2. Develop a logical data model based on the following requirements: (11/17/22)
    a. Derive relations from the conceptual model.

We have the following relationships and their cardinalities from the conceptual model (some have been added/modified based on feedback):

staff (mandatory) performs **examination** (optional) = 1:*
staff (mandatory) manages **clinic** (optional) = 1:1
clinic (mandatory) registers a **pet** (optional) = 1:*
owner (mandatory) owns **pet** (mandatory) = 1:*
clinic (mandatory) employs **staff** (optional) = 1:*
pet (mandatory) receives an **examination** (optional) = 1:*

These relationships were constructed using the following assumptions:

- Some staff members may perform many examinations and some may perform none at all (ex. receptionist at the front desk will likely not perform any examinations, whereas one veteranarian may perform multiple).
- Some staff members may manage no clinics, while others may manage one, but no staff members will manage multiple clinics (this makes sense since not every member of staff is a manager).
- Some clinics may have no pets registered (maybe business is slow), while others may have multiple.
- All owners must own at least one pet, but they can also own multiple. Pets have just one owner, not multiple.
- Some clinics might have no employees (maybe they have shut down temporarily and will rehire in the future), but all employees must be employed at exactly one clinic, not zero or multiple.
- Some pets will have zero examinations (this is the case if their owner registered them but has not brought them to the clinic yet), and other pets may have many examinations.

Using the relationship type and participation constraints for each relationship, we can derive the relations below. The "staff manages clinic" relationship was 1:1, meaning the clinic is the parent because it is optional while the staff is mandatory. All the other relationships were 1:*, so for these the "many" becomes the parent. In each relationship, once we have identified the parent we make its primary key the foreign key of the child. Using this process, we derived the following relations (DOB is now a single attribute since SQL has a date format):

Staff(staffNo, fName, lName, street, city, zip, telephoneNo, DOB, position, salary, clinicNo) **Primary Key** staffNo **Foreign Key** clinicNo **references** Clinic(clinicNo)
Examination(examNo, complaint, description, date, actions, staffNo, petNo) **Primary Key** examNo **Foreign Key** staffNo **references** Staff(staffNo) **Foreign Key** petNo **references** Pet(petNo)
Clinic(clinicNo, name, street, city, zip, telephoneNo) **Primary Key** clinicNo
Pet(petNo, fName, lName, DOB, species, breed, color, clinicNo, ownerNo) **Primary Key** petNo **Foreign Key** clinicNo **references** Clinic(clinicNo) **Foreign Key** ownerNo **references** Owner(ownerNo)
Owner(ownerNo, fName, lName, street, city, zip, telephoneNo) **Primary Key** ownerNo

b. Validate the logical model using normalization to 3NF.

UNF --> 1NF

Assuming we choose correct data types for each attribute, for example zip, telephoneNo, salary, and all of the primary keys should have a numeric data type such as INT, our DBMS shouldn't allow us to insert multiple values into an attribute for the same record (also all of our attributes will have unique names, and the order of our records does not matter). As long as we don't do anything like inserting "Thomas Nick Alex" into the fName column (which will be a VARCHAR), then we should already be in 1NF. We don't have any repeating groups, and our primary keys prevent repeated records.

1NF --> 2NF

For 2NF, we have to remove partial dependendies. This means that every non-primary key attribute should be fully functionally dependent on the entire primary key, which in our case is not a multi-attribute primary key but simply a single number such as staffNo, clinicNo, etc. Therefore, we can easily verify 2NF is true. Since, for example, Staff has primary key staffNo, we know that the fName, lName, street, city, zip, telephoneNo, DOB, position, salary, and clinicNo are all dependent on the specific staff member in question, which is uniquely identified by staffNo. In other words, we know that no matter which attribute in a table we go to, the information is going to be relevant to the primary key (ex. the attribute DOB refers to the date of birth of a specific staff member identified by primary key staffNo). Therefore, we are in 2NF since we don't have any attributes that are not dependent on the primary key.

2NF --> 3NF

For 3NF, we have to make sure we don't have any transitive dependencies. This would be the case if we had one attribute in a table which is dependent on another attribute, which is then in turn dependent on the primary key.

I was going to say the city attribute is a transitive dependency, since it is dependent on zip, but there are some cases where one zip code refers to multiple cities (for example 94608 is used in both Emeryville, CA and parts of Oakland, CA). I was also going to say that telephoneNo is dependent on fName and lName, but since there are landlines you could have one telephone number that refers to multiple people. Another consideration was that a staff member's position determines their salary, but salary is determined by other things as well such as number of years of employment, so you could have two employees with the same position but different salaries, so it is best to keep salary in the Staff table.

However, we can see that a clinic's name is dependent on more than its clinicNo. For example, given a telephoneNo, zip, street, or possibly city, we can determine the name of the clinic (assuming each clinic has a unique name). We can solve this transitive dependency by splitting this information into multiple tables. The Clinic table will become:

Clinic(clinicNo, name)

The name attribute is now a foreign key into ClinicLocation, which will hold location information:

ClinicLocation(name, street, city, zip, telephoneNo)

This introduces a new relationship:

clinic (mandatory) has a **ClinicLocation** (mandatory) = 1:1

One zip may not refer to just one city, but a city can only have one zip. So zip is transitively dependent on city. We can solve this with a Zip table that has city as a primary key, and remove zip from every table that has a zip attribute:

Zip(city, zip)

So, with our addition of the ClinicLocation and Zip tables, our relations become:

Staff(staffNo, fName, lName, street, city, telephoneNo, DOB, position, salary, clinicNo) **Primary Key** staffNo **Foreign Key** clinicNo **references** Clinic(clinicNo) **Foreign Key** city **references** Zip(city)
Examination(examNo, complaint, description, date, actions, staffNo, petNo) **Primary Key** examNo **Foreign Key** staffNo **references** Staff(staffNo) **Foreign Key** petNo **references** Pet(petNo)
Clinic(clinicNo, name) **Primary Key** clinicNo **Foreign Key** name **references** ClinicLocation(name)
ClinicLocation(name, street, city, telephoneNo) **Primary Key** name **Foreign Key** city **references** Zip(city)
Pet(petNo, fName, lName, DOB, species, breed, color, clinicNo, ownerNo) **Primary Key** petNo **Foreign Key** clinicNo **references** Clinic(clinicNo) **Foreign Key** ownerNo **references** Owner(ownerNo)
Owner(ownerNo, fName, lName, street, city, telephoneNo) **Primary Key** ownerNo **Foreign Key** city **references** Zip(city)

Zip(city, zip) **Primary Key** city

c. Validate the logical model against user transactions.

Thanks to our foreign keys, any transactions we need should be supported. For example, if we want to find all staff members that work at the clinic in Miami, we can select * from Staff where clinicNo in (select clinicNo from Clinic where name in (select name from ClinicLocation where city = 'Miami')). This demonstrates that the way we have structured our database is conducive to the queries we might want to make.

d. Define integrity constraints:
   i. Primary key constraints.
   ii. Referential integrity/Foreign key constraints.
   iii. Alternate key constraints (if any).
   iv. Required data.
   v. Attribute domain constraints.
   vi. General constraints (if any).

i. All primary keys must be unique, and cannot be NULL. This includes staffNo for Staff, examNo for Examination, clinicNo for Clinic, name for ClinicLocation, petNo for Pet, ownerNo for Owner, and city for Zip.

ii. References for foreign keys must exist. If we delete a record in a parent table, we can set it to NULL in any child tables or we can delete the record in the child table as well. If we update a parent record, this update should be reflected in all child tables.

If we delete a clinic, we'll just set it to NULL in the Staff and Pet tables since we could assign these staff and pets to another clinic. We will do the same if we delete a record from the zip table for some reason.

If we delete a staff member, we don't want to delete all the examinations they did since we may need to refer to that information later, so we will set NULL. Same thing for if we delete a pet, just in case the owner comes back one day and needs medical records.

If we delete a ClinicLocation, we will also delete the clinic.

If we delete an Owner, we will also delete all their pets.

Staff(staffNo, fName, lName, street, city, telephoneNo, DOB, position, salary, clinicNo) **Primary Key** staffNo **Foreign Key** clinicNo **references** Clinic(clinicNo) ON UPDATE CASCADE ON DELETE SET NULL **Foreign Key** city **references** Zip(city) ON UPDATE CASCADE ON DELETE SET NULL
Examination(examNo, complaint, description, date, actions, staffNo, petNo) **Primary Key** examNo **Foreign Key** staffNo **references** Staff(staffNo) ON UPDATE CASCADE ON DELETE SET NULL **Foreign Key** petNo **references** Pet(petNo) ON UPDATE CASCADE ON DELETE SET NULL
Clinic(clinicNo, name) **Primary Key** clinicNo **Foreign Key** name **references** ClinicLocation(name) ON UPDATE CASCADE ON DELETE CASCADE
ClinicLocation(name, street, city, telephoneNo) **Primary Key** name **Foreign Key** city **references** Zip(city) ON UPDATE CASCADE ON DELETE SET NULL
Pet(petNo, fName, lName, DOB, species, breed, color, clinicNo, ownerNo) **Primary Key** petNo **Foreign Key** clinicNo **references** Clinic(clinicNo) ON UPDATE CASCADE ON DELETE SET NULL **Foreign Key** ownerNo **references** Owner(ownerNo) ON UPDATE CASCADE ON DELETE CASCADE
Owner(ownerNo, fName, lName, street, city, telephoneNo) **Primary Key** ownerNo **Foreign Key** city **references** Zip(city) ON UPDATE CASCADE ON DELETE SET NULL
Zip(city, zip) **Primary Key** city

iii. N/A

iv. N/A

v. Pets, staff, and owners will have a fName and lName no matter what, so we will set these to NOT NULL. Staff should also have a DOB that is NOT NULL. Examinations should always have a NOT NULL complaint, description, date, staffNo, and petNo. The attribute for actions could be NULL if no actions were taken. In Clinic, the name cannot be NULL, and in ClinicLocation the street and city cannot be NULL either. Pets should have a NOT NULL species, but maybe breed and color is not relevant to that species so these can be NULL in this situation (we don't care about a turtle's breed or color).

If someone doesn't have their own address or telephoneNo, we can allow these to be NULL in the cases of owners and staff, but a ClinicLocation should always have an address.

All strings such as fName, lName, street, city, breed, color, species, and name will be VARCHAR(255), DOB and date will have type DATE, telephoneNo and salary will be INTs. In case they are lengthy, we can use LONGTEXT for complaint, description, and actions.

vi. N/A

e. Generate the E-R diagram for the logical level (contains FKs as attributes).