Clinic Master

Karmehr Arora, Kean Onn Lee, Geronimo Aldana

Team 1

CS157A Fall 2023

Table of Contents

Project Requirements 3
Relational Schema 6
Entity-Relationship Diagram 7
MySQL Workbench Tables 8
Project Implementation of Functional Requirements 24
How to Set Up and Run Application 45
Presentation Flow of Application 47
Lesson Learned 48

Project Requirements

This application will have many functional features that will be available. There is only one user of our application and that is the staff of the clinic. The idea is that this application is used by staff to handle the management of the hospital such as patients, other staff, and management of the hospital. Staff users will be able to use all the functionalities listed below.

Functions:

1. **Signup Functions**
    - **a.** Create/Insert into login
        - **i.** Allows users to register into the system via input boxes so they can be recognized and authorized when logging in
        - **ii.** Users must submit a valid set of information consisting of their email, password, age, first name, last name, and address in order to create an account/login and register under the system
2. **Login Functions**
    - **a.** Login to application
        - **i.** Users can verify their credentials by logging into the application
        - **ii.** Their login credentials are checked against the database and only registered users with valid credentials are able to login and access the application
3. **Home Page Function**
    - **a.** Landing spot for users to enter the application
    - **b.** Allows users to transverse throughout the application
4. **Patient Functions**

a. Add/Remove Patient

i. Users will be able to manipulate patients' data. They will be able to add a variety of information from name, address, phone number, billing information, etc.

b. Browse Patients

i. Users will be able to browse through patients through an informational table

5. Appointment Functions

a. Book Patient Appointment/Service

i. Users will be able to book a patient to a Appointment

ii. Each Appointment consists of a service, patient, staff, date/time, and room.

iii. Each Appointment will be made sure to not conflict with other appointments for each room, patient, staff, participating in the appointment.

b. Browse Appointments

i. Users will be able to browse through appointments by patient ID or all appointments at once.

c. Cancel Patient Appointment

i. Users will be able to delete appointment and remove associated cost from patients balance

d. Patient Billing & Payment

i. Users will be able to facilitate patient payments

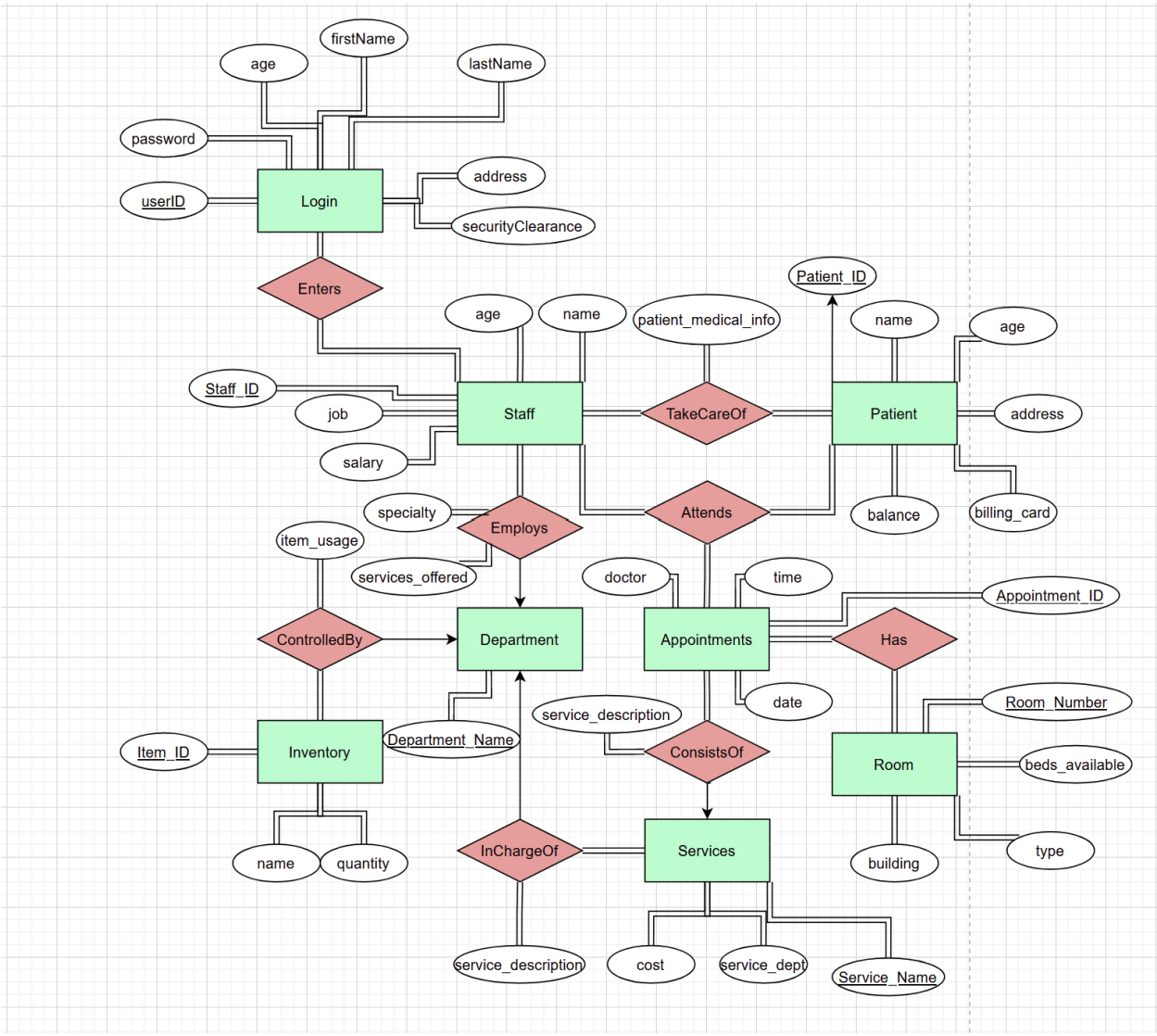
    ii. If an appointment is made, the balance will increase and decrease if the appointment is canceled.

6. **Staff Functions**
    a. Display List of Staff
        i. Users(management) will be able to view all staff that are registered in the system
    b. Add/Remove Staff
        i. Users(management) will be able to add staff or delete staff from the system
    c. Add/Search/View Inventory
        i. Users(staff) will be able to add new inventory instances into the system
        ii. Users(staff) will be able to view a list of all the registered items in the system
7. **Department Functions**
    a. Add/Search/View Rooms
        i. Users(staff) will be able to manipulate room information.
    b. Add/Search/View Services
        i. Users(management) will be able to manipulate services
        ii. Updates InChargeOf Table with related data instances

Relational Schema

- Login(<u>userID</u>, password, age, firstName, lastName, address, securityClearance)
    - Enters(<u>userID</u>, <u>Staff_ID</u>)
- Staff(<u>Staff_ID</u>, name, age, job, salary)
    - TakesCareOf(<u>Patient_ID</u>, <u>Staff_ID</u>, patient_medical_info)
- Patient(<u>Patient_ID</u>, name, age, address, billing_card, balance)
    - Attends(<u>Appointment_ID</u>, <u>Staff_ID</u>, <u>Patient_ID</u>)
- Department(<u>Department_Name</u>, chairperson)
    - InChargeOf(<u>Department_Name</u>, <u>Service_Name</u>, service_description)
    - Employs(<u>Staff_ID</u>, <u>Department_Name</u>, specialty, services_offered)
- Services(<u>Service_Name</u>, cost, service_department)
    - ConsistsOf(<u>Appointment_ID</u>, <u>Service_Name</u>, service_description)
- Appointments(<u>Appointment_ID</u>, time, date)
- Inventory(<u>Item_ID</u>, name, quantity)
    - ControlledBy(<u>Department_Name</u>, <u>Item_ID</u>, item_usage)
- Room(<u>Room_Number</u>, building, type, beds_available)
    - Has(<u>Room_Number</u>, <u>Appointment_ID</u>)

Entity-Relationship Diagram

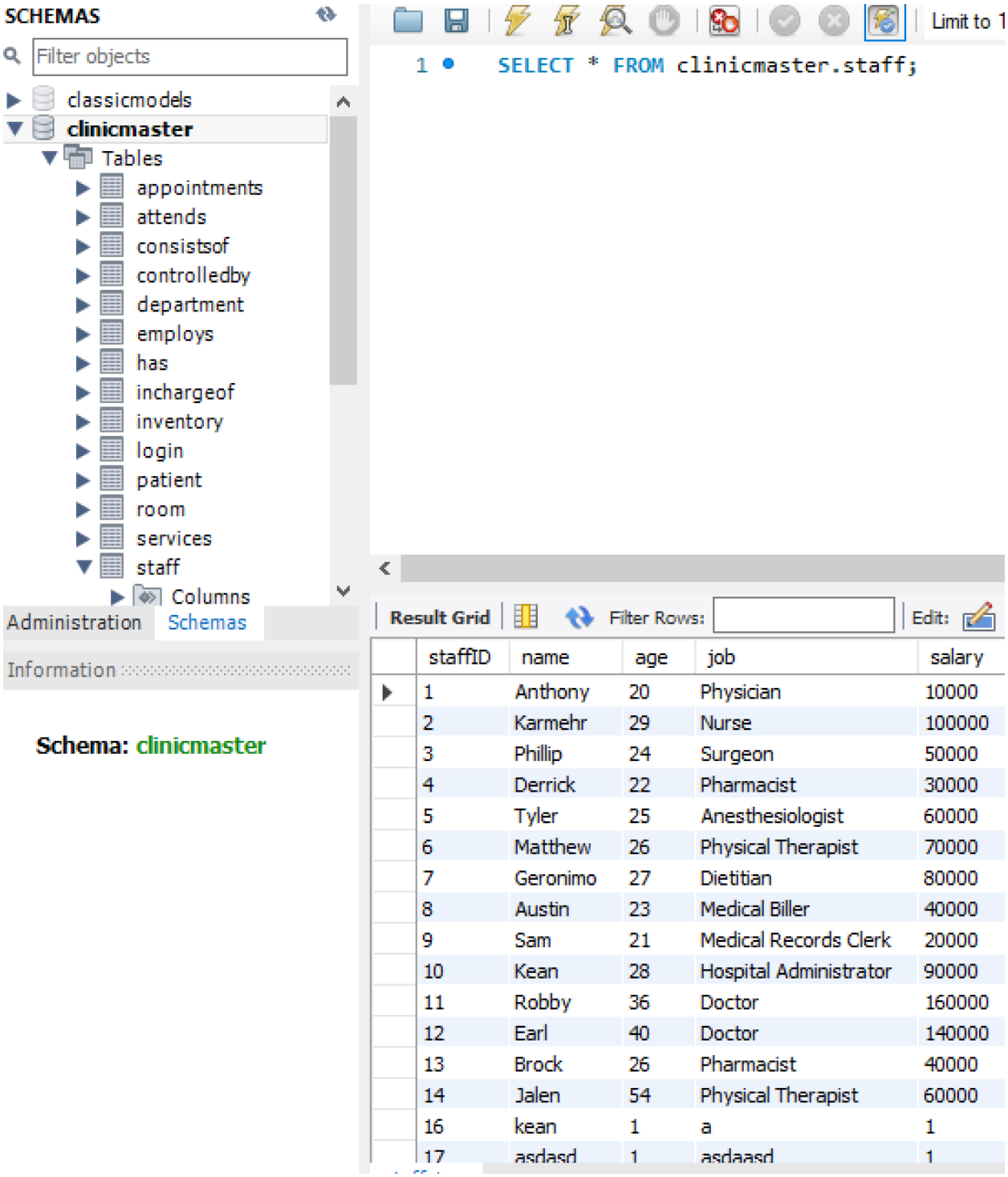


In this diagram software, rounded arrows weren't available, so the following are diagram corrections:

- Staff must be employed by a single department
- Each service must be managed by a single department

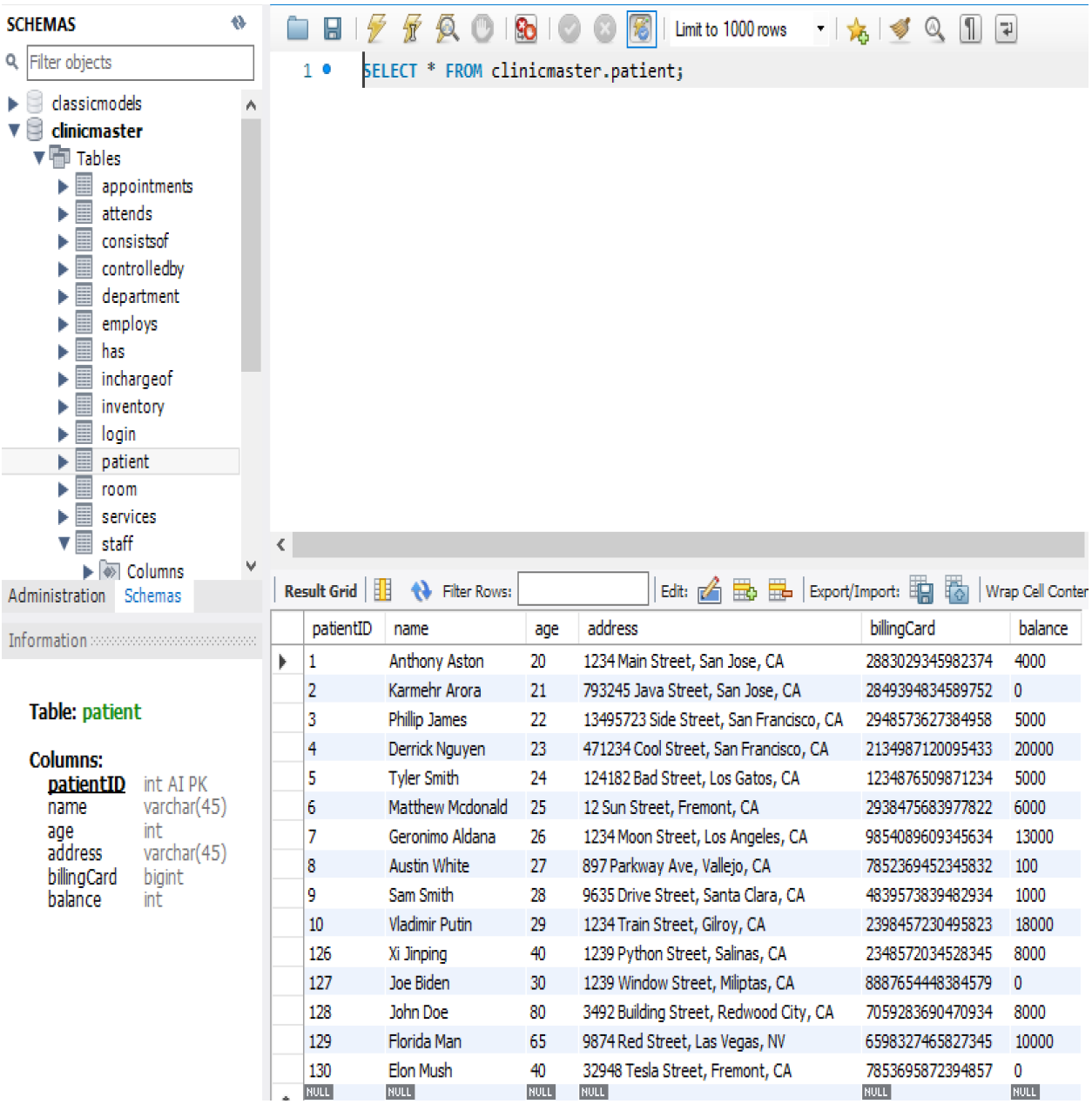
MySQL Workbench Tables

Staff



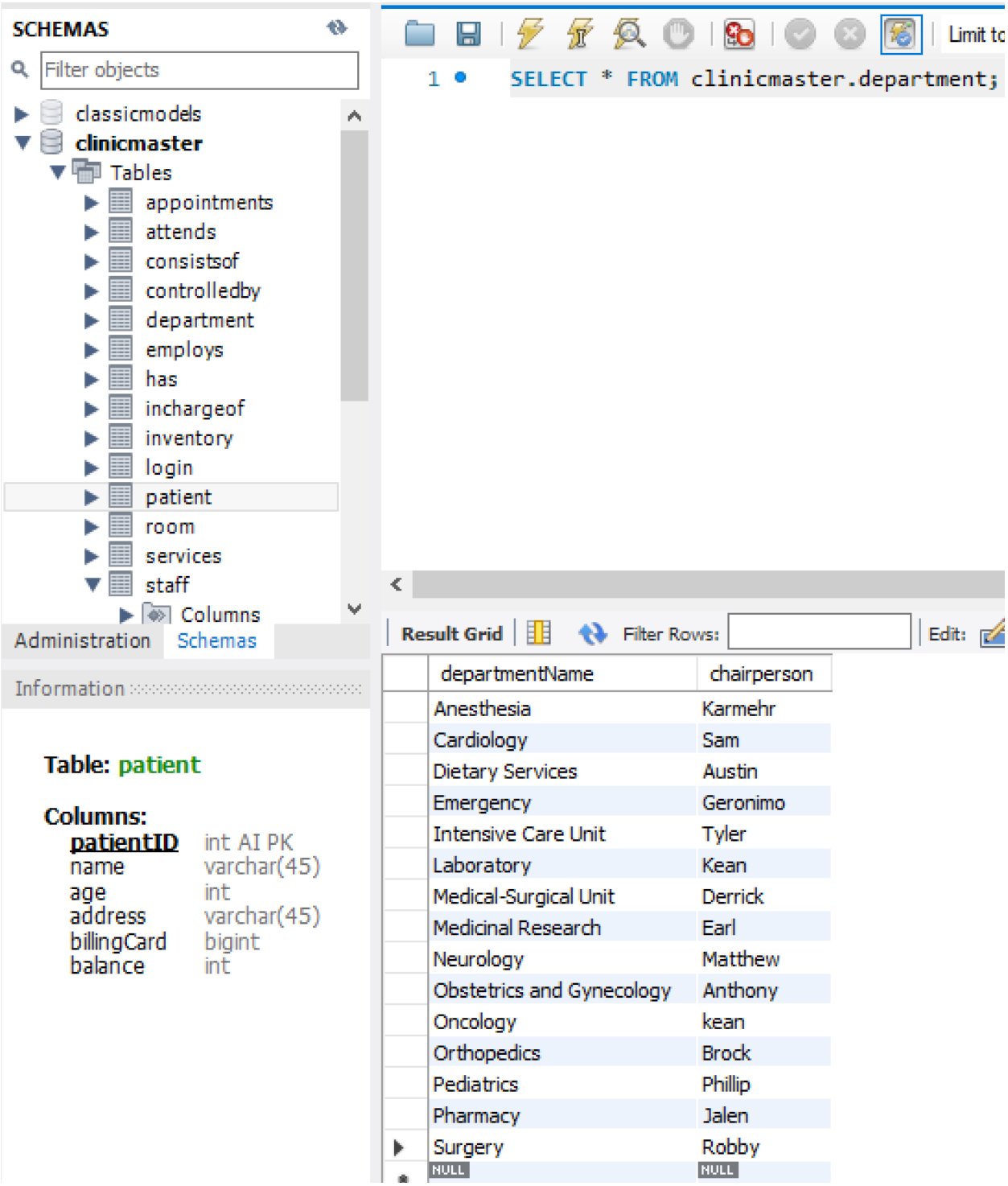
staffID	name	age	job	salary
1	Anthony	20	Physician	10000
2	Karmehr	29	Nurse	100000
3	Phillip	24	Surgeon	50000
4	Derrick	22	Pharmacist	30000
5	Tyler	25	Anesthesiologist	60000
6	Matthew	26	Physical Therapist	70000
7	Geronimo	27	Dietitian	80000
8	Austin	23	Medical Biller	40000
9	Sam	21	Medical Records Clerk	20000
10	Kean	28	Hospital Administrator	90000
11	Robby	36	Doctor	160000
12	Earl	40	Doctor	140000
13	Brock	26	Pharmacist	40000
14	Jalen	54	Physical Therapist	60000
16	kean	1	a	1
17	asdasd	1	asdaasd	1

Patient

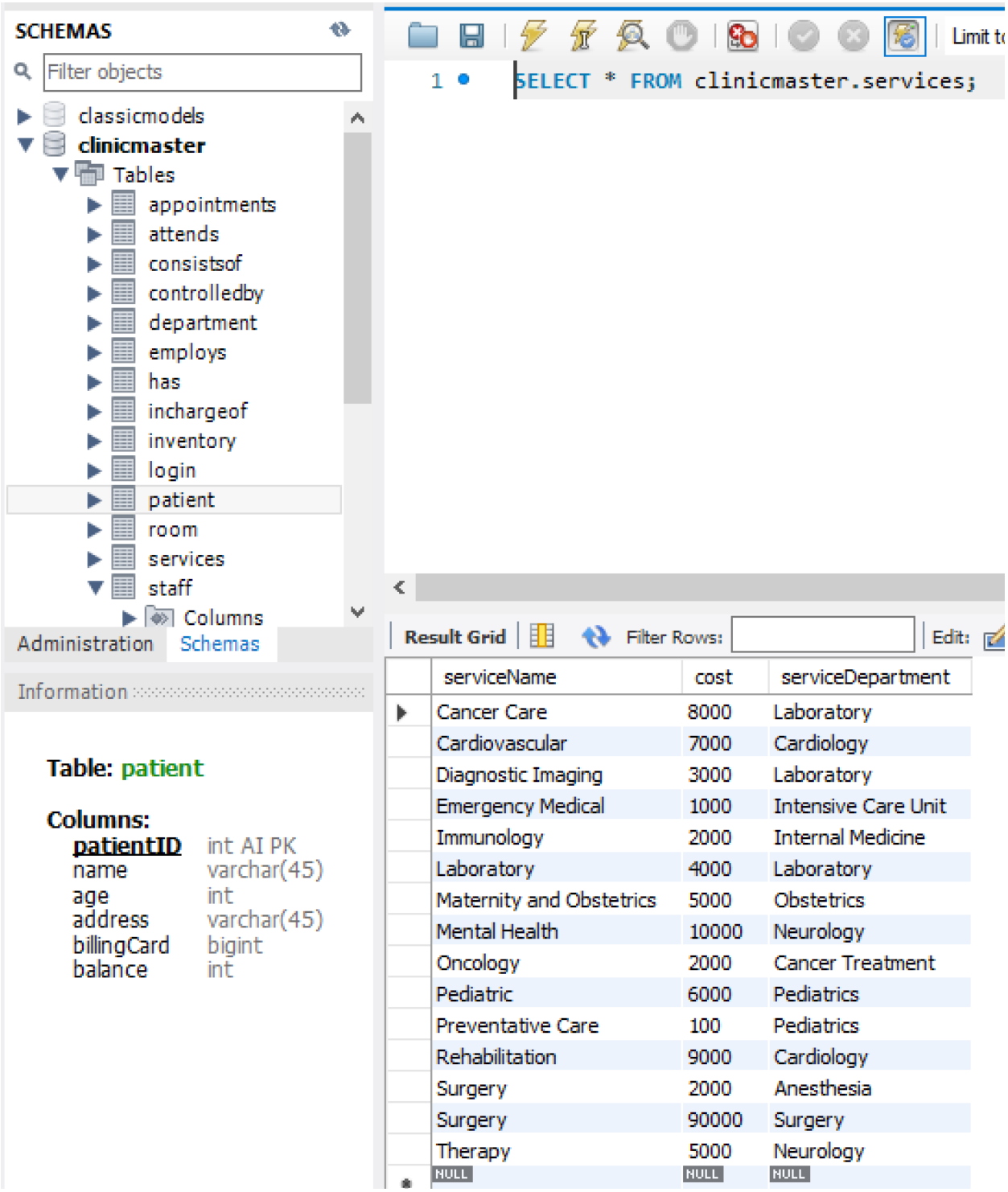


patientID	name	age	address	billingCard	balance
1	Anthony Aston	20	1234 Main Street, San Jose, CA	2883029345982374	4000
2	Karmehr Arora	21	793245 Java Street, San Jose, CA	2849394834589752	0
3	Phillip James	22	13495723 Side Street, San Francisco, CA	2948573627384958	5000
4	Derrick Nguyen	23	471234 Cool Street, San Francisco, CA	2134987120095433	20000
5	Tyler Smith	24	124182 Bad Street, Los Gatos, CA	1234876509871234	5000
6	Matthew Mcdonald	25	12 Sun Street, Fremont, CA	2938475683977822	6000
7	Geronimo Aldana	26	1234 Moon Street, Los Angeles, CA	9854089609345634	13000
8	Austin White	27	897 Parkway Ave, Vallejo, CA	7852369452345832	100
9	Sam Smith	28	9635 Drive Street, Santa Clara, CA	4839573839482934	1000
10	Vladimir Putin	29	1234 Train Street, Gilroy, CA	2398457230495823	18000
126	Xi Jinping	40	1239 Python Street, Salinas, CA	2348572034528345	8000
127	Joe Biden	30	1239 Window Street, Miliptas, CA	8887654448384579	0
128	John Doe	80	3492 Building Street, Redwood City, CA	7059283690470934	8000
129	Florida Man	65	9874 Red Street, Las Vegas, NV	6598327465827345	10000
130	Elon Mush	40	32948 Tesla Street, Fremont, CA	7853695872394857	0
NULL	NULL	NULL	NULL	NULL	NULL

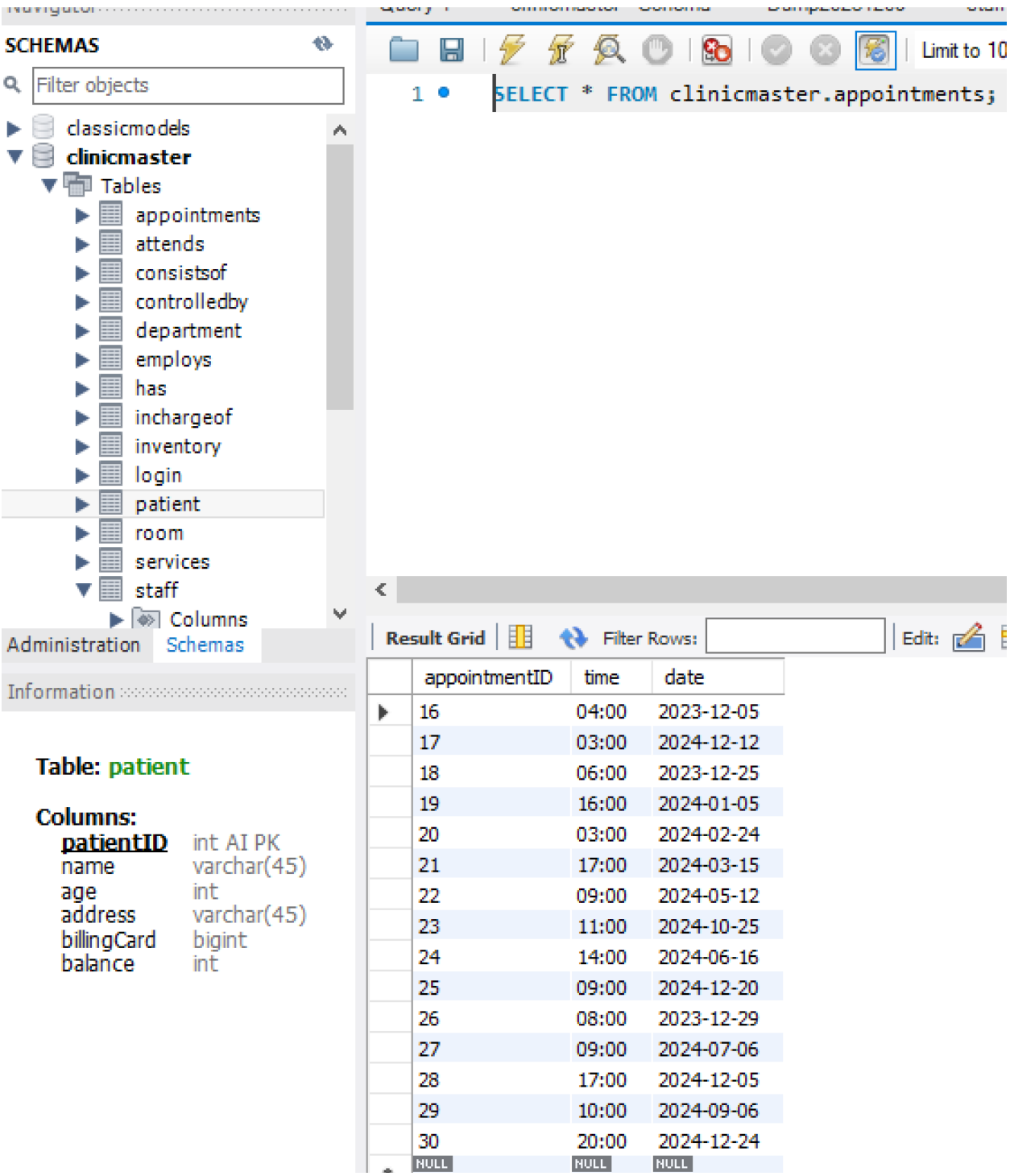
Department


SCHEMAS
Filter objects
classicmodels
clinicmaster
Tables
appointments
attends
consistsof
controlledby
department
employs
has
inchargeof
inventory
login
patient
room
services
staff
Columns
Administration
Schemas
Information
Table: patient
Columns:
patientID int AI PK
name varchar(45)
age int
address varchar(45)
billingCard bigint
balance int
Limit to
1 SELECT * FROM clinicmaster.department;
Result Grid
Filter Rows:
Edit:
departmentName chairperson
Anesthesia Karmehr
Cardiology Sam
Dietary Services Austin
Emergency Geronimo
Intensive Care Unit Tyler
Laboratory Kean
Medical-Surgical Unit Derrick
Medicinal Research Earl
Neurology Matthew
Obstetrics and Gynecology Anthony
Oncology kean
Orthopedics Brock
Pediatrics Phillip
Pharmacy Jalen
Surgery Robby
NULL NULL

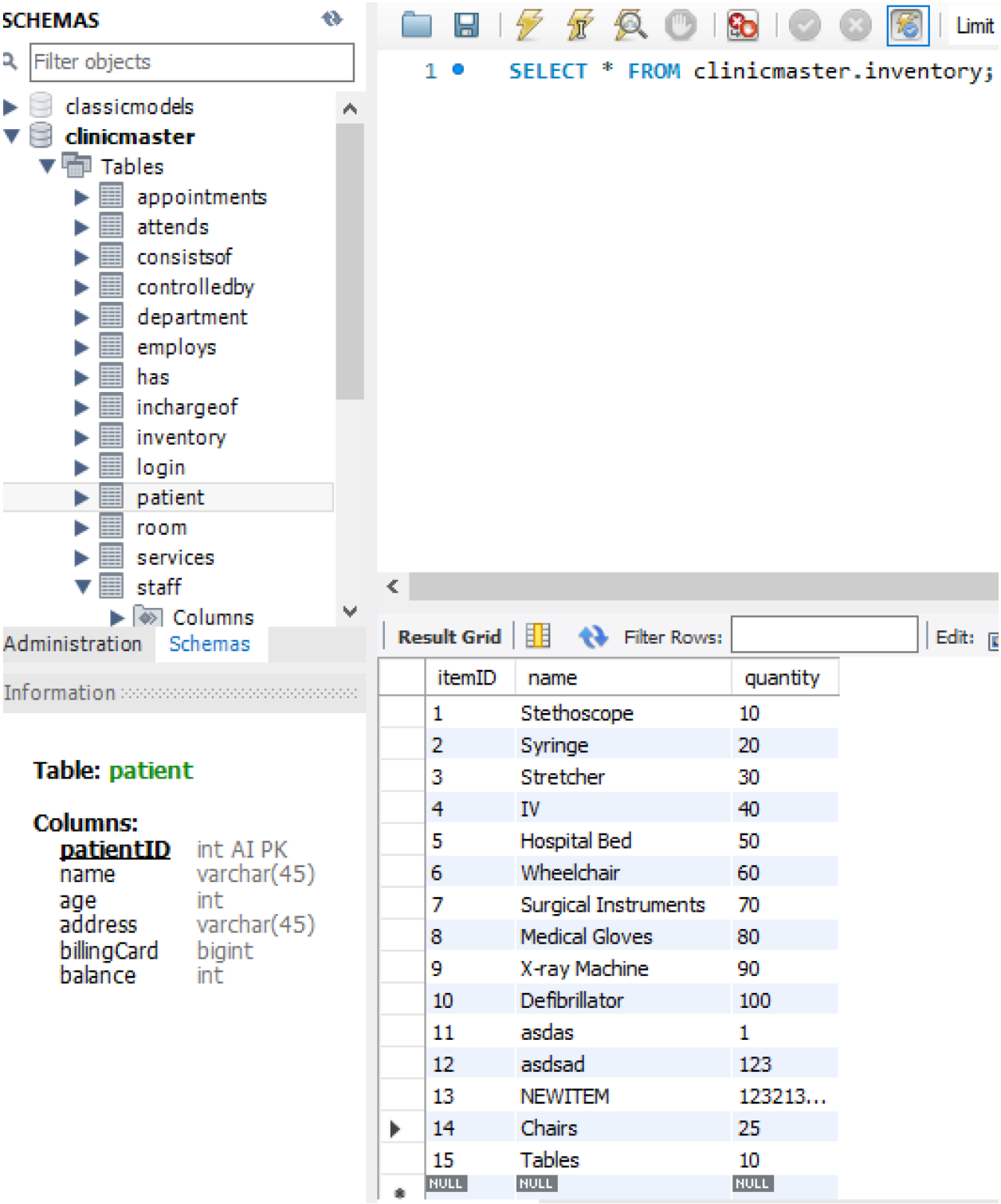
Services


SCHEMAS
Filter objects
classicmodels
clinicmaster
Tables
appointments
attends
consistsof
controlledby
department
employs
has
inchargeof
inventory
login
patient
room
services
staff
Columns
Administration
Schemas
Information
Table: patient
Columns:
patientID int AI PK
name varchar(45)
age int
address varchar(45)
billingCard bigint
balance int
1 SELECT * FROM clinicmaster.services;
Result Grid
Filter Rows:
Edit:
serviceName cost serviceDepartment
Cancer Care 8000 Laboratory
Cardiovascular 7000 Cardiology
Diagnostic Imaging 3000 Laboratory
Emergency Medical 1000 Intensive Care Unit
Immunology 2000 Internal Medicine
Laboratory 4000 Laboratory
Maternity and Obstetrics 5000 Obstetrics
Mental Health 10000 Neurology
Oncology 2000 Cancer Treatment
Pediatric 6000 Pediatrics
Preventative Care 100 Pediatrics
Rehabilitation 9000 Cardiology
Surgery 2000 Anesthesia
Surgery 90000 Surgery
Therapy 5000 Neurology
NULL NULL NULL

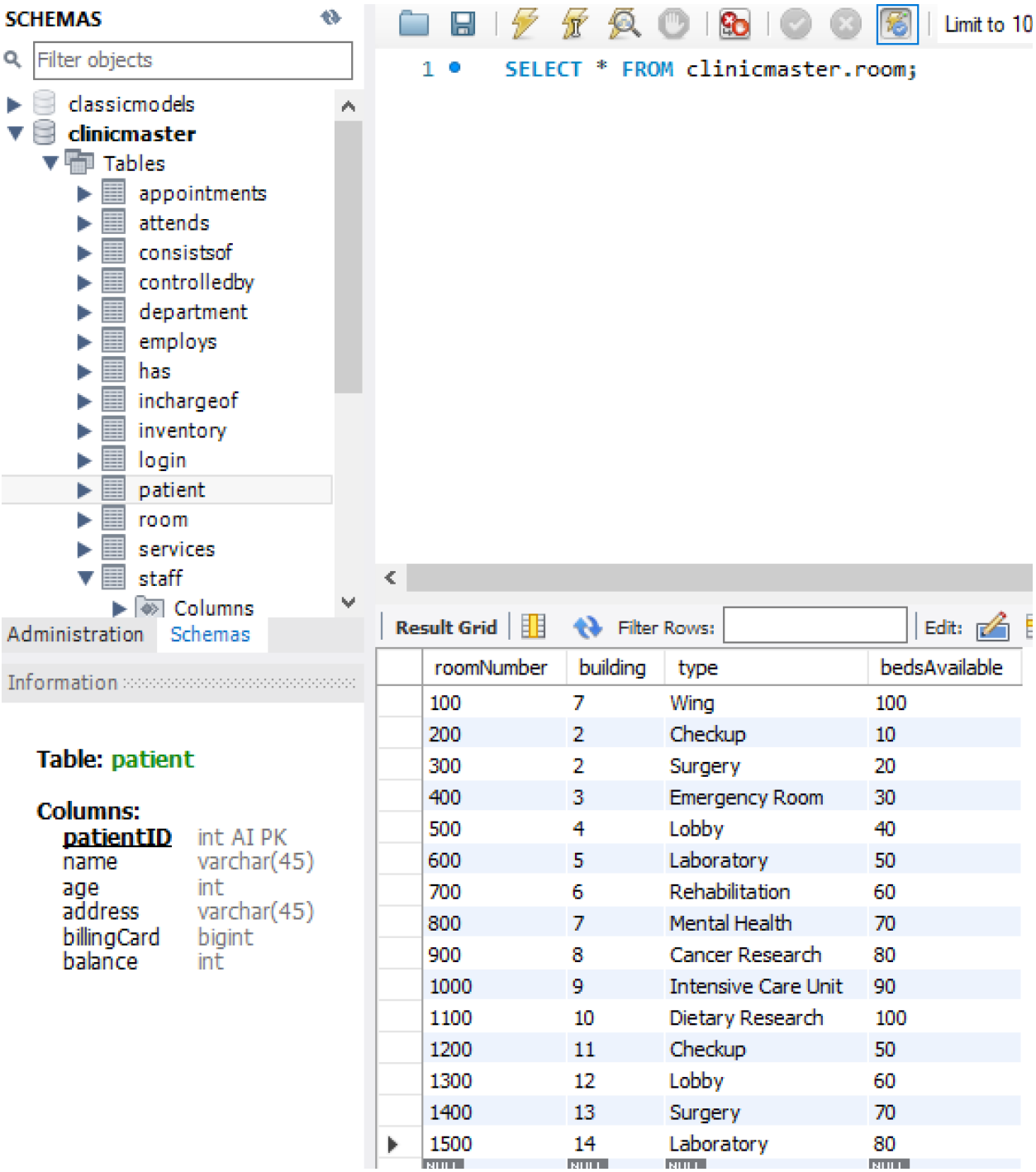
Appointments



Inventory

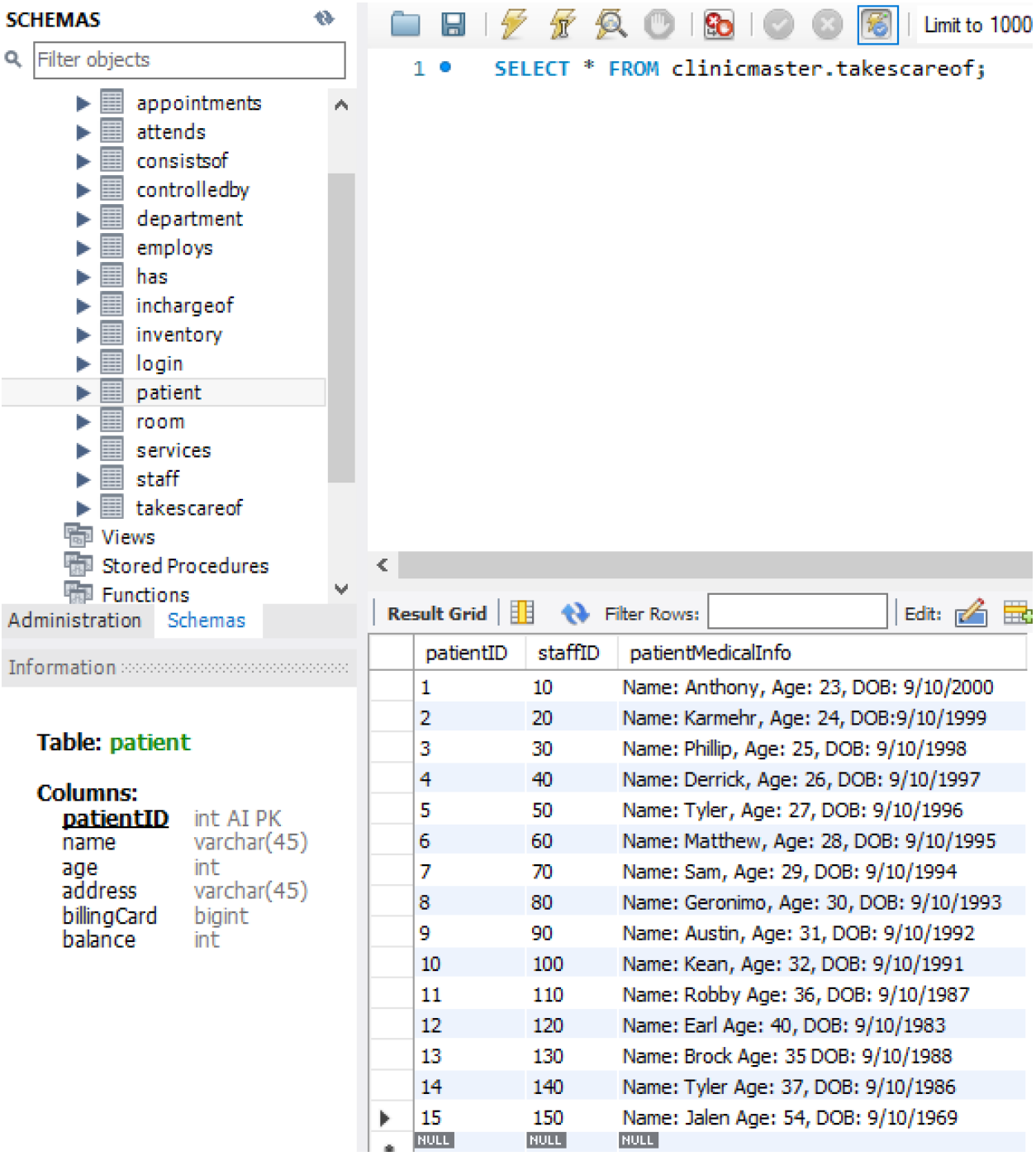

SCHEMAS
Filter objects
classicmodels
clinicmaster
Tables
appointments
attends
consistsof
controlledby
department
employs
has
inchargeof
inventory
login
patient
room
services
staff
Columns
Administration
Schemas
Information
Table: patient
Columns:
patientID int AI PK
name varchar(45)
age int
address varchar(45)
billingCard bigint
balance int
Limit
1 SELECT * FROM clinicmaster.inventory;
Result Grid
Filter Rows:
Edit:
itemID name quantity
1 Stethoscope 10
2 Syringe 20
3 Stretcher 30
4 IV 40
5 Hospital Bed 50
6 Wheelchair 60
7 Surgical Instruments 70
8 Medical Gloves 80
9 X-ray Machine 90
10 Defibrillator 100
11 asdas 1
12 asdsad 123
13 NEWITEM 123213...
14 Chairs 25
15 Tables 10
NULL NULL NULL

Room

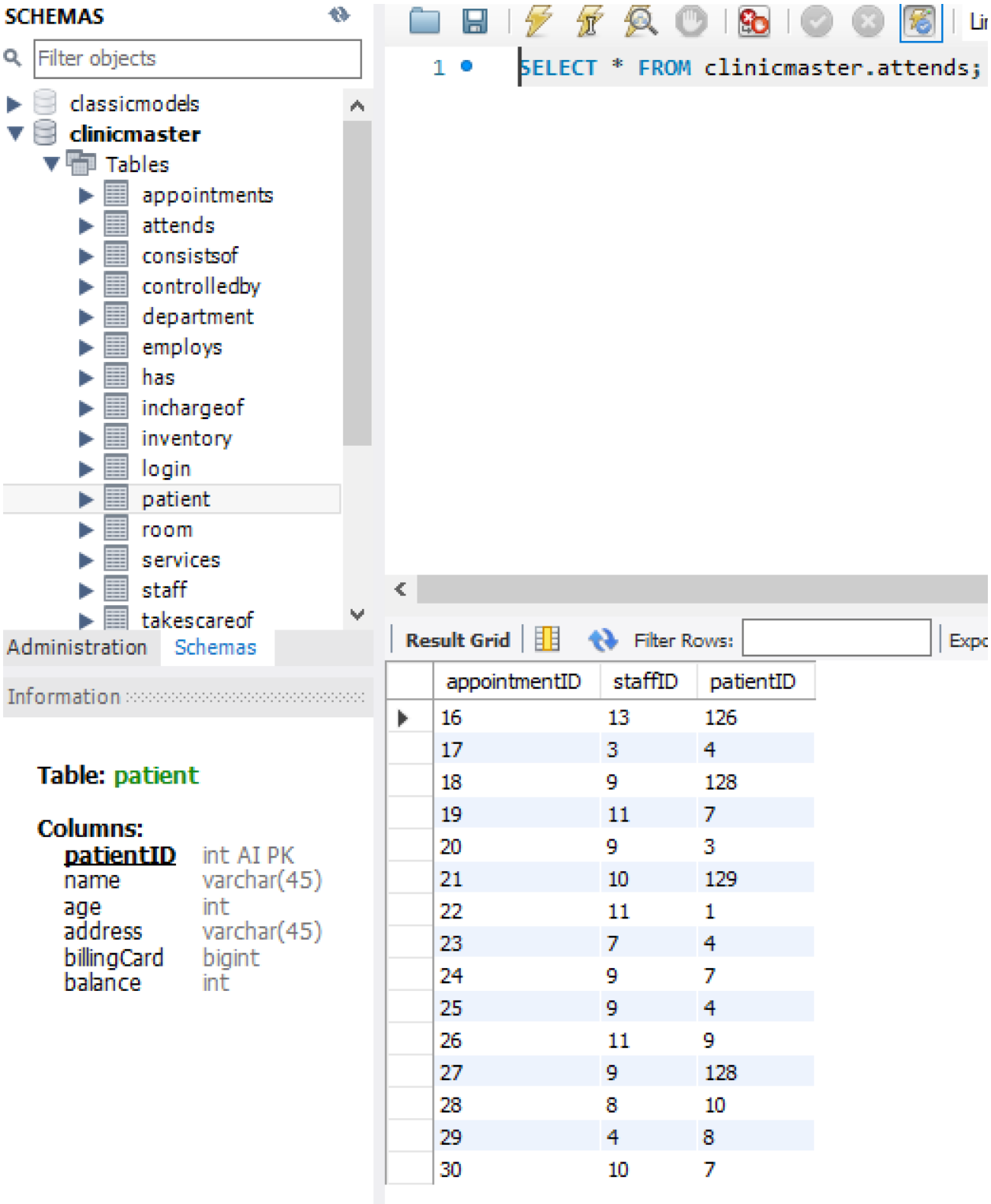


roomNumber	building	type	bedsAvailable
100	7	Wing	100
200	2	Checkup	10
300	2	Surgery	20
400	3	Emergency Room	30
500	4	Lobby	40
600	5	Laboratory	50
700	6	Rehabilitation	60
800	7	Mental Health	70
900	8	Cancer Research	80
1000	9	Intensive Care Unit	90
1100	10	Dietary Research	100
1200	11	Checkup	50
1300	12	Lobby	60
1400	13	Surgery	70
1500	14	Laboratory	80

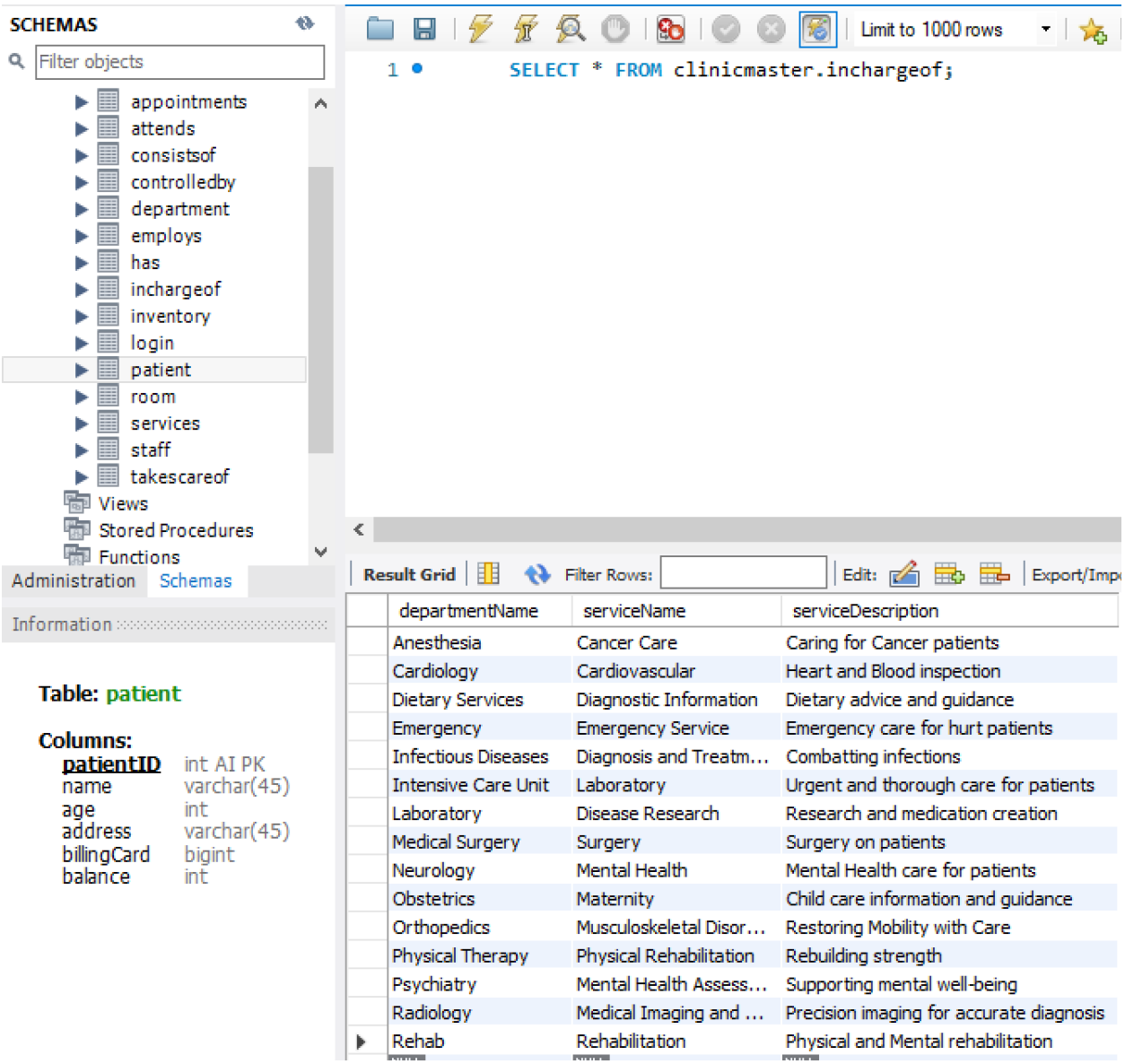
TakesCareOf



Attends

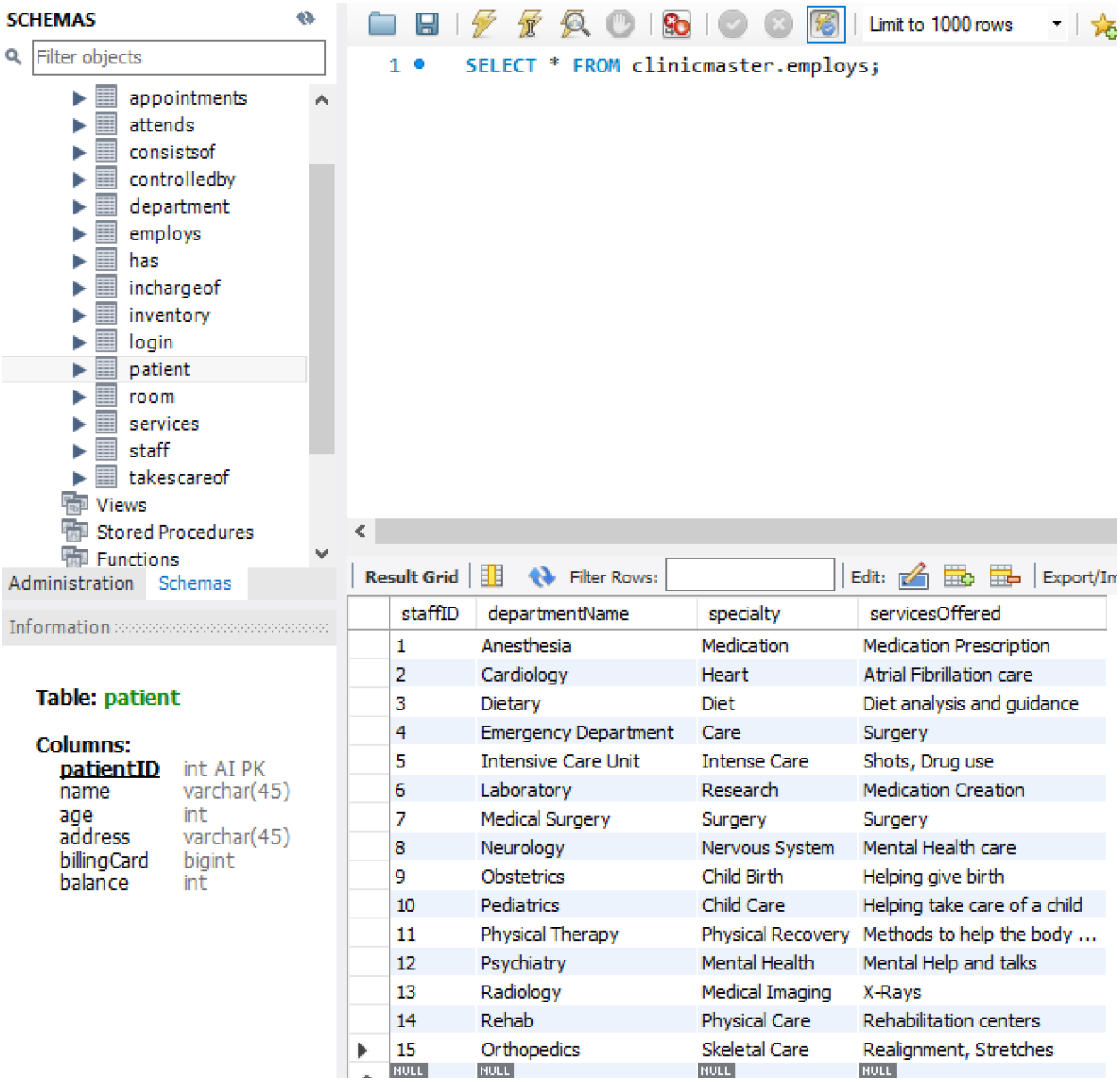

SCHEMAS
Filter objects
classicmodels
clinicmaster
Tables
appointments
attends
consistsof
controlledby
department
employs
has
inchargeof
inventory
login
patient
room
services
staff
takescareof
Administration
Schemas
Information
Table: patient
Columns:
patientID int AI PK
name varchar(45)
age int
address varchar(45)
billingCard bigint
balance int
1 SELECT * FROM clinicmaster.attends;
Result Grid
Filter Rows:
appointmentID staffID patientID
16 13 126
17 3 4
18 9 128
19 11 7
20 9 3
21 10 129
22 11 1
23 7 4
24 9 7
25 9 4
26 11 9
27 9 128
28 8 10
29 4 8
30 10 7

InChargeOf



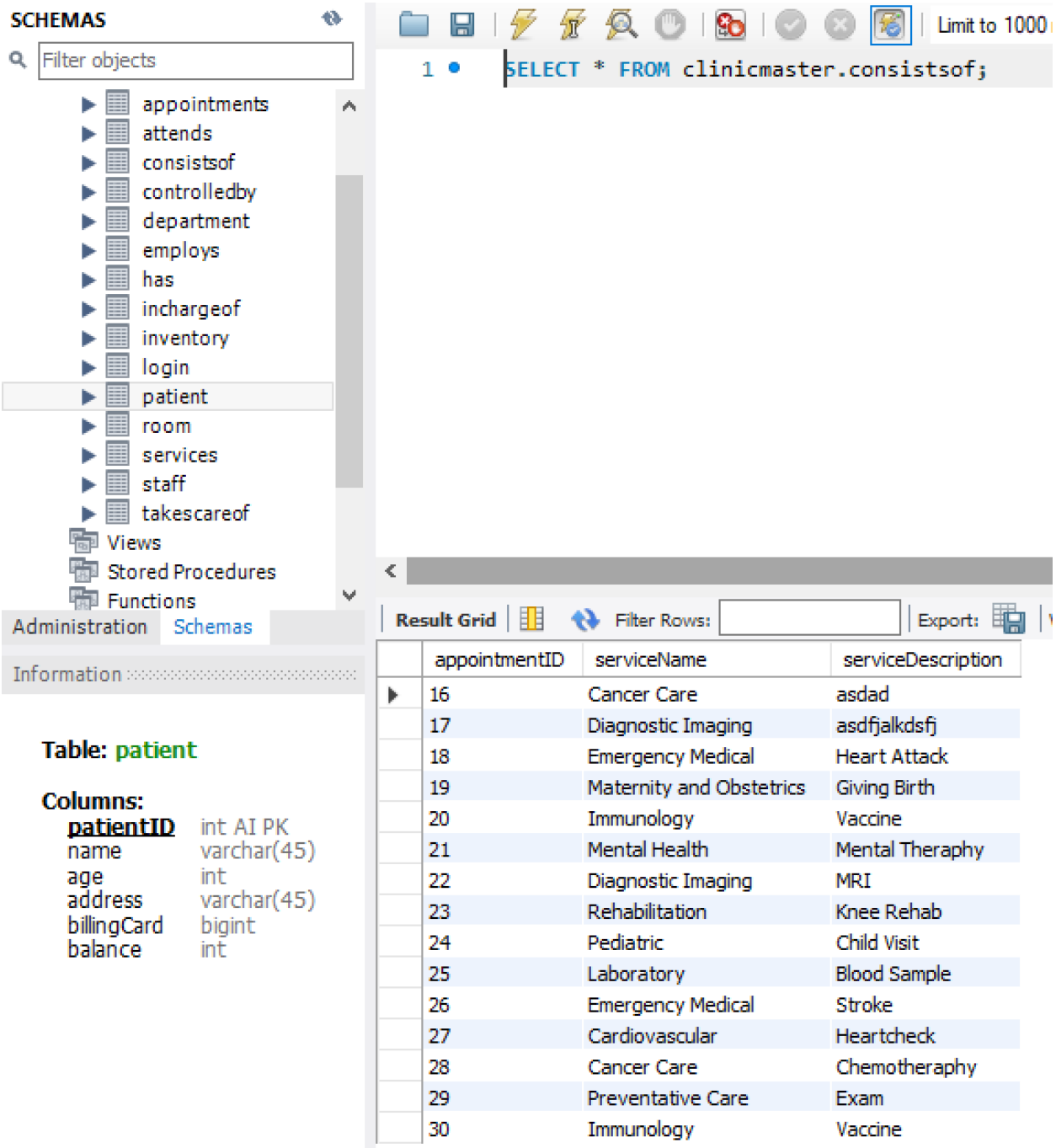
departmentName	serviceName	serviceDescription
Anesthesia	Cancer Care	Caring for Cancer patients
Cardiology	Cardiovascular	Heart and Blood inspection
Dietary Services	Diagnostic Information	Dietary advice and guidance
Emergency	Emergency Service	Emergency care for hurt patients
Infectious Diseases	Diagnosis and Treatm...	Combatting infections
Intensive Care Unit	Laboratory	Urgent and thorough care for patients
Laboratory	Disease Research	Research and medication creation
Medical Surgery	Surgery	Surgery on patients
Neurology	Mental Health	Mental Health care for patients
Obstetrics	Maternity	Child care information and guidance
Orthopedics	Musculoskeletal Disor...	Restoring Mobility with Care
Physical Therapy	Physical Rehabilitation	Rebuilding strength
Psychiatry	Mental Health Assess...	Supporting mental well-being
Radiology	Medical Imaging and ...	Precision imaging for accurate diagnosis
Rehab	Rehabilitation	Physical and Mental rehabilitation

Employs

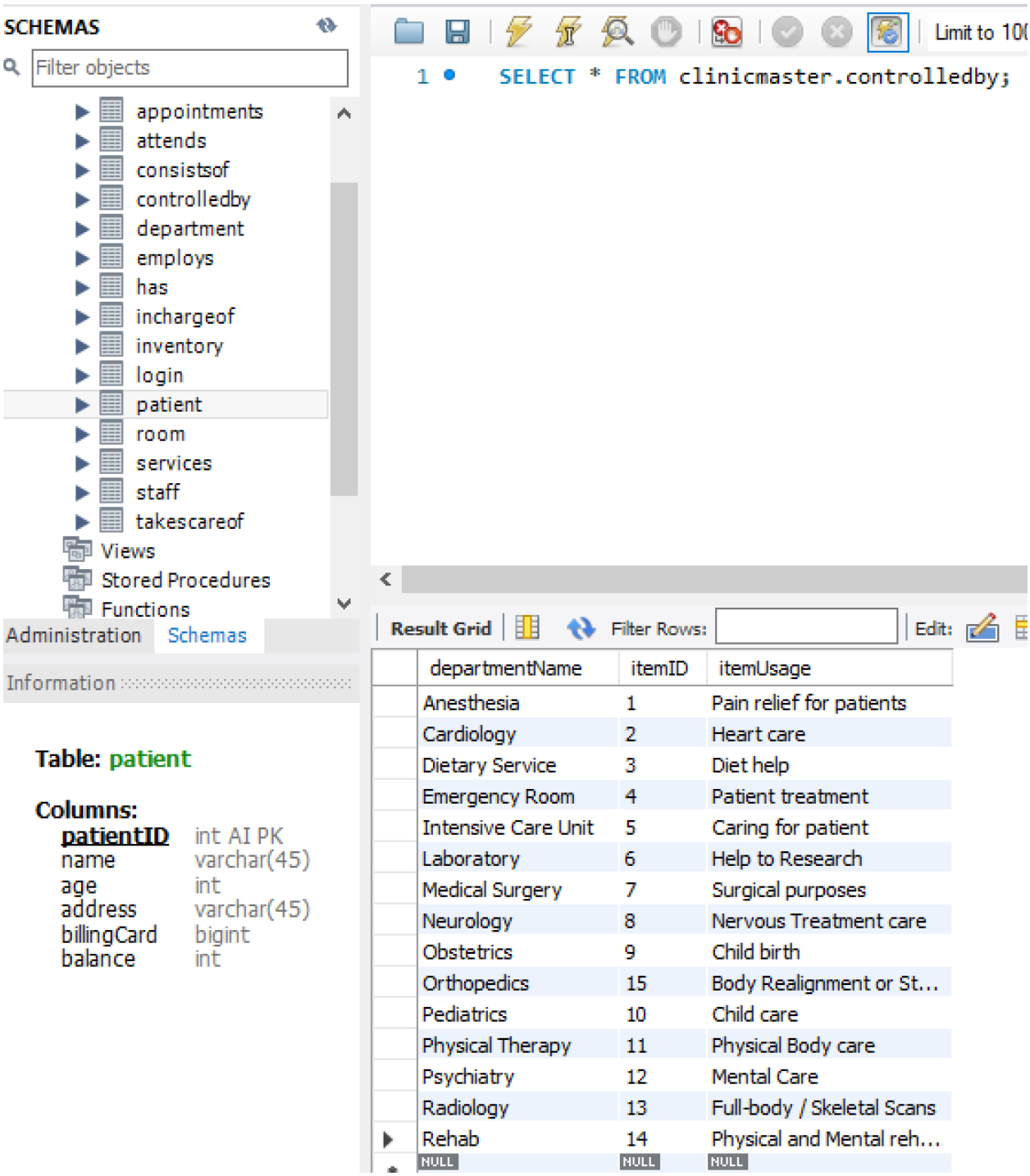


staffID	departmentName	specialty	servicesOffered
1	Anesthesia	Medication	Medication Prescription
2	Cardiology	Heart	Atrial Fibrillation care
3	Dietary	Diet	Diet analysis and guidance
4	Emergency Department	Care	Surgery
5	Intensive Care Unit	Intense Care	Shots, Drug use
6	Laboratory	Research	Medication Creation
7	Medical Surgery	Surgery	Surgery
8	Neurology	Nervous System	Mental Health care
9	Obstetrics	Child Birth	Helping give birth
10	Pediatrics	Child Care	Helping take care of a child
11	Physical Therapy	Physical Recovery	Methods to help the body ...
12	Psychiatry	Mental Health	Mental Help and talks
13	Radiology	Medical Imaging	X-Rays
14	Rehab	Physical Care	Rehabilitation centers
15	Orthopedics	Skeletal Care	Realignment, Stretches
NULL	NULL	NULL	NULL

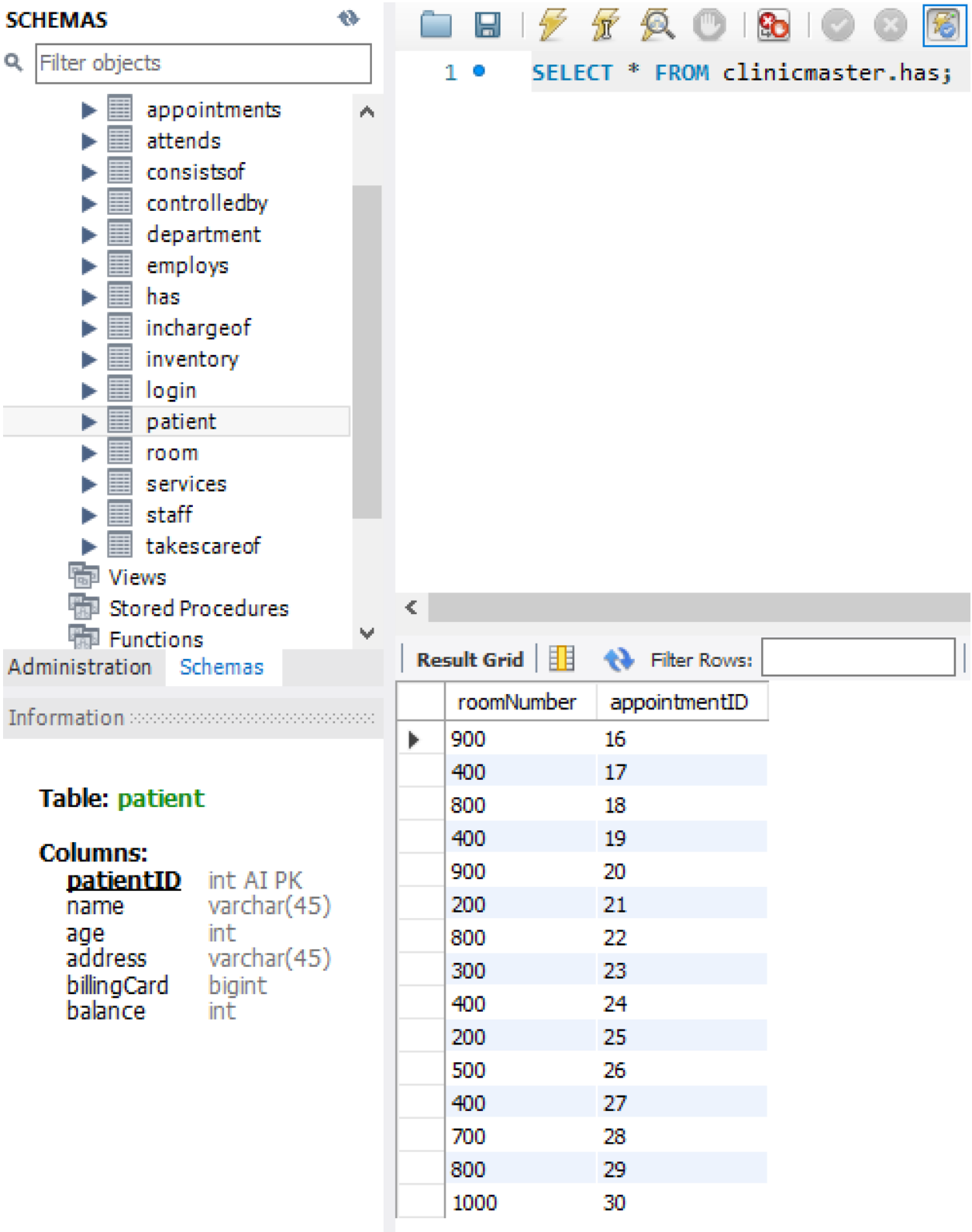
ConsistsOf


SCHEMAS
Filter objects
appointments
attends
consistsof
controlledby
department
employs
has
inchargeof
inventory
login
patient
room
services
staff
takescareof
Views
Stored Procedures
Functions
Administration
Schemas
Information
Table: patient
Columns:
patientID int AI PK
name varchar(45)
age int
address varchar(45)
billingCard bigint
balance int
Limit to 1000
1 SELECT * FROM clinicmaster.consistsof;
Result Grid
Filter Rows:
Export:
appointmentID serviceName serviceDescription
16 Cancer Care asdad
17 Diagnostic Imaging asdfjalkdsfj
18 Emergency Medical Heart Attack
19 Maternity and Obstetrics Giving Birth
20 Immunology Vaccine
21 Mental Health Mental Theraphy
22 Diagnostic Imaging MRI
23 Rehabilitation Knee Rehab
24 Pediatric Child Visit
25 Laboratory Blood Sample
26 Emergency Medical Stroke
27 Cardiovascular Heartcheck
28 Cancer Care Chemotheraphy
29 Preventative Care Exam
30 Immunology Vaccine

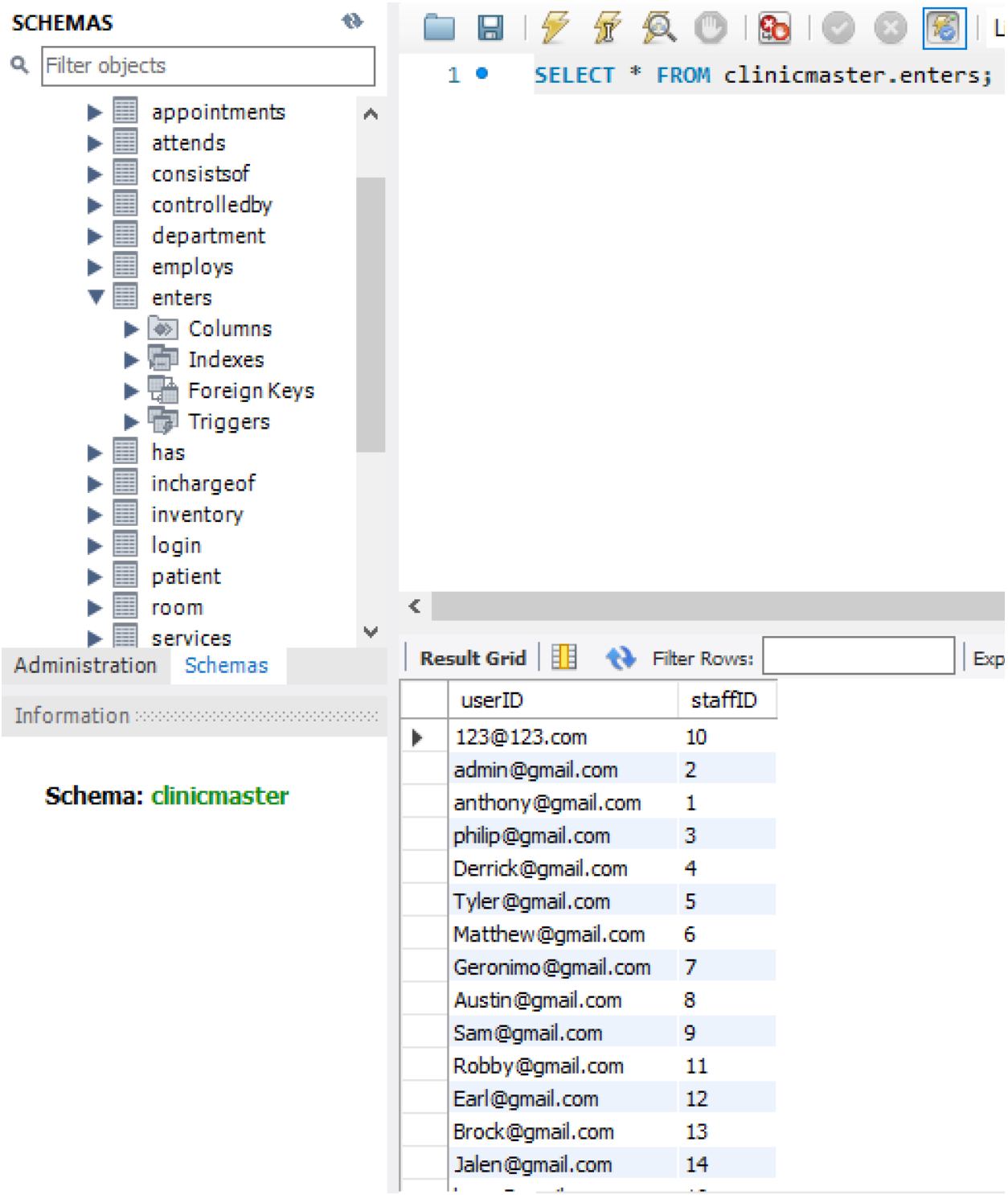
ControlledBy



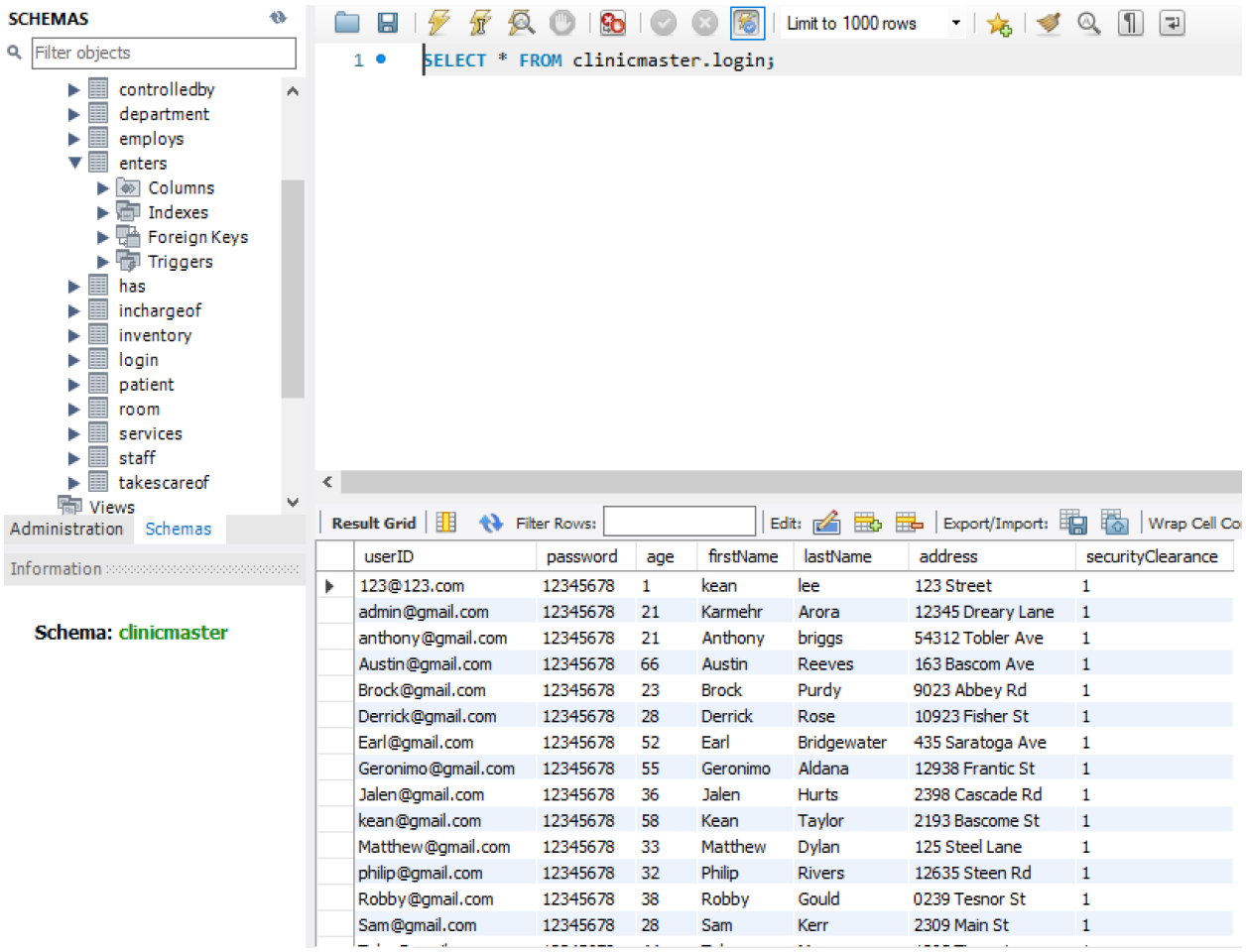
Has


SCHEMAS
Filter objects
appointments
attends
consistsof
controlledby
department
employs
has
inchargeof
inventory
login
patient
room
services
staff
takescareof
Views
Stored Procedures
Functions
Administration
Schemas
Information
Table: patient
Columns:
patientID int AI PK
name varchar(45)
age int
address varchar(45)
billingCard bigint
balance int
1 SELECT * FROM clinicmaster.has;
Result Grid
Filter Rows:
roomNumber appointmentID
900 16
400 17
800 18
400 19
900 20
200 21
800 22
300 23
400 24
200 25
500 26
400 27
700 28
800 29
1000 30

Enters


SCHEMAS
Filter objects
appointments
attends
consistsof
controlledby
department
employs
enters
Columns
Indexes
Foreign Keys
Triggers
has
inchargeof
inventory
login
patient
room
services
Administration
Schemas
Information
Schema: clinicmaster
1 SELECT * FROM clinicmaster.enters;
Result Grid
Filter Rows:
Exp
userID
staffID
123@123.com 10
admin@gmail.com 2
anthony@gmail.com 1
philip@gmail.com 3
Derrick@gmail.com 4
Tyler@gmail.com 5
Matthew@gmail.com 6
Geronimo@gmail.com 7
Austin@gmail.com 8
Sam@gmail.com 9
Robby@gmail.com 11
Earl@gmail.com 12
Brock@gmail.com 13
Jalen@gmail.com 14

Login

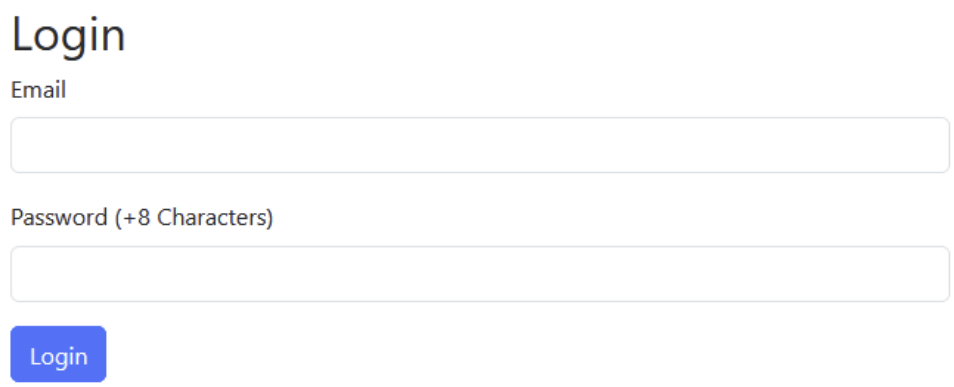


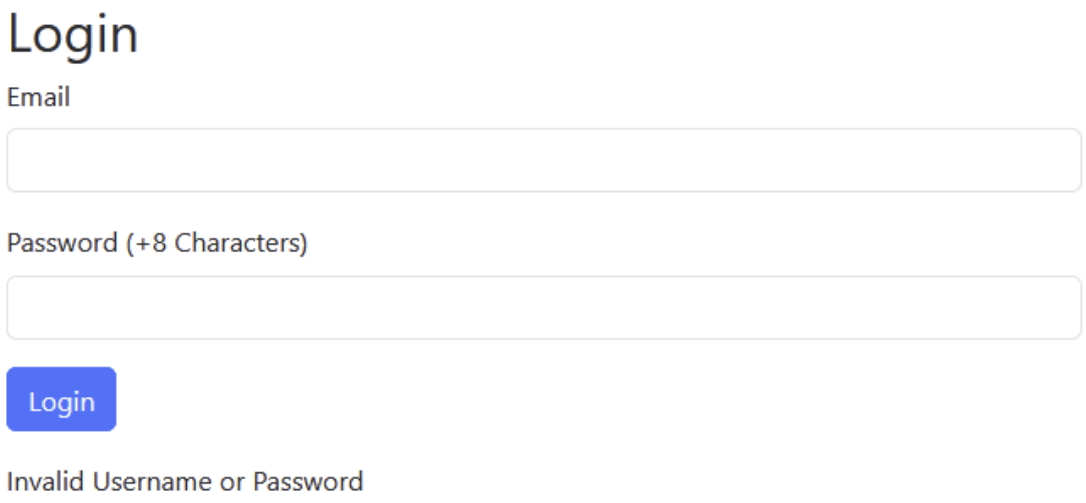
userID	password	age	firstName	lastName	address	securityClearance
123@123.com	12345678	1	kean	lee	123 Street	1
admin@gmail.com	12345678	21	Karmehr	Arora	12345 Dreary Lane	1
anthony@gmail.com	12345678	21	Anthony	briggs	54312 Tobler Ave	1
Austin@gmail.com	12345678	66	Austin	Reeves	163 Bascom Ave	1
Brock@gmail.com	12345678	23	Brock	Purdy	9023 Abbey Rd	1
Derrick@gmail.com	12345678	28	Derrick	Rose	10923 Fisher St	1
Earl@gmail.com	12345678	52	Earl	Bridgewater	435 Saratoga Ave	1
Geronimo@gmail.com	12345678	55	Geronimo	Aldana	12938 Frantic St	1
Jalen@gmail.com	12345678	36	Jalen	Hurts	2398 Cascade Rd	1
kean@gmail.com	12345678	58	Kean	Taylor	2193 Bascome St	1
Matthew@gmail.com	12345678	33	Matthew	Dylan	125 Steel Lane	1
philip@gmail.com	12345678	32	Philip	Rivers	12635 Steen Rd	1
Robby@gmail.com	12345678	38	Robby	Gould	0239 Tesnor St	1
Sam@gmail.com	12345678	28	Sam	Kerr	2309 Main St	1

Project Implementation of Functional Requirements

Project Implementation is split in 7 pages: signupPage.jsp, loginPage.jsp, homePage.jsp, patientView.jsp, appointment.jsp, staffView.jsp, departments.jsp (requirements shown in this order, demonstrates the flow of the application). It is split like this so it is easier to group related functional requirements together in the same page. We will explain each functional requirements implementation below each represented by its number and letter stated in the functional requirements section above.

1. Login





Screenshot of Login Functionality

As you can see in this image above the user can login to the application through this component. Users can enter a valid email and password and click login to do so. If the user enters an incorrect username or password, an error message will be displayed. If a user has been registered and is authorized to proceed into the application, once they are verified, the application will automatically take them to the home page.

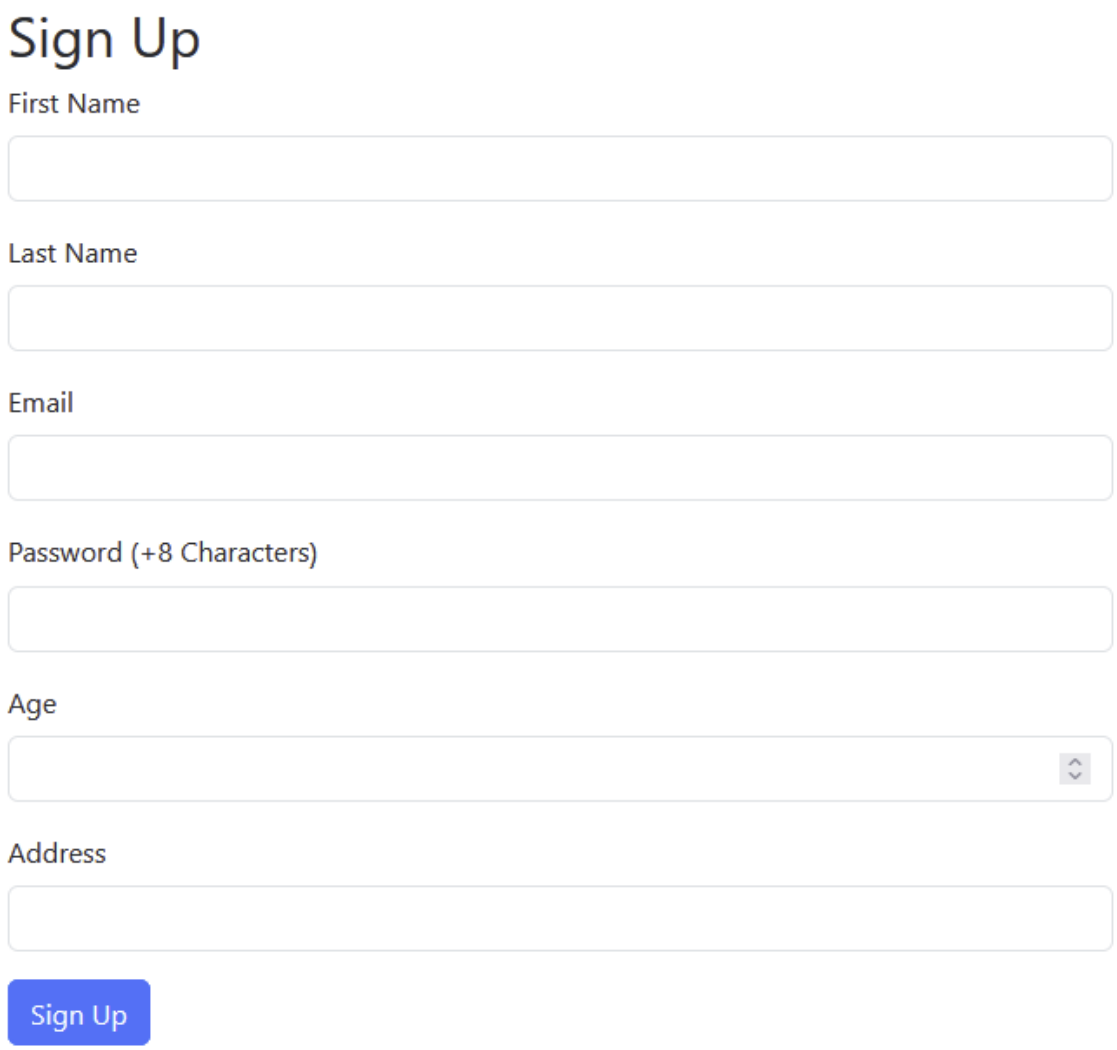
How This Functional Requirement Was Implemented

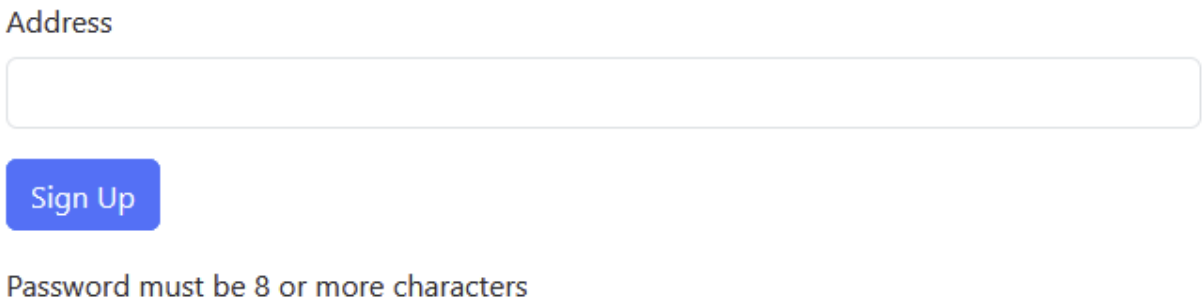
These functional requirements were implemented using a submission form and a submit button to enter the data. In each entry box, users can input their username and password respectively for them to be acceptable inputs.

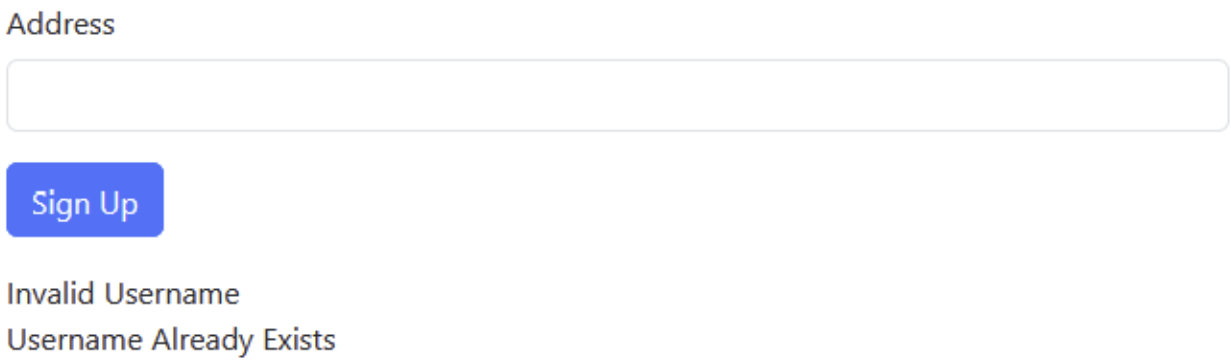
SQL Statements Used In This Functional Requirement

1. Find all instances with the same username and password (should only return zero or one instance). User is verified afterwards using their inputs if the search returns anything.
    a. SELECT * FROM login WHERE userID IN (SELECT ‘“ + request.getParameter(“username”) + “‘ FROM login) AND password IN (SELECT ‘“ + request.getParameter(“password”) + “‘ FROM login)

2. Sign Up


Sign Up
First Name
Last Name
Email
Password (+8 Characters)
Age
Address
Sign Up


Address
Sign Up
Password must be 8 or more characters



Screenshot of Sign Up Functionality

As you can see in this image above the user can Sign Up to the application through this component. Users can enter a valid email and password and other data and click Sign Up to do so. If the user enters an incorrect username, password, or age, an error message will be displayed. If the user enters acceptable data and signs up successfully, their information will be added to login and the application will automatically take them to the login page.

How This Functional Requirement Was Implemented

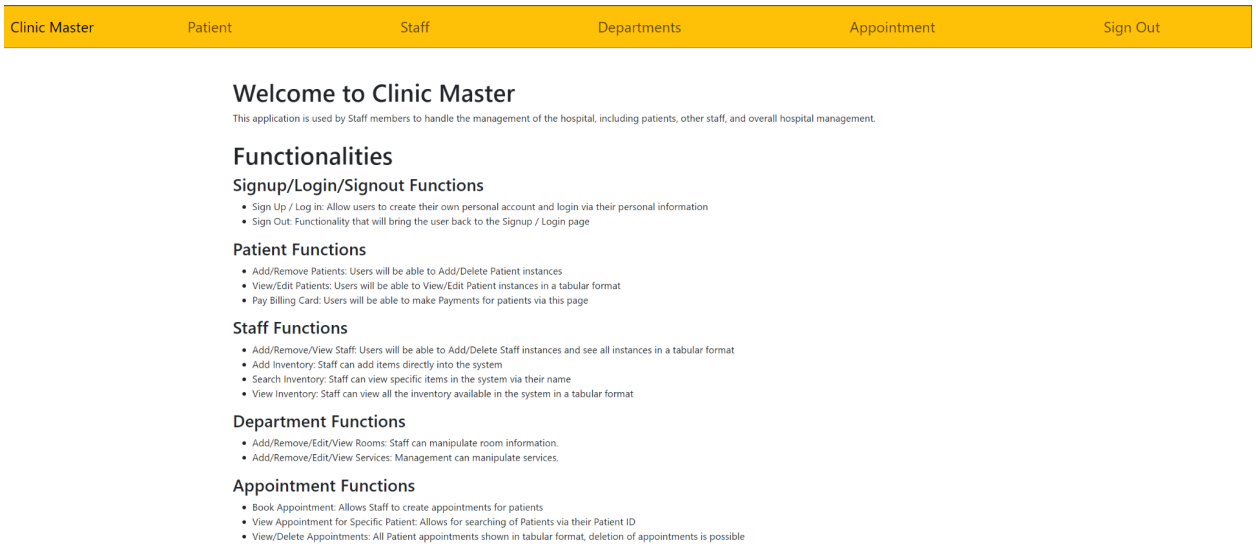
These functional requirements were implemented using a submission form and a submit button to enter the data. In each entry box, users can input their data and they will be verified as acceptable inputs or not. Afterwards, when the user presses the "Sign Up" Button, user data is verified and added to login if acceptable. If the user data isn't acceptable, then an error message will be displayed as shown above.

SQL Statements Used In This Functional Requirement

1. Insert into login
    a. **"INSERT INTO login VALUES("" + username + "', "" + pass + "', " + ageInt + ", "" + firstName + "', "" + lastName + "', "" + address + "', " + 1 + ")**

3. Home Page

Visual of Home Page



The Home Page is a static page that contains text that displays a Welcome message to the user and also shows all of the functionalities that the site has to offer. This is the first page that the user should see that has the Navigation Bar at the top of the page, and this contains buttons that will allow users to redirect to the main page (Clinic Master top right), redirect to the Patient page, redirect to the Staff page, redirect to the Departments page, redirect to the Appointment page. The Sign Out button will redirect the user back to the Sign Up page.

4a. Add/Remove Patient

Visual of Functional Requirement

Screenshot of Add/Remove/Edit Patient Functional Requirement

As shown in the image above the user can input user details and add a patient to the system with a simple form submission. In order to remove a patient from the system in the image there is a table with all the patients and an option to delete the patient.

How This Functional Requirement Was Implemented

In order to add the patient, user input was first required from the user. HTML elements such as form and inputs were used in order to create a form in which users can input data. When the user presses the add button it will add the patient to the system and subsequently to the database. In order to remove the patients from the applicant first we had to display the patients as explained in functional requirement 1b. Within the table a button was added with the tag of the id of the patient. This allowed the delete button to be associated with said patient and thus when pressed is removed from the table and subsequently from the database.

SQL Statements Used In This Functional Requirement

1. Used to add users to database
    a. String.format("INSERT INTO patient(name, age, address, billingCard, balance) VALUES('%s', %s, '%s',%s,%s)",request.getParameter("name"),request.getParameter("age"),request.getParameter("address"),request.getParameter("billingCard"),"0");
2. Used to remove users from the database
    a. String.format("DELETE FROM patient WHERE patientID ="+ request.getParameter("patientID"));

4b .Browse Patients

Visual of Functional Requirement

View/Edit Patient

Patient ID	Name	Age	Address	Billing Card	Balance	Pay Balance	Delete
1	Anthony	20	123 Blvd	500	0	Pay	Delete
2	Karmehr	21	321 Steet	1000	0	Pay	Delete
3	Phillip	22	456 Avenue	1500	0	Pay	Delete
4	Derrick	23	789 Lane	2000	0	Pay	Delete
5	Tyler	24	987 Blvd	2500	0	Pay	Delete
6	Matthew	25	123 Steret	3000	0	Pay	Delete
7	Geronimo	26	654 Avenue	3500	0	Pay	Delete
8	Austin	27	987 Lane	4000	0	Pay	Delete
9	Sam	28	789 Blvd	4500	0	Pay	Delete
10	Kean	29	912 Rd	5000	0	Pay	Delete

Screenshot of Browse Patients Functional Requirement

As shown in the image above the user views all the patients through a table which displays all the inputted user information.

How This Functional Requirement Was Implemented

In order to browse through patients, a table was needed in order to list the user. HTML elements such as table and its related elements were used in order to create a table with data. When the page loads it will automatically load the table with all the patients automatically. The table also features a pay and delete button explained in functional requirements 1a. and 1e.

SQL Statements Used In This Functional Requirement

1. gets all the patients for browsing of patients
    a. stmt.executeQuery("SELECT * FROM patient");

5a. Book Appointments

Visual of Functional Requirement

Screenshot of Booking an Appointment

As shown in the image above an appointment can be booked with a simple form.

How This Functional Requirement Was Implemented

This functional requirement was implemented through the use of form. HTML elements used were form, inputs, and buttons. When the user inputs data and presses the button it will insert the data into the appointment database as well as related relationship tables. Application checks if appointments have conflicts at same data/time with rooms, staff, or patient involved.

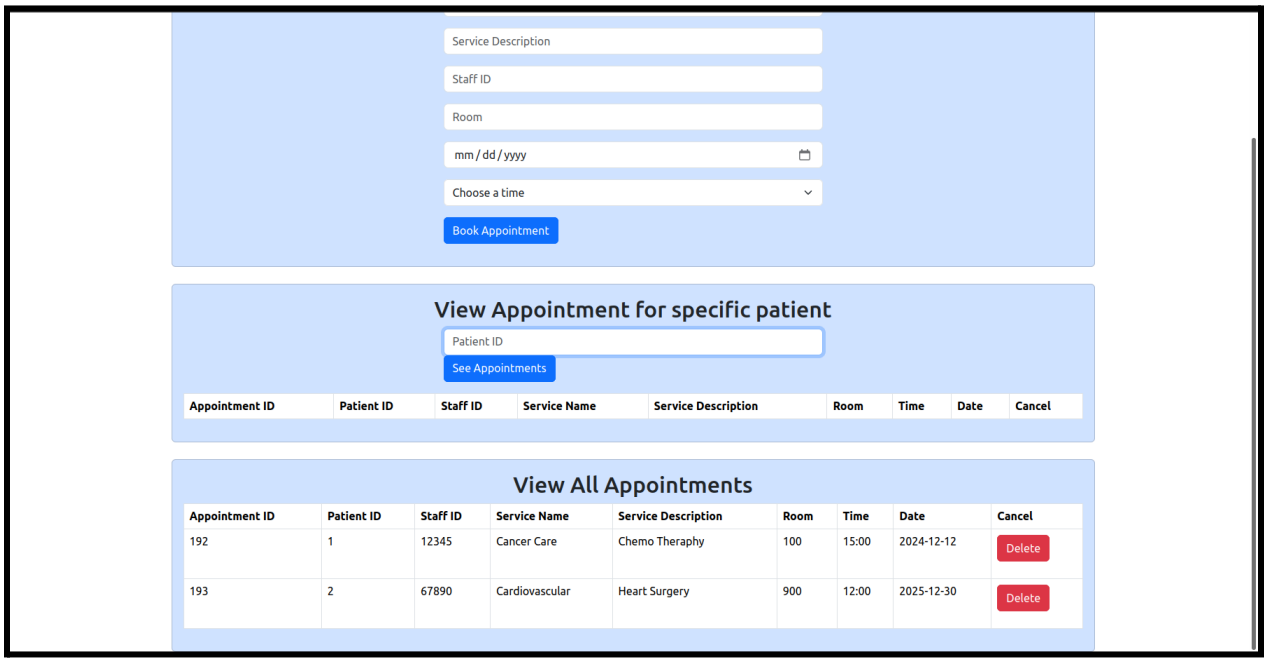
SQL Statements Used In This Functional Requirement

1. checks if there is a room conflict with appointment date and time
    a. stmt1.executeQuery("SELECT * FROM has natural join appointments WHERE roomNumber='" + request.getParameter("room") + "' AND date = '" + request.getParameter("date") + "'AND time = '" + request.getParameter("time") + "'");
2. checks if there is staff conflict with appointment date and time
    a. stmt2.executeQuery("SELECT * FROM attends natural join appointments WHERE staffID='" + request.getParameter("staffID") + "' AND date = '" + request.getParameter("date") + "'AND time = '" + request.getParameter("time")+ "'");
3. checks if there is staff conflict with appointment date and time
    a. stmt3.executeQuery("SELECT * FROM attends natural join appointments WHERE patientID='" + request.getParameter("patientID") + "' AND date = '" + request.getParameter("date") + "'AND time = '" + request.getParameter("time")+ "'");
4. inserts data into appointment table
    a. String sql = String.format("INSERT INTO appointments(date, time) VALUES('%s','%s')",request.getParameter("date"),request.getParameter("time"));

5. gets generated appointment ID
    a. sql = "SELECT appointmentID FROM appointments ORDER BY appointmentID DESC LIMIT 1";
6. inserts appointment into relationship table ‘has’
    a. sql = String.format("INSERT INTO has(appointmentID, roomNumber) VALUES(%s,%s)",appointmentID, request.getParameter("room"));
7. inserts appointment into relationship table ‘consistsof’
    a. sql = String.format("INSERT INTO consistsof(appointmentID, serviceName, serviceDescription) VALUES(%s,'%s','%s')", appointmentID,request.getParameter("serviceName"), request.getParameter("serviceDescription"));
8. inserts appointment into relationship table ‘attends’
    a. sql = String.format("INSERT INTO attends(appointmentID, patientID, staffID) VALUES(%s,%s,%s)",appointmentID, request.getParameter("patientID"),request.getParameter("staffID"));
9. gets the cost from the service
    a. sql = "SELECT cost FROM services WHERE serviceName = '"+request.getParameter("serviceName")+"'";
10. updates the patient’s balance to reflect service’s cost from appointment
    a. sql = "UPDATE patient SET balance = balance +"+ rs4.getInt(1) + " WHERE patientID =" + request.getParameter("patientID");

5b. Browse Appointments

Visual of Functional Requirement



Screenshot of View Appointment or All Appointments

As shown in this image above appointments can be viewed two ways. Through appointments for a specific patient or appointments for all patients.

How This Functional Requirement Was Implemented

This functional requirement was implemented with a form and table. HTML elements were form and table. When a user inputs a patient ID and submits it, it will display the appointments for that patient however if the user wants to view all appointments the second table displays all the appointments.

SQL Statements Used In This Functional Requirement

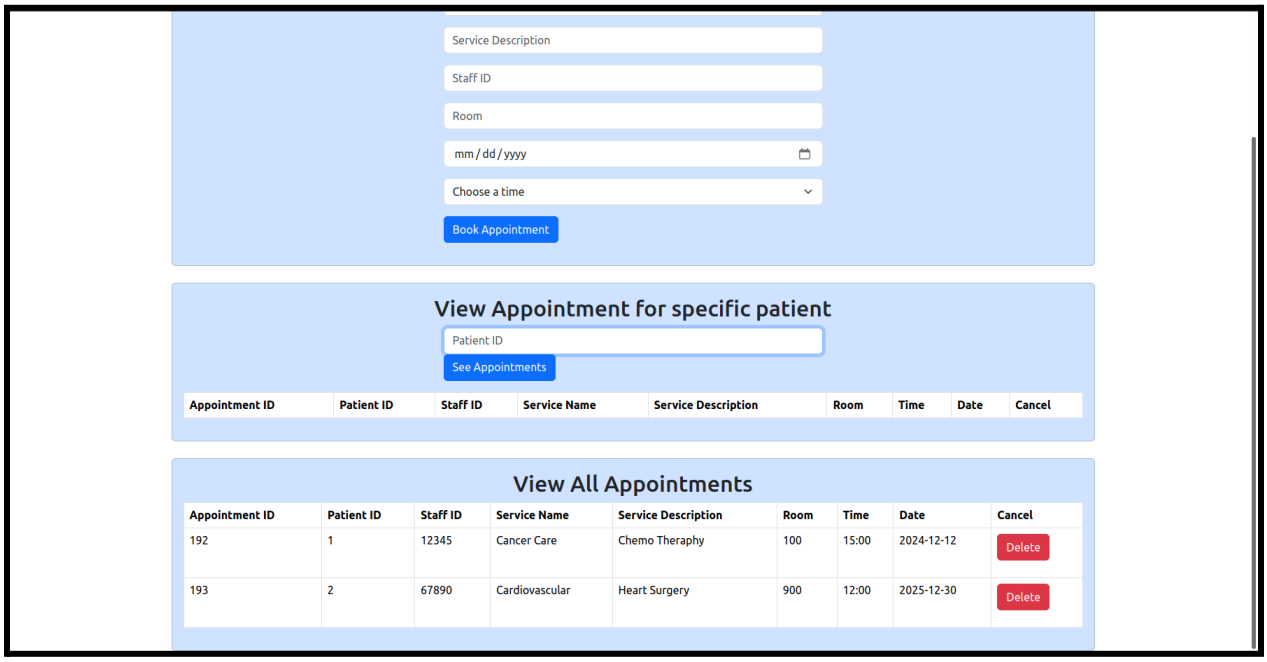
1. Gets appointments for specific patient
    a. stmt1.executeQuery("select appointmentID, patientID, staffID, serviceName, serviceDescription, roomNumber, time, date FROM appointments natural join has

natural join attends natural join consistsof WHERE patientID='" + request.getParameter("patientIDSpecifc")+ "'");

2. Gets related information from relationship tables
    a. stmt2.executeQuery("SELECT patientID, staffID FROM attends WHERE appointmentID=" + rs1.getInt(1));
    b. stmt3.executeQuery("SELECT serviceName, serviceDescription FROM consistsof WHERE appointmentID=" + rs1.getInt(1));
    c. stmt4.executeQuery("SELECT roomNumber FROM has WHERE appointmentID=" + rs1.getInt(1));

5c. Cancel Appointments

Visual of Functional Requirement



Screenshot of Canceling Appointments

As you can see in this image above the user can cancel an appointment by pressing the delete button and view all appointments and also delete an appointment through the specific patient search table.

How This Functional Requirement Was Implemented

This functional requirement was implemented with a button which is in the table mentioned in the explanation for functional requirement 1d. When a user presses the button its associated tuple will be removed from the appointment table and its associated relationship tables through foreign key constraints placed in relationship tables and its method ON DELETE is set to CASCADE meaning it will delete all child rows related to that appointment. It will also decrement its balance in the patient table because when we booked the appointment a cost was added which needs to be removed now.

SQL Statements Used In This Functional Requirement

1. Deletes from appointment table
    a. String.format("DELETE FROM appointments WHERE appointmentID ="+ request.getParameter("appointmentID"));
2. Updates patient balance
    a. sql = "UPDATE patient SET balance = balance -"+ rs5.getInt(1) + " WHERE patientID =" + request.getParameter("patientIDDelete");

5d. Patient Billing and Payment

Visual of Functional Requirement

View/Edit Patient

Patient ID	Name	Age	Address	Billing Card	Balance	Pay Balance	Delete
1	Anthony	20	123 Blvd	500	0	Pay	Delete
2	Karmehr	21	321 Steet	1000	0	Pay	Delete
3	Phillip	22	456 Avenue	1500	0	Pay	Delete
4	Derrick	23	789 Lane	2000	0	Pay	Delete
5	Tyler	24	987 Blvd	2500	0	Pay	Delete
6	Matthew	25	123 Steret	3000	0	Pay	Delete
7	Geronimo	26	654 Avenue	3500	0	Pay	Delete
8	Austin	27	987 Lane	4000	0	Pay	Delete
9	Sam	28	789 Blvd	4500	0	Pay	Delete
10	Kean	29	912 Rd	5000	0	Pay	Delete

Screenshot of Patient Billing and Payment Functional Requirement

As shown in the image above there is a pay button for each column of each patient which updates the patients balance to zero. The payment is not processed through any payment system and it just updates the balance to zero in the database. An assumption is made that the hospital will use a third party payment service so it was deemed that this was sufficient.

How This Functional Requirement Was Implemented

This functional requirement was implemented through the use of a button in a table. For each patient tuple there will be a pay button associated with it in order to update its balance to zero.

SQL Statements Used In This Functional Requirement

1. sets the patients balance to zero simulating a payment
    a. String.format("UPDATE patient SET balance = 0 WHERE patientID ="+ request.getParameter("patientIDPay"));

6a. Display Staff

Visual of Functional Requirement

View/Edit Staff

Staff ID	Name	Age	Job	Salary	
1	Anthony	20	Physician	10000	Delete
2	Karmehr	29	Nurse	100000	Delete
3	Phillip	24	Surgeon	50000	Delete
4	Derrick	22	Pharmacist	30000	Delete
5	Tyler	25	Anesthesiologist	60000	Delete
6	Matthew	26	Physical Therapist	70000	Delete
7	Geronimo	27	Dietitian	80000	Delete
8	Austin	23	Medical Biller	40000	Delete
9	Sam	21	Medical Records Clerk	20000	Delete
10	Kean	28	Hospital Administrator	90000	Delete
11	Robby	36	Doctor	160000	Delete
12	Earl	40	Doctor	140000	Delete
13	Brock	26	Pharmacist	40000	Delete
14	Jalen	54	Physical Therapist	60000	Delete
16	kean	1	a	1	Delete
17	asdasd	1	asdaasd	1	Delete

Screenshot of Staff Table

As you can see in this image above, the user is able to view all of the staff that are registered in the system

How This Functional Requirement Was Implemented

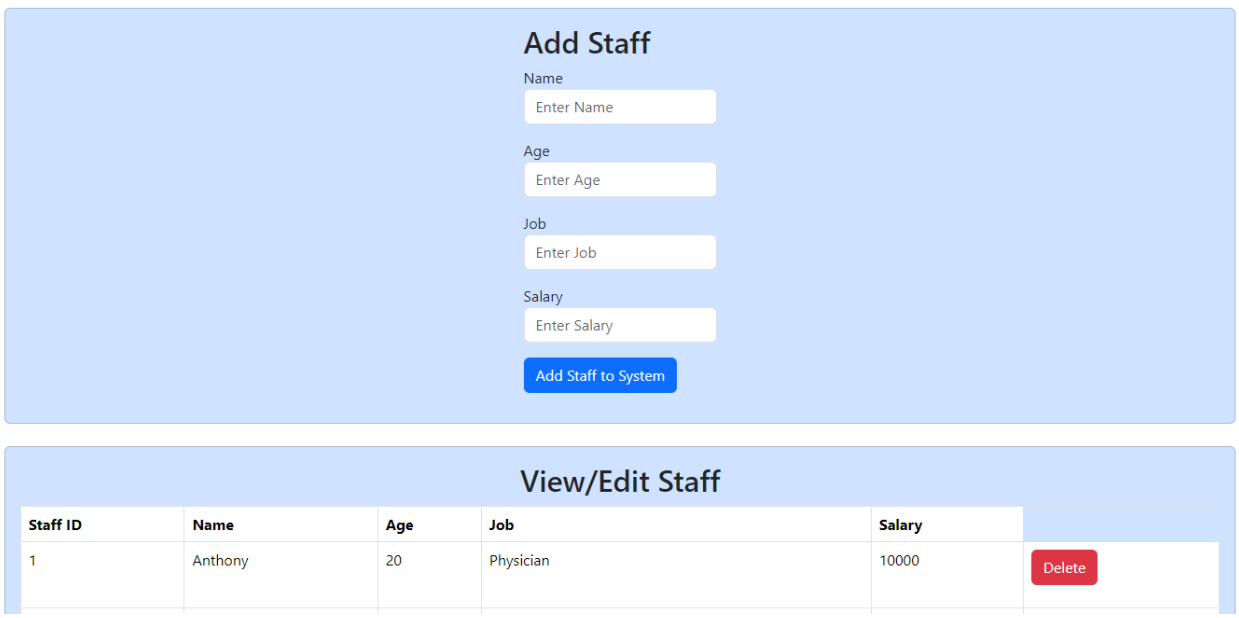
In order to browse through staff, a table was needed in order to list them. HTML elements such as table and its related elements were used in order to create a table with data. When the page loads it will automatically load the table with all the Staff automatically. The table also features a delete button explained in 6b.

SQL Statements Used In This Functional Requirement

2. gets all the staff to display in the table
    a. stmt.executeQuery("SELECT * FROM staff");

6b. Add/Remove Staff

Visual of Functional Requirement



Screenshot of Adding/Removing Staff

As shown in the image above the user can input user details and add a staff to the system with a simple form submission. In order to remove a staff from the system in the image there is a table with all the staff and an option to delete the staff with the red delete button.

How This Functional Requirement Was Implemented

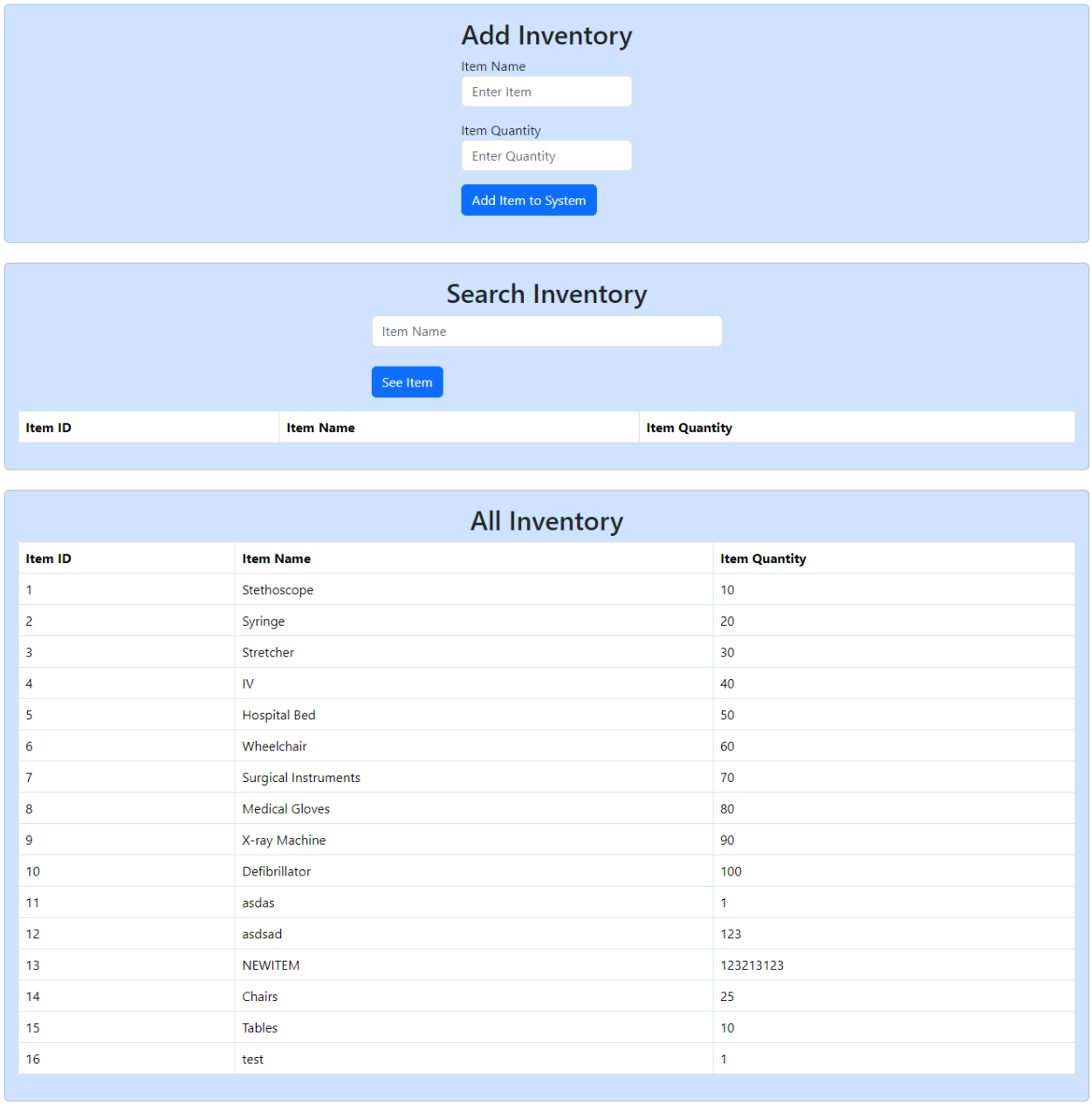
In order to add the staff, user input was first required from the user. HTML elements such as form and inputs were used in order to create a form in which users can input data. When the user presses the add button it will add the staff to the system and subsequently to the database. In order to remove the staff from the application first we had to display the staff as explained in functional requirement 6a. Within the table a button was added with the tag of the id of the staff. This allowed the delete button to be associated with said staff and thus when pressed is removed from the table and subsequently from the database.

SQL Statements Used In This Functional Requirement

3. Used to add staff to database
    a. String.format("INSERT INTO staff (name, age, job, salary) VALUES('%s', %s, '%s',%s)", request.getParameter("name") != null, request.getParameter("age")!= null, request.getParameter("job") != null, request.getParameter("salary") != null);
4. Used to remove staff from the database
    a. String.format("DELETE FROM staff WHERE staffID ="+ request.getParameter("staffID"));

6c. Add/Search/View Inventory

Visual of Functional Requirement



Screenshot of Inventory Features

The image above shows the ability for the user to add an item into inventory via the simple form submission. Once they press “Add Item to System”, it will be registered into the database system and will display under the “All Inventory” table, which is a way for users to view all of the

inventory in the system. The “Search Inventory” form will allow users to search for inventory based on the input they put into the “Item Name” form. If they type ‘s’, all the items that begin with an ‘s’ will show under the “Search Inventory” form in the table provided. The empty table is populated after a valid search is made.

How This Functional Requirement Was Implemented

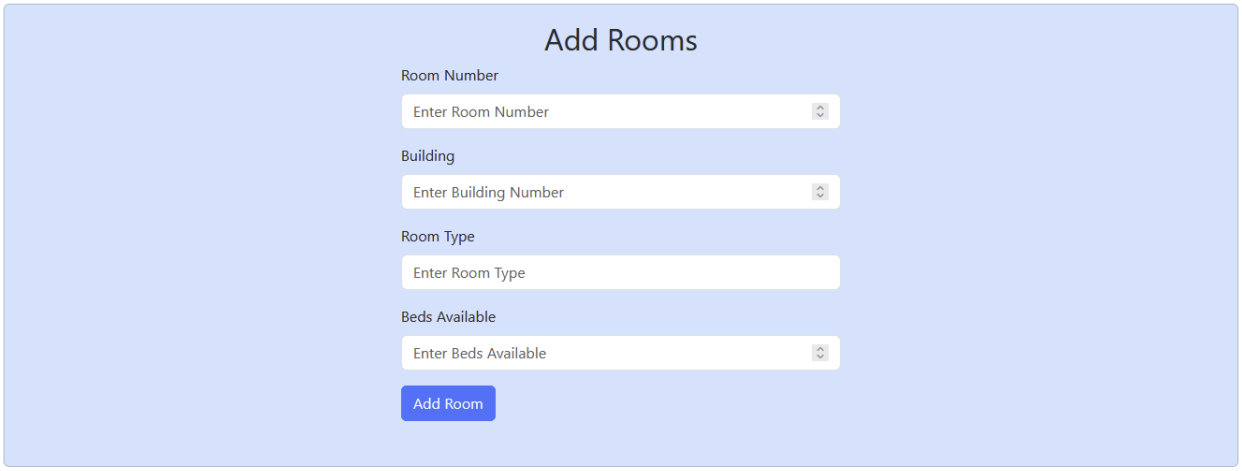
These functional requirements were implemented using a submission form and a submit button to enter the data. For each entry, users can either add an item into the inventory should they have added acceptable inputs, or see the items in inventory given acceptable inputs.

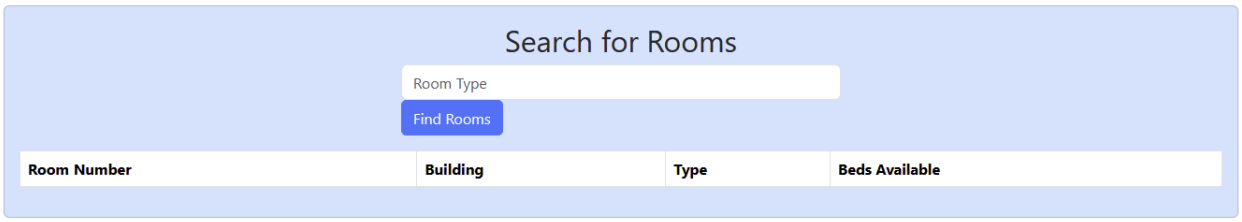
SQL Statements Used In This Functional Requirement

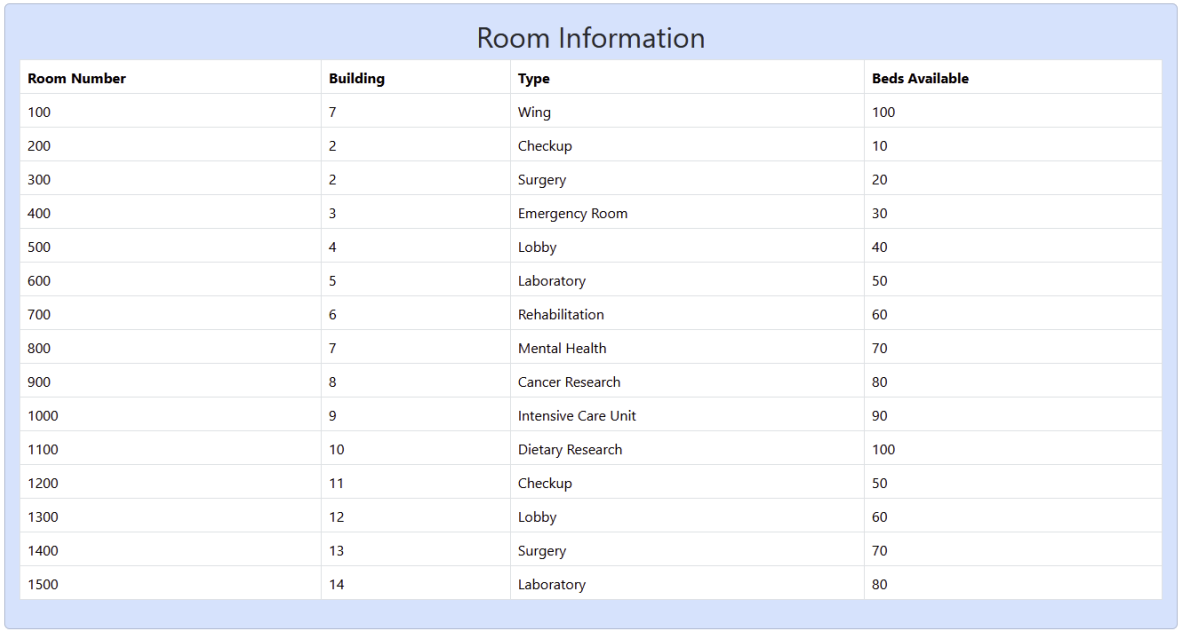
1. Adds to Inventory Table
    a. ("INSERT INTO inventory(name, quantity) VALUES('%s', %s)",request.getParameter("name"),request.getParameter("quantity"));
2. Searches Inventory Table
    a. ("SELECT * FROM inventory WHERE name LIKE '%" + request.getParameter("Item Name") + "%';");
2. Views Inventory Table
    a. ("SELECT * FROM inventory;");

7a. Add/Search/View Rooms

Visual of Functional Requirement







Room Information

Room Number	Building	Type	Beds Available
100	7	Wing	100
200	2	Checkup	10
300	2	Surgery	20
400	3	Emergency Room	30
500	4	Lobby	40
600	5	Laboratory	50
700	6	Rehabilitation	60
800	7	Mental Health	70
900	8	Cancer Research	80
1000	9	Intensive Care Unit	90
1100	10	Dietary Research	100
1200	11	Checkup	50
1300	12	Lobby	60
1400	13	Surgery	70
1500	14	Laboratory	80

Screenshot of Adding, viewing, and searching Rooms

As you can see in this image above the user can add rooms by entering their data and clicking “Add Room”. Users can also search for rooms by entering their search phrase and clicking on “Find Rooms”. Users are always able to view all rooms naturally via this table.

How This Functional Requirement Was Implemented

These functional requirements were implemented using a submission form and a submit button to enter the data. For each entry, users can either add a room should they have added acceptable inputs, or find rooms given acceptable inputs.

SQL Statements Used In This Functional Requirement

3. Adds to Room Table
    a. ("INSERT INTO room VALUES(" + rmNumber + ", " + bd + ", '" + roomType + "', " + ba + ");")
4. Searches Room Table
    a. (SELECT * FROM room WHERE type LIKE ‘% + request.getParameter(“Room Type”) + “%’;”)
5. Views Room Table
    a. SELECT * FROM room;

7b. Add/Search/View Services

Add Services

Service Name

Enter Service Name

Cost

Enter Cost

Service Department

Enter Service Department

Add Service

View Specific Service

Service Name

See Service

Service Name	Cost	Service Department

Services Provided

Service Name	Cost	Service Department
Better Health	1000	Cancer
Cancer Care	8000	Laboratory
Cardiovascular	7000	Cardiology
Caring for Cancer	100000	Cancer Treatment
Diagnostic Imaging	3000	Laboratory
Emergency Medical	1000	Intensive Care Unit
Immunology	2000	Internal Medicine
Laboratory	4000	Laboratory
Maternity and Obstetrics	5000	Obstetrics
Mental Health	10000	Neurology
Oncology	2000	Cancer Treatment
Pediatric	6000	Pediatrics
Preventative Care	100	Pediatrics
Rehabilitation	9000	Cardiology
Surgery	2000	Anesthesia
Surgery	90000	Surgery
Test Service	1000	Cancer Treatment
Therapy	5000	Neurology

Screenshot of Adding, viewing, and searching Services

As you can see in this image above the user can add services by entering their data and clicking “Add Service”. Users can also search for services by entering their search phrase and clicking on “See Service”. Users are always able to view all services naturally via this table.

How This Functional Requirement Was Implemented

These functional requirements were implemented using a submission form and a submit button to enter the data. For each entry, users can either add a service should they have added acceptable inputs, or see services given acceptable inputs.

SQL Statements Used In This Functional Requirement

6. Adds to Services Table
    a. ("INSERT INTO services VALUES('" + serviceName + "', " + cost + ", '" + serviceDepartment + "');")
7. Updates InChargeOf Table
    a. “INSERT INTO InChargeOf VALUES('" + serviceDepartment + "', '" + serviceName + "', '" + serviceDescription + "');”)
8. Searches Services Table
    a. ("Select * from services where serviceName LIKE '%" + request.getParameter("Service Name") + "%';")
9. Views Room Table
    a. SELECT * FROM services;

How to Set Up and Run Application

1. Download Java from http://java.com/
2. Download Apache Tomcat from https://tomcat.apache.org/download-80.cgi
3. Download MySQL from https://www.mysql.com/
4. Download MySQL JDBC Connection and place that jar into “webapps/ROOT” folder of tomcat directory
5. Create the “webapps/ROOT” folder of tomcat directory, for instance we can name is “ClinicMaster”
6. Within that folder in the git terminal run command git clone “https://github.com/karmehr-arora/CS157A-Team1.git”
7. All the application files will show up in the that folder
8. Run the Dump SQL file called “Dump20231210.sql” in MySQL Workbench to get data
9. Make sure the credentials for SQL database in each JSP file is set to your SQL database credentials, by default username is set to root and password is set to root so do not change it if you have root as well.
10. In browser of your choice, go to the homepage which is “signupPage.jsp”
11. From this page you can now interact with the application.

Lesson Learned

Karmehr Arora

During this project, I've had the opportunity to learn a lot of things regarding project/product development. The newest concept to me was the use of the three-tier architecture and splitting up the application into a front-end UI, middleware (which the user interacts with), and Database back-end. It was great to learn how to apply this to our project and fully understand the benefits it presented. For example, not letting users directly manipulate or interact with the data. Also separating the concerns so that the application becomes more maintainable and testable. I also learned how to utilize SQL in a project combined with other languages. SQL utilization within a larger project was something I was confused about in the past, but now I understand how to integrate it into an application. I also felt like I learned about new software such as Tomcat for server usage, and Bootstrap for front-end development. These are tools I haven't taken advantage of before, but now I understand how useful they are.

Kean Onn Lee

I am glad that we were assigned to do this project because I was able to understand the concepts that we learned in class much better as I was able to apply those concepts to the project after I had learned about them. I also learned a lot about web application development and the setup/use of a three tier architecture to assist in developing the website. I had a solid idea of the HTML/CSS/other front-end components of the project, but this project only furthered that knowledge. Also, having to write the SQL statements and interacting with the MySQL database constantly is the most important and challenging part that I had taken from the project. The class and the project also gave me the opportunity to work in a group project environment which gave

me more experience working with others while also using project management resources such as Github. In conclusion, the hands-on project experience was very valuable and I have several valuable takeaways from doing the project.

Geronimo Aldana

I learned many things in the creation of this application, however few lessons stood out. First, I learned how to use new technologies, especially three tier architectures. This will be useful in the future when I start developing applications that interact with a database which is almost all applications. Another thing I learned was how SQL can be used in real world applications. Most database classes teach you the language and where to use it but this project helped in actually implementing it in an application. Finally, the last thing I learned was working in a random group. Most of my classes before allowed us to choose our partners and working in a random group was new to me, however with communication we formed chemistry and synergy throughout the development of this project. Overall, creating this application taught me a lot and I am glad I got to work with Karmehr and Kean in this group project.