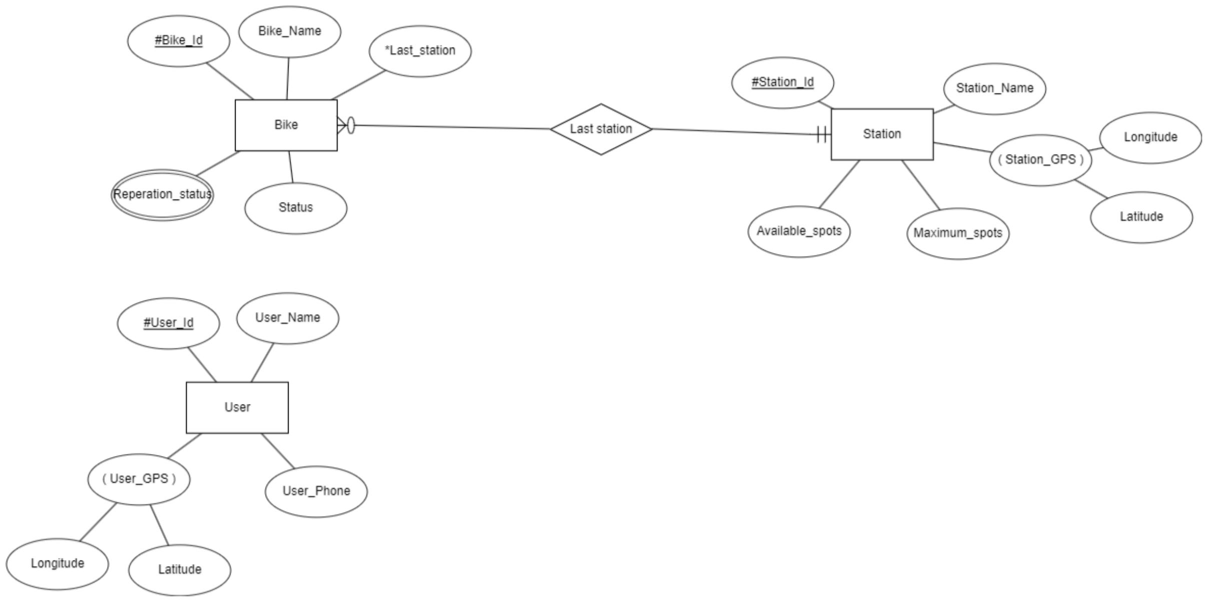
Assignment 1

1a.



1b.

No. There is only a relation between Bike and Station. User has no relations to other entities.

2a.

The solution the junior developer proposes is insufficient for many reasons. Firstly introducing the subscription id attribute. To be able to handle multiple subscriptions per user, like specified in the task, the attribute would need to be multivariable. This would also force status, start, end, type to be multivariable and we need to keep track of indexing, and update the right indexes when starting or ending a subscription. That could become messy over time, and cost extra computation.

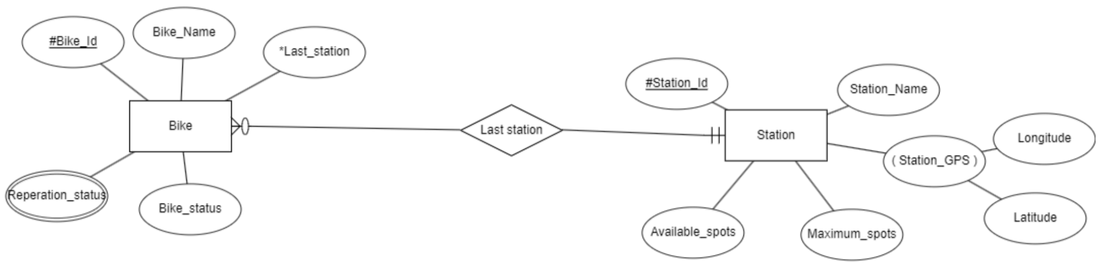
If the user does not have a subscription then it would be null.

Otherwise subscription id would become a primary key candidate, because we would have to repeat user id multiple times. Hence user id is no longer a primary key. This would alter the entity to be more subscription focused, instead of its primary goal to be user focused. This would introduce an unwanted dependency, because we would find information about the user from subscription id. For example, if the user changes phone number, then we could have different phone numbers with the same user id. In short we would be dependent on subscription id, instead of user id.

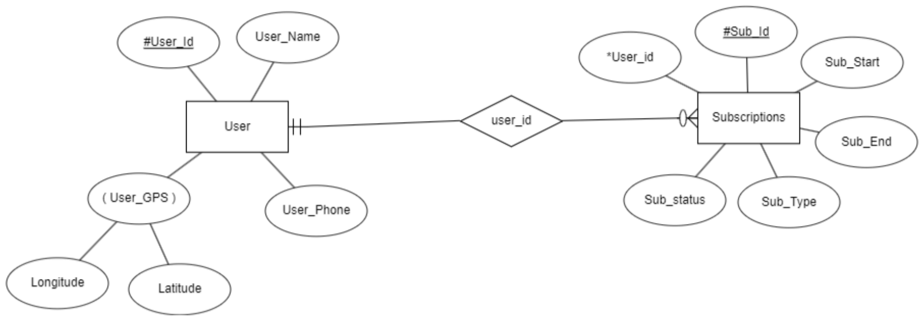
2b.

Subscriptions(#Id, *User_id, Status, Start, End, Type)
CREATE TABLE Subscriptions(Id PRIMARY KEY, User_Id, Status, Start, End, Type, FOREIGN KEY(User_Id) REFERENCES User(Id));

2c.



#Bike_Id
Bike_Name
*Last_station
Bike
Reperation_status
Bike_status
Last station
#Station_Id
Station_Name
Station
Longitude
(Station_GPS)
Latitude
Available_spots
Maximum_spots



#User_Id
User_Name
User
(User_GPS)
User_Phone
Longitude
Latitude
user_id
*User_id
#Sub_Id
Sub_Start
Subscriptions
Sub_End
Sub_status
Sub_Type

3a.

“A Trip has to keep track of the user, the bike, the start time, the end time, the start station and the end station”

The attributes are user_id, bike_id, start_time, end_time, start_station and end station.

You could create a candidate key from the attributes, which would work as a primary key. To identify candidate keys we need to see which combinations would be a key, with its sub-combinations not being candidate keys. No attribute in itself is a primary key. Because they could all be repeated several times. Lets try to identify candidate keys.

User_id and bike_id together alone would not suffice, because the same user could use the same bike.

Adding start_station and end_station would not work either, because the same route could be repeated on the same bike by a user.

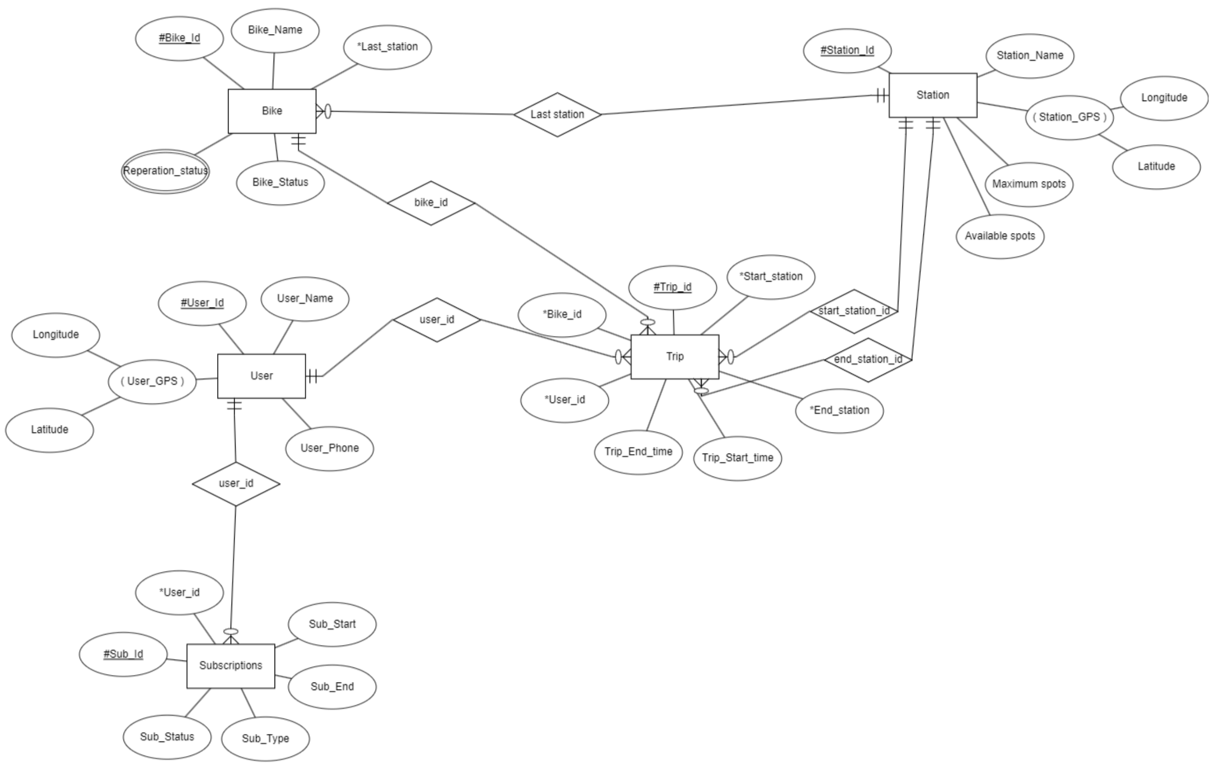
Adding start_time and/or end_time could work, but has some requirements. We know that one bike can not be used by more than one user during a trip. The trip is an interval between start_time and end_time. If the time measurement is fairly accurate and includes a date. If date is excluded then the user could repeat the same route every day. If the measurement is not accurate you could start and end multiple rides on the same bike during one minute.

Thus if the format for start_time and end_time is accurate and includes date. Then one of the two candidate keys (bike_id, start_time) , (bike_id, end_time) could be picked as a primary key. Other combinations building on said keys would not be candidate keys, because they are redundant.

The safest bet is to create a unique new attribute trip_id to be the primary key. User_id is a foreign key to User, and bike_id is a foreign key to Bike. Start_station and End_station are foreign keys to the station.
So the Trip entity could be;

Trip(#*Bike_id, #start_time, end_time, *user_id, *start_station, *end_station)
Or
Trip(#Trip_id, *Bike_id, start_time, end_time, *user_id, *start_station, *end_station)
By the way the task is formulated im leaning towards option 1, since it is specified to identify the attributes from the description. Hence I am unsure if creating an attribute that is not in the description is not what the task intended. Also that 3b asks to specify the primary keys, in plural, so a composite primary key seems reasonable. This is all built on the assumptions made earlier, that the candidate keys are indeed valid. Otherwise Trip_id would be the solution.

3b.



3c.

Normalization secures the database against inconsistencies and often makes them more efficient. The normalization is done sequentially in the stages 1NF, 2NF, 3NF, BCNF, etc.

In 1NF every attribute in a relation has to have a single value, not a combination. All columns must have atomic values of the same datatype, and the order of rows does not matter. So each row must be unique, and defined by a primary key.

In 2NF every attribute has to be dependent on the whole primary key, and 1NF. Else the attribute is not suitable for the entity. For example if an attribute is only dependent on one part of the key, it is not 2NF, and should be moved.

In 3NF every non-key attribute should depend on only the primary key, and not on another attribute. Also be 2NF. For example date of birth and age, where age is dependent on date of birth.

BCNF is a stronger version of 3NF, where every attribute in a table should only depend on the primary key. If the primary key consists of two certain attributes, and an attribute is only dependent on one specific attribute of the primary key. Then the attribute should be moved to a table where the specific attribute from the primary key is the primary key.

3d.

We must start by making all tables in the first normal form.

1NF

- User(#Id, Name, GPS, Phone)
    - Needs to have a unique key
        - Id contains unique values
    - All columns must be atomic of the same datatype.
        - All are atomic and of the same data type except GPS. Composite attributes are not atomic.
        - We can fix this by splitting GPS to Longitude and Latitude as separate attributes in the table.
    - <u>User(#Id, Name, Longitude, Latitude, Phone)</u>
        - Alternatively we could create a new table for gps location.
            - User(#Id, Name, *Gps_id, Phone)
            - Gps(#Gps_Id, Longitude, Latitude)
            - But it would not be necessary, and could cause worse performance with more joint statements.
- Bike(#Id, Name, *Last_station, Reparation status, status)
    - Needs to have a unique key
        - Id contains unique values
    - All columns must be atomic of the same data type without repeating groups
        - All columns except Reperation_status are atomic. Reperation_status is a multivariable attribute. We could not solve it like GPS, and split it to flat_tire, breaks_not_working, etc as attributes. That would be repeating groups. Hence we split it to new tables.
            - Reparations which contain the different types of reparations, with a unique reparation Id.
            - Bike_reperations which has a bike id and a reparation id. Both bike_id and reperation_id are foreign keys from bike and reparations. And together they form a composite primary key.
    - <u>Bike(#Id, Name, *Last_station, status)</u>
    - <u>reparations(#reperation_id, reperation_name)</u>

    - Bike_reperations(#*bike_id, #*reperation_id)
- Subscriptions(#Id, *User_id, Status, Start, End, Type)
    - Has a primary key Id
    - All columns are atomic
- Station(#Id, Name, GPS, Maximum spots, Available spots)
    - Has a primary key Id
    - All columns except GPS are atomic. Same solution as for User.
    - Station(#Id, Name, Longitude, Latitude, Maximum spots, Available spots)
- Trip(#*Bike_id, #start_time, end_time, *user_id, *start_station, *end_station)
    - Has a primary key. Composite key of Bike_id and start_time
    - All columns are atomic.
- Trip(#Trip_id, *Bike_id, start_time, end_time, *user_id, *start_station, *end_station)
    - Has a primary key, Trip_id
    - All columns are atomic

2NF

- User(#Id, Name, Longitude, Latitude, Phone)
    - All attributes are dependent on the primary key Id.
- Bike(#Id, Name, *Last_station, status)
    - All attributes are dependent on primary key
- reparations(#reperation_id, reperation_name)
    - All attributes are dependent on primary key
- Bike_reperations(#*bike_id, #*reperation_id)
    - There are no other attributes than the primary key, so all attributes are dependent on the primary key
- Subscriptions(#Id, *User_id, Status, Start, End, Type)
    - All attributes are dependent on the primary key.
- Station(#Id, Name, Longitude, Latitude, Maximum spots, Available spots)
    - All attributes are dependent on the primary key.
- Trip(#Trip_id, *Bike_id, start_time, end_time, *user_id, *start_station, *end_station)
    - All attributes are dependent on the primary key

3NF

- User(#Id, Name, Longitude, Latitude, Phone)

    - All attributes are dependent on only the primary key Id.
        - People could have the same name, same longitude, same latitude, and same phone. No dependencies between non-key attributes.
        - Longitude or Latitude cannot give latitude or name or Phone. Because multiple users can be at the same place. For example very close to each other or in the same position in different stories of the same building.
        - We know that multiple users can have the same phone number, and same name.
- Bike(#Id, Name, *Last_station, status)
    - All attributes are dependent on only the primary key.
        - We know that Names can be repeated, so they may be not unique. Hence it cannot determine the other attributes.
        - Multiple bikes can have the same last station, and same status.
- reparations(#reperation_id, reperation_name)
    - All attributes are dependent on only the primary key.
    - Reperation_name can only be dependent on reperation_id, since it is the only other attribute, and the primary key.
- Bike_reperations(#*bike_id, #*reperation_id)
    - There are no other attributes than the primary key, so all attributes are dependent on the primary key only.
- **Subscriptions(#Id, *User_id, Status, Start, *Type_id)**
    - Subscription_Types(#Type_id, duration, price)
    - Optional Subscription_EndDates(#*Subscription_Id, End)
        - Where the end is computed from a query. And dependent on only the subscription.
        - Easy to add more attributes to the different types of subscription like price.
            - Is 3NF as long as duration is not unique, so that it determines price. For example student versions that last as long, but are cheaper. Is also 3NF without price.
    - Not all attributes are dependent on only the primary key.
    - The same user can have multiple subscriptions of different types and dates
    - There could be many subscriptions of the same type, so it does not alone define anything else.

    - Many subscriptions could start and end at the same time. We can determine the type based on start and end. This is because the types of subscriptions are time based. Type 1: A month, Type 2: A year, Type 3: A day. We could also determine end with start and type. So {start, end} -> {type} , {start, type} -> {end}. This is not okay in 3NF.
    -
- Station(#Id, Name, Longitude, Latitude, Maximum spots, Available spots)
    - We cannot determine Available spots from Maximum spots. If we for example had another attribute called Occupied spots, then with Maximum and Occupied we could determine Available. But there is no way to determine the Available sports, because we do not have any information in the table about occupied spots. We could create a query with bikes to check the last station and status, and then do computation to determine Available spots, but this would not make Available spots dependent on Maximum spots in this table.
    - It is not described if the names of the stations are unique.
        - Two places could have the same street name. The possibility in Bergen is low, but we have to prove the table against anomalies. I have checked on the municipality's official site of streets, and found two places with the same name.
            - https://www.bergenbyarkiv.no/bergenbyleksikon/arkiv/category/geografi-og-steder/gater-og-veier
            - https://www.bergenbyarkiv.no/bergenbyleksikon/arkiv/1420182
            - https://www.bergenbyarkiv.no/bergenbyleksikon/arkiv/1420181
            - If we were to add these two places as stations, and we only added the name without specification of it being a street or place. Then we would have different stations with the same name. And the information for the stations would only be dependent on the station_id.
        - If all names are unique, then there would be a transitive dependency. Where we could get attributes from the Station name. Can solve this by creating a new table with the names. Station_names(#*Station_id, Name)
    - We don’t know if stations can have the same longitude and latitude or not.

        - If two can have the same, then it would not be transitive. For example if one station is located directly above another station. Like if one is on top a bridge, and the other is under said bridge.
        - But if no station can have the same location (longitude, latitude), then there would be a transitive dependency for the other attributes. Creating a new table Station_location(#*Station_id, Longitude, Latitude), would solve the unwanted dependency.
    - If Name and (Longitude, Latitude) are not unique, the table would remain.
    - Else it would be split to for example.
        - Station_spots(#Station_id, Maximum spots, Available spots)
        - Station_names(#*Station_id, Name)
        - Station_location(#*Station_id, Longitude, Latitude)
- Trip(#Trip_id, *Bike_id, start_time, end_time, *user_id, *start_station, *end_station)
    - All attributes are dependent only on the primary key
    - Multiple of the same bikes, users, start_stations, end_stations, start_time, end_time could be in the table. There are no transitive dependencies.

BCNF

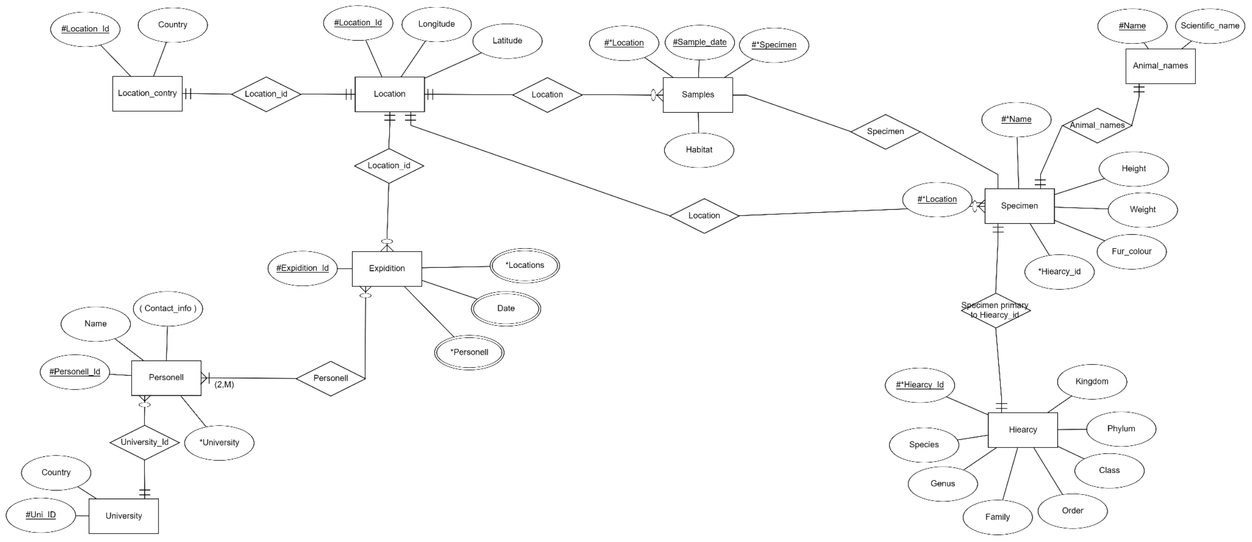
All the tables are already BCNF, because the dependencies is only on the superkey.

Final tables

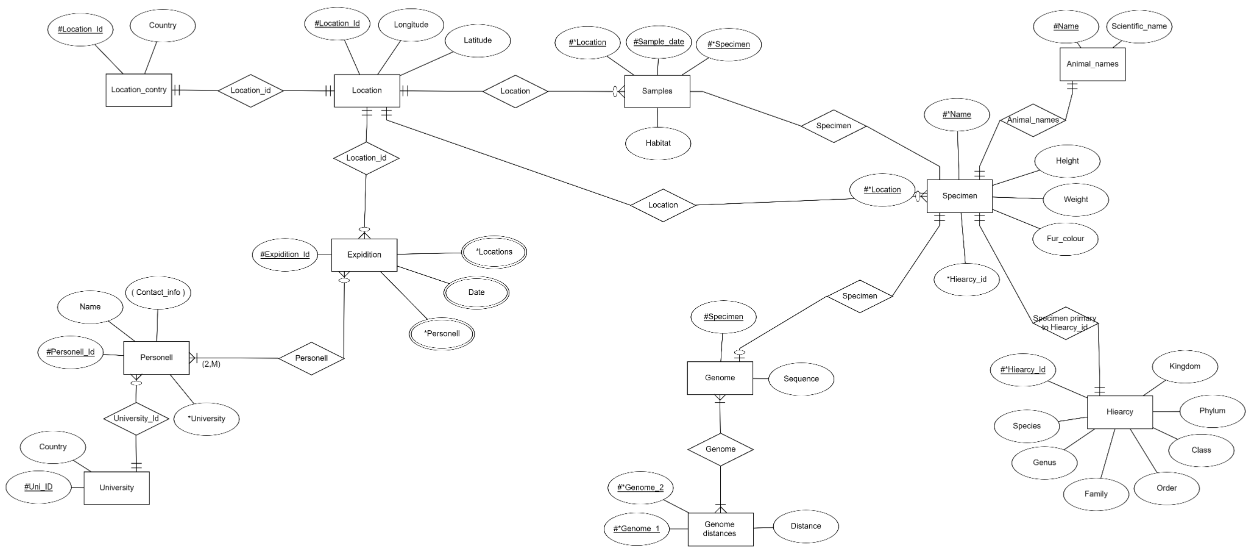
User(#Id, Name, Longitude, Latitude, Phone)
Bike(#Id, Name, *Last_station, status)
reparations(#reperation_id, reperation_name)
Bike_reperations(#*bike_id, #*reperation_id)
Subscriptions(#Id, *User_id, Status, Start, *Type_id)
Subscription_Types(#Type_id, duration)
Optional Subscription_EndDates(#*Subscription_Id, End)
Station(#Id, Name, Longitude, Latitude, Maximum spots, Available spots)
Trip(#Trip_id, *Bike_id, start_time, end_time, *user_id, *start_station, *end_station)

Task 4

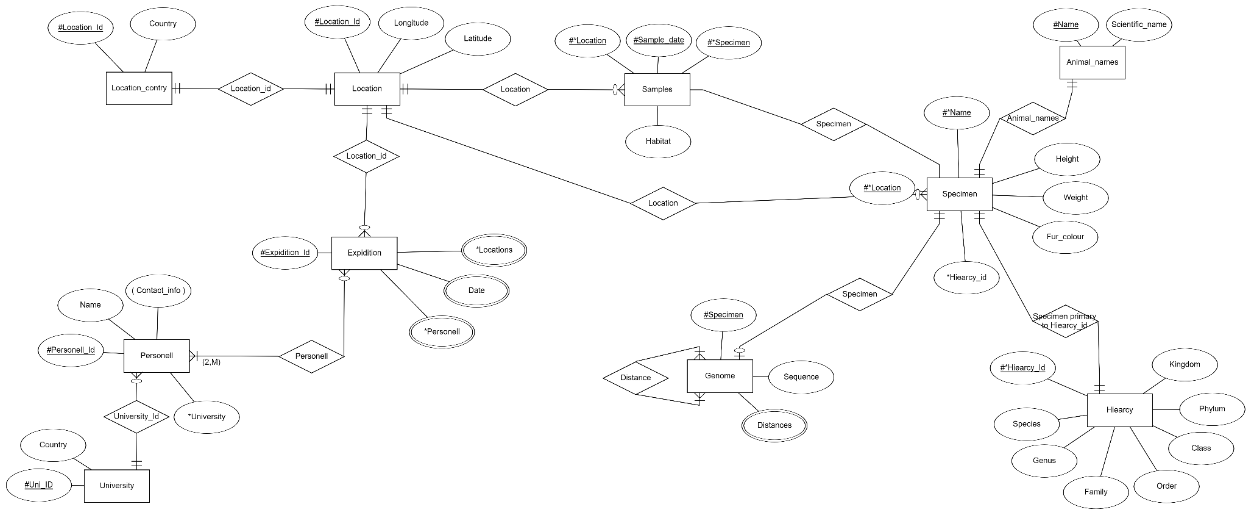
Subproblem 1



Subproblem 2



A bit unsure if it can be solved like this.


#Location_Id
Country
Location_contry
Location_id
#Location_Id
Longitude
Latitude
Location
Location
#*Location
#Sample_date
#*Specimen
Samples
Habitat
Specimen
#Name
Scientific_name
Animal_names
Animal_names
#*Name
Height
Weight
Fur_colour
#*Location
Specimen
Location_id
Location
#Expidition_Id
Expidition
*Locations
Date
*Personell
Name
(Contact_info)
#Personell_Id
Personell
(2,M)
Personell
University_Id
*University
Country
#Uni_ID
University
*Hieacry_id
Specimen
#Specimen
Distance
Genome
Sequence
Distances
Specimen primary to Hieacry_id
#*Hieacry_Id
Kingdom
Hieacry
Phylum
Species
Class
Genus
Family
Order