PES University

100 feet Ring Road, BSK 3rd Stage, Bengaluru 560085

UE17CS252

Database Management Systems

Project Report

Hospital Database Management

PES1201700231 : Adithya Kiran
PES1201700896 : Chirag P Tubakad
PES1201701090 : Nandakrishna

Table of Contents :

Introduction

Databases organize data items and maintain relationships between them in order to facilitate effective information across an organisation. With the advent of technology and a newer perspective of how data is seen , we have come a long way. Data plays a major role in analysis of a company performance or in any scenario where you would want to measure a performance gradient , but to use the data , maintaining proper records and valid inputs is quintessential.

Databases particularly come into picture when to want to store the raw data in a more structured and standardized format. There are also constraints as to what can be entered and what is considered as valid. One of the well known facts today is that there is about 80% unstructured data and of the remaining 20% only 5% is actually being exploited to do analysis and other data driven tasks. Companies are spending a fortune to keep track of all the data which ranges from your basic username and password when you login somewhere or placing order to more classified information such as business transactions and bank details. We see that it’s not just about storing data ,but also keeping them secure while being structured at same time.

Data is invisible yet omnipresent. Given the importance of databases and their use , it is only appropriate we have proper database management systems in place.

Invariably most of the things we do online - browsing ,shopping , online transactions , E - commerce , blogs etc are all powered by well designed and built databases running in the background crunching requests and churning numbers.

Having realized the importance of databases , we set out to built a small , yet a valiant prototype of a hospital database system. Keeping in purview of all the employees running the hospital and important records of patients , we tried to keep it as realistic as possible.

Problem Statement

Our problem statement was to build a as realistic as possible , probably scalable , hospital database management system which comprised of both frontend as well as backend. Various user frontend - backend interactions had to be put into place coupled with sophisticated queries to parse and retrieve data from the database.

The frontend support had to also include various user login and upon authentication preview what is only applicable to that particular person. Essentially data abstraction.

Requirements Specification :

Requirements included us to build a working prototype with the following :

1. Frontend
   a) User login with authentication.
   b) View his/her details including options to change password or other modifiable fields.
   c) Functionality supporting both for the employees as well as patients.
   d) Patients viewing bills and their prescribed medication at their convenience.
   e) Doctors assigning medication to patients and modifying patient records depending on their diagnosis.
   f) Generating patient bills and displaying the current status.
2. Backend
   a) Proper documentation of details and accepting the correct values into the database.
   b) Creating views and displaying the necessary details and hiding sensitive or confidential information.
   c) Keeping the database dynamic and in a valid state at any given time.
   d) To keep as little as NULL values as possible.
   e) Every entity serving the desired purpose.

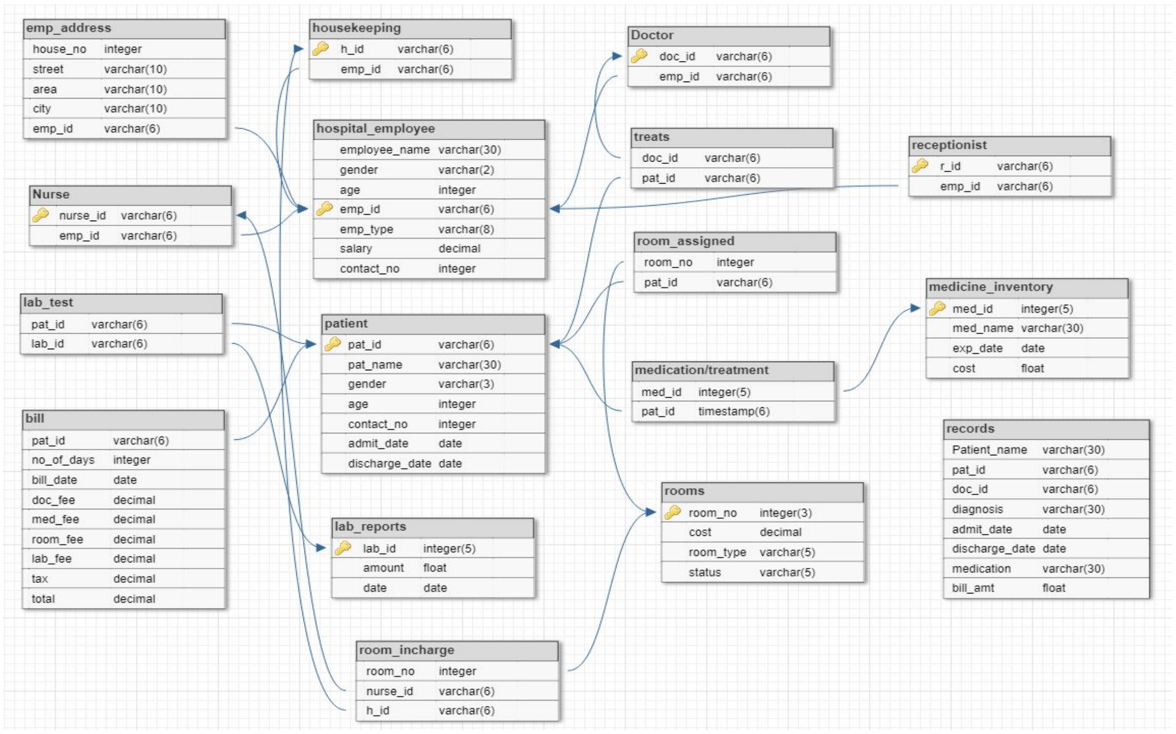
Frontend was built using HTML and a robust bootstrap framework to enhance the functionality as well as make the page as interactive as possible. To make the web pages dynamic , PHP was used alongside the purpose of communication with the backend.
Backend was powered by PostgreSQL which queried and retrieved data accordingly. ER diagrams were made using DIA , a powerful flow control / structure diagram building software.

Design

A very methodological process was followed which started from scratch , going back to the drawing board representing the bare bone of our database , the ER diagram.
The various entities that our database consisted of along with its relationship and participation between relation. This constituted the base of our backend.
Step 1 : ER Diagram

Following the ER diagram , we went ahead to build a more self explanatory and powerful relational schema giving a more in depth view of how the entities looked and the keys referenced to.

Step 2 : Relational Schema

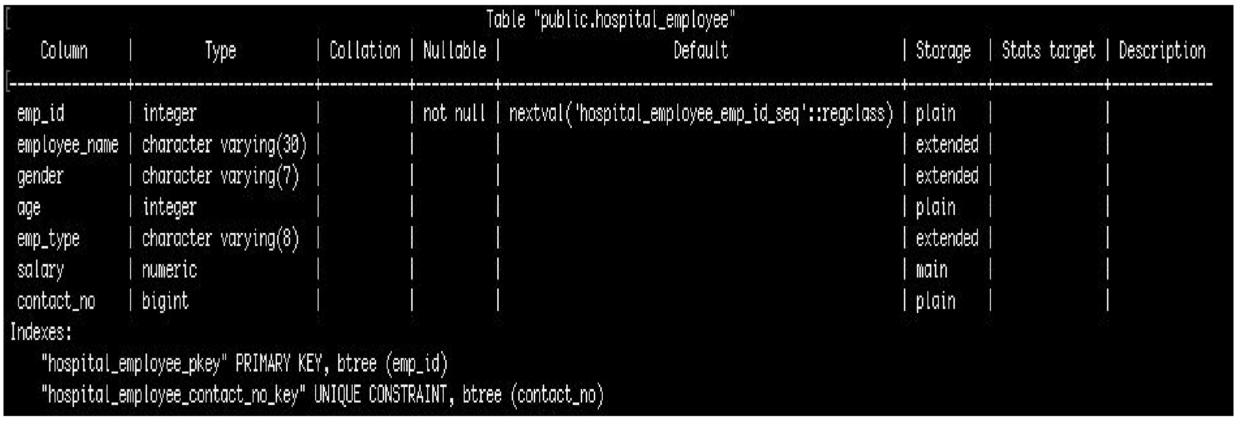


Now , having a blueprint of our database , the next step was to create these entities and tie them together using PostgreSQL.

We wrote a whole lot of queries to build the entities and reference them accurately to their expected attributes. Here are a few to point out to.
Each of the DDL queries are followed by an image of their metadata in postgreSQL.

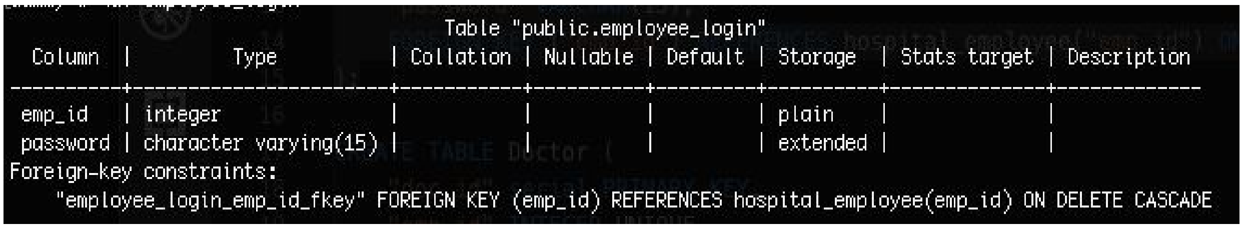
Step 3 : DDL statements

```
CREATE TABLE hospital_employee (
    "emp_id" serial PRIMARY KEY,
    "employee_name" VARCHAR(30),
    "gender" VARCHAR(7),
    "age" INTEGER,
    "emp_type" VARCHAR(8),
    "salary" DECIMAL,
    "contact_no" bigint UNIQUE
);
```



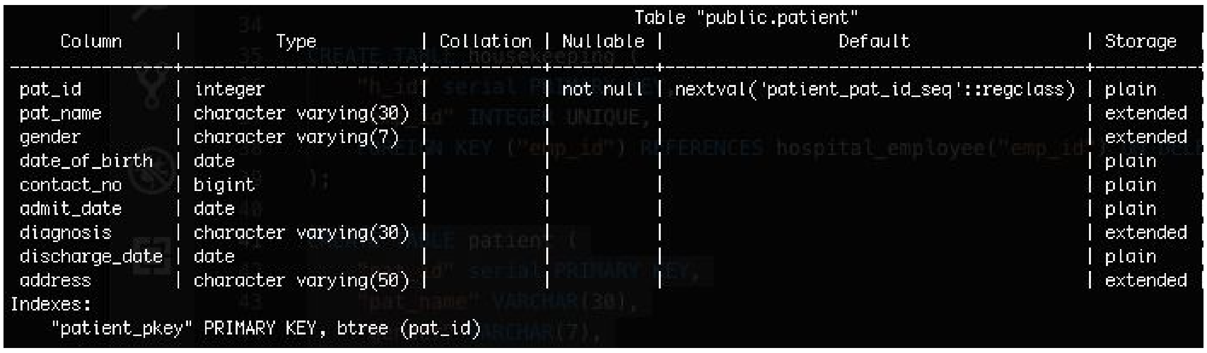
```
                                                  Table "public.hospital_employee"
    Column     |         Type          | Collation | Nullable |                       Default                      | Storage  | Stats target | Description
---------------+-----------------------+-----------+----------+----------------------------------------------------+----------+--------------+-------------
 emp_id        | integer               |           | not null | nextval('hospital_employee_emp_id_seq'::regclass) | plain    |              |
 employee_name | character varying(30) |           |          |                                                    | extended |              |
 gender        | character varying(7)  |           |          |                                                    | extended |              |
 age           | integer               |           |          |                                                    | plain    |              |
 emp_type      | character varying(8)  |           |          |                                                    | extended |              |
 salary        | numeric               |           |          |                                                    | main     |              |
 contact_no    | bigint                |           |          |                                                    | plain    |              |
Indexes:
    "hospital_employee_pkey" PRIMARY KEY, btree (emp_id)
    "hospital_employee_contact_no_key" UNIQUE CONSTRAINT, btree (contact_no)
```

```
CREATE TABLE employee_login(
    "emp_id" INTEGER,
    "password" VARCHAR(15),
    FOREIGN KEY ("emp_id") REFERENCES hospital_employee("emp_id")
ON DELETE CASCADE
);
```



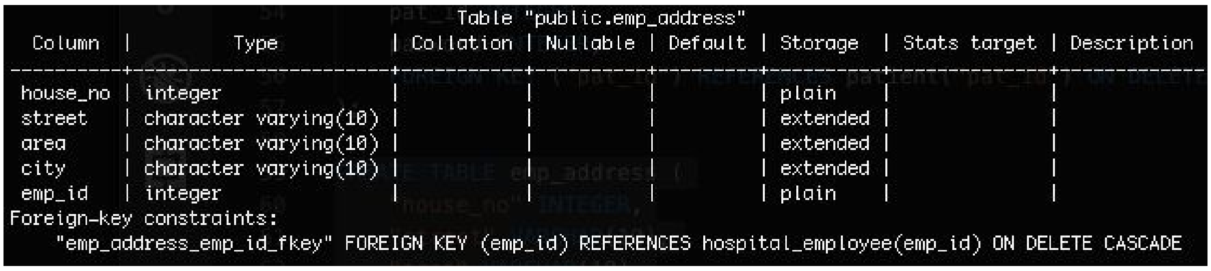
```
                              Table "public.employee_login"
  Column  |         Type          | Collation | Nullable | Default | Storage  | Stats target | Description
----------+-----------------------+-----------+----------+---------+----------+--------------+-------------
 emp_id   | integer               |           |          |         | plain    |              |
 password | character varying(15) |           |          |         | extended |              |
Foreign-key constraints:
    "employee_login_emp_id_fkey" FOREIGN KEY (emp_id) REFERENCES hospital_employee(emp_id) ON DELETE CASCADE
```

```
CREATE TABLE patient (
    "pat_id" serial PRIMARY KEY,
    "pat_name" VARCHAR(30),
    "gender" VARCHAR(7),
    "date_of_birth" DATE,
    "contact_no" bigint,
    "admit_date" DATE,
    "diagnosis" VARCHAR(30),
    "discharge_date" DATE,
    "address" VARCHAR(50)
);
```



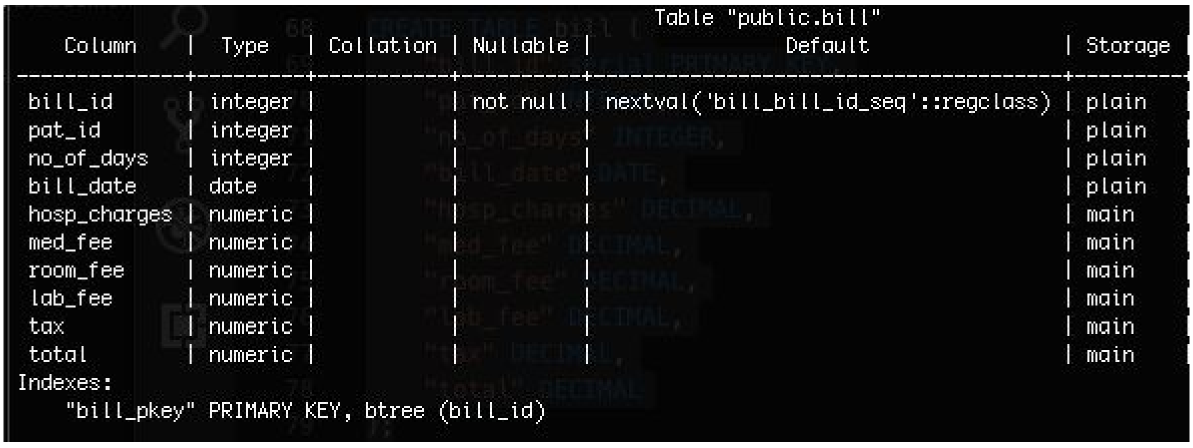
```
                                          Table "public.patient"
    Column     |         Type          | Collation | Nullable |                 Default                  | Storage  |
---------------+-----------------------+-----------+----------+------------------------------------------+----------+
 pat_id        | integer               |           | not null | nextval('patient_pat_id_seq'::regclass) | plain    |
 pat_name      | character varying(30) |           |          |                                          | extended |
 gender        | character varying(7)  |           |          |                                          | extended |
 date_of_birth | date                  |           |          |                                          | plain    |
 contact_no    | bigint                |           |          |                                          | plain    |
 admit_date    | date                  |           |          |                                          | plain    |
 diagnosis     | character varying(30) |           |          |                                          | extended |
 discharge_date | date                 |           |          |                                          | plain    |
 address       | character varying(50) |           |          |                                          | extended |
Indexes:
    "patient_pkey" PRIMARY KEY, btree (pat_id)
```

```
CREATE TABLE emp_address (
    "house_no" INTEGER,
    "street" VARCHAR(10),
    "area" VARCHAR(10),
    "city" VARCHAR(10),
    "emp_id" INTEGER,
    FOREIGN KEY ("emp_id") REFERENCES hospital_employee("emp_id")
ON DELETE CASCADE
);
```



```
                                 Table "public.emp_address"
  Column  |         Type          | Collation | Nullable | Default | Storage  | Stats target | Description
----------+-----------------------+-----------+----------+---------+----------+--------------+-------------
 house_no | integer               |           |          |         | plain    |              |
 street   | character varying(10) |           |          |         | extended |              |
 area     | character varying(10) |           |          |         | extended |              |
 city     | character varying(10) |           |          |         | extended |              |
 emp_id   | integer               |           |          |         | plain    |              |
Foreign-key constraints:
    "emp_address_emp_id_fkey" FOREIGN KEY (emp_id) REFERENCES hospital_employee(emp_id) ON DELETE CASCADE
```

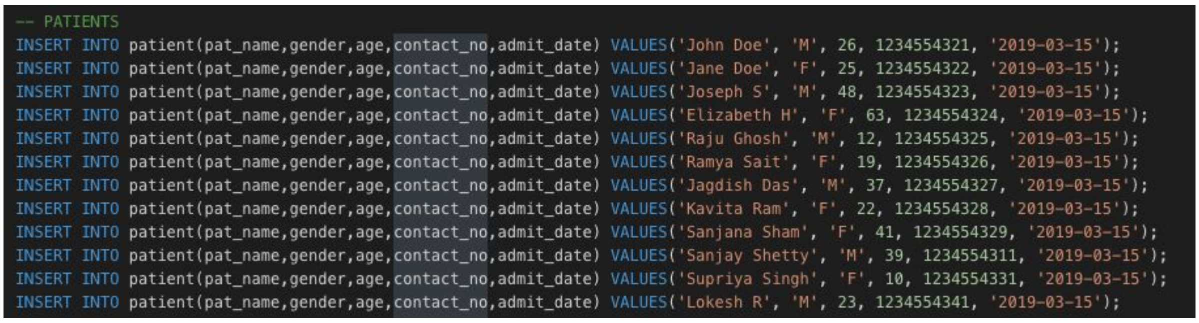
```
CREATE TABLE bill (
    "bill_id" serial PRIMARY KEY,
    "pat_id" INTEGER,
    "no_of_days" INTEGER,
    "bill_date" DATE,
    "hosp_charges" DECIMAL,
    "med_fee" DECIMAL,
    "room_fee" DECIMAL,
    "lab_fee" DECIMAL,
    "tax" DECIMAL,
    "total" DECIMAL
);
```



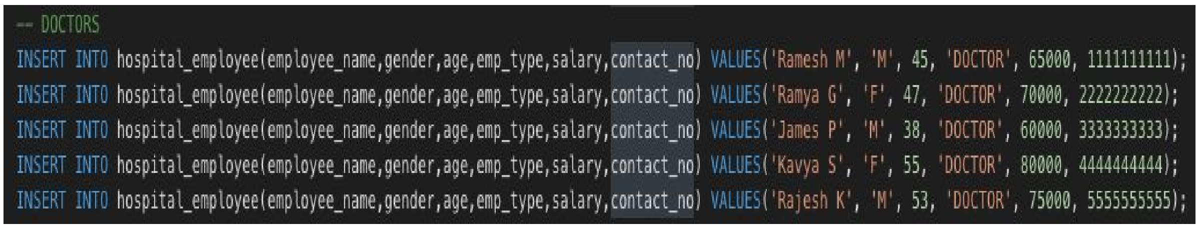
```
                                        Table "public.bill"
    Column    |  Type   | Collation | Nullable |                Default                | Storage
--------------+---------+-----------+----------+---------------------------------------+---------
 bill_id      | integer |           | not null | nextval('bill_bill_id_seq'::regclass) | plain
 pat_id       | integer |           |          |                                       | plain
 no_of_days   | integer |           |          |                                       | plain
 bill_date    | date    |           |          |                                       | plain
 hosp_charges | numeric |           |          |                                       | main
 med_fee      | numeric |           |          |                                       | main
 room_fee     | numeric |           |          |                                       | main
 lab_fee      | numeric |           |          |                                       | main
 tax          | numeric |           |          |                                       | main
 total        | numeric |           |          |                                       | main
Indexes:
    "bill_pkey" PRIMARY KEY, btree (bill_id)
```

With a total of 18 entities , the aforementioned was just to give an inde/example of the work done. With the entities created along with the appropriate input constraints on the data , inserting values is mostly done when deployed because we can’t pre determine the values or inputs. But we pre filled/populated to check functionality and proper updation upon performing various database operations. A bulk insert was incorporated , nevertheless , some of the insert statements are attached.

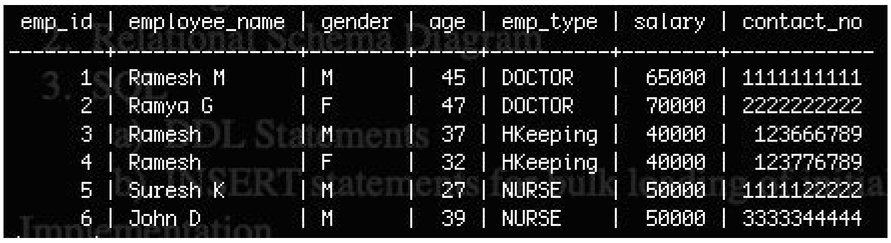
Step 4 : INSERT statements(bulk)



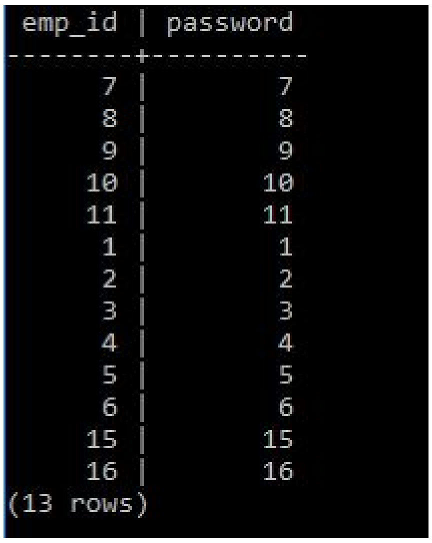
```
-- PATIENTS
INSERT INTO patient(pat_name,gender,age,contact_no,admit_date) VALUES('John Doe', 'M', 26, 1234554321, '2019-03-15');
INSERT INTO patient(pat_name,gender,age,contact_no,admit_date) VALUES('Jane Doe', 'F', 25, 1234554322, '2019-03-15');
INSERT INTO patient(pat_name,gender,age,contact_no,admit_date) VALUES('Joseph S', 'M', 48, 1234554323, '2019-03-15');
INSERT INTO patient(pat_name,gender,age,contact_no,admit_date) VALUES('Elizabeth H', 'F', 63, 1234554324, '2019-03-15');
INSERT INTO patient(pat_name,gender,age,contact_no,admit_date) VALUES('Raju Ghosh', 'M', 12, 1234554325, '2019-03-15');
INSERT INTO patient(pat_name,gender,age,contact_no,admit_date) VALUES('Ramya Sait', 'F', 19, 1234554326, '2019-03-15');
INSERT INTO patient(pat_name,gender,age,contact_no,admit_date) VALUES('Jagdish Das', 'M', 37, 1234554327, '2019-03-15');
INSERT INTO patient(pat_name,gender,age,contact_no,admit_date) VALUES('Kavita Ram', 'F', 22, 1234554328, '2019-03-15');
INSERT INTO patient(pat_name,gender,age,contact_no,admit_date) VALUES('Sanjana Sham', 'F', 41, 1234554329, '2019-03-15');
INSERT INTO patient(pat_name,gender,age,contact_no,admit_date) VALUES('Sanjay Shetty', 'M', 39, 1234554311, '2019-03-15');
INSERT INTO patient(pat_name,gender,age,contact_no,admit_date) VALUES('Supriya Singh', 'F', 10, 1234554331, '2019-03-15');
INSERT INTO patient(pat_name,gender,age,contact_no,admit_date) VALUES('Lokesh R', 'M', 23, 1234554341, '2019-03-15');
```



```
-- DOCTORS
INSERT INTO hospital_employee(employee_name,gender,age,emp_type,salary,contact_no) VALUES('Ramesh M', 'M', 45, 'DOCTOR', 65000, 1111111111);
INSERT INTO hospital_employee(employee_name,gender,age,emp_type,salary,contact_no) VALUES('Ramya G', 'F', 47, 'DOCTOR', 70000, 2222222222);
INSERT INTO hospital_employee(employee_name,gender,age,emp_type,salary,contact_no) VALUES('James P', 'M', 38, 'DOCTOR', 60000, 3333333333);
INSERT INTO hospital_employee(employee_name,gender,age,emp_type,salary,contact_no) VALUES('Kavya S', 'F', 55, 'DOCTOR', 80000, 4444444444);
INSERT INTO hospital_employee(employee_name,gender,age,emp_type,salary,contact_no) VALUES('Rajesh K', 'M', 53, 'DOCTOR', 75000, 5555555555);
```

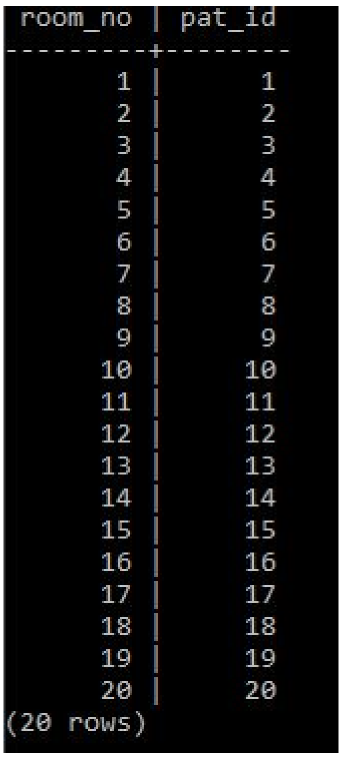


```
 emp_id | employee_name | gender | age | emp_type | salary | contact_no
--------+---------------+--------+-----+----------+--------+------------
      1 | Ramesh M      | M      |  45 | DOCTOR   |  65000 | 1111111111
      2 | Ramya G       | F      |  47 | DOCTOR   |  70000 | 2222222222
      3 | Ramesh        | M      |  37 | HKeeping |  40000 |  123666789
      4 | Ramesh        | F      |  32 | HKeeping |  40000 |  123776789
      5 | Suresh K      | M      |  27 | NURSE    |  50000 | 1111122222
      6 | John D        | M      |  39 | NURSE    |  50000 | 3333344444
```



```
 emp_id | password
--------+----------
      7 |        7
      8 |        8
      9 |        9
     10 |       10
     11 |       11
      1 |        1
      2 |        2
      3 |        3
      4 |        4
      5 |        5
      6 |        6
     15 |       15
     16 |       16
(13 rows)
```

The above image contains the employee ID and password for them to login through our website and view details. For the sake of sanity , the employee ID , first when created , their password is defaulted to the same. They are also provided with an option to change their password at a later stage for their security.



```
 room_no | pat_id
---------+--------
       1 |      1
       2 |      2
       3 |      3
       4 |      4
       5 |      5
       6 |      6
       7 |      7
       8 |      8
       9 |      9
      10 |     10
      11 |     11
      12 |     12
      13 |     13
      14 |     14
      15 |     15
      16 |     16
      17 |     17
      18 |     18
      19 |     19
      20 |     20
(20 rows)
```

Indicating the patients in each room. Upon further updation , the rooms might have patients with alternating ID's.

Now , once we have successfully created the entities and populated them , it is imperative we check them by reading and updating values to see if the consequent changes are reflected. CRUD operations play a vital role in making the database dynamic. It basically helps us to make changes as and when it is needed making them flexible.

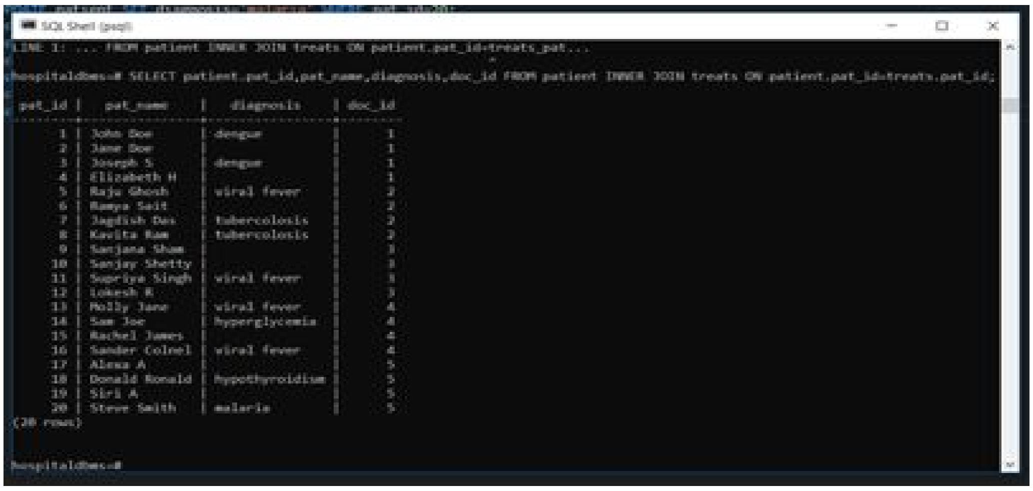
Step 5 : CRUD Operations

```
UPDATE room_incharge SET h_id= (SELECT emp_id FROM hospital_employee WHERE employee_name=emp_name AND contact_no=ph_no)  WHERE room_no=r_no;

DELETE FROM hospital_employee WHERE emp_id=employee_id;

SELECT pat_id, room_no FROM patient, rooms WHERE pat_name=p_name AND contact_no=ph_no AND room_no=r_no;

UPDATE patient SET discharge_date=d_date WHERE pat_id=p_id;

UPDATE room_incharge SET h_id= (SELECT emp_id FROM hospital_employee WHERE employee_name=emp_name AND contact_no=ph_no)  WHERE room_no=r_no;
```

The above mentioned CRUD operations is only a few of the many used to alter the contents of the database. However these are hardcoded to check the functionality of the tables. The queries are more dependent on the what are the options we are giving the end user on the frontend. Accordingly we write queries to perform what is asked for.

Step 6 : Complex SQL queries

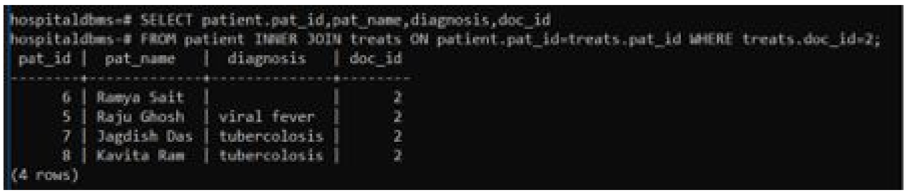
```
SELECT patient.pat_id,pat_name,diagnosis,doc_id
FROM patient INNER JOIN treats ON patient.pat_id=treats.pat_id;
```



```
LINE 1: ... FROM patient INNER JOIN treats ON patient.pat_id-treats_pat...
                                                                  ^
hospitaldbms=# SELECT patient.pat_id,pat_name,diagnosis,doc_id FROM patient INNER JOIN treats ON patient.pat_id=treats.pat_id;
 pat_id |   pat_name    |   diagnosis    | doc_id
--------+---------------+----------------+--------
      1 | John Doe      | dengue         |      1
      2 | Jane Doe      |                |      1
      3 | Joseph S      | dengue         |      1
      4 | Elizabeth H   |                |      1
      5 | Raju Ghosh    | viral fever    |      2
      6 | Ramya Sait    |                |      2
      7 | Jagdish Das   | tubercolosis   |      2
      8 | Kavita Ram    | tubercolosis   |      2
      9 | Sanjana Sham  |                |      3
     10 | Sanjay Shetty |                |      3
     11 | Supriya Singh | viral fever    |      3
     12 | Lokesh R      |                |      3
     13 | Molly Jane    | viral fever    |      4
     14 | Sam Joe       | hyperglycemia  |      4
     15 | Rachel James  |                |      4
     16 | Sander Colnel | viral fever    |      4
     17 | Alexa A       |                |      5
     18 | Donald Ronald | hypothyroidism |      5
     19 | Siri A        |                |      5
     20 | Steve Smith   | malaria        |      5
(20 rows)

hospitaldbms=#
```

Displays all the patients with their corresponding diagnosis and the doctor treating them. This provides a more easy graphical representation instead of looking up the patient ID in two separate tables.

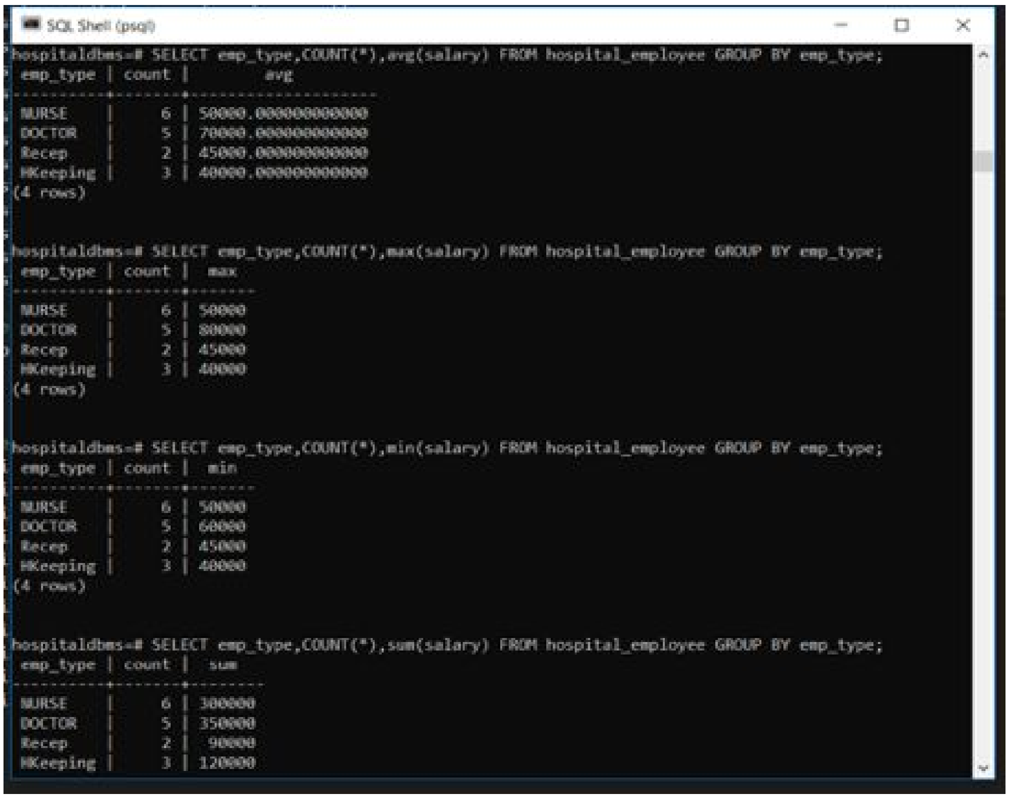
```
SELECT patient.pat_id,pat_name,diagnosis,doc_id
FROM patient INNER JOIN treats ON patient.pat_id=treats.pat_id
WHERE treats.doc_id=2;
```



```
hospitaldbms=# SELECT patient.pat_id,pat_name,diagnosis,doc_id
hospitaldbms-# FROM patient INNER JOIN treats ON patient.pat_id=treats.pat_id WHERE treats.doc_id=2;
 pat_id |  pat_name   |  diagnosis   | doc_id
--------+-------------+--------------+--------
      6 | Ramya Sait  |              |      2
      5 | Raju Ghosh  | viral fever  |      2
      7 | Jagdish Das | tubercolosis |      2
      8 | Kavita Ram  | tubercolosis |      2
(4 rows)
```

Say , as a daily routine , doctors want to review all the patients they are treating. By just entering their ID is displays a list of all their patients along with diagnosis. This helps a doctor to keep track as well as monitor patients regularly.

```
SELECT emp_type,COUNT(*),avg(salary) FROM hospital_employee

GROUP BY emp_type;

SELECT emp_type,COUNT(*),min(salary) FROM hospital_employee

GROUP BY emp_type;

SELECT emp_type,COUNT(*),max(salary) FROM hospital_employee

GROUP BY emp_type;

SELECT emp_type,COUNT(*),sum(salary) FROM hospital_employee

GROUP BY emp_type;
```

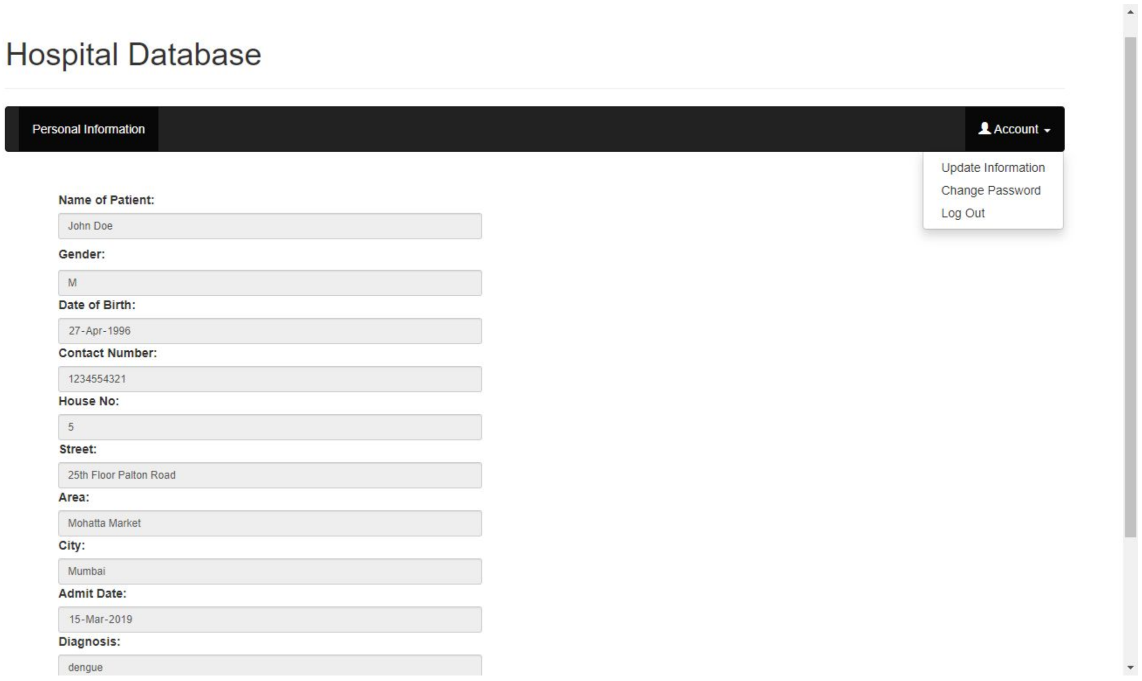


```
hospitaldbms=# SELECT emp_type,COUNT(*),avg(salary) FROM hospital_employee GROUP BY emp_type;
 emp_type | count |        avg
----------+-------+--------------------
 NURSE    |     6 | 50000.000000000000
 DOCTOR   |     5 | 70000.000000000000
 Recep    |     2 | 45000.000000000000
 HKeeping |     3 | 40000.000000000000
(4 rows)


hospitaldbms=# SELECT emp_type,COUNT(*),max(salary) FROM hospital_employee GROUP BY emp_type;
 emp_type | count |  max
----------+-------+-------
 NURSE    |     6 | 50000
 DOCTOR   |     5 | 80000
 Recep    |     2 | 45000
 HKeeping |     3 | 40000
(4 rows)


hospitaldbms=# SELECT emp_type,COUNT(*),min(salary) FROM hospital_employee GROUP BY emp_type;
 emp_type | count |  min
----------+-------+-------
 NURSE    |     6 | 50000
 DOCTOR   |     5 | 60000
 Recep    |     2 | 45000
 HKeeping |     3 | 40000
(4 rows)


hospitaldbms=# SELECT emp_type,COUNT(*),sum(salary) FROM hospital_employee GROUP BY emp_type;
 emp_type | count |  sum
----------+-------+--------
 NURSE    |     6 | 300000
 DOCTOR   |     5 | 350000
 Recep    |     2 |  90000
 HKeeping |     3 | 120000
```

These queries are for internal assessment , such as finding out the average salary of all the employees , number of employees , number of employees in each category , what's the maximum salary , who is underpaid etc. This can also be used as a measure to check the hospital administration.

Once we drill down to the very minute details of the backend and ensure proper working of the same , we move on to building the front end.
The front end is relatively a more challenging aspect because we have to keep in mind the visual representation at the same time device proper ways to link the front end to the backend.

Frontend



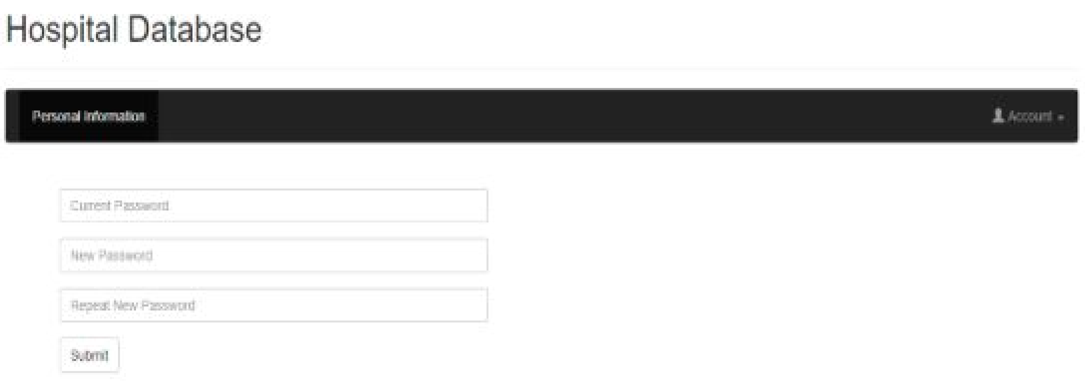
Hospital Database

Personal Information | Account

Update Information
Change Password
Log Out

Name of Patient: John Doe
Gender: M
Date of Birth: 27-Apr-1996
Contact Number: 1234554321
House No: 5
Street: 25th Floor Palton Road
Area: Mohatta Market
City: Mumbai
Admit Date: 15-Mar-2019
Diagnosis: dengue

The form by default , as soon as the patient logs in displays all their details and their current status. In case of any diagnosis , that too is displayed bridging the gap between a patient and doctor and their prescribed medication.

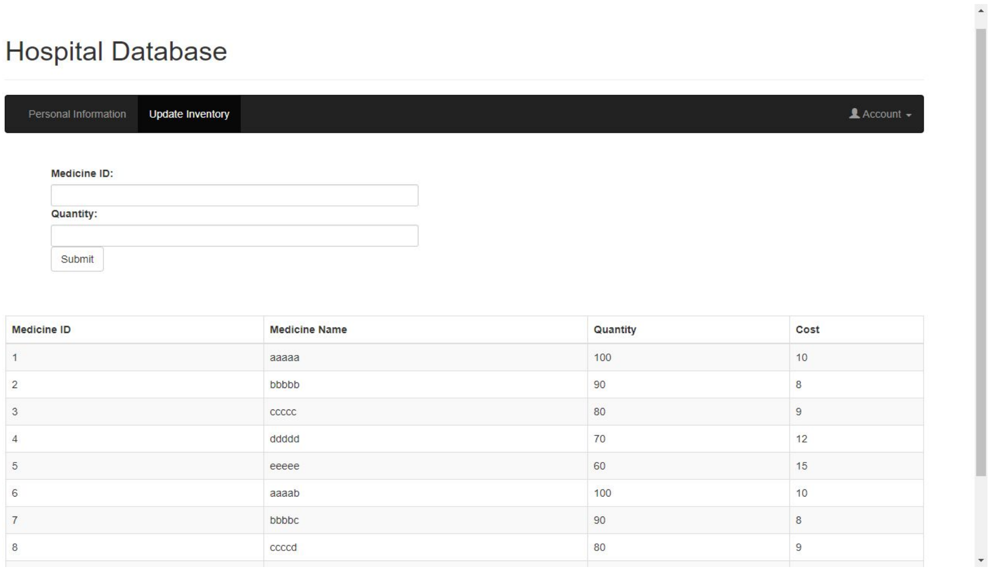


Hospital Database

Personal Information | Account

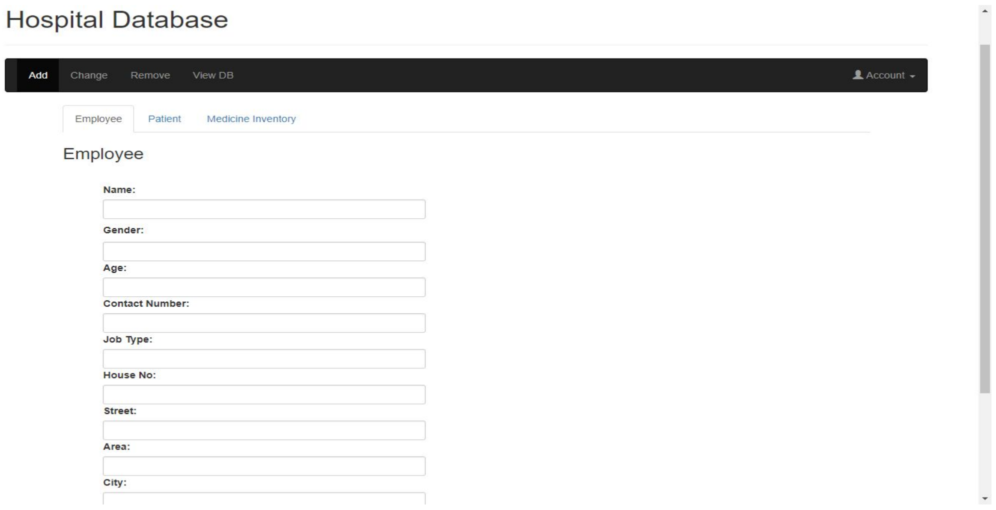
Current Password
New Password
Repeat New Password
Submit

Added functionalities include updating their information such as contact number , address or they could even change their password.

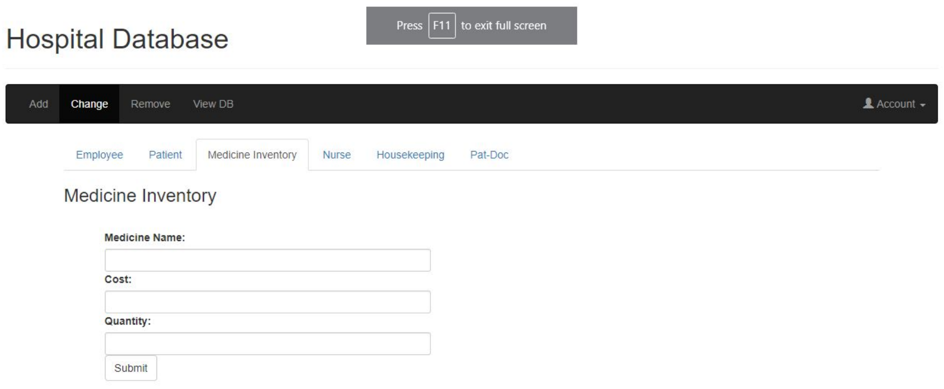


Medicine ID	Medicine Name	Quantity	Cost
1	aaaaa	100	10
2	bbbbb	90	8
3	ccccc	80	9
4	ddddd	70	12
5	eeeee	60	15
6	aaaab	100	10
7	bbbbc	90	8
8	ccccd	80	9

This is the page for inventory management. The database not only keeps track of their employees and patients , but also does inventory management. Which comprises of the medicines available and the quantity of them. It is crucial for hospitals because it helps them keep stock and emergencies might occur at any moment.



This tab is mostly for admin control where they make new entries of doctors and other staff members. This is not modifiable. Only information such as their number , address can be changed. Only attributes susceptible to change.



An extension/added functionality of the admin which happens to be the receptionist in our project.

Conclusion

Keeping in purview of all the possible employees , staff and everyone involved in running an hopital , we structure and store the data of each of them keeping records of patients as well. With an in-detail explanation of each and every aspect of the project , it is imperative we extrapolate it to the point where it is deployable. We also provided flexibility to the end user to change his/her employee or patient details making our program much more powerful , interactive and consistent.

Proposed Enhancements

1) Currently our program gives more control to the receptionist whereas we could actually develop another login for a system administrator who has master control of the whole database.
2) Making our website more elaborate by adding a dedicated home page for the hospital and extra pages for about the hospital. Currently it serves the purpose of the accepting data and storing it on the backend.
3) Building a more robust backend by being able to store media as well. For instance we are tabulating all the details of every employee , whereas if we could enhance it to store the employee photo for cross verification so that their ID is not being misused. We could also develop to store fingerprints of all employees so that they could scan their finger and login and logoff.

References

1) Textbook : Fundamentals of Database Systems, Ramez Elamsri, Shamkant B Navathe , 7th edition
2) www.tutorialspoint.com
3) www.w3schools.com

Tools and Technologies

1) DIA : For drawing the ER diagram
2) PostgreSQL phppgadmin : For complete backend development
3) HTML and Bootstrap : For frontend development and making the web pages interactive.
4) PHP : For linking the front end with the backend.