# **PROJECT REPORT**

# **DOCTOR-PATIENT APPOINTMENt DATABASE cum app**





**Ayushi Jain, Sujoy Roy, Vaibhav Kaushik, Rachit Trivedi, Rajat Rai.**

**CS231 – DATABASE MANAGEMENT SYSTEM**

**Professor Amit Kumar**

# VIEW OF THE PROJECT

The main aim of this project is to provide a well-functioning database and an app which allows a hassle-free booking of appointment between Doctor-Patient in Hospitals of private sector. All the functionalities of a role model Hospitals are taken into consideration and have been credited in this project.

# ABOUT THE PROJECT

There has been a melancholy amongst patients of standing/Long waiting in queue for several hours in OPD or for simple diagnosis which in turn make them annoyed and disinterests them from getting themselves regularly diagnosed. In world of Technology at our fingertips, People prefer to have an APP which can sought the very problem with ease and comfort, rather long waiting in Queues. This Doctor-Patient Appointment App rely on the concept of databases to fulfil people’s wishes. This project tends to demonstrate the working of the database of Interaction between Doctors and Patients on a large scale.

# PROBLEM STATEMENT

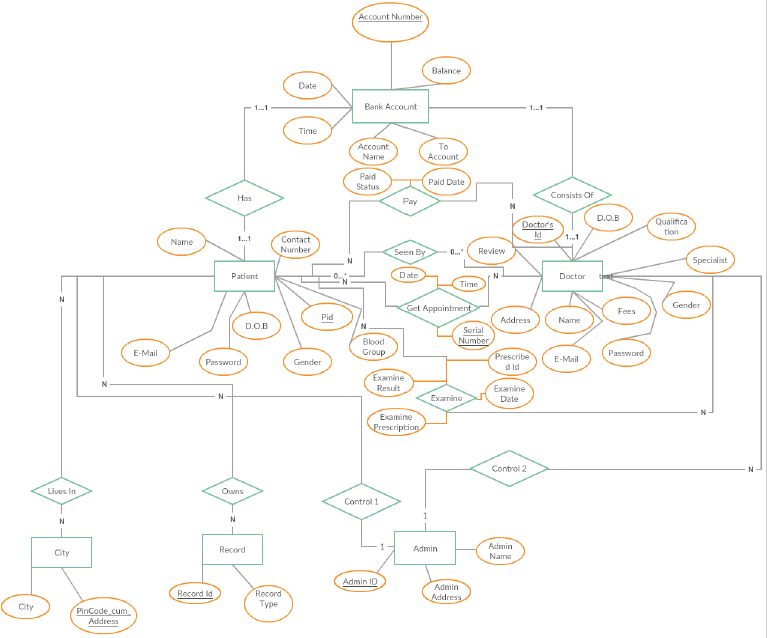
Consider the following information about an online appointment system database.

* Patients have name, patient ID, Date of Birth, Gender, Password, Email, Contact number, Blood Group.
* Bank Accounts have Date, Time, Account Name, To Account, Balance and Account Number.
* Doctors have Name, Address, Doctor’s ID, DOB, Qualifications, Specialization, Gender, Password, Fees, Review and Email.
* Patient can undergo examination prescribed by doctor which has prescribed id, examination results, examination date and Examination prescription.
* Patient lives in a City which has City Name, Pincode\_cum\_Address.
* Patient can have some records which will have Record ID and Record Type.
* Admin can have Admin Name, Admin ID, and Admin Address.
* Patient can get doctor’s appointment which has serial number, Date and Time.
* Patient can pay doctors which has Payment status and Payment Date.
* Patient and Doctor Can have only one bank account.
* Many patients can pay one doctor.
* The appointment request of a patient can be seen by a doctor.
* Any number of patients can get any number of appointments from different doctors.
* A doctor can prescribe multiple tests to many patients.
* One patient can live in many cities and can have several records.

This project is for the demo purpose of showing the working of an online appointment database.

Note: Also, Patient ID, Doctor’s ID and Account No are maintained by the website, and do not relate to any government authorized ID (like driving license, PAN, etc.)

# ENTITY – RELATIONSHIP DIAGRAM



## **ER TO RELATIONAL MODEL CONVERSION**

# RELATIONAL SCHEMA

Consider the following information about an online appointment system database.

Patients (name varchar , patient\_ID int , DOB int, Gender char, Password varchar, Email varchar , Contact\_Number int, Blood\_Group varchar)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **PID** | Name | Contact Number | D.O.B | Gender | E-MAIL | PASSWORD | Blood Group |

Bank\_Accounts ( Date date, Time time, Account\_Name varchar, To\_Account int, Balance int, Account\_Number int).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Acc. No.** | Acc. Name | Balance | To Account | Date | Time |

Doctors ( Name varchar, Address varchar, Doctor\_ ID int, DOB int, Qualifications varchar, Specialization varchar, Gender char, Password varchar, Fees int, Review varchar, Email varchar).

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DID** | Name | D.O.B | Qualification | Specialization | EMAIL | Password | Gender | Fees | Address | Review |

Examination (Prescribed id int, examination\_Results varchar, examination\_Date date, Examination\_Prescription varchar)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PID.DID** | Prescribed ID | Examine Prescription | Date | Result |

City (City\_Name varchar, Pincode\_cum\_ Address varchar)

|  |  |
| --- | --- |
| **PinCode\_Cum\_Address** | City |

Records (Record\_ID int, Record\_Type varchar)

|  |  |
| --- | --- |
| **Record Id** | Record Type |

Admin (Admin\_Name varchar, Admin\_ID int, Admin Address varchar)

|  |  |  |
| --- | --- | --- |
| **Admin Id** | Admin Name | Admin Address |

Appointment (Date date , serial number int, Time time)

|  |  |  |  |
| --- | --- | --- | --- |
| **PID.DID** | Serial Number | Date | Time |

Pay (Payment\_status int, Payment\_Date)

|  |  |  |
| --- | --- | --- |
| **PID.DID** | Date | Status |

# SQL CODE

1. Bank Account

Create table Bank\_Account

(

Account\_number integer (25) not null,

Account\_name char (30),

To\_account integer (20),

Balance integer (20),

B\_acc\_Date varchar(30), - For Date

B\_acc\_TimeTime, - Time

Primary key(Account\_number),

Unique key(Account\_number)

);

\*\*\*Relation between bank Account and Patient & Doctor: ----b/w bank acc & patient

* Has relation:

Create table Has

(

Pid integer (10),

Primary key (Pid.Account Number),

FOREIGN KEY (Account Number) REFERENCES Account Bank

FOREIGN KEY(Pid) REFERENCES Patient,

);

* Consists relation: -------------------- b/w bank acc and Doctor:

Create table Consists

( Doctor’s\_id integer (10),

Primary key (Doctor’s\_id. Account Number),

FOREIGN KEY (Account Number) REFERENCES Account Bank

FOREIGN KEY(Doctor’s\_id) REFERENCES Doctor);

1. Patient

Create table patient

(

Pid integer (10),

Name varchar (20),

Contact\_number integer (10),

Email varchar (20),

Password varchar (20),

Gender char (10),

Blood\_group varchar (5),

D.O.B varchar (10),

Primary key (Pid),

Unique key(Pid)

);

\*\*\*Relation b/w patient and city:

* Lives in:

Create table lives in

(

Pid integer (10),

Pincode\_cum\_Address varchar (50),

Primary key (Pid. Pincode\_cum\_Address),

FOREIGN KEY(Pid) REFERENCES Patient,

FOREIGN KEY(Pincode\_cum\_Address) REFERENCES City

);

1. City

Create table city (

Pincode\_cum\_Address varchar (50),

City\_name char (20),

Primary key (Pincode\_cum\_Address),

Unique key (Pincode\_cum\_Address)

);

\*\*\*Relation: b/w patient and record:

* Owns:

Create table Owns

( Pid integer (10),

Record\_id integer (10),

PRIMARY KEY (Pid.Record Id);

FOREIGN KEY(Pid) REFERENCES Patient,

FOREIGN KEY(Record\_id) REFERENCES Record

);

1. Records

Create table records

(

Record\_id integer (10),

Record\_type varchar (20),

Primary key(Record\_id),

Unique key(Record\_id),

);

1. Doctor

Create table doctor

(

Doctor\_id integer (10),

Name char (10),

Address varchar (20),

D.O.B varchar (10),

Qualifications varchar (20),

Specialist varchar (10),

Gender varchar (10),

Fees double,

Password varchar (20),

Review char (20),

Email varchar (10),

Primary key (Doctor\_id),

Unique key(Doctor\_id)

);

\*\*\*\*Relation b/w doctor and patient:

* Seen by:

Create table lives in

(

Pid integer (10),

Doctor’s\_id integer (10),

Pid.Doctor’s id integer (10),

PRIMARY KEY(Pid.Doctor’s id),

FOREIGN KEY(pid) REFERENCES Patient,

FOREIGN KEY(Doctor’s Id) REFERENCES Doctor

);

\*\*\* Relation Pay: b/w doctor and patient:

* Pay

Create table Pay(

Pid integer (10),

Doctor\_id integer (10),

Pid\_Doctors\_Id integer (20),

Paid\_Date varchar(20), -For Date

Paid\_Status varchar(20), -For Time

PRIMARY KEY(Pid.Doctor\_id),

FOREIGN KEY(pid) REFERENCES Patient,

FOREIGN KEY(Doctor’s Id) REFERENCE Doctor

);

\*\*\* Relation Examine: b/w doctor and patient:

* Examine

Create table Examine

( Pid integer (10),

Doctor\_id integer (10),

Examine\_Prescription varchar(50),

Examine\_Result varchar(20),

Prescription\_Id integer (10),

PRIMARY KEY(Pid.Doctor\_id),

FOREIGN KEY(pid) REFERENCES Patient,

FOREIGN KEY(Doctor’s Id) REFERENCE Doctor

);

1. Admin

Create table admin

(

Admin\_id integer (10),

Admin\_name char (10),

Admin\_address varchar (20),

Primary key(Admin\_id),

Unique key(Admin\_id)

);

\*\*\*Relation b/w doctor and patient:

* Get Appointment:

Create table Get Appointment

(

App\_Date varchar(30),

App\_Time varchar(30),

Serial\_Number integer (20),

Doctor’s\_id integer (10),

Pid integer (10),

Pid.Doctor’s\_ Id integer (10);

PRIMARY KEY(pid.Doctor’s id),

UNIQUE KEY(Serial Number),

FOREIGN KEY(pid) REFERENCES Patient,

FOREIGN KEY(Doctor’s Id) REFERENCES Doctor

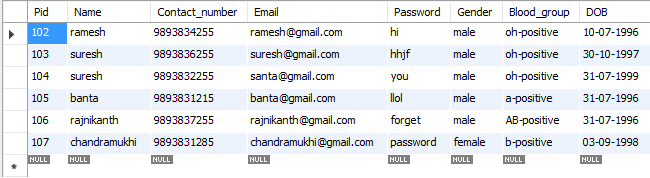
);

# NORMALIZATION

For the normalization process, we have recognized the functional dependencies in each table and then proceed to achieve the best suited normal form.

## Patient:

Patients table has Eight (8) attributes: Pid, Name, contact number, Email Id, Password, Gender, Blood\_group and D.O.B. Here, Pid is unique for each Patient and hence is the primary key. However, it may be possible that two different patients in different locations have same name,gender, Blood Group and a DOB and different too(vice-versa). Moreover, Email.Password – only Email to be precise (A.B->x.y.z) is required as a key to access any information of particular patient which also determines every other attribute except Pid. So Pid, E-mail and contact number are uniquely determined. Thus, we can’t find a specific patient’s address using patient name, Gender, Blood Group and DOB(vice-versa) only as it may **return multiple values**.



Patient (**Pid**, Name, contact number, Email Id, Password, Gender, Blood\_group , D.O.B)

**Primary Key**: Pid

**Candidate keys**: Pid, contact number, E-mail id.

**Prime Attributes**: Pid, contact number, E-mail id.

**Functional dependencies -**

Pid-> Name, contact number, Email Id, Password, Gender, Blood\_group , D.O.B

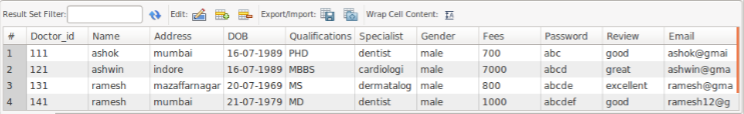
Email->Pid, Name, contact number, Password, Gender, Blood\_group , D.O.B

Contact number-> Pid, Name, Email Id, Password, Gender, Blood\_group , D.O.B

This table is, therefore, already in BCNF (as all the values on the left of functional dependencies are a super key) and no mulit-valued dependency is found, thus in **4NF**

## DOCTORS

Doctors table also has Eleven (11) attributes, which too are bit similar to Patients table, i.e. Doctor\_id, Name, Address, D.O.B, Qualifications, Specialist, Gender, Fees, Password, Review, Email. The Doctor\_id is chosen here as the primary key as it is unique to a Doctor and can be used to determine the Address, D.O.B, Qualifications, Specialist, Gender, Fees, Password, Review, E-mail of the Doctor or the Name of the Doctor. Both Doctor\_id and E-mail are uniquely determined. On the other hand, it may be possible that two Doctors based in different locations can have the same name, D.O.B, Qualifications, Specialist, Gender, Fees, Review. So we can’t fetch the address of the Doctor based on the name and every other attribute only.



Doctor ( Doctor\_id, Name, Address, D.O.B, Qualifications, Specialist, Gender, Fees, Password, Review, Email)

Primary Key: Doctor\_id

Candidate keys: Doctor\_id, E-mail id.

Prime Attributes: Doctor\_id, E-mail id.

**Functional dependencies -**

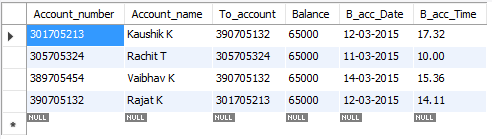
Doctor\_id-> Name, Address, D.O.B, Qualifications, Specialist, Gender, Fees, Password, Review, Email

Email id-> Doctor\_id, Name, Address, D.O.B, Qualifications, Specialist, Gender, Fees, Password, Review

This table is, therefore, already in BCNF (as all the values on the left of functional dependencies are a super key) and no mulit-valued dependency is found, thus in **4NF.**

## BANK ACCOUNT

In the Account Bank Table, there are 6 attributes, namely: Account\_number, Account\_name, To\_account, Balance, Date and Time. As given in problem statement, one account bank can be opened by only one Doctor and patient. Hence, the Account\_number can be used to identify the Doctor and Patient using “Consists” and “has” Relation Respectively.



Bank Account (Account\_number, Account\_name, To\_account, Balance, Date, Time)

Primary Key: Account\_number

Candidate keys: Account\_number

Prime Attributes: Account\_number

**Functional dependencies -**

Account\_number->Account\_name, To\_account, Balance, Date, Time

* This table is thus in **BCNF** as all the values on the left of functional dependencies are a super key. Also, none of the non-key attribute is determining each other.
* But mulit-valued dependency is found. Due to which there is a violation of 4NF. These MVD’s are the following:

Name->-> Account\_number

To\_account->-> Account\_number

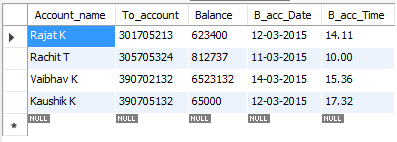
Balance->-> Account\_number

Date->-> Account\_number

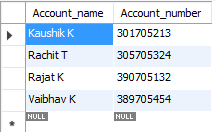
Time->-> Account\_number

So we decompose the table to make it in 4NF

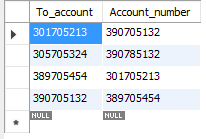
Bank Account 1(Account\_name, To\_account, Balance, Date, Time)



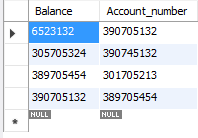
Bank Account 2(Account\_Name, Account\_number)



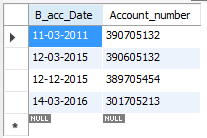
Bank Account 3(To\_account , Account\_number)



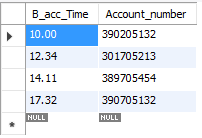
Bank Account 4(Balance , Account\_number)



Bank Account 5(Date , Account\_number)

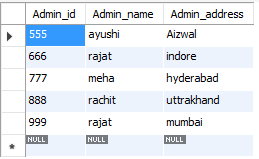


Bank Account 6(Time , Account\_number)



## ADMIN

The orders table has got 3 attributes. They are: Admin\_id, Admin\_name, Admin\_address. An Admin\_id is specific to a particular admin only. However, it may be possible that two different Admin in different locations have same name. Thus, we can’t find a specific Admin’s address using Admin name only as it may return multiple values.



Admin ( Admin\_id, Admin\_name, Admin\_address)

Primary Key: Admin\_id

Candidate keys: Admin\_id

Prime Attributes: Admin\_id

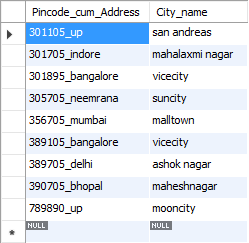
**Functional dependencies -**

Admin\_id->Admin\_name, Admin\_address

This table is, therefore, already in BCNF (as all the values on the left of functional dependencies are a super key) and no mulit-valued dependency is found, thus in **4NF.**

## City

The orders table has got 2 attributes. They are: Pincode\_cum\_address, and City\_name. Pincode\_cum\_address is unique for every Town, district or a city. However, it may be possible that two different City in different states have same name. Thus, we can’t find a specific city’s address using City\_name only as it may return multiple values.



City (Pincode\_cum\_address, and City\_name)

Primary Key: Pincode\_cum\_address

Candidate keys: Pincode\_cum\_address

Prime Attributes: Pincode\_cum\_address

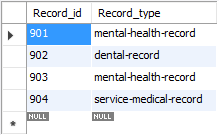
**Functional dependencies -**

Pincode\_cum\_Address->City\_name

* This table is, therefore, already in BCNF (as all the values on the left of functional dependencies are a super key) and no mulit-valued dependency is found, thus in **4NF.**

## RecorD

The Record table has got 2 attributes. They are: Record\_id and Record\_type. Record\_id is unique. However, it may be possible that two different Record id in different Hospitals may have same Record\_type.



Record (Record\_id, Record\_type)

Primary Key: Record\_id

Candidate keys: Record\_id

Prime Attributes: Record\_id

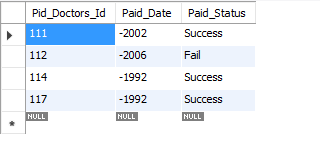
**Functional dependencies -**

Record\_id->Record\_type

* This table is, therefore, already in BCNF (as all the values on the left of functional dependencies are a super key ) and no mulit-valued dependency is found, thus in **4NF.**

## Pay- RElation between doctor and patient

The Pays table has got 3 attributes. They are: Pid.Did, Paid Status and Paid Date. In this ( Pid.Doctor\_id ) is specific and a primary key. Also, since we are having many patients performing transaction in the Pay, the Pid.Doctor\_id is determining Paid Status and Paid Date too.



Pay (Pid.Did, Paid Status, Paid Date)

Primary Key: Pid.Did

Candidate keys: Pid.Did

Prime Attributes: Pid.Did

**Functional dependencies -**

Pid.Doctor\_id->Paid Status ,Paid Date

But the catch here is that the Paid Status can be defined by Pid.Doctor\_id and Paid Status undergoes multivalued Dependency, following are the multivalued dependencies possible:

Pid.Doctor\_id->-> Paid Status Pid.Doctor\_id->-> Paid Date Paid Date->->Pid.Doctor\_id Paid Date->->Paid Status

Paid Status->->Paid Date Paid Status->->Paid Date

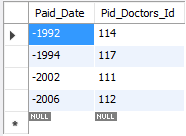
Here, Pid.Doctor\_id is the candidate key but Paid Date and Paid Status aren’t and they are present on the L.H.S but not a key. Following are the MVD which doesn’t have a key on the L.H.S:

Paid Date->->Pid.Doctor\_id, Paid Date->->Paid Status, Paid Status->->Paid Date, Paid Status->-> Pid.Doctor\_id

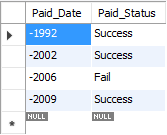
Therefore, these are the cases where it violates 4NF. So we decompose the table to make it in 4NF.

Pay(Pid.Did, Paid Status, Paid Date): No decomposition required.

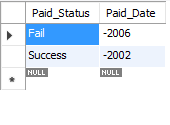
Pay (Paid Date, Pid.Did)



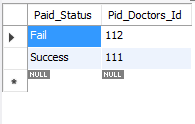
Pay (Paid Date, Paid Status)



Pay (Paid Status, Paid Date)

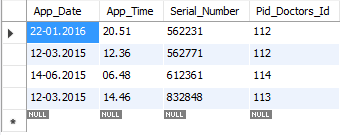


Pay ( Paid Status, Pid.Did)



## get appointment- RElation between doctor and patient

The Get\_Appointment table has got 4 attributes. They are: Pid.Doctor\_id, Serial No, Date and Time. In this ( Pid.Doctor\_id ) is specific and a primary key. Serial number is also a key. Also, since we are having many patients Booking their appointment with various Doctors on different intervals of Time, the Pid.Doctor\_id is determining Serial No, Time and Paid Date too.



Get\_Appointment (Pid.Doctor\_id, Serial No, Date,Time)

Primary Key: Pid.Doctor\_id

Candidate keys: Pid.Doctor\_id, Serial No

Prime Attributes: Pid.Doctor\_id, Serial No

**Functional dependencies -**

Pid.Doctor\_id-> Serial No, Date,Time

Serial No-> Pid.Doctor\_id, Date,Time

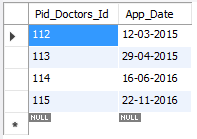
Following are the multivalued dependencies possible:

Pid.Doctor\_id->-> Serial no.Time

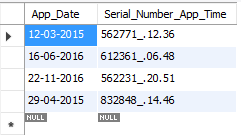
Date->-> Serial no.Time

Here, Pid.Doctor\_id is the candidate key but Date isn’t and it is present on the L.H.S but not a key. Following are the MVD which doesn’t have a key on the L.H.S: Date->-> Serial no.Time. Therefore, this is the only case where it violates 4NF. So we decompose the table to make it in 4NF.

Get\_Appointment (Pid.Doctor\_id, Date)



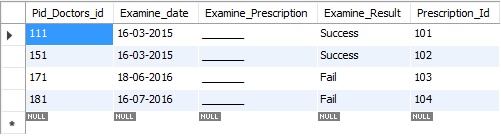
Get\_Appointment1 (Date, Serial No. Time)



Hence, this table is thus in **4NF**.

## Examine- RElation between doctor and patient

The pays table has got 5 attributes. They are: Pid.Doctor\_id, Prescription\_id, Examine Prescription, Examine Date and Examine Result. In this (Pid.Doctor\_id) is specific and a primary key. Prescription id is unique. Also, since we are having various patients being examined by several doctors, the Pid.Doctor\_id, Examine Prescription, Examine Date, Examine Result are determining Prescription\_id in all cases respectively.



Examine (Pid.Doctor\_id, Prescription\_id, Examine Prescription, Date, Result)

Primary Key: Pid.Doctor\_id

Candidate keys: Pid.Doctor\_id, Prescription\_id

Prime Attributes: Pid.Doctor\_id, Prescription\_id

**Functional dependencies -**

Pid.Doctor\_id-> Prescription\_id, Examine Prescription, Date, Result

Prescription\_id-> Pid.Doctor\_id, Examine Prescription, Date, Result

But the catch here is that the Prescription\_id is be defined by Pid.Doctor\_id, Examine Prescription, Examine Date and Examine Result undergoes multivalued Dependency:

Pid.Doctor\_id->-> Prescription\_id.

Examine Prescription->-> Prescription\_id.

Examine Date->->Prescription\_id.

Examine Result->-> Prescription\_id.

Here, Pid.Doctor\_id is the candidate key but all others are not and are present on the L.H.S which are not a key. Following are the MVD which don’t have a key on the L.H.S:

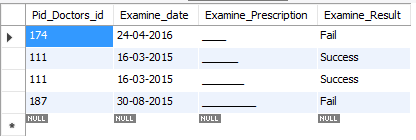
Examine Prescription->-> Prescription\_id.

Examine Date->->Prescription\_id.

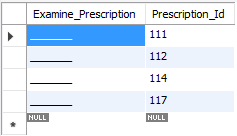
Examine Result->-> Prescription\_id.

Therefore, this is the only case where it violates 4NF . . So we decompose the table to make it in 4NF.

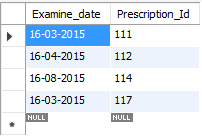
Examine1 (Pid.Doctor\_id, Examine Prescription, Date, Result)



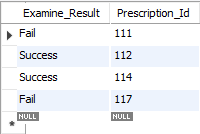
Examine2 (Examine Prescription, Prescription\_id)



Examine 3(Date, Prescription\_id)



Examine 4(Result, Prescription\_id)

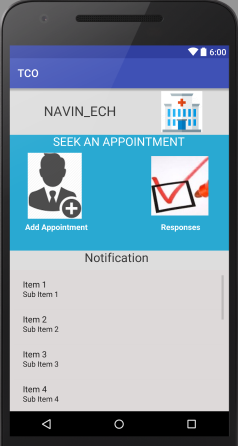
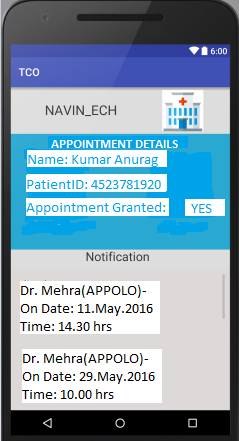


# Showcasing Real-Time App Processing:

As promised, we are done with our App. Only few issues with linking some segments of Service with our App Activity. For Reference regarding service, you can look upon this link: <https://github.com/Zaiku1972/docapp/blob/master/v1/index.php>

Rest you can view below:

Figure 1: Welcome Interface Figure 2: Patient Interface after Booking an appointment.

# CONCLUSION

Through this project, we were able to show the basic functioning of Doctor-Patient based online Appointment HUB. Moreover, we have been able to create our requisite service and Beta-Phase working App which is additional w.r.t D.B.M.S Project. Along with that, we were also able to show the process of normalizing the tables. We performed BCNF normalization on each of the tables. Thus our Database The reason for choosing that was:

* Data integrity was achieved.
* Amount of data duplicated was reduced to a great extent.

# GROUP MEMBERS

* RACHIT TRIVEDI– U101114FCS110
* VAIBHAV KAUSHIK – U101114FCS0152
* AYUSHI JAIN-U101114FCS
* RAJAT RAI– U101113FCS221
* SUJOY ROY– U101114FCS191

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-o-\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_