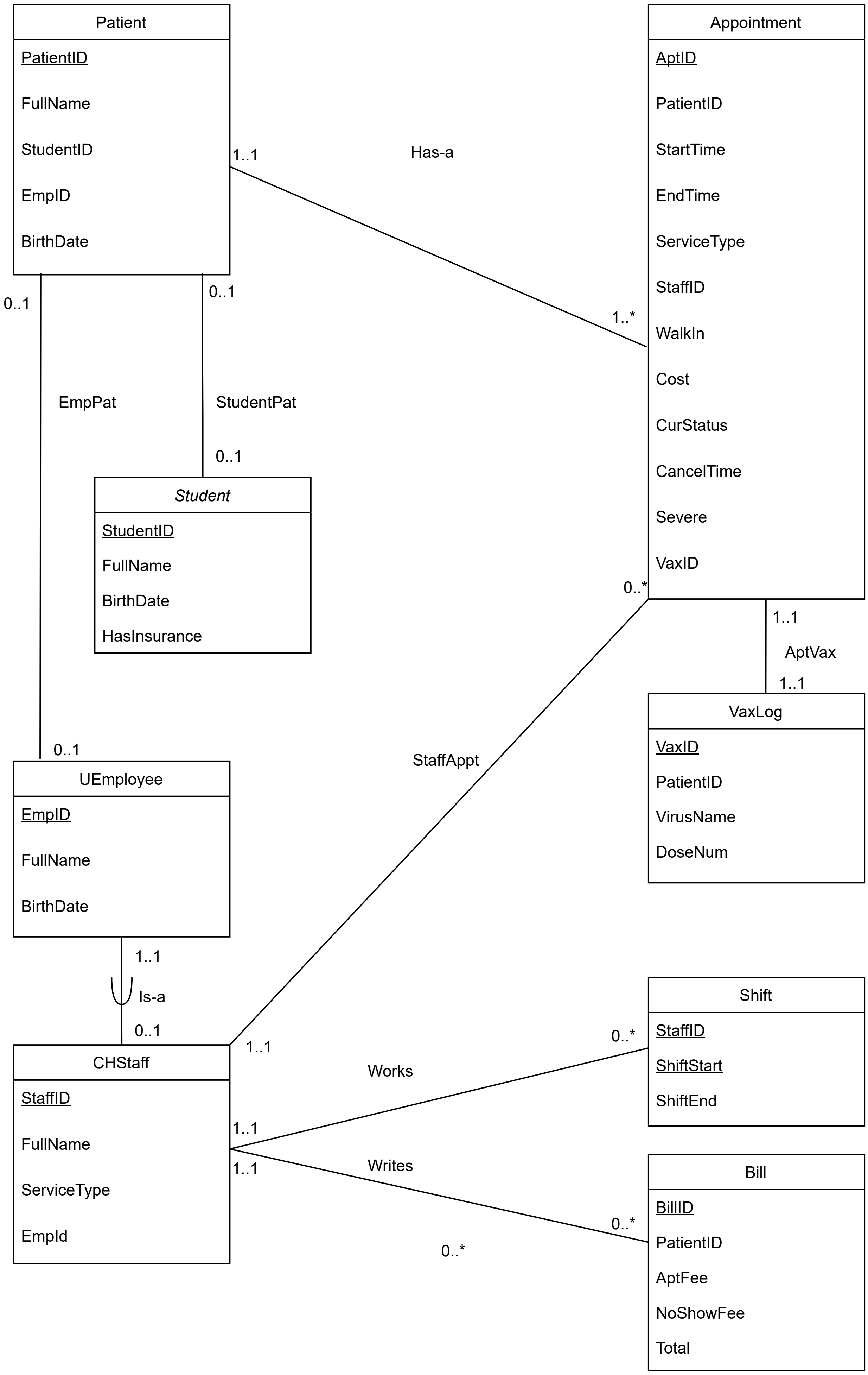
CSC460, Dr. McCann, Haris Riaz, Aayush Pinto, Spring 2022, 5/2 12:30p
Andrew Logan (adlogan), Trong Nguyen (trongnguyen),
Dane Norville (danenorville), Anh Nguyen Phung (anhnguyenphung)

Conceptual Database Design E-E-R Diagram



Design rationale / high-level text description (conceptual database design:

In order to encapsulate all required functionality determined by the assignment description, required functionalities, and queries, we created 8 entities: Student, UEmployee, Patient, CHStaff, Appointment, VaxLog, Shift, and Bill.

Student: We created Student to represent a University of Arizona student, who is identified by their StudentID. The HasInsurance field in Student is constrained to having values of 'Yes' or 'No'. If the value is 'Yes', a Student will be billed at the student discounted rate. If 'No', they are billed at the regular rate. The StudentID cannot be updated by a user.

UEmployee: We created UEmployee to represent any University of Arizona employee (including Campus health staff). They are identified by their EmpID. The EmpID cannot be updated by a user.

Patient: We created Patient to represent a Campus Health Patient (can be a Student, Employee, or both). Since a Patient can either be a Student, Employee, or both, we allowed StudentID and EmpID to be able to take on NULL values, and our Java program prevents both values from being NULL simultaneously. StudentID is a FK to Student and EmpID is a FK to Employee. The PatientID cannot be updated by a user.

CHStaff: We created CHStaff to represent a Campus Health employee. CHStaff is a subset of UEmployee (IS-A) and is identified by their StaffID. Their ServiceType is constrained to 4 values: ['General Medicine', 'CAPS', 'Laboratory and Testing', 'Immunization'] to represent the 4 services that Campus Health offers. EmpId is a FK to UEmployee. The StaffID cannot be updated by a user.

Appointment: We created Appointment to represent an appointment at Campus Health, which is identified by the AptID. We allowed CancelTime, Severe, and VaxID to be NULL because a Patient may not cancel, we assume if Severe is NULL the appointment is not severe, and the appointment may not be for a vaccine, respectively. PatientID is the Patient going in for the appointment from StartTime to EndTime. ServiceType is constrained to the same values as ServiceType in CHStaff. Walkin is constrained to 'Yes' or 'No', Severe is constrained to 'Yes' 'No', or NULL, and CurStatus is constrained to ['Complete', 'Incomplete', 'Cancelled', 'No Show']. An appointment will be scheduled initially as incomplete, and will be updated to 'Complete' if the appointment was successfully completed, 'Cancelled' if the appointment was cancelled more than an hour in advance, or 'No show' if the appointment was not cancelled more than an hour in advance or the patient did not show up to the appointment. PatientID is a FK to Patient, StaffID is a FK to CHStaff, and VaxID is a FK to VaxLog. The AptID cannot be updated by a user.

VaxLog: We created VaxLog to represent the log of all vaccines appointments that have been created (including those that have been completed, are incomplete, were cancelled, or for ones that patients didn't show up for). This status can be found using the PK-FK connection of VaxID in Appointment to get the CurStatus. PatientID is a FK to Patient which can be used to gain access to the birthdate of a Patient (which is useful for determining vaccine eligibility). VirusName is the name of the virus being vaccinated against, and DoseNum is the number of the vaccine dose in the case of a series of vaccines (e.g COVID-19 has 4 doses).

Shift: We created Shift to represent the shift of a campus health staff (e.g. 9 - 5). A shift is identified by the StaffID and ShiftStart.

Bill: We created Bill to represent a bill charged to a patient after their appointment. We assumed that all patients have a Bursar's account, but we knew much less about how the Bursar's office actually charges account. So given their patientID, we can find the employeeID or studentID to charge the correct account. PatientID is a foreign key connection the Patient relation. When a Campus Health staff is manually entering a bill, they can calculate the ApptFee given the cost of the appointment and the insurance plan from the Patient relation. If a patient does not show up, there is an additional $25 no-show fee. The total is the sum of the ApptFee and NoShowFee.

Logical Database Design Relational Schema

Patient				
PatientID	FullName	StudentID	EmpID	BirthDate

CHStaff			
StaffID	FullName	ServiceType	EmpId

Student			
StudentID	FullName	BirthDate	HasInsurance

UEmployee		
EmpID	FullName	BirthDate

Appointment											
AptID	PatientID	StartTime	EndTime	ServiceType	StaffID	WalkIn	Cost	CurStatus	CancelTime	Severe	VaxID

VaxLog			
VaxID	PatientID	VirusName	DoseNum

Shift		
StaffID	ShiftStart	ShiftEnd

Bill				
BillID	PatientID	AptFee	NoShowFee	Total

Normalization Analysis

Candidate Keys:

Patient: {PatientID}, {StudentID, EmpID}
CHStaff: {StaffID}, {EmpID}
Student: {StudentID}
Employee: {EmpID}
Appointment: {AptID}, {PatientID, StartTime}, {PatientID, EndTime}, {StaffID, StartTime}, {StaffID, EndTime},
VaxLog: {VaxID}
Shift: {StaffID, ShiftStart}, {StaffID, ShiftEnd}
Bill: {BillID}

Primary Key Functional Dependencies:

Patient: PatientID → PatientID, FullName, EmpID, StudentID, BirthDate
CHStaff: StaffID → StaffID, FullName, ServiceType, EmpID
Student: StudentID → StudentID, FullName, BirthDate, HasInsurance
Employee: EmpID → EmpID, FullName, BirthDate
Appointment: AptID → Apt ID, PatientID, StartTime, EndTime, ServiceType, StaffID, WalkIn, Cost, CurStatus, CancelTime, Severe, VaxID
VaxLog: VaxID → VaxID, PatientID, VirusName, DoseNum, CurStatus
Shift: StaffID, ShiftStart → StaffID, ShiftStart, ShiftEnd
Bill: BillID → BillID, PatientID, AptFee, NoShowFee, Total

Normalization Level: Boyce-Codd Normal Form

In our analysis, we were able to find more candidate keys in addition to our created primary keys. Most of these candidate keys are composite keys. We elected to create primary keys for simplicity in our queries. For all candidate keys listed above, there obviously exists a functional dependency where the candidate key functionally determines all of the attributes in the relation. We found that our relational schema achieves Boyce-Codd Normal Form. That is, for every functional dependency of the form X → A, X is a superkey (X contains a candidate key). We found no functional dependencies X → A where A was not the entire set of attributes of the given relation. See the example below of our reasoning:

> In the appointment relation, we discovered that because our JDBC functionality enforces that no patient can book overlapping appointments, this produced a few new composite candidate keys with the patient and staff and the start and end times. Consider the key {PatientID, StartTime}, because this composite key is a candidate key, it functionally determines all of the other attributes in appointment. This preserves the requirements for BCNF.

Query 5 Description

Our fifth query prompts the user to input a COVID-19 dose number (1, 2, 3, or 4) and returns a list of all the patients who have completed that dose.

Specifically, this query answers the question: "Which patients have completed COVID-19 dose number X at UA Campus Health?"

This query is useful for looking into COVID-19 vaccination trend statistics in the Tucson community. With this query, Campus Health staff can see which and how many UA students and employees have completed a given dose, which gives valuable insight into things like:

- UA campus' herd immunity
- UA campus' vaccination posture
- Immunization agenda for subsequent doses 2 and 4