VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY FACULTY OF COMPUTER SCIENCE AND ENGINEERING

DATABASE SYSTEMS LAB (CO2014)

Assignment 2 Report

HOSPITAL MANAGEMENT SYSTEM

Advisor:	Dr. Trần Minh Quang	
Class:	CC08	
Students:	Nguyễn Hoàng	1952255
	Cao Bá Huy	1952713
	Lưu Chấn Hưng	1952063

HO CHI MINH CITY, NOVEMBER 2021

Contents

1 Improvement

Recalling from our previous Hospital management database, we think that we can make some improvements to make our database more suitable to be used in real life hospitals. Therefore, we will make some changes by adding some constraints between some specific entities:

- Firstly, a patient may be diagnosed with many diseases at the same time. Thus, we will let the attribute Diagnosis in the PATIENT RECORD be a multivalued attribute.
- Secondly, we will add a relationship called Provide between DOCTOR and TREATMENT. A Doctor can provide many treatments.
- Thirdly, we add a relationship called Join between DOCTOR and APPOINTMENT. A doctor may have one or many appointments, but an appointment is only joined by 1 doctor.
- Finally, we add a relationship called Cost between TREATMENT and PAYMENT. A treatment will only cost a payment.

That is all about our improvement, in the next section will discuss about the normal forms and modify the tables as well as the logical design.

2 Normalization checking

2.1 Checking for violation of first normal form

2.1.1 Definition of first normal form

- First normal form (1NF) is considered to be part of the formal definition of a relation in the basic (flat) relational model; historically, it was defined to disallow multivalued attributes, composite attributes, and their combinations.
- It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.
- Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple. In other words, 1NF disallows relations within relations or relations as attribute values within tuples. The only attribute values permitted by 1NF are single atomic (or indivisible) values.

2.1.2 Determine the violations

In our design, we have a multivalued attribute called maintenance date. This attribute belongs to the relation MEDICAL EQUIPMENT and stores the date(s) this medical equipment got checked to assure it functions normally.

Moreover, we still have another multivalued attribute called Diagnosis from the PATIENT RECORD which we just added on the above section that violates 1NF.

2.1.3 Solution

First of all we will determine the violation of MEDICAL EQUIPMENT relation because the attribute called Maintenance dates stores a set of value, there by violating the 1NF.

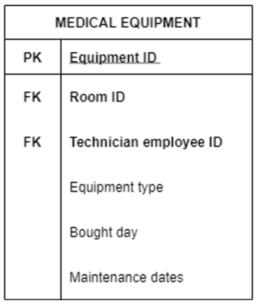

Figure 1: MEDICAL EQUIPMENT relation violates the 1NF

Next, we will remove the attribute Maintenance date and create another relation called EQUIPMENT MAINTENANCE DATE which includes the PK Equipment ID from the MEDICAL EQUIPMENT and the attribute called Date. The primary key of this relation is the combination of 2 mentioned attributes: {Date, Equipment ID}.

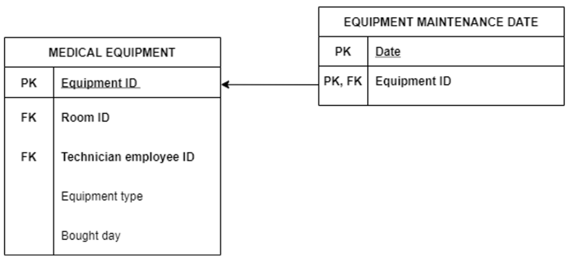

Figure 2: The result tables after we remove the attribute Maintenance date and create a new table to assure 1NF

Moreover, we still have another multivalued attribute called Diagnosis from the PATIENT RECORD. Similarly, we will perform the same steps as above. We will first remove the attribute Diagnosis and create another relation called PATIENT DIAGNOSIS which includes the PK Record ID from the PATIENT RECORD table and the attribute called Diagnosis. The primary key of this relation is the combination of 2 mentioned attributes: {Diagnosis, Record ID}.

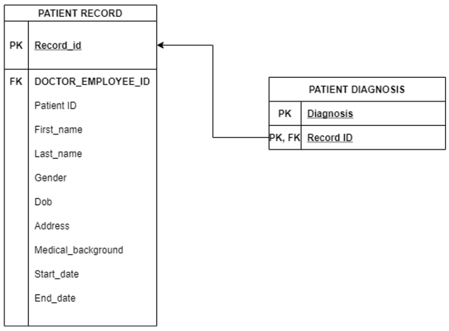

Figure 3: The result tables after we remove the attribute Diagnosis and create a new table to assure 1NF

2.2 Checking for violation of second normal form

2.2.1 Definition of second normal form

Second normal form (2NF) is based on the concept of full functional dependency. A functional dependency X → Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold anymore; that is, for any attribute $A \epsilon X$, $(X - \{A\})$ does not functionally determine Y.

⇒ A relation schema R is in 2NF if every non-prime attribute A in R is fully functionally dependent on the primary key of R.

2.2.2 Determine the violations

Luckily, we can say that for each relation in our database, every non-prime attributes are fully dependent on the their respective primary key. Therefore, we can proudly say that our database is in 2NF.

2.3 Checking for violation of third normal form

2.3.1 Definition of third normal form

Third normal form (3NF) is based on the concept of transitive dependency. A functional dependency X → Y in a relation schema R is a transitive dependency if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of and both X → Z and Z → Y hold.

⇒ According to Codd's original definition, a relation schema R is in 3NF if it satisfies 2NF and no non-prime attribute of R is transitively dependent on the primary key.

2.3.2 Determine the violations

In our design, the PATIENT RECORD has violated the 3NF.

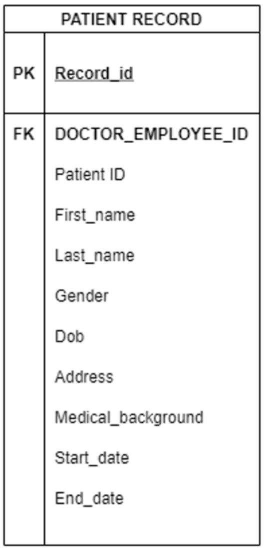

Figure 4: The PATIENT RECORD relation

This relation is in 2NF but not in 3NF because of the transitive dependency. To be more specific, we have the following dependencies:

- Record ID $\rightarrow$ {patient ID, Doctor employee ID, first name, last name, DOB, gender, address, medical background, start date, end date}
- Patient ID $\rightarrow$ {first name, last name, dob, gender, address}

This relation is not in 3F because of the transitive dependency of first name, last name, day of birth , gender and address on Record ID via Patient ID (Patient ID is a non-prime attribute).

2.3.3 Solution

We can normalize PATIENT RECORD relations by decomposing it into the two 3NF relation schemas:

- The first relation schema will be R1(Patient ID, Name,DOB, Gender, Address) where the Patient ID is the primary key.
- The second relation schema will be R2(Record ID,Doctor employee ID, Patient ID, Medical background,start date, end date) where Record ID is the primary key and Patient ID is the foreign key that refers to primary key of R1.

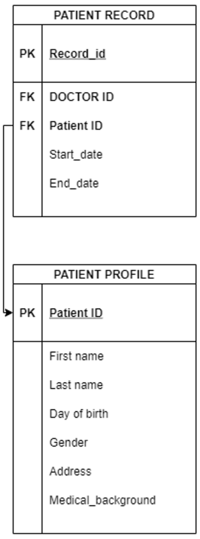

Figure 5: The decomposition of the MEDICAL RECORD table into 2 tables.

3 New logical design and physical

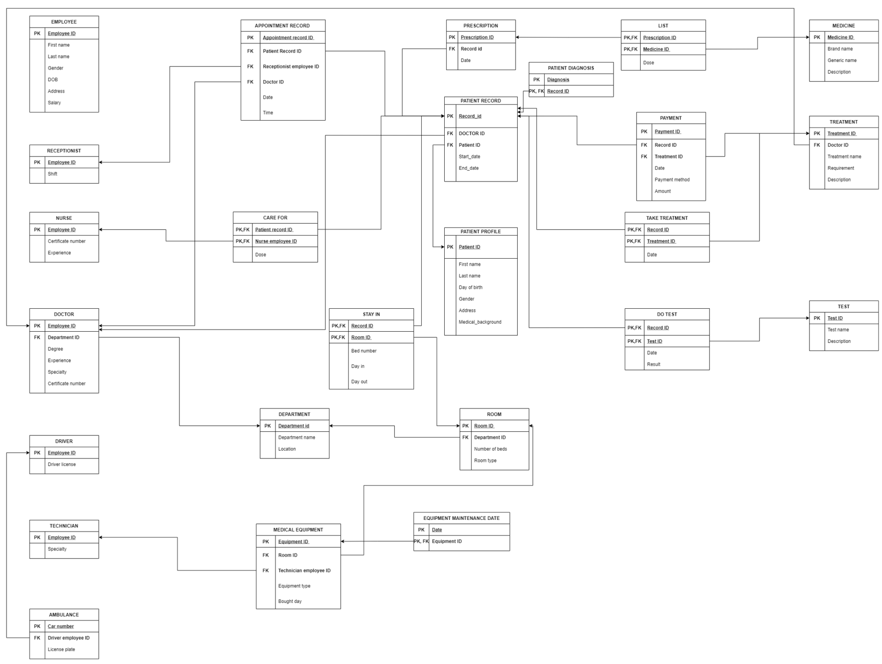

Figure 6: The new logical design that has been improved and assures all NFs

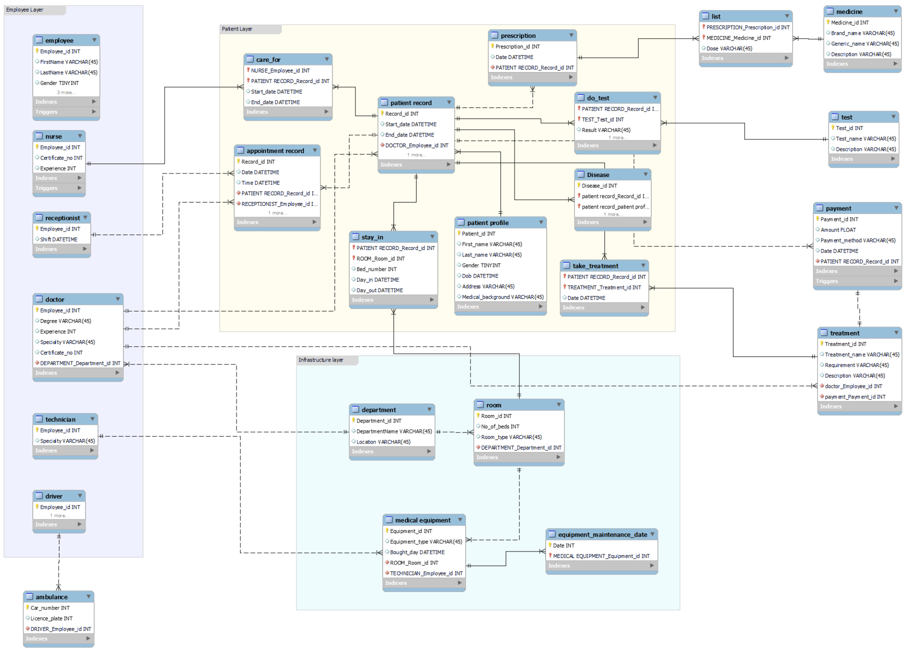

Figure 7: The new physical design implemented with MySQL Workbench

Similar to assignment 1, we will use a feature called Forward engineering in MySQL workbench to convert the design to SQL code.

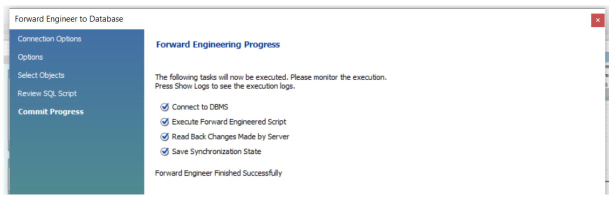

Figure 8: Forward engineering finished successfully.

Our database is now ready to be used.

4 Implementation completion

4.1 Constraints specification

First step to complete our implementation of the database is to complete a set of constraints, especially the referential integrity constraints of the foreign keys. Luckily, MySQLWorkbench provides us with a simple interface to set these constraints.

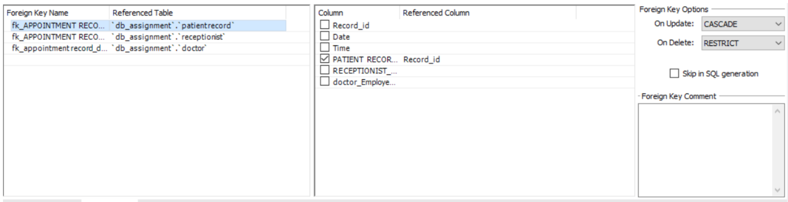

Figure 9: MySQL Workbench foreign keys options features

As we can see, the window on the right side allow us the ability to set the options when DELETE and UPDATE.We have 4 total options to choose (RESTRICT, CASCADE, SET NULL, NO ACTION). Now we will have to go through each of our foreign keys and set the suitable option.

- On DOCTOR table, we have 1 FK referenced to DEPARTMENT table. We would set CASCADE on both ON UPDATE AND ON DELETE.The reason for this choice is due to a business rule that when a DEPARTMENT is removed, every doctors should be removed as well.
- For the rest of our tables, we would choose CASCADE ON UPDATE and RESTRICT ON DELETE. When the referenced table UPDATE their value, it is obvious that the referencing table should UPDATE as well, so we choose CASCADE ON UPDATE. For ON DELETE options, we choose RESTRICT because when a referenced table is deleted, we don't have to delete the referencing table as well because they represents two distinct entities.

4.2 Triggers

4.2.1 Trigger on Employee's Salary

The first trigger we would like to set is a constraint on the input of Salary column in EMPLOYEE table. If a person wants to insert a negative value into our table, we would automatically set it to 0 as default.

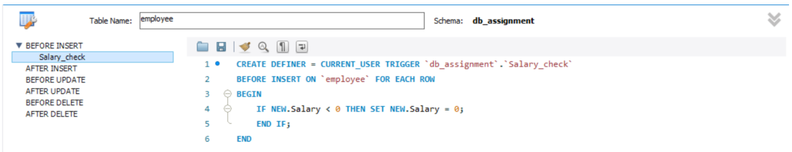

Figure 10: Trigger on Employee's Salary

4.2.2 Trigger on Experience of NURSE

The second trigger is similar to the first one in terms of checking BEFORE_INSERT constraint. In this case, we would also set a default value of 0 whenever a person accidentally inserted a negative value of Experience column of NURSE table.

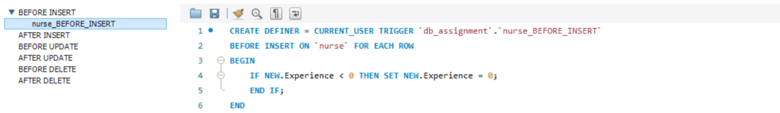

Figure 11: Trigger on Nurse's Experience

4.2.3 Trigger on date of Appointment record

The final trigger in our assignment is about the default value for a DATETIME type value, which is one of the most common usage of trigger. We would create a trigger to automatically set the current date in to the

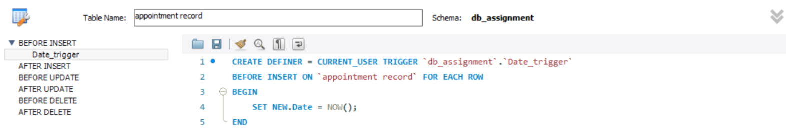

Figure 12: Trigger on Appointment Record

4.3 Queries

4.3.1 Retrieve a full form of medical records

As we mentioned earlier, one big improvement of our database after normalizing is the spilt of MEDICAL RECORD table into 2 sub tables, PATIENT PROFILE and PATIENT RECORD. However, when we want to get the information of the whole medical record, we have to join those 2 table.

```sql
SELECT * FROM db_assignment.`patient profile`;
```

Patient_id	First_name	Last_name	Gender	Dob	Address	Medical_background
1	K	Huynh	1	2000-05-04 00:00:00	NULL	NULL
2	L	Vu	1	1994-03-03 00:00:00	NULL	NULL
3	M	Ngo	0	1980-01-02 00:00:00	NULL	NULL
4	N	Nguyen	0	1960-05-06 00:00:00	NULL	heart disease
5	O	Nguyen	1	1990-06-04 00:00:00	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 13: PATIENT PROFILE table

```sql
SELECT * FROM db_assignment.`patient record`;
```

Record_id	Start_date	End_date	DOCTOR_Employee_id	patient profile_Patient_id
1	2019-05-12 00:00:00	2020-01-01 ...	2	1
2	2018-04-05 00:00:00	2021-06-07 ...	3	2
3	2020-04-06 00:00:00	2021-08-09 ...	1	3
4	2017-04-04 00:00:00	2018-05-01 ...	1	4
5	2019-02-04 00:00:00	2021-05-04 ...	1	5
6	2021-01-01 00:00:00	2021-04-05 ...	1	4
NULL	NULL	NULL	NULL	NULL

Figure 14: PATIENT RECORD table

```sql
SELECT *
FROM db_assignment.`patient record` as r
INNER JOIN db_assignment.`patient profile` as p ON  p.Patient_id = r.`patient profile_Patient_id`
```

Record_id	Start_date	End_date	DOCTOR_Employee_id	patient profile_Patient_id	Patient_id	First_name	Last_name	Gender	Dob	Addres
1	2019-05-12 00:00:00	2020-01-01 ...	2	1	1	K	Huynh	1	2000-05-04 00:00:00	NULL
2	2018-04-05 00:00:00	2021-06-07 ...	3	2	2	L	Vu	1	1994-03-03 00:00:00	NULL
3	2020-04-06 00:00:00	2021-08-09 ...	1	3	3	M	Ngo	0	1980-01-02 00:00:00	NULL
4	2017-04-04 00:00:00	2018-05-01 ...	1	4	4	N	Nguyen	0	1960-05-06 00:00:00	NULL
5	2019-02-04 00:00:00	2021-05-04 ...	1	5	5	O	Nguyen	1	1990-06-04 00:00:00	NULL
6	2021-01-01 00:00:00	2021-04-05 ...	1	4	4	N	Nguyen	0	1960-05-06 00:00:00	NULL

Figure 15: Query and Result

As a result of the above query, we have a full form of a medical record.

4.3.2 Get all prescriptions of patient "Nguyen N"

In this query, we would like to retrieve all prescriptions of a patient named "Nguyen N". As we already know, patient First name and Last name is stored in the PATIENT PROFILE table.While prescriptions are stored in PRESCRIPTION table, which have a foreign key referencing to Record id of PATIENT RECORD.Therefore, we first have to lookup the Record id of patient Nguyen N, then join with Prescription table on that Record id.

```sql
SELECT *
FROM db_assignment.`prescription` as pre
INNER JOIN (
    SELECT r.`Record_id`
    FROM db_assignment.`patient record` as r
    INNER JOIN db_assignment.`patient profile` AS p ON r.`patient profile_Patient_id` = p.`Patient_id`
    WHERE p.First_name = "N" AND p.Last_name = "Nguyen") AS t
ON t.`Record_id` = pre.`PATIENT RECORD_Record_id`
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Prescription_id	Date	PATIENT RECORD_Record_id	Record_id
4	2017-04-06 00:00:00	4	4
6	2021-01-03 00:00:00	6	6

Figure 16: List of prescriptions used by Nguyen N

From the result we could conclude that Nguyen N has 2 prescriptions.

4.3.3 Get the number patients treated by doctor "Nguyen A"

For the last query, we will get the number of patients treated by doctor Nguyen A. From our database design, we know that employee names are stored in EMPLOYEE table. Therefore, we have to find the Employee id of doctor Nguyen N, then find the number of patients treated by him.

```sql
SELECT COUNT(*)
FROM db_assignment.`patient record` as r
WHERE r.DOCTOR_Employee_id =
    (SELECT d.Employee_id
     FROM db_assignment.doctor as d
    INNER JOIN db_assignment.employee as e
    ON d.Employee_id = e.EMPLOYEE_id
    WHERE e.FirstName = "A" AND e.LastName = "Nguyen")
```

COUNT(*)
4

Figure 17: Number of patients of doctor Nguyen A

4.4 Stored procedures

4.4.1 Get monthly report of patient records

For our first store procedure, we would like to store the query to retrieve the complete form of a medical record like the first query. Because of that, we just need to store the query above into a procedures called Get_report.

```sql
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `Get_report`()
BEGIN
    SELECT *
    FROM db_assignment.`patient record` as r
    INNER JOIN db_assignment.`patient profile` as p ON  p.Patient_id = r.`patient profile_Patient_id`;
END$$
DELIMITER ;
```

Figure 18: Procedure creation code

```sql
CALL `db_assignment`.`Get_report`();
```

Record_id	Start_date	End_date	DOCTOR_Employee_id	patient profile_Patient_id	Patient_id	First_name	Last_name	Gender	Dob	Addres
1	2019-05-12 00:00:00	2020-01-01 ...	2	1	1	K	Huynh	1	2000-05-04 00:00:00	NULL
2	2018-04-05 00:00:00	2021-06-07 ...	3	2	2	L	Vu	1	1994-03-03 00:00:00	NULL
3	2020-04-06 00:00:00	2021-08-09 ...	1	3	3	M	Ngo	0	1980-01-02 00:00:00	NULL
4	2017-04-04 00:00:00	2018-05-01 ...	1	4	4	N	Nguyen	0	1960-05-06 00:00:00	NULL
5	2019-02-04 00:00:00	2021-05-04 ...	1	5	5	O	Nguyen	1	1990-06-04 00:00:00	NULL
6	2021-01-01 00:00:00	2021-04-05 ...	1	4	4	N	Nguyen	0	1960-05-06 00:00:00	NULL

Figure 19: Procedure call result

4.4.2 Get number of patients stays in one department

Next, a department would like to know how many patients have stayed in their building. Therefore we create a procedure call named Get_number_of_patients to get the number of patients stays in one department.Regarding the SQL code, we have to how many patients have stayed in which rooms that belongs to which department.

```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `Get_number_of_patients`()
BEGIN
    SELECT  t.Department_id, t.DepartmentName, COUNT(*) AS "Number of patients"
    FROM `db_assignment`.`stay_in` AS s
    INNER JOIN (
        SELECT *
        FROM `db_assignment`.`room` AS r
        INNER JOIN `db_assignment`.`department` AS d ON r.DEPARTMENT_Department_id = d.Department_id
        ) AS t
    ON t.Room_id = s.ROOM_Room_id
    GROUP BY t.Department_id;
END$$
```

Figure 20: Procedure creation

```sql
CALL `db_assignment`.`Get_number_of_patients`();
```

Department_id	DepartmentName	Number of patients
3	Pediatrics	2
5	Gastroenterology	1

Figure 21: Procedure call result

From the result, we could say that there are 2 patients stay in Pediatrics department and 1 patient stay in Gastroenterology department.

4.4.3 Get a list of tests that were made by the hospital

Last but no least, our final procedure is about getting a list of tests that were made by the hospital. We know that a hospital conducts a lot of tests. Therefore we have to get a report about all kinds of tests have been made.

```sql
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `Get_test_report`()
BEGIN
    SELECT *
    FROM db_assignment.`do_test` as do
    INNER JOIN db_assignment.`test` as test ON do.TEST_Test_id = test.Test_id;
END$$
DELIMITER ;
```

Figure 22: Procedure call result

```sql
CALL `db_assignment`.`Get_test_report`();
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

PATIENT RECORD_Record_id	TEST_Test_id	Result	Date	Test_id	Test_name	Description
1	5	NULL	2019-05-13 00:00:00	5	Blood test	NULL
2	2	NULL	2018-04-06 00:00:00	2	CT Scan	NULL
3	3	NULL	2020-04-07 00:00:00	3	MRI	NULL
4	1	NULL	2017-04-05 00:00:00	1	Echocardiogram	NULL
5	4	NULL	2019-02-05 00:00:00	4	X-Ray	NULL
6	1	NULL	2021-01-02 00:00:00	1	Echocardiogram	NULL

Figure 23: Procedure call result

From our query, we could see the whole list of tests were conducted by the hospitals.

5 Database security

5.1 Types of database security

Database security is a broad area that addresses many issues, including the following:

- Various legal and ethical issues regarding the right to access certain information, for example, some information may be deemed to be private and cannot be accessed legally by unauthorized organizations or persons.
- Policy issues at the governmental, institutional, or corporate level regarding what kinds of information should not be made publicly available, for example, credit ratings and personal medical records.
- System-related issues such as the system levels at which various security functions should be enforced, for example, whether a security function should be handled at the physical hardware level, the operating system level, or the DBMS level.
- The need in some organizations to identify multiple security levels and to categorize the data and users based on these classifications, for example, top secret, secret, confidential, and unclassified.

5.2 Threats to databases

Threats to databases can result in the loss of degradation of some or all of the following commonly accepted security goals:

- Loss of integrity: Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creating, inserting, and updating data; changing the status of data; and deleting data. Integrity is lost if unauthorized changes are made to the data by either intentional or accidental acts. If the loss of system or data integrity is not corrected, continued use of the contaminated system or corrupted data could result in inaccuracy, fraud, or erroneous decisions.
- Loss of availability. Database availability refers to making objects available to a human user or a program who/which has a legitimate right to those data objects. Loss of availability occurs when the user or program cannot access these objects.
- Loss of confidentiality. Database confidentiality refers to the protection of data from unauthorized disclosure. The impact of unauthorized disclosure of confidential information can range from violation of the Data Privacy Act to the jeopardization of national security. Unauthorized, unanticipated, or unintentional disclosure could result in loss of public confidence, embarrassment, or legal action against the organization.

5.3 Solution

When considering the threats facing databases, it is important to remember that the database management system alone cannot be responsible for maintaining the confidentiality, integrity, and availability of the data. Rather, the database works as a part of a network of services, including applications, Web servers, firewalls, SSL terminators, and security monitoring systems. Because security of an overall system is only as strong as its weakest link, a database may be compromised even if it would have been perfectly secure on its own merits.

To protect databases against these types of threats, four kinds of countermeasures can be implemented:

1. Access control
2. Inference control
3. Flow control
4. Data encryption

5.3.1 Access control

A DBMS typically includes a database security and authorization subsystem that is responsible for ensuring the security of portions of a database against unauthorized access. It is now customary to refer to two types of database security mechanisms:

- Discretionary Access control. These are used to grant privileges to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update).
- Mandatory access control. These are used to enforce multilevel security by classifying the data and users into various security classes (or levels) and then implementing the appropriate security policy of the organization. For example, a typical security policy is to permit users at a certain classification (or clearance) level to see only the data items classified at the user's own (or lower) classification level. An extension of this is role-based security, which enforces policies and privileges based on the concept of organizational roles.

5.3.2 Inference control

Inference control in databases, also known as Statistical Disclosure Control (SDC), is a discipline that seeks to protect data so they can be published without revealing confidential information that can be linked to specific individuals among those to which the data correspond. SDC is applied to protect respondent privacy in areas such as official statistics, health statistics, e-commerce (sharing of consumer data), etc. Since data protection ultimately means data modification, the challenge for SDC is to achieve protection with minimum loss of the accuracy sought by database users.

5.3.3 Data encryption

A control measure is data encryption, which is used to protect sensitive data (such as credit card numbers) that is transmitted via some type of communications network. Encryption can be used to provide additional protection for sensitive portions of a database as well. The data is encoded using some coding algorithm. An unauthorized user who accesses encoded data will have difficulty deciphering it, but authorized users are given decoding or decrypting algorithms (or keys) to decipher the data.

5.3.4 Flow control

The flow control helps prevents information from flowing in such a way that it reaches unauthorized users. A flow policy lists out the channels through which information can flow. It also defines security classes for data as well as transactions.

6 Develop an application on top of the Database system

Any database system would expose an interface so that the users can hook on to this connection and perform tasks on the database. The big problem is different databases require different properties for a connection to be established. To not put a burden on the database users, we as developers need to do the technical work and only expose what the users need.

With that having been said, we decided to create a webserver that provides a RESTful API. We will demonstrate the application on the AMBULANCE table. Our server lives at `localhost:3000` as seen here.

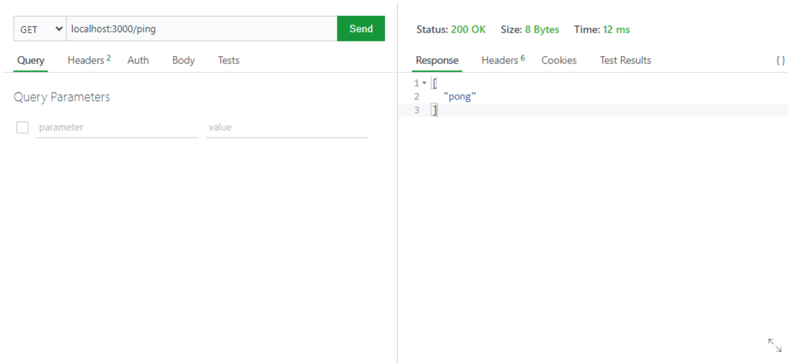

Figure 24: Test ping to the application

- Create: We will add data to the table at /ambulance/new using POST method and form-data. We will insert 2 entries into the table.

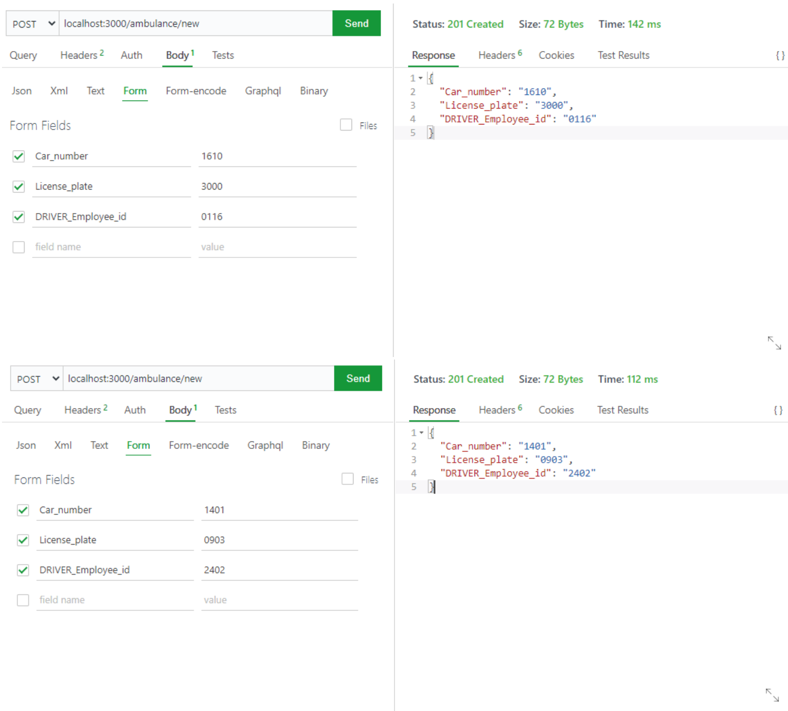

Figure 25: Adding entries to the table

- Retrieve: We will get data from the table at /ambulance/all using GET method.

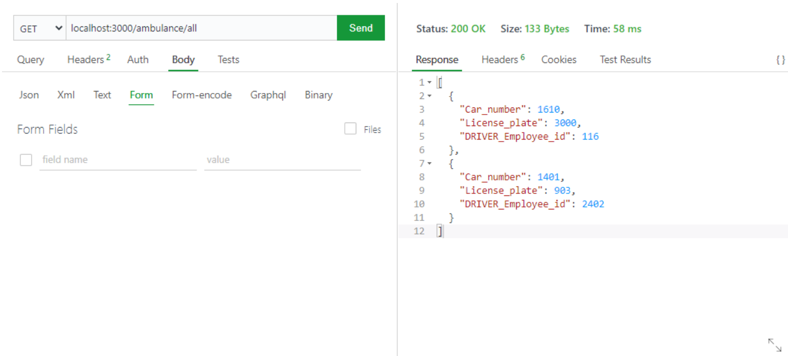

Figure 26: Get all entries of table

We also provide a method to get data of one entry from the table at /ambulance/<Car_number> using GET method.

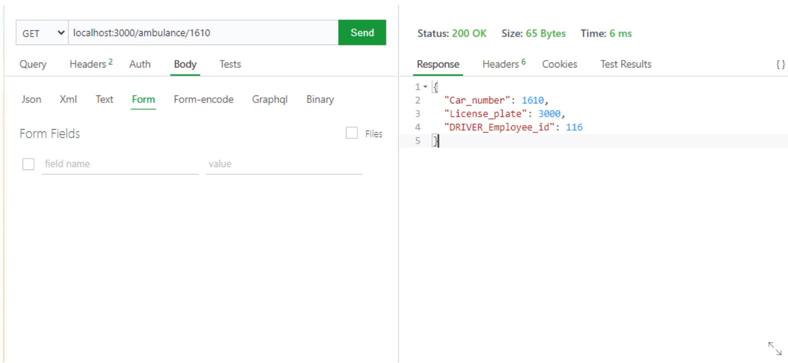

Figure 27: Get one entry of table

- Delete: We will now remove one entry from the table at /ambulance/<Car_number>/delete using DELETE method.

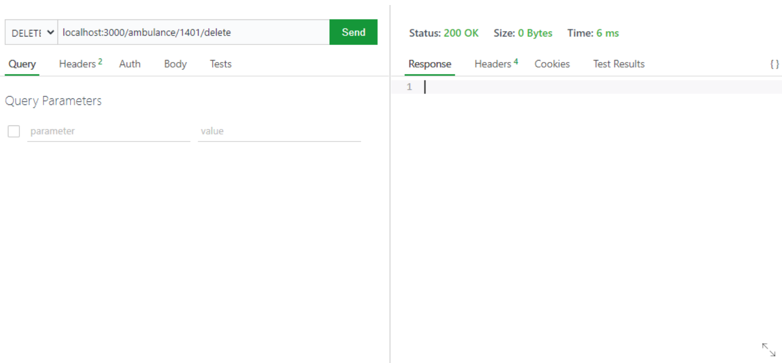

Figure 28: Delete the entry whose Car number is 1401

We call /ambulance/all again to verify that the entry has been deleted.

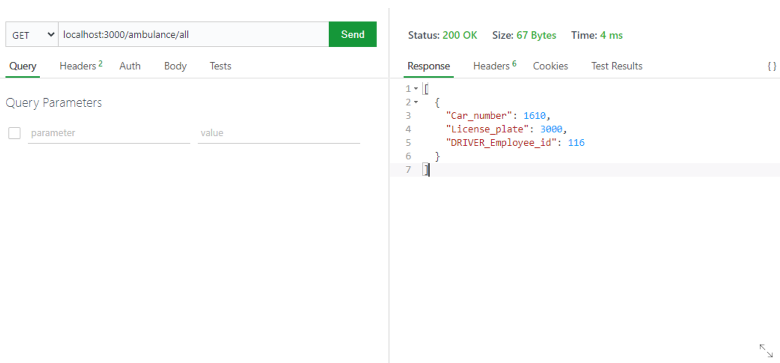

Figure 29: The result after we delete the entry whose Car number is 1401

You can find the full source code at: https://github.com/Smithienious/CO2014-Asg/tree/dev/rest-api

7 Conclusion

Through this assignment, we have learned how to improve our database and make it more suitable for business and real life uses. First of all, we have discussed about database normalization and applied it straight away to our Hospital database management to reduce the redundant data and enhance the consistency of the whole database. Second of all, we have talked about the database security such as the types of database security, the security threats and some solutions that are usually considered to the database security problem. Finally, we have written some complex queries, triggers, stored procedures as well as create a simple application above this database. Therefore, we can say that although there are still many things to learn, accomplishing this assignment has surely helped us to understand clearer about the structure of a database as well as the process of designing databases in a logical way for business purposes.

References

[1] R. Elmasri & S.B. Navathe (2016): Fundamentals of Database Systems, 7th Edition, Addison-Wesley.

[2] Nate Lord (2020): Definiton of Data encryption, accessed with: https://digitalguardian.com/blog/what-data-encryption

[3] TutorialsPoint: Database security, accessed with: https://www.tutorialspoint.com/distributed_dbms/distributed_dbms_database_security_cryptography.htm

[4] Josep Domingo-Ferrer, Inference Control in Statistical Databases, accessed with: https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-39940-9_203

[5] GeeksforGeeks: Different types of MySQL Triggers (with examples), accessed with: https://www.geeksforgeeks.org/different-types-of-mysql-triggers-with-examples/

[6] MySQL: Trigger Syntax and Examples, accessed with: https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html

[7] W3school: SQL Stored Procedures for SQL Server, accessed with: https://www.w3schools.com/sql/sql_stored_procedures.asp