Instituto Superior Técnico - UL

Project2-SIBD

Database Modeling - Group 30

Authors:	*IST ID:*
Afonso Costa	83986
Bárbara Silva	86950
Gonçalo Mestre	87005

Professor Bruno Martins

Professor Francisco Regateiro

8 of November of 2019

Chapter 1

Database Creation

To start this part of the project it was needed to write the SQL instructions to create the database. Here we also used the suggestion of the professor, to make the *VAT_nurse* entry of the table *consultation_assistant* part of the primary key, since without it, for consultations with more than one nurse, there would be more than one row with the same primary key values. We also changed the SQL reserved words like procedure to proceduretable or put an underscore after the word, so we wouldn't have any problemas with these reserved words. The corresponding code to create all the database is shown next.

```sql
SET FOREIGN_KEY_CHECKS=0;
DROP TABLE IF EXISTS employee;
DROP TABLE IF EXISTS phone_number_employee;
DROP TABLE IF EXISTS receptionist;
DROP TABLE IF EXISTS doctor;
DROP TABLE IF EXISTS nurse;
DROP TABLE IF EXISTS client;
DROP TABLE IF EXISTS phone_number_client;
DROP TABLE IF EXISTS permanent_doctor;
DROP TABLE IF EXISTS trainee_doctor;
DROP TABLE IF EXISTS supervision_report;
DROP TABLE IF EXISTS appointment;
DROP TABLE IF EXISTS consultation;
DROP TABLE IF EXISTS consultation_assistant;
DROP TABLE IF EXISTS diagnostic_code;
DROP TABLE IF EXISTS diagnostic_code_relation;
DROP TABLE IF EXISTS consultation_diagnostic;
DROP TABLE IF EXISTS medication;
DROP TABLE IF EXISTS prescription;
DROP TABLE IF EXISTS proceduretable;
DROP TABLE IF EXISTS procedure_in_consultation;
DROP TABLE IF EXISTS procedure_radiology;
DROP TABLE IF EXISTS teeth;
DROP TABLE IF EXISTS procedure_charting;
SET FOREIGN_KEY_CHECKS=1;


CREATE TABLE employee
  (VAT INTEGER,
  name VARCHAR(35),
  birth_date DATE,
  street VARCHAR(35),
  city VARCHAR(35),
  zip VARCHAR(35),
  IBAN INTEGER,
  salary INTEGER,
  PRIMARY KEY(VAT),
  UNIQUE(IBAN),
  CHECK(salary > 0));


CREATE TABLE phone_number_employee
  (VAT INTEGER,
  phone INTEGER,
  PRIMARY KEY(VAT, phone),
  FOREIGN KEY(VAT) REFERENCES employee(VAT) ON DELETE CASCADE);


CREATE TABLE receptionist
  (VAT INTEGER,
  PRIMARY KEY(VAT),
  FOREIGN KEY(VAT) REFERENCES employee(VAT) ON DELETE CASCADE);


CREATE TABLE doctor
  (VAT INTEGER,
  specialization VARCHAR(15),
  biography VARCHAR(255),
  email VARCHAR(35),
  PRIMARY KEY(VAT),
  FOREIGN KEY(VAT) REFERENCES employee(VAT) ON DELETE CASCADE,
  UNIQUE(email));

```

```

CREATE TABLE nurse
  (VAT INTEGER,
  PRIMARY KEY(VAT),
  FOREIGN KEY(VAT) REFERENCES employee(VAT) ON DELETE CASCADE);


CREATE TABLE client
  (VAT INTEGER,
  name VARCHAR(35),
  birth_date DATE,
  street VARCHAR(35),
  city VARCHAR(15),
  zip VARCHAR(15),
  gender VARCHAR(15),
  age INTEGER,
  PRIMARY KEY(VAT),
  CHECK(age > 0));


CREATE TABLE phone_number_client
  (VAT INTEGER,
  phone INTEGER,
  PRIMARY KEY(VAT, phone),
  FOREIGN KEY(VAT) REFERENCES client(VAT) ON DELETE CASCADE);


CREATE TABLE permanent_doctor
  (VAT INTEGER,
  years INTEGER,
  PRIMARY KEY(VAT),
  FOREIGN KEY(VAT) REFERENCES doctor(VAT) ON DELETE CASCADE);


CREATE TABLE trainee_doctor
  (VAT INTEGER,
  supervisor INTEGER,
  PRIMARY KEY(VAT),
  FOREIGN KEY(VAT) REFERENCES doctor(VAT) ON DELETE CASCADE,
  FOREIGN KEY(supervisor) REFERENCES permanent_doctor(VAT) ON DELETE CASCADE);


```

```sql
CREATE TABLE supervision_report
  (VAT INTEGER,
  date_timestamp TIMESTAMP,
  description VARCHAR(255) NOT NULL,
  evaluation INTEGER,
  PRIMARY KEY(VAT, date_timestamp),
  FOREIGN KEY(VAT) REFERENCES trainee_doctor(VAT) ON DELETE CASCADE,
  CHECK(evaluation >= 1),
  CHECK(evaluation <= 5));


CREATE TABLE appointment
  (VAT_doctor INTEGER,
  date_timestamp TIMESTAMP,
  description VARCHAR(255),
  VAT_client INTEGER,
  PRIMARY KEY(VAT_doctor, date_timestamp),
  FOREIGN KEY(VAT_doctor) REFERENCES doctor(VAT) ON DELETE CASCADE,
  FOREIGN KEY(VAT_client) REFERENCES client(VAT) ON DELETE CASCADE);


CREATE TABLE consultation
  (VAT_doctor INTEGER,
  date_timestamp TIMESTAMP,
  SOAP_S VARCHAR(255),
  SOAP_O VARCHAR(255),
  SOAP_A VARCHAR(255),
  SOAP_P VARCHAR(255),
  PRIMARY KEY(VAT_doctor, date_timestamp),
  FOREIGN KEY(VAT_doctor, date_timestamp) REFERENCES appointment(VAT_doctor,
    date_timestamp) ON DELETE CASCADE);


CREATE TABLE consultation_assistant
  (VAT_doctor INTEGER,
  date_timestamp TIMESTAMP,
  VAT_nurse INTEGER,
  PRIMARY KEY(VAT_doctor, date_timestamp, VAT_nurse),
  FOREIGN KEY(VAT_doctor, date_timestamp) REFERENCES appointment(VAT_doctor,
    date_timestamp) ON DELETE CASCADE,
  FOREIGN KEY(VAT_nurse) REFERENCES nurse(VAT) ON DELETE CASCADE);

```

```

CREATE TABLE diagnostic_code
  (ID VARCHAR(15),
  description VARCHAR(255),
  PRIMARY KEY(ID));


CREATE TABLE diagnostic_code_relation
  (ID1 VARCHAR(15),
  ID2 VARCHAR(15),
  type_ VARCHAR(255),
  PRIMARY KEY(ID1, ID2),
  FOREIGN KEY(ID1) REFERENCES diagnostic_code(ID) ON DELETE CASCADE,
  FOREIGN KEY(ID2) REFERENCES diagnostic_code(ID) ON DELETE CASCADE);


CREATE TABLE consultation_diagnostic
  (VAT_doctor INTEGER,
  date_timestamp TIMESTAMP,
  ID VARCHAR(15),
  PRIMARY KEY(VAT_doctor, date_timestamp, ID),
  FOREIGN KEY(VAT_doctor, date_timestamp) REFERENCES consultation(VAT_doctor, date_timestamp) ON DELETE CASCADE,
  FOREIGN KEY(ID) REFERENCES diagnostic_code(ID) ON DELETE CASCADE);


CREATE TABLE medication
  (name VARCHAR(35),
  lab VARCHAR(35),
  PRIMARY KEY(name, lab));


CREATE TABLE prescription
  (name VARCHAR(35),
  lab VARCHAR(35),
  VAT_doctor INTEGER,
  date_timestamp TIMESTAMP,
  ID VARCHAR(15),
  dosage INTEGER,
  description VARCHAR(255),
  PRIMARY KEY(name, lab, VAT_doctor, date_timestamp, ID),
  FOREIGN KEY(name, lab) REFERENCES medication(name, lab) ON DELETE CASCADE,
```

```sql
  FOREIGN KEY(VAT_doctor, date_timestamp, ID) REFERENCES
    consultation_diagnostic(VAT_doctor, date_timestamp, ID) ON DELETE CASCADE);


CREATE TABLE proceduretable
  (name VARCHAR(35),
  type_ VARCHAR(35),
  PRIMARY KEY(name));


CREATE TABLE procedure_in_consultation
  (name VARCHAR(35),
  VAT_doctor INTEGER,
  date_timestamp TIMESTAMP,
  description VARCHAR(255),
  PRIMARY KEY(name, VAT_doctor, date_timestamp),
  FOREIGN KEY(name) REFERENCES proceduretable(name) ON DELETE CASCADE,
  FOREIGN KEY(VAT_doctor, date_timestamp) REFERENCES consultation(VAT_doctor, date_timestamp) ON DELETE CASCADE);


CREATE TABLE procedure_radiology
  (name VARCHAR(35),
  file_ VARCHAR(35),
  VAT_doctor INTEGER,
  date_timestamp TIMESTAMP,
  PRIMARY KEY(name, file_, VAT_doctor, date_timestamp),
  FOREIGN KEY(name, VAT_doctor, date_timestamp) REFERENCES
    procedure_in_consultation(name, VAT_doctor, date_timestamp) ON DELETE CASCADE);


CREATE TABLE teeth
  (quadrant INTEGER,
  number_ INTEGER,
  name VARCHAR(15),
  PRIMARY KEY(quadrant, number_));


CREATE TABLE procedure_charting
  (name VARCHAR(35),
  VAT INTEGER,
  date_timestamp TIMESTAMP,
  quadrant INTEGER,
  number_ INTEGER,
  desc_ VARCHAR(255),
  measure INTEGER,
  PRIMARY KEY(name, VAT, date_timestamp, quadrant, number_),
  FOREIGN KEY(name, VAT, date_timestamp) REFERENCES
    procedure_in_consultation(name, VAT_doctor, date_timestamp) ON DELETE CASCADE,
  FOREIGN KEY(quadrant, number_) REFERENCES teeth(quadrant, number_) ON DELETE CASCADE);
```

Chapter 2

Table Population

To populate the tables we created with data, we wrote a script in python to generate a *.sql* file that populates all the tables, because it was needed to populate with large amounts of data for some tables. Since it is quite a big file, we decided not to put it on the report and just on the zip file that is being submitted.

Chapter 3

Queries

3.1 Query1

In this query we were asked to list the VAT, name, and phone number(s) for all clients that had consultations with the doctor named Jane Sweettooth, ordering the list by the alphabetical order for the names.

In here we believe it is relevant to mention that we didn’t put any specification for the order, since mysql orders by default on ascending order, which was the supposed order. It is also of importance to justify that we have done a left outer join, so we could still preserve the lines for the clients that didn’t have phone number.

```sql
SELECT DISTINCT c.VAT, c.name, phone
FROM employee AS e, appointment AS a, consultation AS con, client AS c LEFT OUTER
    JOIN phone_number_client
ON c.VAT = phone_number_client.VAT
WHERE e.VAT = con.VAT_doctor
AND con.VAT_doctor = a.VAT_doctor AND con.date_timestamp = a.date_timestamp
AND c.VAT = a.VAT_client
AND e.name = ’Jane Sweettooth’
ORDER BY c.name;
```

3.2 Query2

In this query it was asked to list the name of all trainee doctors who had a report associated to an evaluation score below the value of three, or with a description that contains the term insufficient. It was asked to list with the name, the VAT of the trainee, the name of the doctor that made the evaluation, the evaluation score and the textual description for the evaluation report. It was also asked to order the results according to the evaluation score in descending order.

Here it is relevant to notice that we made a subquery to get the name of the permanent doctor together with the rest of the information, since we get both the name of the trainee doctor and the name of the permanent doctor from the employee table, and was not possible to get both on the main part of the query. It is also to notice that here we had to specify the type of ordering, since the default is ascending and not descending.

```
SELECT e.name, t.VAT,
    (SELECT e1.name
    FROM employee AS e1
    WHERE e1.VAT = t.supervisor),
    r.evaluation, r.description
FROM trainee_doctor AS t, employee AS e, supervision_report AS r
WHERE t.VAT = e.VAT
AND r.VAT = t.VAT
AND (r.evaluation < 3
OR r.description LIKE ’%insufficient%’)
ORDER BY r.evaluation DESC;
```

3.3 Query3

Here we were asked to list the name, city and VAT for all clients where the most recent consultation has the objective part of the SOAP notes mentioning the terms gingivitis or periodontitis. For this query it is to notice the use of the SQL *LIKE* operator, so we could find the asked terms in the text for the corresponding soap note. It is also to notice that we had to use a subquery to find the max time_stamp so we could have the last consultation.

```
SELECT name, city, VAT
FROM client AS cl, appointment AS a1, consultation AS c
WHERE cl.VAT = a1.VAT_client
AND a1.VAT_doctor = c.VAT_doctor
AND a1.date_timestamp = c.date_timestamp
AND (SOAP_O LIKE ’%gingivitis%’ OR SOAP_O LIKE ’%periodontitis%’)
AND c.date_timestamp = (SELECT MAX(a2.date_timestamp) FROM appointment a2 WHERE a1.
    VAT_client = a2.VAT_client)
ORDER BY name;
```

3.4 Query4

For this query the goal was to list the name, VAT and address of all clients of the clinic that have had appointments but that never had a consultation.

In this query we had to use a subquery to get the vat of all clients that had an appointment, but have never had a consultation, we made sure of that by having the consultation count equal to 0.

```sql
SELECT DISTINCT name, VAT, street, city, zip
FROM client
WHERE VAT IN (SELECT a.VAT_client
  FROM appointment a LEFT OUTER JOIN consultation c
  ON a.VAT_doctor = c.VAT_doctor AND a.date_timestamp = c.date_timestamp
  GROUP BY VAT_client
  HAVING COUNT(c.date_timestamp) = 0);
```

3.5 Query5

For this query it was needed to present the code of the diagnostic together with it's description. It was also needed to list the number of distinct medication names that have been prescribed to treat that condition. The results were supposed to be sorted according to the number of distinct medication names, in ascending order.

In here it was important to make a left outer join, so we could include the diagnostic codes for which there was no prescription.

```sql
SELECT COUNT(DISTINCT p.name) AS medication_usage, d.ID, d.description
FROM diagnostic_code AS d LEFT OUTER JOIN prescription AS p
ON d.ID = p.ID
GROUP BY d.ID
ORDER BY COUNT(DISTINCT p.name);
```

3.6 Query6

We had problems testing this query since it takes a long time to run. In this query we should present the average number of nurses/assistants, procedures, diagnostic codes, and prescriptions involved in consultations from the year 2019, respectively for clients belonging to two age groups: less or equal to 18 years old, and more than 18 years old.

```sql
SELECT
    AVG(n0.cnurse) AS Nurse_avg_less_18,
    AVG(n1.cnurse) AS Nurse_avg_more_18,
    AVG(p0.cproced) AS Procedures_avg_less_18,
    AVG(p1.cproced) AS Procedures_avg_more_18,
    AVG(d0.diag) AS Diagnostics_avg_less_18,
    AVG(d1.diag) AS Diagnostics_avg_more_18,
    AVG(pr0.presc) AS Prescriptions_avg_less_18,
    AVG(pr1.presc) AS Prescriptions_avg_more_18
FROM
    (SELECT COUNT(DISTINCT VAT_nurse) as cnurse
    FROM consultation_assistant AS ca, appointment AS a, client AS c
    WHERE ca.VAT_doctor = a.VAT_doctor
    AND ca.date_timestamp = a.date_timestamp
    AND a.VAT_client = c.VAT
    AND a.date_timestamp LIKE ’2019%’
    AND c.age <= 18
    GROUP BY a.VAT_doctor, a.date_timestamp) AS n0,

    (SELECT COUNT(DISTINCT VAT_nurse) as cnurse
    FROM consultation_assistant AS ca, appointment AS a, client AS c
    WHERE ca.VAT_doctor = a.VAT_doctor
    AND ca.date_timestamp = a.date_timestamp
    AND a.VAT_client = c.VAT
    AND a.date_timestamp LIKE ’2019%’
    AND c.age > 18
    GROUP BY a.VAT_doctor, a.date_timestamp) AS n1,

   (SELECT COUNT(DISTINCT pc.name) AS cproced
    FROM procedure_in_consultation AS pc, appointment AS a, client AS c
    WHERE pc.VAT_doctor = a.VAT_doctor
    AND pc.date_timestamp = a.date_timestamp
    AND a.VAT_client = c.VAT
    AND a.date_timestamp LIKE ’2019%’
    AND c.age <= 18
    GROUP BY a.VAT_doctor, a.date_timestamp) AS p0,

    (SELECT COUNT(DISTINCT pc.name) AS cproced
    FROM procedure_in_consultation AS pc, appointment AS a, client AS c
    WHERE pc.VAT_doctor = a.VAT_doctor
    AND pc.date_timestamp = a.date_timestamp
    AND a.VAT_client = c.VAT
    AND a.date_timestamp LIKE ’2019%’
    AND c.age > 18
    GROUP BY a.VAT_doctor, a.date_timestamp) AS p1,

   (SELECT COUNT(DISTINCT cd.ID) AS diag
    FROM consultation_diagnostic AS cd, appointment AS a, client AS c
    WHERE cd.VAT_doctor = a.VAT_doctor
    AND cd.date_timestamp = a.date_timestamp
    AND a.VAT_client = c.VAT
    AND a.date_timestamp LIKE ’2019%’
    AND c.age <= 18
    GROUP BY a.VAT_doctor, a.date_timestamp) AS d0,

    (SELECT COUNT(DISTINCT cd.ID) AS diag
    FROM consultation_diagnostic AS cd, appointment AS a, client AS c
    WHERE cd.VAT_doctor = a.VAT_doctor
    AND cd.date_timestamp = a.date_timestamp
    AND a.VAT_client = c.VAT
    AND a.date_timestamp LIKE ’2019%’
    AND c.age > 18
    GROUP BY a.VAT_doctor, a.date_timestamp) AS d1,

   (SELECT COUNT(DISTINCT pr.name) AS presc
    FROM prescription AS pr, appointment AS a, client AS c
    WHERE pr.VAT_doctor = a.VAT_doctor
    AND pr.date_timestamp = a.date_timestamp
    AND a.VAT_client = c.VAT
    AND a.date_timestamp LIKE ’2019%’
    AND c.age <= 18
    GROUP BY a.VAT_doctor, a.date_timestamp) AS pr0,

    (SELECT COUNT(DISTINCT pr.name) AS presc
    FROM prescription AS pr, appointment AS a, client AS c
    WHERE pr.VAT_doctor = a.VAT_doctor
    AND pr.date_timestamp = a.date_timestamp
    AND a.VAT_client = c.VAT
    AND a.date_timestamp LIKE ’2019%’
    AND c.age > 18
    GROUP BY a.VAT_doctor, a.date_timestamp) AS pr1
;
```

3.7 Query7

In here it was asked for each diagnostic code, present the name of the most common medication used to treat that condition (i.e., the medication name that more often appears associated to prescriptions for that diagnosis).

```
SELECT
  c.ID AS ’diagnosis’,
  name AS ’medication name’,
  COUNT(name) AS 'Most common medication (has been prescripted __ times)'
FROM
  prescription p,
  consultation_diagnostic c
WHERE
  c.ID=p.ID
  AND c.VAT_doctor=p.VAT_doctor
  AND c.date_timestamp=p.date_timestamp
  AND c.date_timestamp LIKE '2019%'
GROUP BY
  c.VAT_doctor,
  c.date_timestamp,
  c.id
ORDER BY
  c.ID asc;
```

3.8 Query8

For query number 8 it was needed to list alphabeticaly, the names and labs for the medications that, in the year 2019, have been used to treat ”dental cavities”, but have not been used to treat any infectious disease”.

To obtain this list we made two sub-queries, one for selecting the medications which have been used to treat ”dental cavities” and other for selecting all the medications that were never used to cure “infectious disease”. Once these sub-queries were obtained, was made a interception between them, so we could get the table entries corresponding only to the result of both sub-queries.

```
SELECT DISTINCT
  den_cav.den_name AS medication_name,
  den_cav.den_lab AS medication_lab
FROM
  (SELECT DISTINCT
    p.name AS inf_name,
    p.lab AS inf_lab
  FROM
    diagnostic_code d,
    prescription p
  WHERE
    d.description!='infectious disease'
    AND d.ID=p.ID
    AND p.date_timestamp LIKE '2019%'
  ORDER BY
```

```
    d.id
  ) AS inf_dis
  INNER JOIN
  (SELECT DISTINCT
    p.name AS den_name,
    p.lab AS den_lab
  FROM
    diagnostic_code d,
    prescription p
  WHERE d.description=’dental cavities’
    AND d.ID=p.ID
    AND p.date_timestamp LIKE ’2019%’
  ORDER BY
    d.id
  ) AS den_cav
  ON
    (inf_dis.inf_name=den_cav.den_name
    AND inf_dis.inf_name=den_cav.den_name)
ORDER BY
  den_cav.den_name,
  den_cav.den_lab asc;
```

3.9 Query9

The objective on this last query was to list the names and addresses of clients that have never missed an appointment in 2019.

Here we chose that the way to go was to select the clients that had the same count of appointments and consultations in the table, so to go with it we made two subquerys, one to select the information of the clients and the count of appointments they had and another to select the information of the clients and the count of consultations they had. Then we joined both subquerys, making as a condition for them to have the same count. Since there can’t be a consultation without an appointment, the count of appointments was always higher or equal, which meant that when it was equal the client never missed an appointment.

```
SELECT app.name, app.street, app.city, app.zip
FROM (SELECT name, street, city, zip, COUNT(a.date_timestamp) AS count_a
    FROM client AS c, appointment AS a
    WHERE a.VAT_client = c.VAT
    AND a.date_timestamp LIKE ’2019%’
    GROUP BY c.VAT) AS app
JOIN
```

```
    (SELECT name, street, city, zip, COUNT(con.date_timestamp) AS count_b
    FROM client AS c, appointment AS a, consultation AS con
    WHERE a.VAT_client = c.VAT
    AND a.VAT_doctor = con.VAT_doctor
    AND a.date_timestamp LIKE '2019%'
    AND a.date_timestamp = con.date_timestamp
    GROUP BY c.VAT) co
ON app.count_a = co.count_b
AND app.name = co.name
AND app.street = co.street
AND app.city = co.city
AND app.zip = co.zip;
```

Chapter 4

Indexes

In here it was asked for us to suggest database indexes that could be used to improve the performance of the first two queries from the list of information needs.

4.1 Index Query 1

For the first query we believe the best option for an index that would improve the performance of this query is an index for the name of the clients, since most of the time of the query might taken be ordering the data and the data is ordered by the names of the clients. We thought it would be also useful to improve performance to create an index for the VAT of the client on the appointment table, since we match this with the VAT from the client table, and it would be easy if we could have an easy way to access orderly the VAT of the client on the appointment table. For the same reasons we thought it would be also useful to have an index for the VAT of the client on the phone_number_client table. To implement these indexes we used the SQL code shown next.

```sql
CREATE INDEX client_name
ON client(name);

CREATE INDEX appointment_vat_client
ON appointment(VAT_client);

CREATE INDEX phone_client_vat
ON phone_number_client(VAT);
```

4.2 Index Query 2

For the second query, since the results need to be ordered by the evaluation score, we thought it would be useful to create an index for the evaluation on the supervision report. Also, as on the subquery to select the name of the permanent doctor we compare with the VAT of the supervisor on the trainee_doctor table, we decided to create an index for this VAT of the supervisor also. The SQL instructions to implement these indexes are shown next.

```sql
CREATE INDEX score_trainee
ON supervision_report(evaluation);

CREATE INDEX supervisor
ON trainee_doctor(supervisor);
```

Chapter 5

Changes

5.1 Change 1

In this change we were asked to update the city and street of the doctor named *Jane Sweettooth* to a different city of our choice. The SQL instruction we used to implement this is shown next

```
UPDATE employee
SET street = ’Rua dos Dentinhos’, city = ’Dentatown’
WHERE employee.name = ’Jane Sweettooth’
```

5.2 Change 2

In this change we were asked to increase the salaries (5%) of all the doctors that had more than 100 appointments in 2019. In order to do it, we used a subquery to count the number of appointments for each doctor in the year of 2019.

```
update employee e
set salary = 1.05*salary
where ( select count(*)
  from appointment a
  where e.VAT = a.VAT_doctor
  and year(a.date_timestamp)=2019
  group by VAT_doctor) > 100;
```

5.3 Change 3

In this change we were asked to delete the doctor named ’Jane Sweettooth’ and all the appointments and consultations, including the corresponding diagnostics, procedures and prescriptions made by that doctor from the database. We were also asked to delete the diagnostic codes and the procedures that were only performed/assigned by this doctor. To delete every thing related to this doctor from the database, we had to first start by deleting the data from the *procedure* table and the *diagnostic_code* table, because to do it we used information from the doctor we want to delete, and then just after deleting this data, we delete the employee ”Jane Sweettooth” from the table. This only works without causing problems with the the tables that inherit a foreign key from the employee table and consequently from those tables to others, because when creating the tables and inserting the foreign key constraints we used **ON DELETE CASCADE**, so that when the foreign key is deleted from the table that supplies it, it is still deleted on that table.

```
delete from proceduretable
where exists (select pc.name
  from procedure_in_consultation pc, employee e
  where pc.VAT_doctor=e.VAT
  and e.name=’Jane Sweettooth’)
and not exists (select pc.name
  from procedure_in_consultation pc, employee e
  where pc.VAT_doctor=e.VAT
  and e.name<>’Jane Sweettooth’)
;

delete from diagnostic_code
where exists (select cd.ID
  from consultation_diagnostic cd, employee e
  where cd.VAT_doctor=e.VAT
  and e.name=’Jane Sweettooth’)
and not exists (select cd.ID
  from consultation_diagnostic cd, employee e
  where cd.VAT_doctor=e.VAT
  and e.name<>’Jane Sweettooth’)
;

delete from doctor
where VAT in (select VAT from employee
where name = ’Jane Sweettooth’);
```

5.4 Change 4

In this change we were asked 3 things. First, to find the diagnosis code corresponding to ’gingivitis’. Second, to insert a new diagnostic code in the table corresponding to ’periodontitis’. Last, to change the diagnostic codes from ’gingivitis’ to ’periodontitis’, for all clients that have an average gap above 4 for a consultation/diagnostic. We decided to put here the code that gives us a table with the procedure name, the doctor VAT, the date of the procedure, the average gap and the diagnostic code description, if the average gap is bigger than 4 and the corresponding diagnosis code description is ’gingivitis’. This way we know which diagnostic codes should be changed in the consultation diagnostic for each client. To update the consultation diagnostic we used a subquery that sees if the average gap measure of one consultation is bigger than 4.

```sql
select ID
from diagnostic_code
where description = ’gingivitis’;

INSERT INTO diagnostic_code
VALUES (’A-1069’, ’periodontitis’);

select pc.name, pc.VAT, pc.date_timestamp, avg(pc.measure) as average_gap, dc.description
from consultation_diagnostic cd inner join procedure_charting pc
on cd.VAT_doctor = pc.VAT
inner join diagnostic_code dc
on cd.ID = dc.ID
where cd.date_timestamp = pc.date_timestamp
and cd.ID IN (select ID from diagnostic_code where description = ’gingivitis’)
group by pc.VAT, pc.date_timestamp
having avg(measure) > 4;

update consultation_diagnostic cd
set cd.ID = (select dc.ID from diagnostic_code dc where dc.description = ’periodontitis’)
where (select avg(measure) from procedure_charting pc
      where pc.VAT=cd.VAT_doctor
    and pc.date_timestamp=cd.date_timestamp
    group by pc.VAT, pc.date_timestamp) > 4
and cd.ID = (select dc.ID from diagnostic_code dc where dc.description = ’gingivitis’);

select pc.name, pc.VAT, pc.date_timestamp, avg(pc.measure) as average_gap, dc.description
from consultation_diagnostic cd inner join procedure_charting pc
on cd.VAT_doctor = pc.VAT
inner join diagnostic_code dc
on cd.ID = dc.ID
```

```
where cd.date_timestamp = pc.date_timestamp
and cd.ID IN (select ID from diagnostic_code where description = 'periodontitis)
group by pc.VAT, pc.date_timestamp;
```

Chapter 6

Views

6.1 View 1

In here we had to create the view over the tables in the database model corresponding to the following relational schema.

dim_date(date_timestamp, day, month, year)

IC: date_timestamp corresponds to a date existing in consultations

We decided to also provide the instruction to drop the view with the SQL instructions to create, to make sure that there is no view with the same name in the database.

```
DROP view IF EXISTS dim_date;
CREATE VIEW dim_date AS
SELECT date_timestamp, DAY(date_timestamp) AS day, MONTH(date_timestamp) AS month,
    YEAR(date_timestamp) AS year
FROM consultation;
```

6.2 View 2

In here we had to create the view over the tables in the database model corresponding to the following relational schema.

dim_client(VAT, gender, age)

VAT: FK(client)

We decided to also provide the instruction to drop the view with the SQL instructions to create, to make sure that there is no view with the same name in the database.

```
DROP view IF EXISTS dim_client;
CREATE VIEW dim_client AS
SELECT VAT, gender, age
FROM client;
```

6.3 View 3

In here we had to create the view over the tables in the database model corresponding to the following relational schema.

dim_client(<u>VAT</u>, gender, age)

IC: zip corresponds to a zip code existing in clients

We decided to also provide the instruction to drop the view with the SQL instructions to create, to make sure that there is no view with the same name in the database.

```
DROP view IF EXISTS dim_location_client;
CREATE VIEW dim_location_client AS
SELECT zip, city
FROM client;
```

6.4 View 4

In here we had to create the view over the tables in the database model corresponding to the following relational schema.

facts_consults(VAT,date,zip,num_procedures,num_medications,num_diagnostic_codes)
VAT: FK(dim_client) date: FK(dim_date) zip: FK(dim_location_client)

We decided to also provide the instruction to drop the view with the SQL instructions to create, to make sure that there is no view with the same name in the database.

The main goal in this view was to sum, separately, all the medicines prescribed, all the procedures made and all the diagnostic codes in every consultation. To select this fields we had to consider that some of them have null spaces, so it was used the LEFT OUTER JOIN sql command, to join even if there are null fields. We had to make two route of left outer join, one for the prescriptions (to get the number of medications by the ID field, and the number of diagnostic_code by the name field) and other for the procedure in consultation.

```
DROP view IF EXISTS dim_location_client;
CREATE VIEW facts_consult AS
SELECT
  dc.VAT,
  dd.date_timestamp AS 'date',
  dl.zip,
  COUNT(DISTINCT pres.name) as num_medications,
  COUNT(DISTINCT pres.ID) as num_diagnostic_codes,
  COUNT(DISTINCT proc.name) as num_procedures
FROM
```

```sql
  dim_client dc,
  dim_date dd,
  dim_location_client dl,
  client cl,
  appointment a
  LEFT OUTER JOIN consultation c
  ON (c.date_timestamp = a.date_timestamp AND c.VAT_doctor=a.VAT_doctor)
  LEFT OUTER JOIN prescription pres
  ON (c.date_timestamp = pres.date_timestamp AND c.VAT_doctor=pres.VAT_doctor)
  LEFT OUTER JOIN procedure_in_consultation proc
  ON (c.date_timestamp = proc.date_timestamp AND c.VAT_doctor=proc.VAT_doctor)
WHERE
  a.VAT_client=dc.VAT
  AND cl.VAT=dc.VAT
  AND cl.zip=dl.zip
  AND dd.date_timestamp=pres.date_timestamp
  AND dd.date_timestamp=proc.date_timestamp
GROUP BY
  dc.VAT,
  dd.date_timestamp
ORDER BY
  dc.VAT,
  dd.date_timestamp;
```