Semester - V

Course Name – CST305 Database Management System

Project Title - Hospital Database Management System

Group Members		
Roll Number	**Name**	**Batch**
2021UCP1136	Priyanshu Tatu	A2 CSE
2021UCP1144	Garvit Mittal	A2 CSE
2021UCP1148	Priyanshu Raj	A2 CSE
2021UCP1165	Mohit Kumar	A2 CSE
2021UCP1181	Shivansh Nautiyal	A2 CSE

a. Description of Project :

The purpose of this project is to develop a system that handles different workflows related to a hospital. The main task is to develop a software that registers, stores, and meaningfully manipulates the information related to the patients, staff and doctors, appointments,visits, medicines, etc. in order to have some meaningful conclusions.

Key Functionalities:

1) Managing **Patients' Information**:

- For **deriving the age of a patient, given the DOB**, we have created a function followed by a procedure. The function (**age(patientid int)**) with parameter patient id returns the current age of that patient. Then, if the current age is greater than 18 then the procedure (**patient_age_msg(patientid int)**) will **print a message** 'Patient is a Major', else it will print the message 'Minor Patient'.

- To **get an overview related to the blood groups of all the patients present in the database**, we have created a function (**patient_bloodgroup(bloodgroup varchar(10))**) with parameter blood group. It returns the total number of patients of that particular blood group.

- To get the **details of the patients who has the maximum number of appointments** in this hospital, we have created a procedure **patient_with_maxApps().**

- To **display the patients who took appointment on the weekends**, we have created a procedure called **display_patients_on_weekend()**

- To **get the information about the patients who took an appointment during current month,** we have created a procedure **pat_this_month().**

- In our database, we have a table for all the patients in the hospital, and then a separate table for indoor patients. However, we do not have **a table that shows the details of outdoor patients.**

- Atlast, we have generated a **bill** for a visit of a patient, which includes a patient's visit charge, doctor charge, medicine charge etc. And generates the total amount to be paid.

2) Managing **Appointment Related Information**:

- To **get the trend of appointments for each weekday**, we have created a function with parameter i, **total_patients_on_this_day(dayno int)** that returns the number of patients who have appointments on day i. (1=Sunday, 2=Monday, 3=Tuesday, 4=Wednesday, 5=Thursday, 6=Friday, 7=Saturday)

- The above function is followed by a procedure (**patient_count_msg(dayno int)**) that **prints a message** 'Many patients on this day!!' if the number of patients is greater than 2 on that day, otherwise it prints the message 'Not many patients on this day!' (For bigger database we need to take a bigger number than 2)

- To **display patient-wise appointment details**, we have created a procedure called display_patient_appointment_info().

- To **display year-wise appointment details,** we have created a procedure called **display_yearwise_appointments().** Also, all the patients who have taken the appointment might not end up visiting the hospital. So, to **display year-wise visits records**, we have created an additional procedure called **display_yearwise_visits(which_year year).**

3) Managing Staff, Rooms, and Medicines related to Information:

- If Staff_dutystarttime is equal to Staff_dutyendtime then a trigger will print **an error message, on inserting the data in the staff table**: "Duty start time cannot be same as Duty end time..." and won't allow the user to insert that data.

- We have created a trigger which triggers when any record is updated in the Medicines table. When the **value of any field is updated, we keep track of before and after values**

in the table “medicines_updates” for each field of the Medicines table. The “medicines_updates” table contains the fields change_date, field_name, before_value and after_value.

4) Managing **Admin/Login Details**:

- We have created a **simple login and user registration** system using authentication from Admin User Table.

- There are some functionalities like creating a **new administrato**r (if the person is new to the system), **logout** and **password change** (altering the password column of the user table).

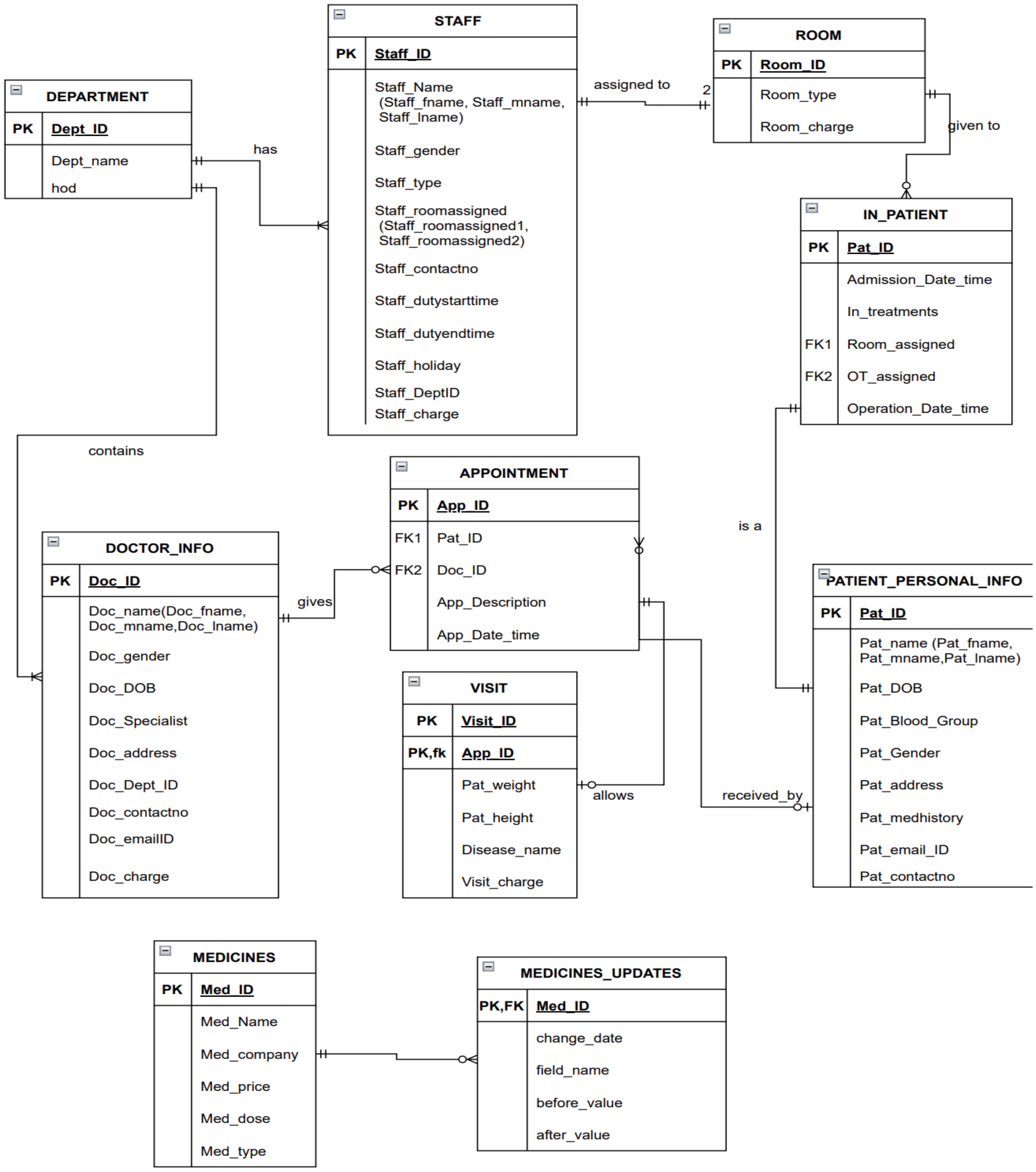
- As a matter of Privacy Policy we don't save ‘password’ directly but instead store the HashCode corresponding to it using **Password_Hash().**

5) Managing other information:

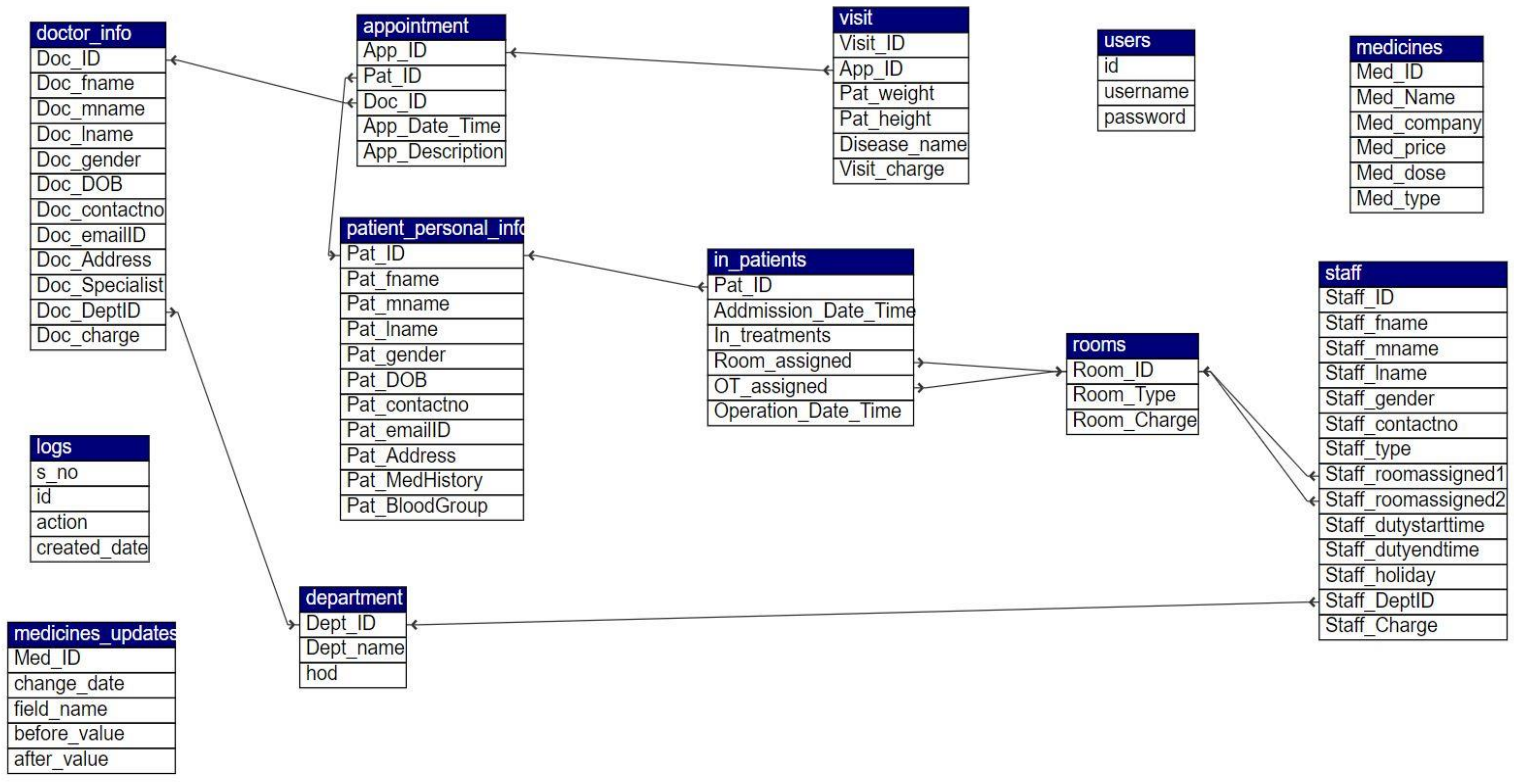
- Whenever it is needed to permanently delete a record from patient_personal_info table, a trigger will automatically **delete all child records** from appointment, and in-patients tables. For this it is required to apply the same trigger on Appointment, and Visit table.

- For **managing patient log details**, we have created a trigger that keeps the track of new entries entered into patient_personal_info in a table called “logs table”

ER DIAGRAM


STAFF
PK Staff_ID
Staff_Name (Staff_fname, Staff_mname, Staff_lname)
Staff_gender
Staff_type
Staff_roomassigned (Staff_roomassigned1, Staff_roomassigned2)
Staff_contactno
Staff_dutystarttime
Staff_dutyendtime
Staff_holiday
Staff_DeptID
Staff_charge
ROOM
PK Room_ID
Room_type
Room_charge
DEPARTMENT
PK Dept_ID
Dept_name
hod
has
assigned to
2
given to
IN_PATIENT
PK Pat_ID
Admission_Date_time
In_treatments
FK1 Room_assigned
FK2 OT_assigned
Operation_Date_time
contains
APPOINTMENT
PK App_ID
FK1 Pat_ID
FK2 Doc_ID
App_Description
App_Date_time
is a
DOCTOR_INFO
PK Doc_ID
Doc_name(Doc_fname, Doc_mname,Doc_lname)
Doc_gender
Doc_DOB
Doc_Specialist
Doc_address
Doc_Dept_ID
Doc_contactno
Doc_emailID
Doc_charge
gives
PATIENT_PERSONAL_INFO
PK Pat_ID
Pat_name (Pat_fname, Pat_mname,Pat_lname)
Pat_DOB
Pat_Blood_Group
Pat_Gender
Pat_address
Pat_medhistory
Pat_email_ID
Pat_contactno
VISIT
PK Visit_ID
PK,fk App_ID
Pat_weight
Pat_height
Disease_name
Visit_charge
allows
received_by
MEDICINES
PK Med_ID
Med_Name
Med_company
Med_price
Med_dose
Med_type
MEDICINES_UPDATES
PK,FK Med_ID
change_date
field_name
before_value
after_value

Relational Model



Functional Dependencies

1) Dept_ID ➔ Dept_name, hod
2) Staff_ID ➔Staff_fname, Staff_mname, Staff_lname, Staff_gender, Staff_contactno, Staff_type, Staff_roomassigned1, Staff_roomassigned2, Staff_dutystarttime, Staff_dutyendtime, Staff_holiday, Staff_DeptID, Staff_Charge
3) Room_ID ➔Room_type, Room_charge

4) Doc_ID ➔Doc_ID, Doc_fname, Doc_mname, Doc_lname, Doc_gender, Doc_DOB, Doc_contactno, Doc_emailID, Doc_Address, Doc_Specialist, Doc_DeptID, Doc_charge
5) App_ID ➔ Pat_ID, Doc_ID, App_Description, App_Date_Time
6) Pat_ID➔Pat_ID, Pat_fname, Pat_mname, Pat_lname, Pat_gender, Pat_DOB, Pat_contactno, Pat_emailID, Pat_Address, Pat_MedHistory, Pat_BloodGroup
7) Pat_ID ➔ Addmission_Date_Time, In_treatments, Room_assigned, OT_assigned, Operation_Date_Time
8) Visit_ID,App_ID ➔Pat_weight, Pat_height, Disease_name, Visit_charge
9) Med_ID ➔Med_Name, Med_company, Med_price, Med_dose, Med_type
10) Med_ID ➔change_date, field_name, before_value, after_value

Finding candidate keys of all the relations:

(Dept_ID)+ = attr(department)

(Staff_ID)+=attr(staff)

(RoomID)+=attr(rooms)

(Doc_ID)+=attr(doctor_info)

(App_ID)+=attr(appointment)

(Pat_ID)+=attr(patient_personal_info)

(Pat_ID)+=attr(in_patients)

(Visit_ID,App_ID)+=attr(visit)

(Med_ID)+=attr(medicines)

(Med_ID)+=attr(medicines_updates)

Here attr(REL) represents attribute set of relation named REL.

Since left hand side of all the functional dependencies are superkeys (candidate keys) or there are no prime attributes which are determining other prime attributes in the same table so we can justify that all the relations are in Boyce Codd Normal Form. Hence it is also implied that all the relations are in 3NF, 2NF as well as 1NF.