Atal Bihari Vajpayee- Indian Institute of Information Technology and Management

विश्वजीवनामृतं ज्ञानम्

Database Management System Project

Quarantine Center Management Portal

Submitted to:

Dr. Neetesh Kumar

Website of the Project:

https://theninza.github.io/quarantineportal/

(**Default Password is 1234**

Submitted by:

Nikhil Gupta (BCS2019-036)
Harshit Singh (BCS2019-025)
Aditi Singh (BCS2019-001)
Yashpal Parmar (BCS2019-073)

TABLE OF CONTENTS

Introduction

Covid-19 Quarantine Centers:

Quarantine centers, as the name suggests, are the hub where the people, who are suspected to have Covid-19 disease are admitted. The main motive of these centers is to make sure that the spread of the disease can be stopped by not letting the suspect to come in the contact of others in case the infection is for real. Following the outbreak of corona virus, the government established tens of thousands of these quarantine centers. The schools, hotels, clinics, that were kept closed due to nationwide lockdown have been converted into quarantine centers.

Brief about the project:

The project tries to demonstrate a Quarantine Center Management Portal at different levels (details in section III). The Portal let users manage the centers, patients and staffs at those centers and requests (demand of some service) from the center.

The project uses MySQL for the management of database system and its tables are normalized to 3NF for managing data redundancy. The whole database along with all the constraints can be made available to the local machine by running the **script.sql** file given with this report.

The source code for the file can be found at following links:

Frontend: https://github.com/TheNinza/quarantineportal

Backend: https://github.com/TheNinza/dbs-backend

Contributions:

Nikhil Kumar Gupta (2019BCS-036)

Made the complete application (frontend and backend) and deployed it.

Aditi Singh (2019BCS-001)

Designed the Database, ER diagram, Normalized the tables.

Harshit Singh (2019BCS-025)

Wrote SQL queries for the operations on the database.

Yashpal Parmar (2019BCS-073)

Edited the project report. Provided sample data for the project.

DATABASE DESIGN

ENTITIES AND ATTRIBUTES:

I. **User**
   A. user_id
   B. user_name
   C. user_email
   D. user_phone.
II. **user_role**
   A. user_role_id
   B. user_role_name
   C. user_role_description
III. **center**
   A. center_id
   B. center_name
   C. center_address
   D. center_contact_number
   E. number_patients
   F. number_staffs
IV. **center_type**
   A. center_type_id
   B. center_type_description
   C. staff
   D. staff_id
   E. staff_name
   F. staff_contact_number
   G. working_hours
V. **staff**
   A. staff_id
   B. staff_name
   C. staff_contact_number
   D. staff_working_hours
VI. **staff_role**
   A. role_id
   B. role_name
   C. role_description

VII. **patient**

A. patient_id
B. patient_name
C. date_of_admission
D. stay_duration
E. patient_address
F. patient_status

VIII. **req_id**

A. request_id
B. request_description

IX. **request_status**

A. status_id
B. status_name

RELATIONS:

1. 'manages' is a *one-to-many* relation between 'user' and 'center'
   User_id FOREIGN KEY in 'center' from 'user'

2. 'belongs to' is a *many-to-one* relation between 'center' and 'center_type'
   center_type_id FOREIGN KEY in 'center' from 'center_type

3. 'is admitted at' is a *many-to-one* relation between 'patient' and 'center'
   center_id FOREIGN KEY in 'patient' from 'center'

4. 'works at' is a *many-to-one* relation between 'staff' and 'center'
   center_id FOREIGN KEY in 'staff' from 'center'

5. 'sends' is a *one-to-many* relation between 'center' and 'service_request'
   center_id FOREIGN KEY in 'service_request' from 'center'

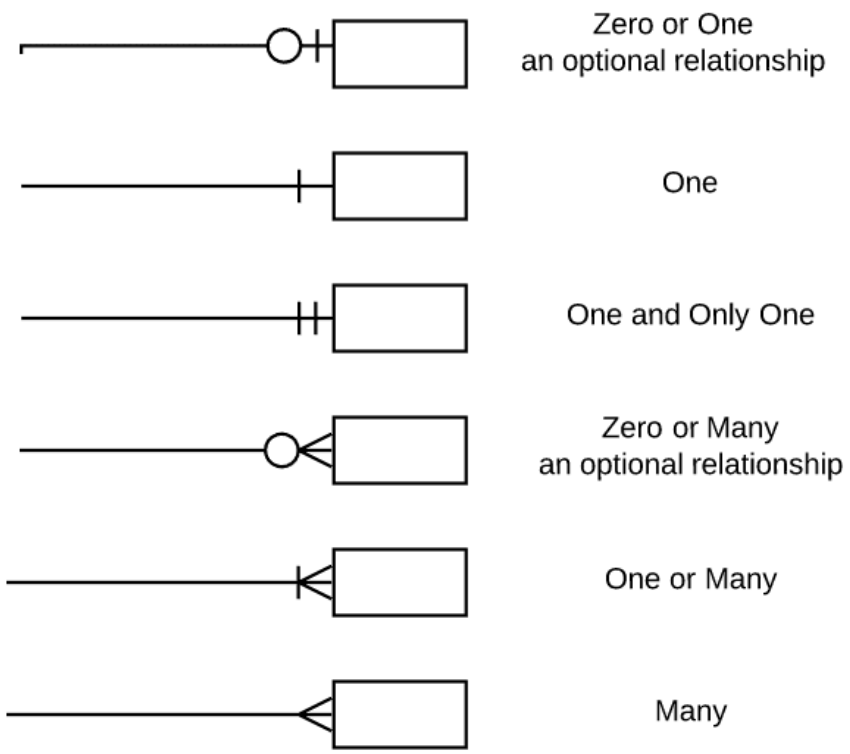
6. 'processes' is a *one-to-many* relation between 'user' and 'service_request'
   user_id FOREIGN KEY in 'service_request' from 'user

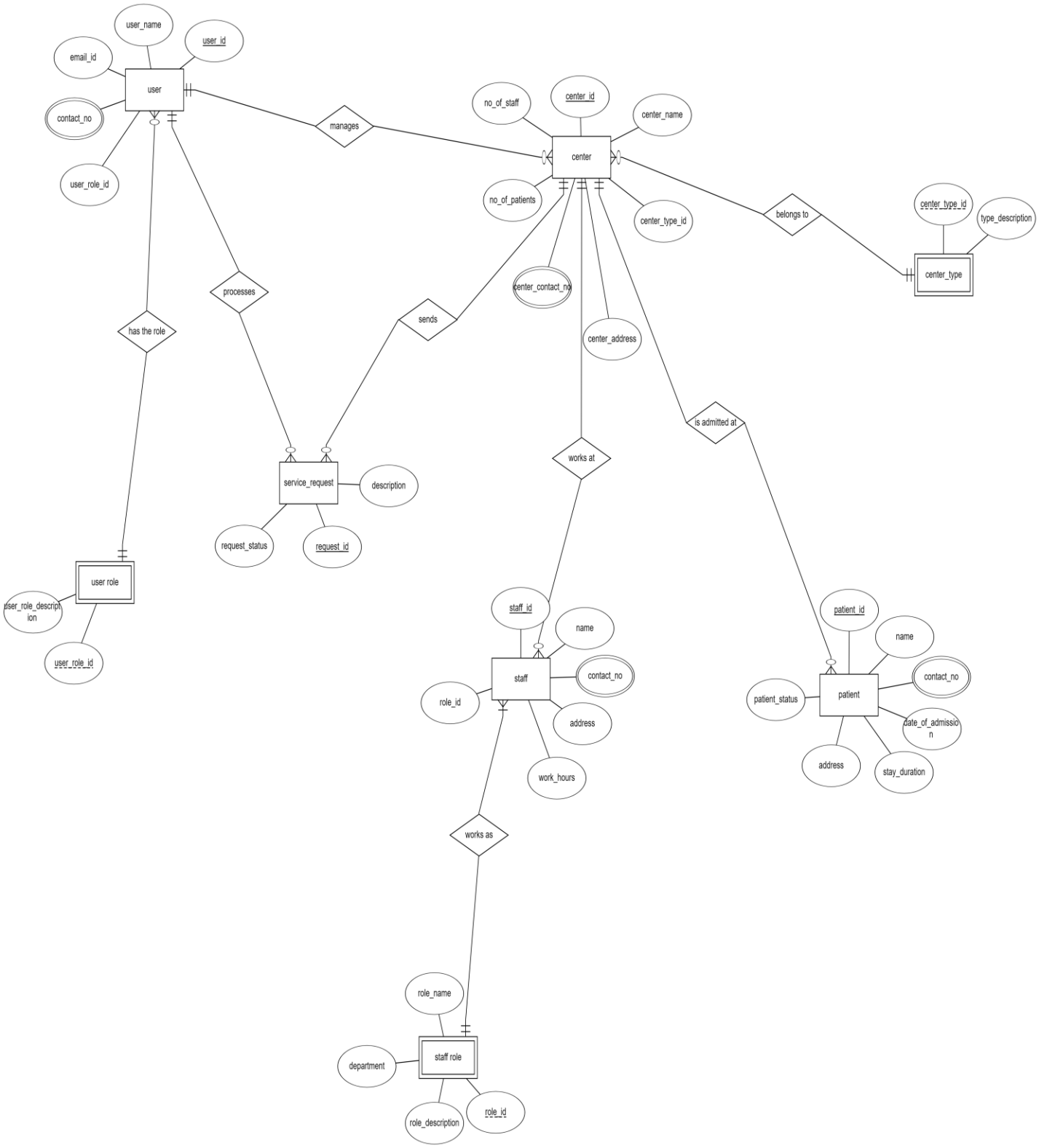
7. 'has the role' is a *many-to-one* relation between 'user' and 'user_role'
   user_role_id FOREIGN KEY in 'user' from 'user_role'

8. ‘works as’ is a *many-to-one* relation between ‘staff’ and ‘staff_role’
   role_id FOREIGN KEY in ‘staff’ from ‘staff_role’

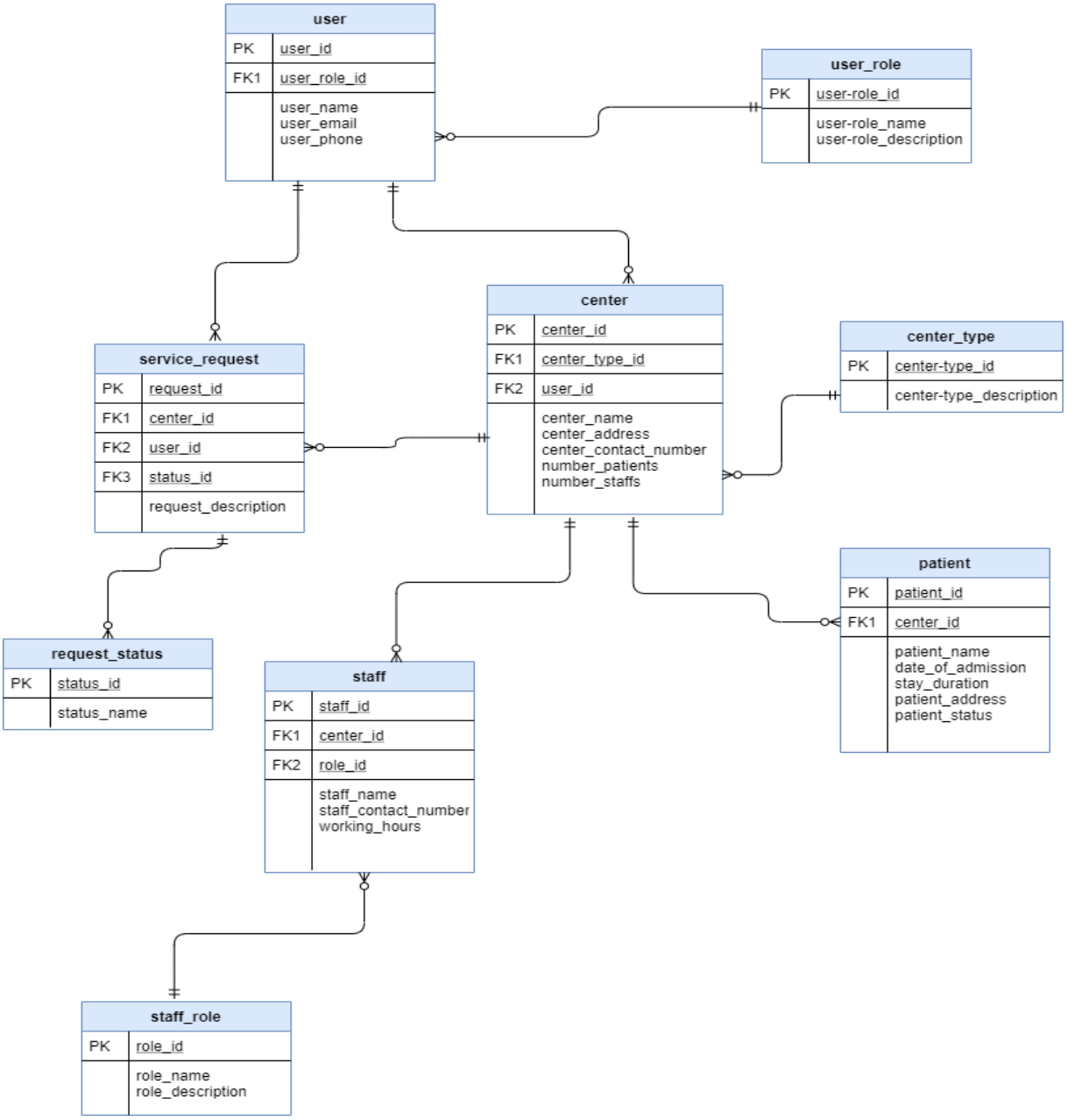
ENTITY RELATIONSHIP DIAGRAM

Crow’s foot notation:




user_name
user_id
email_id
user
contact_no
user_role_id
manages
no_of_staff
center_id
center_name
center
no_of_patients
center_type_id
belongs to
center_type_id
type_description
center_type
center_contact_no
processes
sends
has the role
center_address
is admitted at
works at
service_request
description
request_status
request_id
user role
user_role_descripti
ion
user_role_id
staff_id
name
staff
contact_no
role_id
address
work_hours
patient_id
name
patient
contact_no
patient_status
date_of_admissio
n
address
stay_duration
works as
role_name
staff role
department
role_description
role_id

RELATIONAL SCHEMA DIAGRAM



NORMALISATION

R1 - center_info

A. center_id
B. center_name
C. center_type_id
D. center_type_description
E. center_address
F. center_contact_number
G. number_patients
H. number_staffs

F1-{ A→ B,C,D,E,F,G,H

C→ D }

After normalisation to 3NF:

R11 - center

(A, B, C, E, F, G, H) F11- {A→ BCEFGH}

R12- center_type

(C,D) F12-{C→ D}

R2- user_info

A. user_id
B. user_role_id
C. user-role_name
D. user-role_description
E. user_name
F. user_email
G. user_phone

F2- {A→ BCDEFG

B→ CD}

After normalisation to 3NF

R21- user

(A,B,E,F,G) F21-{A→ BEFG}

R22- user_role

(B,C,D) F22-{B→ CD}

R3- staff_info

A. staff_id
B. center_id
C. role_id
D. role_name
E. role_description
F. staff_name
G. staff_contact_number
H. working_hours

F3-{ A→ BCDEFGH

C→ DE}

After normalisation to 3NF

R31- staff

(A,B,C,F,G,H) F31-{A→ BCFGH}

R32- staff_role

(C,D,E) F32-{C→ DE}

R4- request_info

A. Request_id
B. Center_id
C. User_id
D. Status_id
E. Status_name
F. request_description

F4- {A→ BCDEF

D→ E}

After normalisation to 3NF

R41- service_request

(A,B,C,D,F) F41-{A→ BCDF}

R42- request_status

(D,E) F42-{D→ E}

R5- patient

A. Patient_id
B. Center_id
C. patient_name
D. date_of_admission
E. stay_duration
F. patient_address
G. patient_status

F5-{A→ BCDEFG}

It is normalised to 3NF

Tables After Normalization

center_Id	center_type_id	center_name	user_id	center_contact_number	center_address	number_patients	number_staffs
1	2	Xijiao	10	361-824-3919	62208 Jackson Way	100	27
2	3	Halteu	4	938-776-3251	13 Wayridge Terrace	4	17
3	3	Campo Formoso	3	639-918-6441	5 Ramsey Park	96	22
4	2	Bendungan	5	504-780-1082	24391 Roth Avenue	77	16
5	3	Hengyang	2	596-584-2835	123 Arrowood Way	52	11
6	4	Xipi	4	140-394-7696	29087 Fisk Way	6	24
7	3	Fīrūzābād	4	448-811-9234	54205 Randy Lane	46	22
8	1	Lama	7	752-265-2487	95841 Clyde Gallagher Way	60	22
9	2	Ortigueira	2	951-538-4473	502 Mayer Avenue	85	14
10	3	Pitanga	2	829-868-4173	567 Reindahl Center	75	17
11	4	Delok	9	746-962-1248	9 Ilene Junction	15	16
12	1	Dovhe	3	325-114-1523	686 Lerdahl Court	78	6

center

Center_type_id	Center_type_description
1	School
2	Hospital
3	Hotel
4	Custom Quarantine Centers

center_type

user_Id	user_role_id	user_name	user_phone	user_email
1	2	Janaye Uden	945-168-9155	juden0@drupal.org
2	2	Gayelord Crop	235-190-9563	gcrop1@upenn.edu
3	2	Berna Scahill	625-359-9830	bscahill2@exblog.jp
4	1	Cynthia Sircomb	986-816-0701	csircomb3@army.mil
5	1	Weston Fewell	865-647-9009	wfewell4@dot.gov
6	2	Daven Tutchings	123-895-2169	dtutchings5@tripadvisor.com
7	1	Dwayne Ferrick	185-887-7565	dferrick6@desdev.cn
8	2	Barb Spincks	604-108-3611	bspincks7@com.com
9	2	Renell Paddell	144-779-5492	rpaddell8@photobucket.com
10	2	Stephie Bodker	742-821-5460	sbodker9@uiuc.edu

user

User_role_id	User_role_name	User_role_description
1	Database Administrator	It has control to all the functionality of the system
2	Center Manager	Manages the individual center
3	Government Official	It manages the centers

user_role

staff_id	center_id	staff_name	role_id	staff_contact_number	working hours
1	7	Jessika Renfrew	2	710-672-3461	1600-1600
2	19	Bevan Northall	1	157-855-3104	1400-1300
3	3	Kelcy Stocks	1	666-305-7177	1800-1100
4	16	Vivia Mostin	3	765-998-4582	1500-1100
5	4	Ami Kamen	3	367-704-6408	1600-1000
6	4	Thorin Jillins	1	407-933-7189	1200-1200
7	18	Chev Bachman	1	971-851-6017	1100-1300
8	18	Shauna Lorryman	4	949-932-7502	1700-1600
9	4	Stillman O'Dowling	3	371-372-0499	1700-1000
10	17	Wrennie Craney	5	905-398-3085	1300-1400
11	1	Hailee Destouche	5	428-747-8847	1400-1200

staff

Role_id	Role_name	Role_description
1	Nurse	takes care of the health requirements of the patients.
2	Sweeper	Cleans the Quarantine Center
3	Cook	Makes Food for all
4	Labour	Does the physical works as needed.
5	Electrician	Manages the electrical components

staff_role

Request_id	Center_id	User_id	Status_id	Request_description
1	14	15	3	eco-centric
2	10	19	1	concept
3	19	9	2	Pre-emptive
4	12	13	1	analyzer
5	1	19	3	object-oriented
6	19	2	2	array
7	9	9	3	secondary
8	13	11	1	Adaptive
9	4	11	1	eco-centric
10	11	18	3	Multi-layered
11	11	17	2	Organized

service_request

Status_id	Status_name
3	discarded
2	processed
1	unsolved

request_status

Patient_id	Center_id	Patient_name	Date_of_admission	Patient_address	Patient_status
1	1	Martyn Camillo	16/04/2020	35866 Union Place	User-centric
2	8	Herman Murricanes	27/01/2020	3239 Huxley Junction	methodology
3	17	Kaylee Cranham	02/03/2020	6 Elmside Road	high-level
4	9	Rhoda Powlesland	17/04/2020	5 Stuart Street	Multi-layered
5	13	Tyne Bashford	07/05/2020	0 Dawn Lane	regional
6	8	Jacki Robardey	05/04/2020	59 Moose Way	Exclusive
7	1	Salaidh Clerc	23/04/2020	00639 Forest Dale A...	multi-state
8	15	Sergeant Scoone	03/03/2020	6689 Thackeray Point	attitude
9	12	Bernadine Cakes	28/04/2020	41 Bluestem Court	Assimilated
10	17	Alyce Janic	12/05/2020	12140 Shasta Terrace	Expanded
11	11	Hansiain Flageul	13/03/2020	6314 Melrose Point	solution-oriented
12	18	Ewell Narbett	03/03/2020	2 Schmedeman Trail	analyzing

patient

Implementation of tables:

The tables and the database can be created by the SQL scripts given in the project directories.

The code for making those tables is given below:

1. user_role

```sql
CREATE TABLE `user_role` (
    `user_role_id` int NOT NULL AUTO_INCREMENT,
    `user_role_name` varchar(45) NOT NULL,
    `user_role_description` varchar(200) NOT NULL,
    PRIMARY KEY (`user_role_id`)
);
```

2. user

```sql
CREATE TABLE `user` (
  `user_id` int NOT NULL AUTO_INCREMENT,
  `user_name` varchar(45) NOT NULL,
  `user_email` varchar(45) DEFAULT NULL,
  `user_phone` varchar(45) NOT NULL,
  `user_role_id` int DEFAULT NULL,
  PRIMARY KEY (`user_id`),
  KEY `user-role_id_idx` (`user_role_id`),
  CONSTRAINT `users_role` FOREIGN KEY (`user_role_id`) REFERENCES `user_role` (`user_role_id`)
);
```

3. center_type

```sql
CREATE TABLE `center_type` (
    `center_type_id` int NOT NULL AUTO_INCREMENT,
    `center_type_description` varchar(200) NOT NULL,
    PRIMARY KEY (`center_type_id`)
);
```

4. center

```sql
CREATE TABLE `center` (
  `center_id` int NOT NULL AUTO_INCREMENT,
  `center_name` varchar(50) NOT NULL,
  `center_address` varchar(100) NOT NULL,
  `center_contact_number` int DEFAULT NULL,
  `number_patients` int DEFAULT '0',
  `number_staffs` int DEFAULT '0',
  `center_type_id` int DEFAULT NULL,
  `user_id` int DEFAULT NULL,
  PRIMARY KEY (`center_id`),
  UNIQUE KEY `center_manager` (`user_id`),
  KEY `center_type` (`center_type_id`),
  CONSTRAINT `center_manager` FOREIGN KEY (`user_id`) REFERENCES `user` (`user_id`) ON DELETE SET NULL,
  CONSTRAINT `center_type` FOREIGN KEY (`center_type_id`) REFERENCES `center_type` (`center_type_id`)
);
```

5. patient

```sql
CREATE TABLE `patient` (
  `patient_id` int NOT NULL AUTO_INCREMENT,
  `patient_name` varchar(50) NOT NULL,
  `date_of_admission` varchar(20) NOT NULL,
  `stay_duration` int NOT NULL,
  `patient_address` varchar(100) DEFAULT NULL,
  `patient_status` varchar(100) DEFAULT NULL,
  `center_id` int DEFAULT NULL,
  PRIMARY KEY (`patient_id`),
  KEY `patient_center` (`center_id`),
  CONSTRAINT `patient_center` FOREIGN KEY (`center_id`) REFERENCES `center` (`center_id`) ON DELETE SET NULL ON UPDATE CASCADE
);
```

6. service_request

```sql
CREATE TABLE `service_request` (
  `request_id` int NOT NULL AUTO_INCREMENT,
  `request_description` varchar(200) NOT NULL DEFAULT 'invalid request',
  `center_id` int DEFAULT NULL,
  `user_id` int DEFAULT NULL,
  `status_id` int DEFAULT '1',
  PRIMARY KEY (`request_id`),
  KEY `req_status` (`status_id`),
  KEY `user_processing` (`user_id`),
  KEY `from_center` (`center_id`),
  CONSTRAINT `from_center` FOREIGN KEY (`center_id`) REFERENCES `center` (`center_id`) ON DELETE SET NULL ON UPDATE SET NULL,
  CONSTRAINT `req_status` FOREIGN KEY (`status_id`) REFERENCES `req_status` (`status_id`),
  CONSTRAINT `user_processing` FOREIGN KEY (`user_id`) REFERENCES `user` (`user_id`) ON DELETE SET NULL ON UPDATE SET NULL
);
```

7. req_status

```
CREATE TABLE `req_status` (
  `status_id` int NOT NULL AUTO_INCREMENT,
  `status_name` varchar(10) NOT NULL,
  PRIMARY KEY (`status_id`),
  UNIQUE KEY `req_status_status_name_uindex` (`status_name`)
);
```

8. staff

```
CREATE TABLE `staff` (
  `staff_id` int NOT NULL AUTO_INCREMENT,
  `staff_name` varchar(100) NOT NULL,
  `staff_contact_number` int NOT NULL,
  `role_id` int DEFAULT NULL,
  `center_id` int DEFAULT NULL,
  `working_hours` varchar(20) NOT NULL,
  PRIMARY KEY (`staff_id`),
  KEY `staff_center` (`center_id`),
  KEY `staff_role` (`role_id`),
  CONSTRAINT `staff_center` FOREIGN KEY (`center_id`) REFERENCES `center` (`center_id`) ON DELETE SET NULL ON UPDATE CASCADE,
  CONSTRAINT `staff_role` FOREIGN KEY (`role_id`) REFERENCES `staff_role` (`role_id`)
);
```

9. staff_role

```
CREATE TABLE `staff_role` (
  `role_id` int NOT NULL AUTO_INCREMENT,
  `role_name` varchar(50) NOT NULL,
  `role_description` varchar(200) NOT NULL,
  PRIMARY KEY (`role_id`),
  UNIQUE KEY `staff_role_role_name_uindex` (`role_name`)
);
```

We added constraints like FOREIGN KEY in the code itself so that the system handles all the error at the database level.

Functioning of the Database Application

The user can access the website having one of the three following roles.

a. Database Administrator (The manager of the whole database)
b. Government Official (Controls different quarantine centers)
c. Center Manager (Controls only one quarantine center)

Login:

After the user clicks on login (assuming he/she is logging as 'Database Administrator') the backend performs the following query on the database:

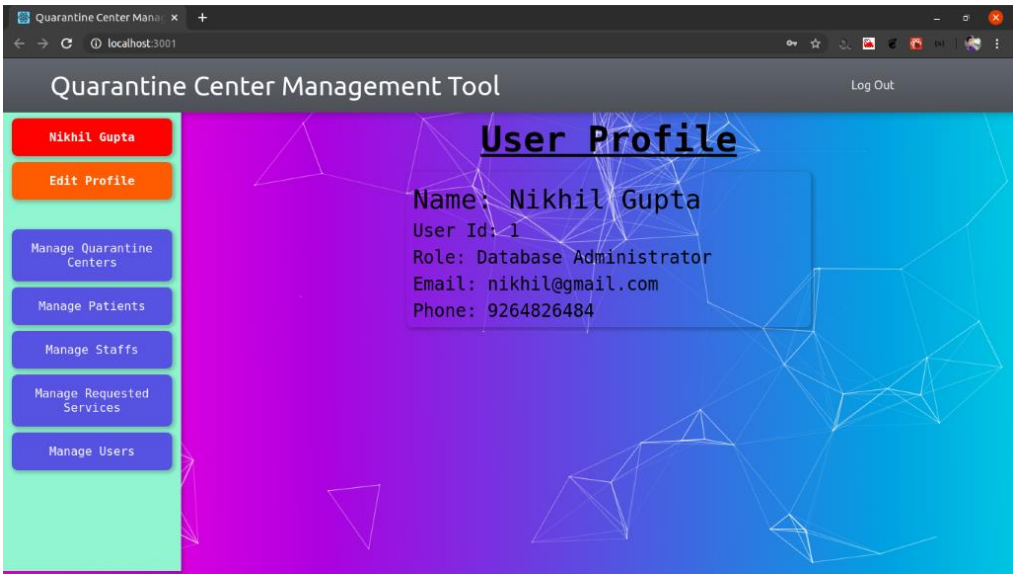
```
select * from user
    inner join user_role ur on user.user_role_id = ur.user_role_id
    where ur.user_role_name = 'Database Administrator';
```

And logs you in with the first result of the query.

Similarly, you can login as other two roles.

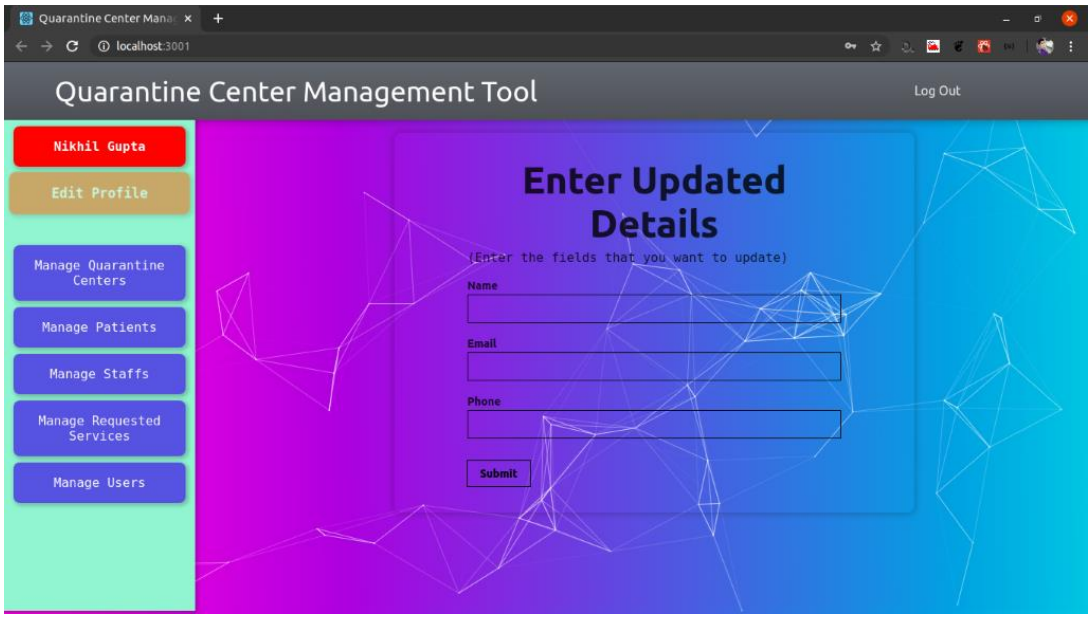
User Profile:

The User Profile for each user looks as following:



Edit Profile:

The Edit Profile section of every user role looks as following:



The user can only the Name, Email and Phone of his entry in the database. Major changes can only be done by the Database administrator in Manage Users Section

For each of the above-mentioned sections (say, user_name for the user_id 1) to be updated, the database performs the following query:

```
update user set user_name = 'Updated Name' where user_id = 1;
```

Similarly, for other sections the backend performs the similar query.

The Sidebar:

The Sidebar contains the actions that the user can perform on the database. Given below are the sidebars of all the roles.

Database Administrator

Government Official

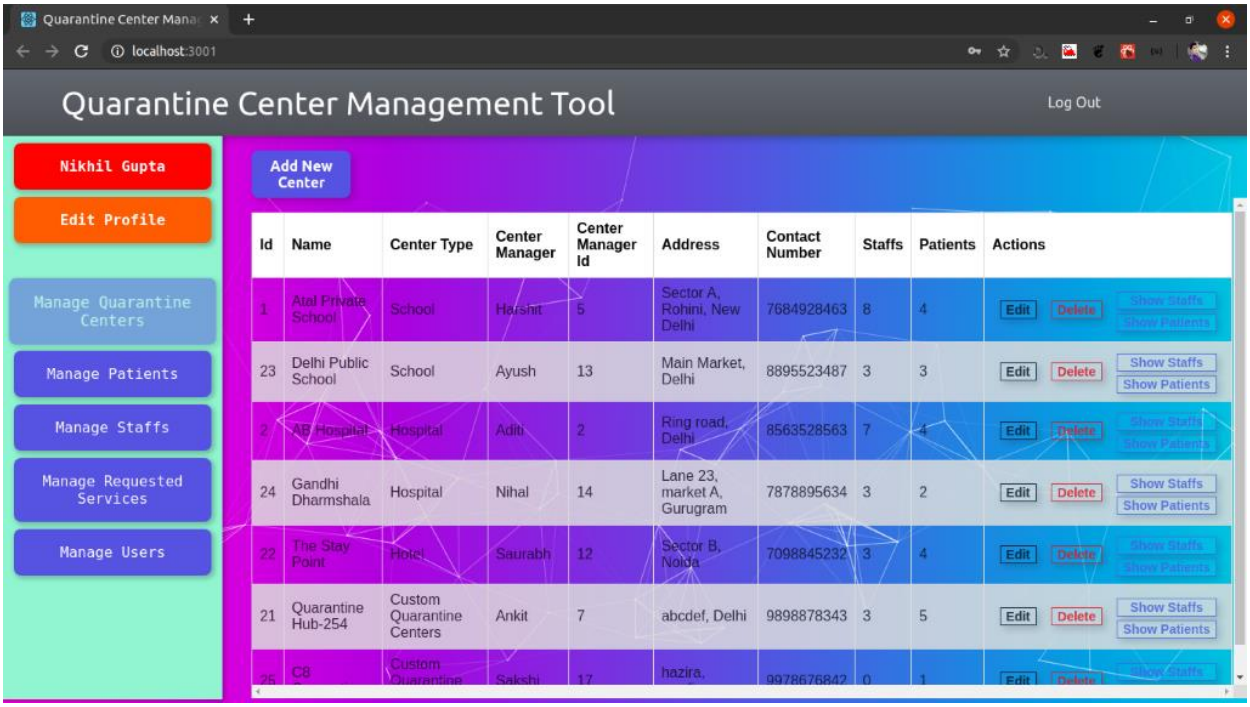
Center Manager

As you can clearly see, not all users can perform all actions, and this is completely kept taken off by the frontend.

Also, there are several restrictions in using those actions. Let's discuss the those and the SQL queries related to them.

Manage Quarantine Centers:

This can only be accessed by the Database Administrator and the Government Official.



For accessing all the centers along with their center managers and type, we perform following query:

```
select * from center
    inner join center_type ct on center.center_type_id = ct.center_type_id
    left join user u on center.user_id = u.user_id;
```

Note that, we are doing a left join on ‘user’ table because we also want to get those Quarantine centers, which are not assigned any manager and can be assigned one.

Also, each time when retrieving the centers, we also update the numbers of staffs and patients in that center using the following query:

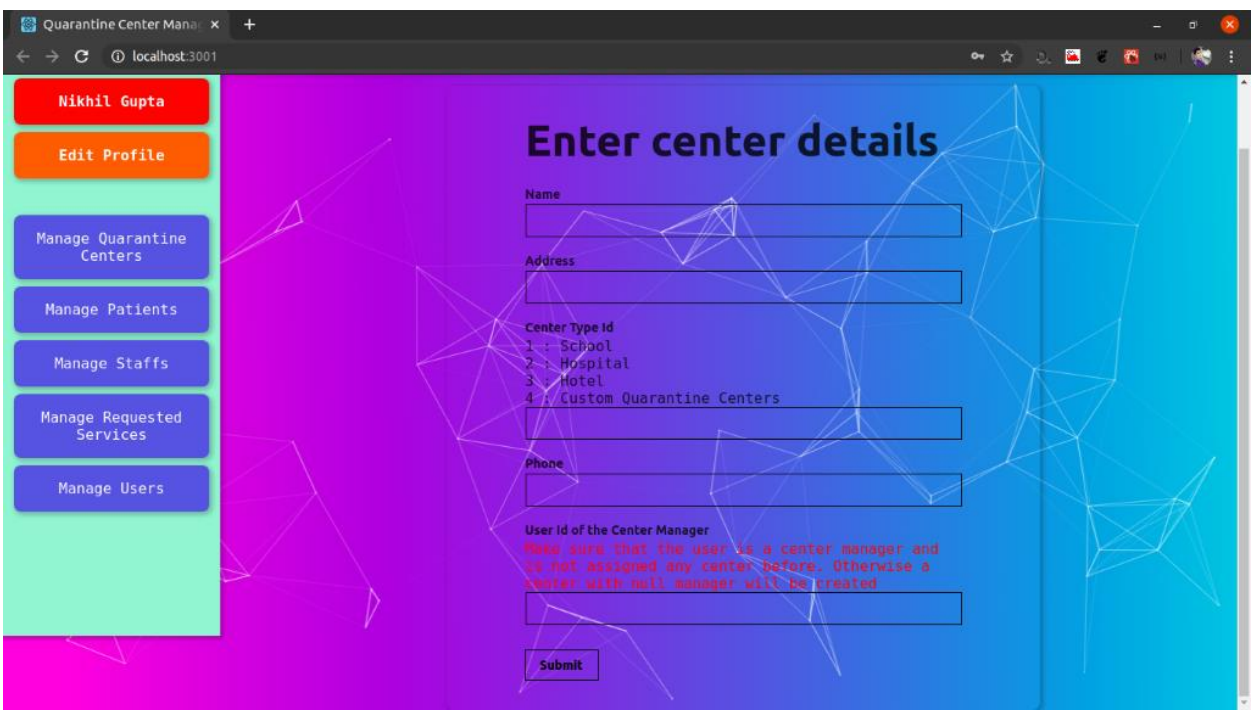
```
update center
    set number_staffs = (
        select count(*) from staff where staff.center_id = center.center_id
    );

update center
    set number_patients = (
        select count(*) from patient where patient.center_id = center.center_id
    );
```

Above query updates all the rows of the center table.

Adding a new Quarantine Center:

A new quarantine center can be added to the list using the New Center button. The new-center section looks like following:



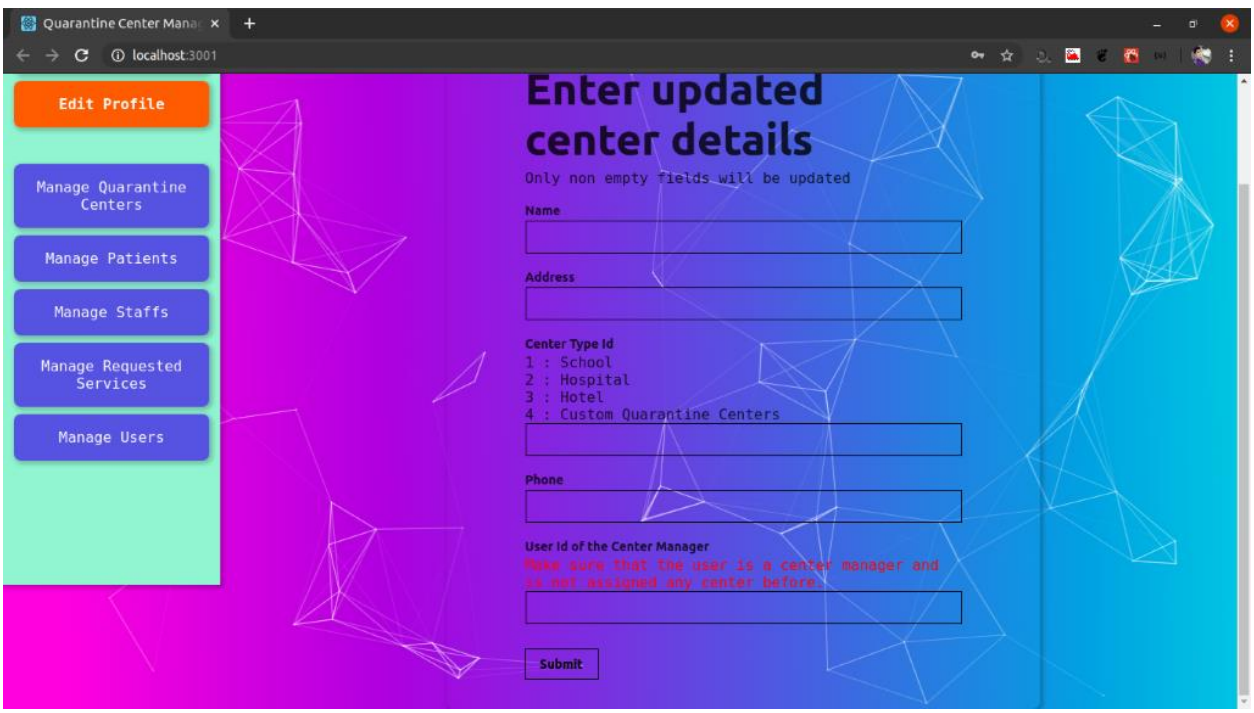
After entering all the fields and clicking the submit button, the backend performs a query similar to the following one:

```
insert into center
    (center_name, center_address, center_contact_number, center_type_id, user_id)
    values
        ('center name','center_address','contact number', 22, 2);
```

Note that we are not giving the center_id , number_patients, number_staffs values because all the primary keys of the database including the center_id is set to follow the AUTO INCREMENT method for self-generation and number_patients and number_staffs are set default to value 0.

Edit Center:

Any quarantine center can be edited by clicking the corresponding EDIT button in the Actions column. The section looks like following:



Again, the number_patients and number_staffs section is not provided for making an edit on those because we are already updating those from ‘patient’ and ‘staff’ table as mentioned earlier.

Now for making an edit on each field (say, name), we run the following query:

```
update center set center_name = 'updated name' where center_id = 22;
```

Similarly, we run similar queries for editing other sections.

Delete Center:

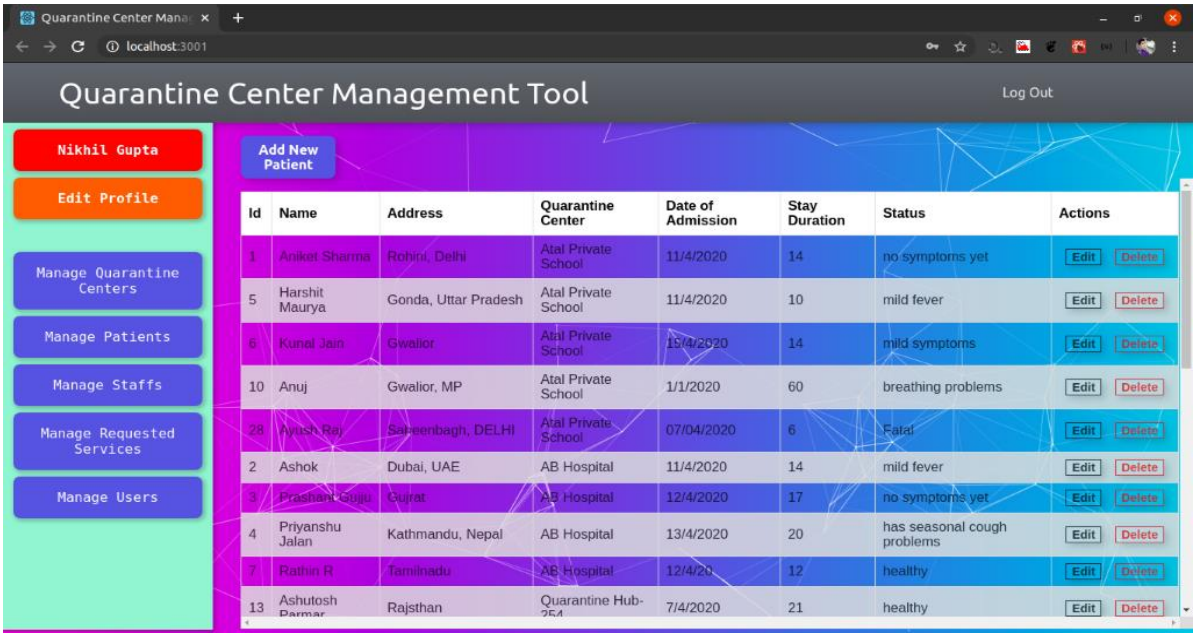
Any quarantine center can be deleted by clicking the corresponding DELETE button in the Actions column.

We run the following query for the deletion.

```
DELETE FROM center WHERE center_id = 22;
```

Manage Patients

This action is available to all users. But when logged in as Center Manager, the user can only see the patients of his/her quarantine center. The section looks as following when viewed as a database administrator:



For getting the patients, we perform the following query:

```
select * from patient inner join center c on patient.center_id = c.center_id;
```

New Patient:

(For rest of the report, the application interface is same as that of Manage Quarantine Center. The website can be accessed by the address provided on the first page. We really encourage you to check the website.)

For a new patient we run the following query:

```
insert into patient
    ( patient_name, date_of_admission, stay_duration, patient_address, patient_status, center_id)
    values (
        'patient_name', '4/4/2020', 14, 'address', 'status', 21
    );
```

Edit Patient:

Any patient's details can be edited by clicking the corresponding EDIT button in the Actions column.

For making an edit in a section (say, patient's name), we run the following query:

```
update patient set patient_name = 'new name' where patient_id = 15;
```

Similarly, we run similar queries for editing other sections.

Delete Patient:

Any patient's details can be deleted by clicking the corresponding DELETE button in the Actions column.

We run the following query for deleting the patient record:

```
delete from patient where patient_id =15;
```

Manage Staffs

This action is available to all users. But when logged in as Center Manager, the user can only see the staffs of his/her quarantine center.

For getting the staffs, we perform the following query:

```
select * from staff
    inner join staff_role sr on staff.role_id = sr.role_id
    inner join center c on staff.center_id = c.center_id;
```

New Staff:

For a new staff we run the following query:

```
insert into staff
    (staff_name, staff_contact_number, role_id, center_id, working_hours)
    values (
        'name', 'number', 2, 22, 1400-2000
    );
```

Edit Staff:

Any Staff's details can be edited by clicking the corresponding EDIT button in the Actions column.

For making an edit in a section (say, staff's name), we run the following query:

```
update staff set staff_name = 'new name' where staff_id = 17;
```

Similarly, we run similar queries for editing other sections.

Delete Staff:

Any Staff's details can be deleted by clicking the corresponding DELETE button in the Actions column.

We run the following query for deleting the staff' record:

```
delete from staff where staff_id = 17;
```

Manage Service Requests

This action is available to all users.

But when logged in as Center Manager, the user can only see the request of his/her quarantine center, can only create or delete requests.

When logged in as Government Official, user cannot create new requests. He can only respond to the request, either process it or discard it.

When logged in a Database Administrator, the user can perform all the actions, i.e., create, delete, process or discard a request.

The action to edit a request is not given to any user, as we wanted to create a constraint for the functioning of the project to demonstrate that we need to understand the proper functioning of the project which is given to us.

For getting the requests, we perform the following query:

```
select * from service_request
    inner join req_status rs on service_request.status_id = rs.status_id
    inner join center c on service_request.center_id = c.center_id
    inner join user u on service_request.user_id = u.user_id;
```

New Request:

For a new staff we run the following query:

```
insert into service_request (request_description, center_id) values ('description', 23);
```

Delete Request:

Any request can be deleted by clicking the corresponding DELETE button in the Actions column.

We run the following query for deleting the request:

```
delete from service_request where request_id = 19;
```

Manage Users

This action is available only to Database Administrator.

For getting the users, we perform the following query:

```
select * from user
    inner join user_role ur on user.user_role_id = ur.user_role_id;
```

New User:

For a new user we run the following query:

```
insert into user
    ( user_name, user_email, user_phone, user_role_id)
    values (
        'name', 'email', 'phone', 3
    );
```

Edit User:

Any User's details can be edited by clicking the corresponding EDIT button in the Actions column.

For making an edit in a section (say, user's name), we run the following query:

```
update user set user_name = 'new name' where user_id = 3;
```

Similarly, we run similar queries for editing other sections.

Delete User:

Any User's details can be deleted by clicking the corresponding DELETE button in the Actions column.

We run the following query for deleting the user's record:

```
delete from user where user_id = 3;
```

Conclusions

Additional features of the website

1. The website also features a component showing covid-19 status of India on the homepage.
2. The whole website is a single page website. So, it is so fast in terms of response.
3. Most of the actions, on completion, give window alerts in the browser itself.

Accomplishments

1. Skills to manage a huge database system.
2. Handling complex SQL queries.
3. Managing data redundancy.
4. Connecting a database to the application.
5. Running parallel queries on the database.