HospitalDB

Yan Ling Cheung, Thomas Lerner

Introduction:

Hospitals need tools for recording, managing, and checking appointments. A web-based database can be accessed by staff at any computer in the hospital to quickly and efficiently check a patient's appointments or schedule a new one.

Application Requirements Specifications

HospitalDB will allow users to enter patients into the database using a form designed to efficiently convey the type of information needed. This patient submission form will restrict the user submit data that is compatible with the format accepted by the database.

It will also grant users the capability to search through patients by name and displays relevant information to confirm patient identity, then gives the option to view that patient's appointments or to match them with a caregiver (which will be searched for in in a similar style) in order to schedule an appointment between them.

The appointment scheduling page will allow users to input a desired date, and will by default use today's date. A list of appointment slots will be displayed, and if one is selected an appointment will be automatically scheduled (relevant information has already been gathered and stored, including pID, eID, bID, and desired date and time). Appointment slots that conflict with the database constraints will not be displayed.

Upon patient submission and appointment submission a success splash screen will be displayed to confirm that the inputs were valid.

All pages will have "back" buttons that will direct the user to the page before their current one in the workflow of the website. Success screens will redirect to the index page.

Database Requirements Specifications

We will record the dates and times of appointments, the caregiver(s) assigned to those appointments, and the patients that scheduled them. This information will be used to create a system such that staff cannot be overbooked and overflowing appointments must be queued for later dates.

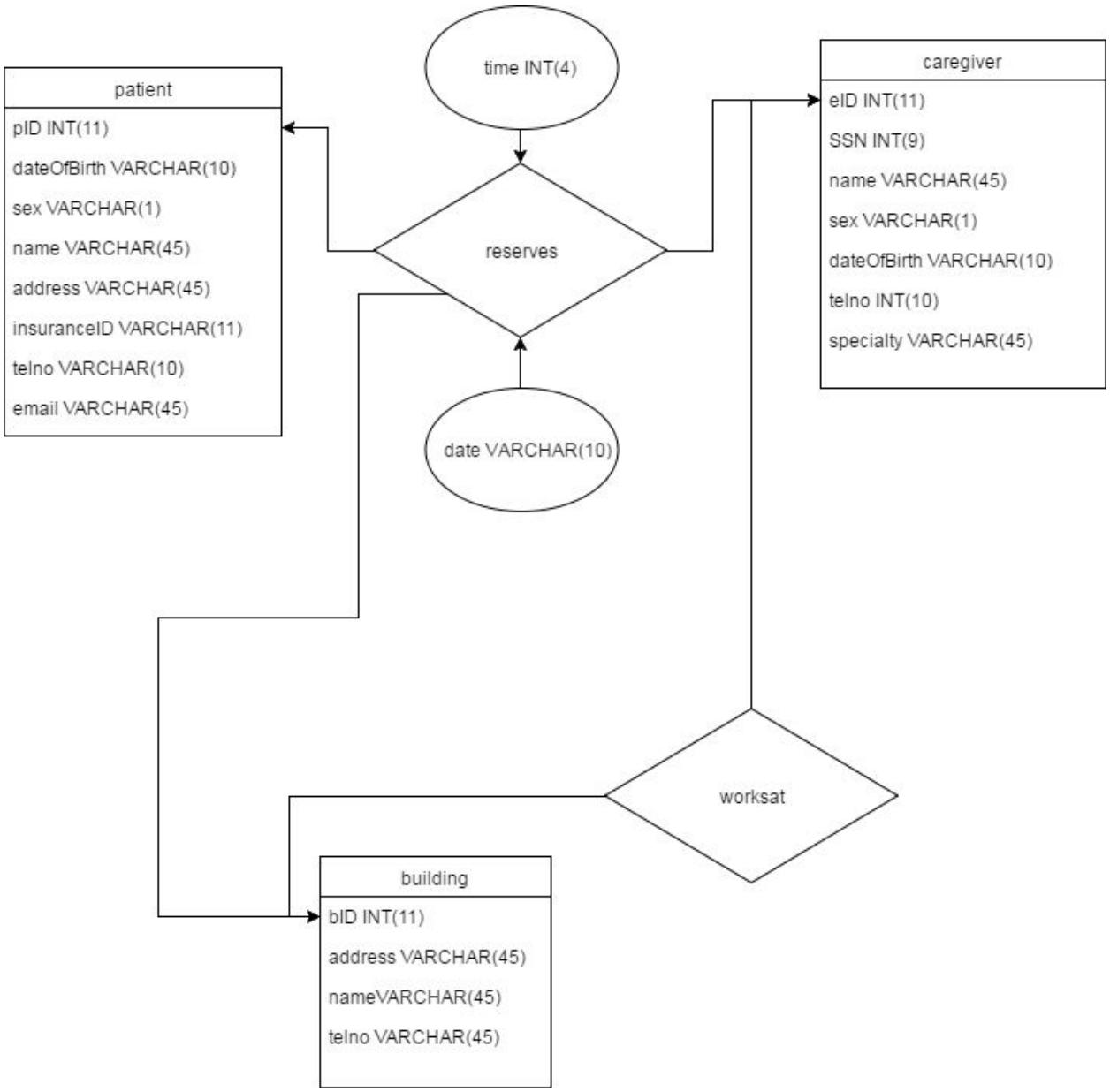
Caregivers, buildings, and worksat data should be added manually to the database, and their IDs should auto-increment. Our application could handle caregiver and building management, but it doesn't seem appropriate for an appointment scheduling tool intended for use by clerks and other non-managerial hospital staff to have that capability. If it were to be implemented, it would be very similar to the patient submission form and success pages, but would have data inputs customized to the data to be entered.

Patients and reserves should be able to be created using intuitive forms and selections, and their data should be sanitized before submission. It should be possible to create patients and reserves quickly and often to facilitate fast triage and scheduling over phones or in person.

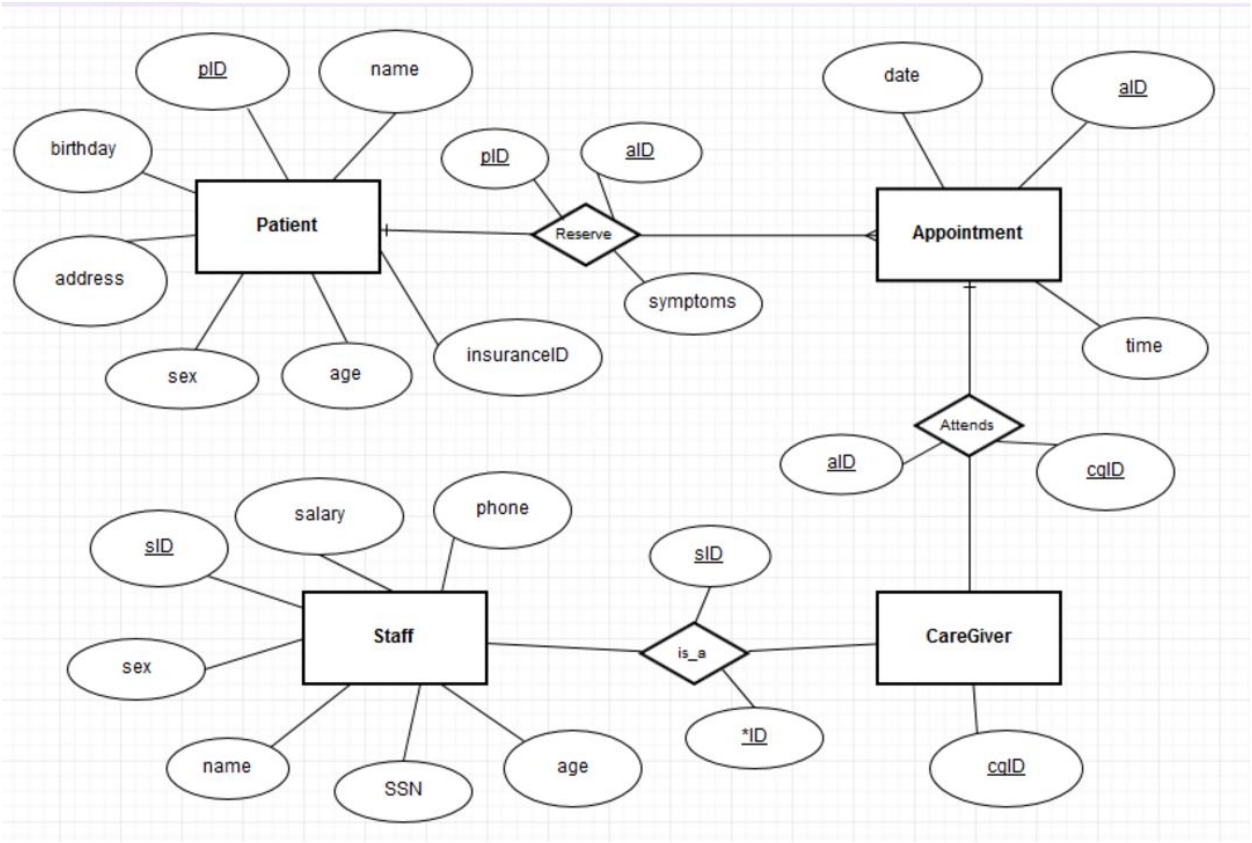
Reserves will have data constraints such that no caregiver can be scheduled for multiple appointments at the same time, and no building can have more than five appointments

scheduled at any one time. These constraints will be enforced by not allowing appointments to be scheduled during timeslots that would violate these conditions It is assumed that all appointment time slots will be one hour long.

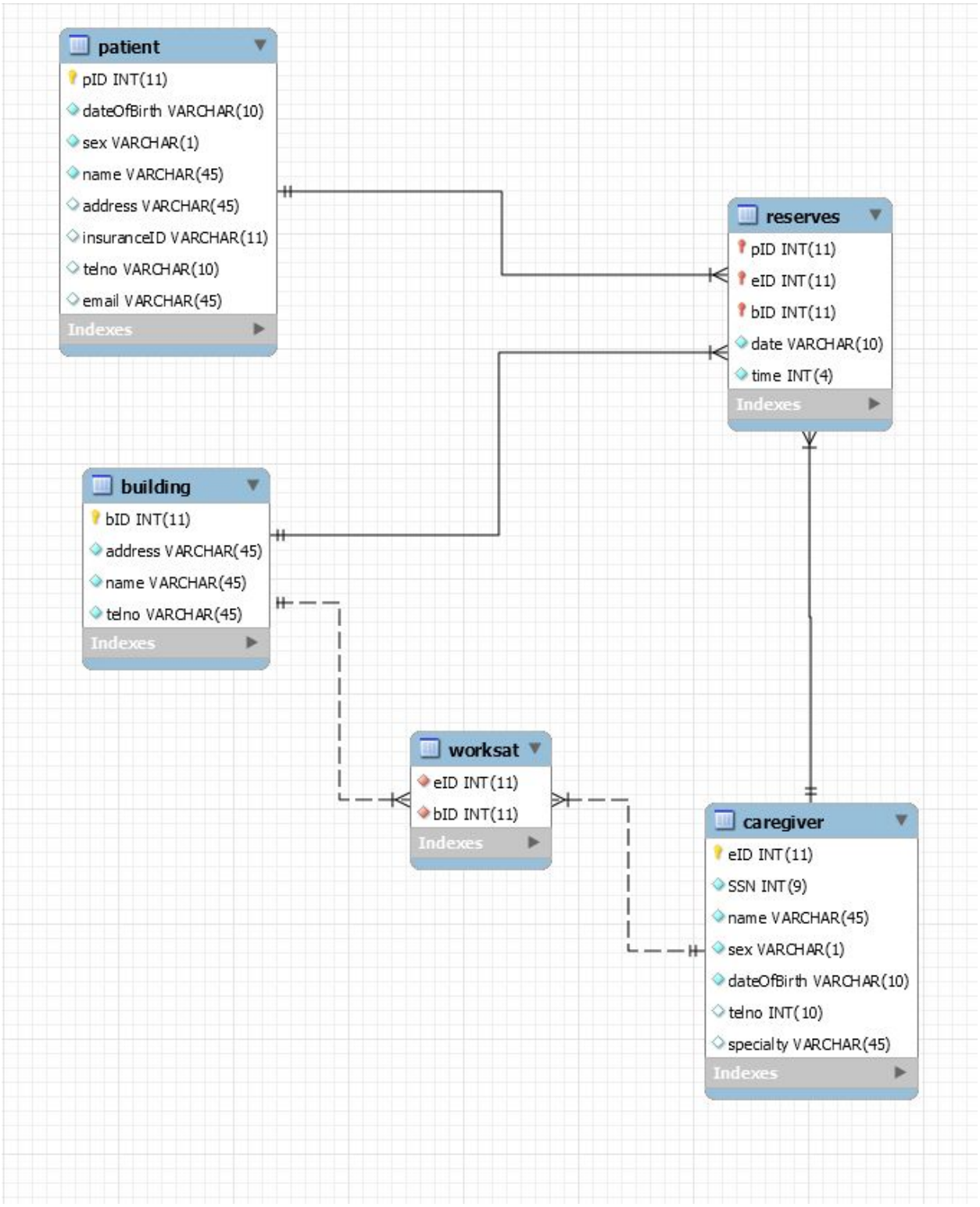
ER Data Diagram and Revisitation:



This is the final ER diagram created for the project. The original ER diagram and database structured was flawed and ineffective. After reworking it, it is much more effective. The old ER diagram is displayed below:


pID
name
birthday
Patient
address
sex
age
insuranceID
pID
aID
Reserve
symptoms
date
aID
Appointment
time
Attends
aID
cgID
salary
phone
sID
Staff
sex
name
SSN
age
sID
is_a
*ID
CareGiver
cgID

Relational Model:



Tables:

- Patient: pID, dateOfBirth, sex, name, address, insuranceID, telno, email
    - pID is auto-incremented
- Reserves: pID, eID, bID, date, time
    - pID is a foreign key referencing Patient
    - eID is a foreign key referencing CareGiver
    - bID is a foreign key referencing Building
- CareGiver: eID, SSN, name, sex, dateOfBirth, telno, specialty
    - eID is auto-incremented
- Worksat: eID, bID
    - eID is a foreign key referencing CareGiver
    - bID is a foreign key referencing Building
- Building: bID, address, name, telno
    - bID is auto-incremented

Legend:

- Underlined: primary key

Functional Dependencies:

- Patient: pID → name, dateOfBirth, sex, address, telno, email, insuranceID
- Reserves: pID, date, time → eID, bID
- CareGiver: eID → SSN, name, sex, dateOfBirth, specialty
- Worksat: eID → bID
- Building: bID → address, name, telno

Constraints**:**

- Primary keys are not null and unique. This is easily managed because primary keys are autogenerated, so input sanitization is not needed.
- Foreign keys reference primary keys in other tables and cascade on deletion (though data removal is currently unimplemented)

Tools and Languages Used:

- SQL (through MySQL Workbench)
- JDBC
- Java (through JSP)
- HTML and Javascript
- Apache Tomcat
- Heroku (during development)

Queries Used in our System:

To see the appointments a patient has scheduled (given inputpID):

```
SELECT c.name, r.date, r.time, b.name
FROM Caregiver c, Building b, Reserves r
INNER JOIN Patient
      ON Patients.pID = r.pID
WHERE Patient.pID = inputpID
      AND c.eID = r.eID
      AND b.bID = r.bID;
```

$\Pi_{C.name,\ R.date,\ R.time,\ B.name}((\rho_B(Building) \bowtie_{B.bID = R.bID} (\rho_C(CareGiver) \bowtie_{C.eID = R.eID} ((\rho_R(Reserves)) \bowtie_{r.pID = p.pID} (\Pi_{P.pID}(\sigma_{P.pID = inputpID}\ \rho_P(Patient)))))))$

Search patients by name (given inputName)

```
SELECT *
FROM patient
WHERE name = inputName;
```

$\{P \mid P \in Patient \wedge P[name] = inputName\}$

$\sigma_{name = inputName}(Patient)$

Search caregivers by name (given inputName)

```
SELECT *
FROM caregiver
WHERE name = inputName;
```

$\{C \mid C \in CareGiver \wedge C[name] = inputName\}$

$\sigma_{name = inputName}(CareGiver)$

Find timeslots where caregiver is busy (given inputeID, inputDate):

```
SELECT r.time
FROM Reserves r
WHERE r.eID = inputeID
      AND r.date = inputDate;
```

$\{t \mid (\exists\ r)(r \in Reserves \wedge t[time] = r[time] \wedge r[eID] = inputeID \wedge r[date] = inputDate\}$

Find timeslots where hospitals are overbooked (given inputeID, inputDate) :

```
SELECT r.time
FROM Reserves r
INNER JOIN Worksat
      ON Worksat.bID = r.bID;
WHERE Worksat.eID = inputeID
      AND r.date = inputDate
GROUP BY r.time
HAVING COUNT(r.time) >= 5;
```

Find unavailable timeslots (given inputeID, inputDate):

```
SELECT r.time
FROM Reserves r
WHERE r.eID = inputeID
      AND r.date = inputDate
UNION
      SELECT r.time
      FROM Reserves r
      INNER JOIN Worksat
            ON Worksat.bID = r.bID
      WHERE Worksat.eID = inputeID
            AND r.date = inputDate
GROUP BY r.time
HAVING COUNT(r.time) >= 5;
```

Conclusions:

The process of learning JSP and implementing it was very difficult. In the future however it will be much easier to create database projects of this kind because the information gained by doing this project has been extremely valuable. In the end, this project was very heavy for a two-person team to handle and I regret that the end User Interface is somewhat barebones because of it.

Special thanks go to Professor Ozsoyoglu for help with our ER diagram and database structure.

Appendix 1. Installation Manual:

- Install Apache Tomcat 6.0
- Copy the JSP files in the “Webapp JSP Files” folder into your “Tomcat 6.0 > webapps > ROOT” folder
- Using any Database Management Software, open the Schema.sql file
- Create new entries for caregivers, buildings, and worksat OR run the included InserPeople and InsertData files (WARNING: Since the IDs of the caregivers, buildings, and patients are autogenerated, it is likely that the InsertData file will have incorrect IDs

specified for its relationships. You may need to search for the new IDs using your DBMS and change InsertData appropriately)
- Run the database and Tomcat, then visit localhost:8080 or wherever you specified your Tomcat server to run on your browser

Appendix 2. Users Manual:

- After caregivers, buildings, and the worksat relation have been created by hospital managerial staff, you can use this tool to create patient entries and schedule and view appointments.
- Start on the application homepage. If the patient you are interested in does not already exist in the database, click “Create Patient”.
- From there, fill out the following form with relevant information. Fields with an (*) next to them are required.
- When finished, click “Create Patient” again. If you were successful, you should see a success screen. Congratulations! Now click “Back” to return to the home page.
- With your patient in the database, click “Find Patient” from the home page.
- From there, search for your patient’s name or some part of it, or leave the search blank to browse all patients. Click “Search Patients” and look for an entry matching your patient.
- Now you can view the patient’s existing appointments by clicking “Show Appointments”, or match them with a caregiver to make a new appointment.
- If you are creating a new appointment, click “Find Caregiver”.
- From there, use the caregiver search like you used the patient search to find the desired caregiver and click “Schedule”.
- Now specify the desired date (today’s date is automatically selected) and create an appointment in one of the available slots using the “Create Appointment” button.

Appendix 3. Programmers Manual:

- The database includes three tables for which data must be given through manual submission via your DBMS. These are caregiver, building, and worksat.
- The pID, eID, and bID of patients, caregivers, and buildings respectively are auto-incremented, so adding worksat relations will require searching for and using the generated keys.
- The worksat table is used to link caregivers and the buildings they work at, while reserves links patients, caregivers, and buildings.
- The reserves table is used to store appointment data.
- The default username for the database is “root”, and the default password is “password”.