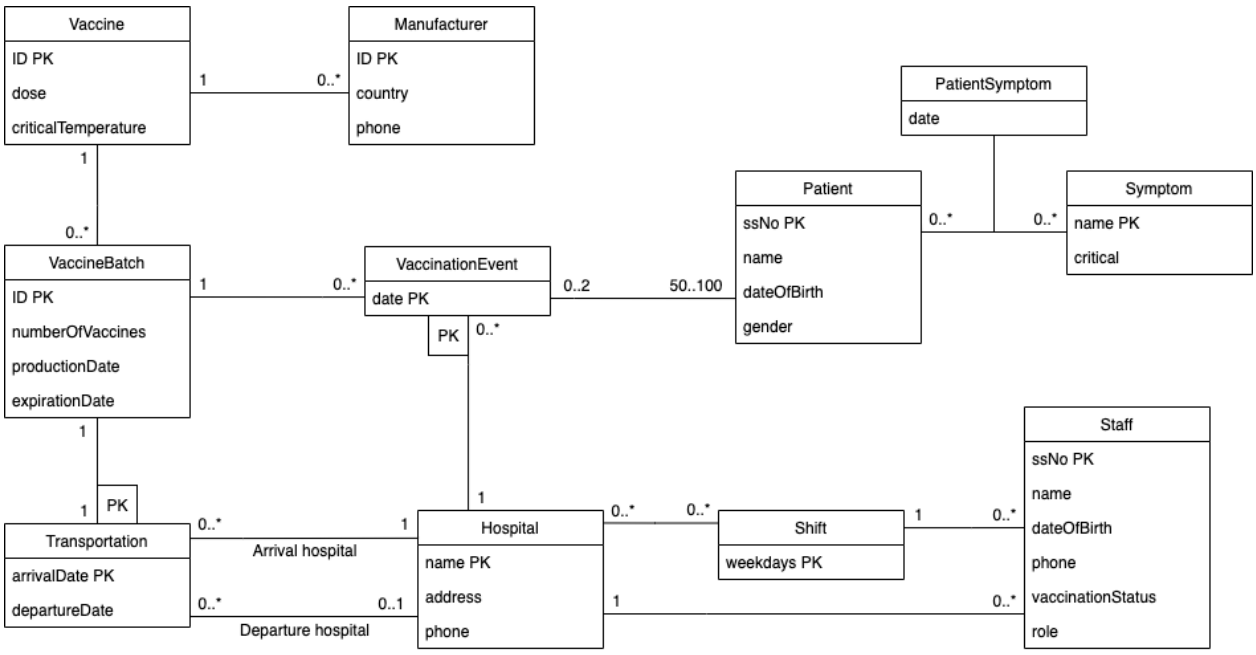
CS-A1153 - Databases group project
Vaccine distribution

Group 15
Linh Ngo 906502
Rajat Kaul 890906
GuTing Huang 885791
Pouya Adibnezhad 818409
Anna Valldeoriola Cardó 885869

UML diagram

The UML diagram of the preliminary design of the database is as follows:



A vaccine can be manufactured by multiple manufacturers, however each manufacturer makes only one type of vaccine hence the one-many association between Vaccine and Manufacturer classes. The multiplicity is 0..* on the Manufacturer side to signify that a vaccine can be added to the database before a manufacturer for it is saved since the id of the vaccine is a foreign key in the Manufacturer relation.

There is a one-many association between Vaccine and VaccineBatch classes since a batch contains only one type of vaccine and a specific type of vaccine can be used in multiple batches. Here on the VaccineBatch multiplicity is 0..* rather than 1..* to allow the data of a vaccine being added to the database before it is assigned to a batch.

It is assumed that it is always possible to transport a vaccine batch as a whole on the arrival date so there is a one-one association between VaccineBatch and Transportation classes. Here the arrival date in the Transportation class is not enough to define a transportation tuple uniquely so the primary key of the VaccineBatch which is the id of the batch is borrowed. A transportation tuple always has an arrival hospital so there is a many-one association between Transportation and Hospital classes to signify that, however as at first a batch is delivered to a hospital from the manufacturer, and not another hospital, the multiplicity on the departure hospital association is 0..1 on the hospital side. On the many sides of both associations the multiplicity is 0..* and a hospital is not enforced to be an arrival or departure destination so that the transportation log can be populated gradually during the vaccination program.

Since there is no difference between a clinic and a hospital in this application, both are modeled using the Hospital class. As a requirement a hospital has its own weekly vaccination shifts for the staff and each staff has exactly one shift. The association between Hospital and Shift is a many-many one to allow a hospital to have different shifts for its staff and to allow different hospitals to have overlapping shifts. Since a staff member has exactly one shift the association between Staff and Shift is one-many and a staff member is assumed to always have a shift assigned.
Since a hospital has at most one vaccination shift plan for each weekday storing a shift as a string containing the days of the week is enough. In the implementation it should also be enforced that a staff is always assigned one of the shifts of their own hospital. Shifts are stored in a separate relation, as opposed to being added to the Staff relation to avoid the loss of data in case of removing staff, and also that the retrieval of shifts of a hospital is not dependent on having a staff member assigned to that shift.
Moreover as a hospital has multiple staff and a staff works at exactly one hospital there is a one-many association between Staff and Hospital classes. The multiplicity 0..* on the Staff side is to allow entering the data of a hospital to the database before assigning the staff.

To record the vaccination events the class VaccinationEvent is added with the event date and the name of the hospital as its primary key to comply with the requirement of each hospital having at most one vaccination event on a day. The association between VaccinationEvent and Hospital is a one-many association to emphasize that each vaccination event happens in one hospital, and a hospital can arrange any number of vaccination events as long as they are on different days. There is a similar association between VaccinationEvent and VaccineBatch since as a requirement only one vaccine batch can be used in a vaccination event but each batch can be used in multiple vaccination events. Again 0..* multiplicities are used for the ease of data entry.
Since 50-100 patients will get vaccinated on each vaccination event, the multiplicity of the association between Patient and VaccinationEvent is 50..100 on the Patient side. It is assumed

that the list of the patients to be vaccinated in each event is prepared beforehand so it is enforced that the information of the patients exists in the database before a vaccination event is created and at least 50 of the patients are added to the event when it is created.
On the other hand depending on the type of the vaccine a patient attends 1 or 2 vaccination events hence the multiplicity 0..2 on the VaccinationEvent side. In the implementation it should be enforced that in case a patient needs to attend a second vaccination event, the type of the vaccine used in that event is the same as the first event they have been to.
The information on the vaccination status of a patient is not stored on the Patient class since that information can be retrieved by knowing the number of events a patient has been to from the relation between Patient and VaccinationEvent classes, the type of vaccine used in those events from the association between VaccinationEvent and VaccineBatch and the number of required doses of that vaccine from the Vaccine relation.
Similarly the number of vaccines used in a vaccination event is not stored since that information can be easily retrieved from the number of patients vaccinated in an event.

Lastly, the information on the criticality of symptoms is stored in the Symptoms relation with the name as the primary key, and there is a many-many association between that class and Patient to record the data of the symptoms observed in a patient. The date of symptoms is stored as part of the PatientSymptom association class. An association class is used here since it is assumed that recording the latest date a symptom is observed in a patient is enough and the date of the symptom is not required to determine the primary key.

Relational data model

The following are the schemas of the relations in the database, with key attributes specified by underline:

Manufacturer(ID, country, phone, vaccineID)
Vaccine(ID, dose, criticalTemperature)
VaccineBatch(ID, vaccineID, numberOfVaccines, productionDate, expirationDate)
Transportation(batchID, arrivalDate, departureDate, arrivalHospital, departureHospital)
Hospital(name, address, phone)
Staff(ssNo, name, dateOfBirth, phone, vaccinationStatus, role, hospital, shift)
Shift(weekdays)
HospitalShift(hospital, weekdays)
VaccinationEvent(hospital, date, batchID)
Patient(ssNo, name, dateOfBirth, gender)
PatientVaccinationEvent(ssNo, eventDate, eventHospital)
Symptom(name, critical)
PatientSymptom(ssNo, symptomName, date)

Functional dependencies

This section contains functional dependencies in each relation listed in the previous section. In deriving the dependencies it is assumed that phone numbers and addresses are unique.

In the Manufacturer relation:
ID → country, phone, vaccineID
phone → ID, country, vaccineID

In the Vaccine relation:
ID → dose, criticalTemperature

In the VaccineBatch relation:
ID → vaccineID, numberOfVaccines, productionDate, expirationDate

In the Transportation relation:
batchID, arrivalDate → departureDate, arrivalHospital, departureHospital

In the Hospital relation:
name → address, phone
address → name, phone
phone → name, address

In the Staff relation:
ssNo → name, dateOfBirth, phone, vaccinationStatus, role, hospitalName, shift
phone → ssNo, name, dateOfBirth, vaccinationStatus, role, hospitalName, shift

In the VaccinationEvent relation:
hospital, date → batchID

In the Patient relation:
ssNo → name, dateOfBirth, gender

In the Symptom relation:
name → critical

In the PatientSymptom relation:
ssNo, symptomName → date

In the Shift, HospitalShift, and PatientVaccinationEvent relations there are no non-trivial functional dependencies.

Boyce-Codd Normal Form

In all the functional dependencies listed in the previous section the left hand side attributes are super keys since it is possible to determine every other attribute in the tuple by knowing them which means none of the functional dependencies violate BCNF so all the relations are in Boyce-Codd Normal Form and no further decomposition is required.

Anomalies and redundancies

Since all the relations are in BCNF then there is no redundancy in the database, also there is no delete and update anomaly.

There are however a few insert anomalies in the database. In the Manufacturer and VaccineBatch relations, id of the vaccine is a foreign key so a vaccine is required to be stored in the database before the information of its manufacturers or a vaccine batch of its type is added.
In the Transportation relation id of the vaccine batch forms the primary key with the arrival date so the batch is required to exist in the database before a transportation record is created. Also arrival hospital and departure hospital are foreign keys in the relation which cause insert anomaly. Similarly in the VaccinationEvent relation hospital is required to form the primary key and the id of the batch is a foreign key.
In the Staff relation, shift and hospital are foreign keys and are required to exist before a staff tuple is created.
In HospitalShift, PatientVaccinationEvent, and PatientSymptom relations the primary key is formed by combining primary keys of other relations which also introduce insert anomaly.