00:00:00
P005	Jasmine Gomez	Female	1952-06-15 00:00:00
P006	Toby Verthongen	Male	1967-04-25 00:00:00
P010	Felix Henderson	Male	1962-06-24 00:00:00
NULL	NULL	NULL	NULL

Figure 29: Patient table after new patient is added.

Next, the patient log table is shown by

```
SELECT* FROM PatientLog;
```

Figure (30) shows the result, where it is seen that the new patient is added to the patient log table along with a time stamp.

PatientID	Name	Gender	Birth	LogTime
P010	Felix Henderson	Male	1962-06-24 00:00:00	2017-04-06 21:52:40
NULL	NULL	NULL	NULL	NULL

Figure 30: Patient log table after new patient is added.

Thereby a trigger which ensures to keep a record of added patients with time stamp is implemented.

9.5 Event

It is important that the hospital to ensure that both the information in the appointment table is backed up if an incident should occur in the database, such that they would be able to restore the information and proceed the treatment of patients regardless of that. First, a back up table is created by:

```
CREATE TABLE BackAppointment LIKE Appointment;\\
```

Then, a back up procedure is created:

```
DELIMITER //
CREATE PROCEDURE AppointmentBackup ()
BEGIN
DELETE FROM BackAppointment;
INSERT INTO BackAppointment SELECT* FROM Appointment;
```

```
END; //
DELIMITER ;\\
```

Finally, an event is created:

```
CREATE EVENT AppointmentEvent
ON SCHEDULE EVERY 1 DAY
STARTS '2017-04-06 23:15:00'
DO CALL AppointmentBackup;
SET GLOBAL event\_scheduler = 1;
```

The content of the appointment backup table is checked:

```
SELECT* FROM BackAppointment;\\
```

As expected the table is empty, which is seen from the output in figure (31)

Appointment...	Date	Doctor_Doctor...	Doctor_Name	Patient_PatientID	Patient_Name
NULL	NULL	NULL	NULL	NULL	NULL

Figure 31: Newly added appointment backup table before backup event.

When the same query is run after the event has occurred the appointment backup table is a copy of the appointment table. This is seen in figure (32).

Appointment...	Date	Doctor_Doctor...	Doctor_Name	Patient_PatientID	Patient_Name
A001	2007-04-05 00:00:00	d124567	Lauren Holmes	P005	Jasmine Gomez
A002	2013-06-01 00:00:00	d162198	Andrew Lowe	P001	Sarah Wu
A003	2011-03-01 00:00:00	d143267	Carrie Miller	P001	Sarah Wu
A004	2004-04-11 00:00:00	d124567	Lauren Holmes	P006	Toby Verthongen
A005	2002-02-02 00:00:00	d143267	Carrie Miller	P004	Alina Ott
A006	2000-10-09 00:00:00	d162198	Andrew Lowe	P003	John Woodgate
NULL	NULL	NULL	NULL	NULL	NULL

Figure 32: Appointment backup table after backup event.

Now, an appointment for the patient Felix Henderson, who was added to the patient table in the previous section, is added to the appointment table:

```
INSERT Appointment VALUES
('A009','2017-05-01 10:05:00','d124567','Lauren Holmes','P010','Felix Henderson')
    ;
```

The result is seen in the appointment table:

```
SELECT * FROM Appointment;
```

The result is shown in figure (33)

Appointment...	Date	Doctor_Doctor...	Doctor_Name	Patient_PatientID	Patient_Name
A001	2007-04-05 00:00:00	d124567	Lauren Holmes	P005	Jasmine Gomez
A002	2013-06-01 00:00:00	d162198	Andrew Lowe	P001	Sarah Wu
A003	2011-03-01 00:00:00	d143267	Carrie Miller	P001	Sarah Wu
A004	2004-04-11 00:00:00	d124567	Lauren Holmes	P006	Toby Verthongen
A005	2002-02-02 00:00:00	d143267	Carrie Miller	P004	Alina Ott
A006	2000-10-09 00:00:00	d162198	Andrew Lowe	P003	John Woodgate
A009	2017-05-01 10:05:00	d124567	Lauren Holmes	P010	Felix Henderson
NULL	NULL	NULL	NULL	NULL	NULL

Figure 33: Appointment table after appointment is inserted.

And we notice that the appointment backup table has not changed, since the backup is only run once a day:

```
SELECT * FROM BackAppointment;
```

Which is seen in figure (34)

Appointment...	Date	Doctor_Doctor...	Doctor_Name	Patient_PatientID	Patient_Name
A001	2007-04-05 00:00:00	d124567	Lauren Holmes	P005	Jasmine Gomez
A002	2013-06-01 00:00:00	d162198	Andrew Lowe	P001	Sarah Wu
A003	2011-03-01 00:00:00	d143267	Carrie Miller	P001	Sarah Wu
A004	2004-04-11 00:00:00	d124567	Lauren Holmes	P006	Toby Verthongen
A005	2002-02-02 00:00:00	d143267	Carrie Miller	P004	Alina Ott
A006	2000-10-09 00:00:00	d162198	Andrew Lowe	P003	John Woodgate
NULL	NULL	NULL	NULL	NULL	NULL

Figure 34: Appointment backup table after appointment is inserted.

Thereby a backup event is implemented in the hospital database.