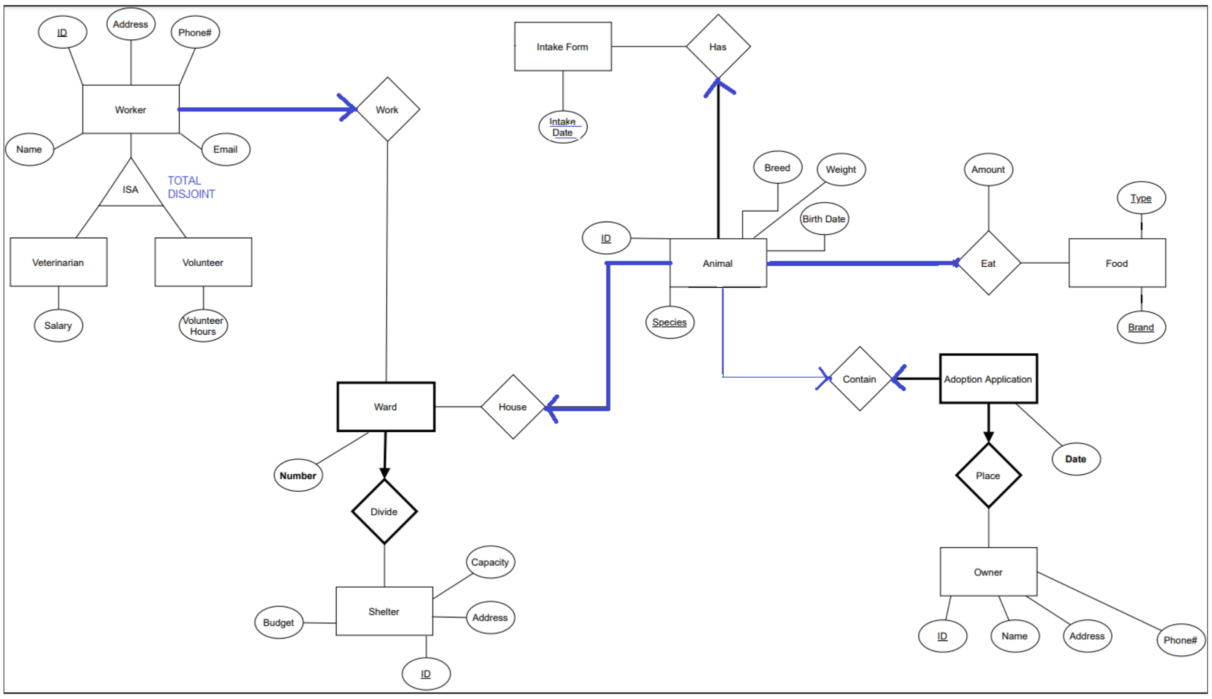
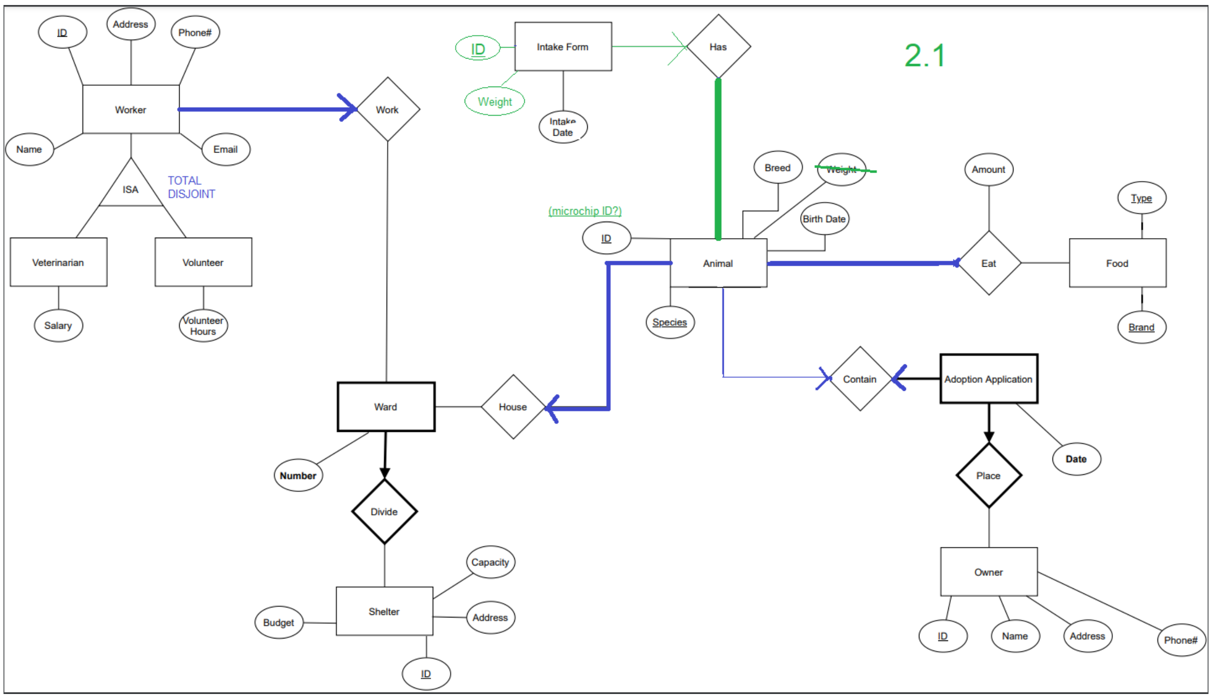


Changes: er_diagram 2.0

Animal has exactly 1 intake form, rather than 1 to many
IntakeDate is a key
Animal has exactly 1 Ward, rather than 0 to many (not registered in our system if not associated with a Ward)
Animal has 1 to many Foods now rather than 0 to 1 (must have food, but not only one, or can't change brand. Perhaps exactly 1?)
Animal has 0 to 1 AdoptionApplication, rather than 0 to many
AdoptionApplication has exactly 1 Animal, rather than 1 to many (or should we keep it as 1 to many?)
Worker has exactly 1 Ward, rather than 0 to many
Added TOTAL DISJOINT to ISA: a Worker must be either a Veterinarian XOR a Volunteer
TODO: properly update ER diagram, don't use Dan's janky MS Paint job lol

2.1



Change Intake Form to Animal, Animal could have more than one intake form because it could be brought back to shelter, so give Intake Form ID as key, change date to non-key, make intake form one to one with animal (put arrow coming from IntakeForm side, Intake form ID)
Add Price per Kilogram to Food

Relational Model Schema:

Animal(ID, Species, Breed, Weight, BirthDate, **WardNumber*, ShelterID***)
AdoptionApplication(Date, **Owner Name, Postal Code, Street Number, Street Name, AnimalID*, AnimalSpecies***)
Owner(Name, Postal Code, Street Number, Street Name, City, Province, Phone#)
Eat(**AnimalID**, **Species**, **FoodType***, **FoodBrand***, Amount)
Eat(**AnimalID**, **Species**, **FoodType, FoodBrand**, Amount)
Food(Type, Brand, Price per Kilogram)
IntakeForm(IntakeDate, IntakeID)
Ward(Number, **ShelterID**)
Shelter(SID, Address, City, Budget, Capacity)
Worker(ID, Name, Address, Phone#, Email, **Ward#*, ShelterID***)
Worker(Name, Postal Code, Street Number, Street Name, City, Province, Phone#, Email, **Ward#*, ShelterID***)
Veterinarian(**ID**, Salary)
Volunteer(**ID**, VolunteerHours)
Veterinarian(**Name, Postal Code, Street Number, Street Name,** Salary)
Volunteer(**Name, Postal Code, Street Number, Street Name,** VolunteerHours)
* not Null
Used ‘Method 2’ for ISA relationship, separate tables for superclass and each subclass

No tables for relationships Work, Divide, House, Has, Contain, or Place because they can be merged with the 'many' side
TODO: set domain for each attribute

FDs

Intake form -> animal
animal -> food
adoption application -> owner

~~TODO~~: "Identify the functional dependencies in your relations, including the ones involving all candidate keys (including the primary key).
You should show the steps taken for the decomposition. Should there be errors, and no work is shown, no partial credit can be awarded without steps shown.
Note: *In your list of FDs, there must be some kind of valid FD other than those identified by a PK or CK – in at least two relations, so that they are at most 2NF.* If you observe that no relations have FDs other than the PK and CK(s), then *you will have to intentionally add some (meaningful) attributes to show valid FDs.* We want you to get a good normalization exercise. Your design must go through a normalization process."

Owner(Owner Name, Street Number, Postal Code, Street Name, Province, City)

Postal Code → Province, City

- Breaks BCNF, non-prime attribute functionally dependant on subset of candidate key of relation

R1(Postal Code, City, Province)
R2(Owner Name, Street Number, Postal Code, Street Name, Phone#)

Eat(**AnimalID**, **Species**, **FoodType**, **FoodBrand**, Amount)

Species → FoodType

- Breaks 2NF, which is great, we have to normalize this relation so that no subset of (**AnimalID**, **Species**) determines any other attribute

AnimalID, **Species** → **FoodType**, **FoodBrand**, Amount
R1(**Species**, **Food Type**) R2(**Species, AnimalID**, **Food Brand**, Amount)

TODO: Determine whether or not Breed → Species is in 3NF. If it is, we need to add attributes to a relation that don't satisfy 3NF, and then normalize both that one and **Species** → **FoodType**
I think the rest should be pretty elementary, with the key determining every other attribute (eg Key → EveryOtherAttribute) **TODO**: type the rest up

~~TODO~~: Normalization, after ensuring we have two FDs that are < 3NF

Randy:
TODO: SQL DDL for all tables
TODO: Set up on Oracle
Chris:
TODO: Populate tables with 5+ tuples
Dan:
TODO: “write a list of all the variety of queries that are proposed for your application, in plain English”. For example, “Insertion: Add a customer to the supermarket membership list.”
TODO: Type up all

Insert Operation – Provide an interface for the user to specify some input for the insert operation. You can choose which table the user should insert to but you cannot hardcode any values. For example, you may choose to have a GUI that allows a fictional user to enter in details for a new employee. The table to insert to has been predetermined (the employee table) but the various details should be dependent on the user of the GUI.
Insertion: Add an animal to our database of animals, ensuring it has an intake form and a ward.
Add an owner to our list of animal owners
… and so on, should be able to insert tuples into every table we have

Delete Operation – Implement a cascade-on-delete situation. Provide an interface for the user to specify some input for the deletion operation. You can choose which table to delete from but you cannot specify which tuple to delete. For example, you can allow your fictional user to delete an employee (so you predetermine the delete operation is for the Employee table) but you cannot predetermine which employee is to be deleted.
Delete: If an Owner requests to be removed from our database of Animal Owners, all Adoption Applications under the Owner must also be removed
Remove an Animal from our database if they die, then Intake Form must also be deleted
Remove a Food if the company stops stocking that specific brand and type

Update Operation – Provide an interface for the user to specify some input for the update operation. You can choose which table to update but the user should be able to specify which attribute(s) in that table to update the value(s) for.
Update: be able to change the Weight, WardNumber, or ShelterID for an Animal that has been reweighed or moved

Selection – Create one query of this category and provide an interface for the user to specify the values of the selection conditions to be returned. Example:

```
SELECT …
FROM …
WHERE Field1 = :Var1 AND Field2 > :Var2
```

The user must have the ability to choose which table and which attributes to select on as well as the values of the selection conditions. I.e., you cannot predetermine which table the selection is for.
Selection:
Select animals or an animal from any table with any specified attributes

Projection – Create one query of this category, with 3-5 attributes in the projection condition, but not SELECT *. The user should be able to select t any number of attributes to project on. Provide an interface for the user to execute this query.
“we have not covered projection yet, so we cannot answer this question.”
Projection: Select any subset of Animal’s attributes to view, such as {Species, Breed, Weight, BirthDate} for example, useful to see physical properties of an Animal without other information

Join – Create one query in this category, which joins at least 2 tables and performs a meaningful query, and provide an interface for the user to execute this query. The user must provide at least one value to qualify in the WHERE clause (e.g. join the Customer and the Transaction table to find the names and phone numbers of all customers who have purchased a specific item).
Join: Join Animal, AdoptionApplication, and Owner tables, to find all animals that have been adopted and where they are

Aggregation with Group By – Create one query that requires the use of distinct aggregation (min, max, average, or count are all fine), and provide an interface (e.g., HTML button/dropdown, etc.) for the user to execute this query. You can hardcode this query (i.e., you do not have to allow for user input) but you must display the tuples in the result within the GUI. The resulting tuples should be presented in a clear manner (e.g., if you use a table to display the results, there should be column headings that give context to the displayed tuples).
Aggregation with Group By: Display all Animals grouped by Species and ordered by

Aggregation with Having – Create one meaningful query that requires the use of a HAVING clause, and provide an interface (e.g., HTML button/dropdown, etc.) for the user to execute this query. You can hardcode this query (i.e., you do not have to allow for user input) but you must display the tuples in the result within the GUI. The resulting tuples should be presented in a clear manner (e.g., if you use a table to display the results, there should be column headings that give context to the displayed tuples).
Aggregation with Having: Find Shelters that have total Veterinarian Salary below a certain threshold, to identify which ones can afford to hire another Vet

Nested Aggregation with Group By – Create one query that finds some aggregated value for each group (e.g., use a nested subquery, such as finding the average number of items purchased per customer, subject to some constraint) and provide an interface (i.e., HTML button, etc.) for the user to execute this query. Some examples for the Sailors table are below.

Note the difference between this query and the above Aggregation Query. You must use separate distinct queries for this criterion and the Aggregation Query (i.e., do not double dip). You can hardcode this query (i.e., you do not have to allow for user input) but you must display the tuples in the result within the GUI. The resulting tuples should be presented in a clear manner (e.g., if you use a table to display the results, there should be column headings that give context to the displayed tuples).
Nested Aggregation with Group By: Display the average weight of each species of animal grouped by shelter ID where calculated age is less than one year, to find a shelter that has more puppies and kittens.

Division– Create one query of this category and provide an interface (i.e., HTML button, etc.) for the user to execute this query (e.g., find all the customers who bought all the items). You can hardcode this query (i.e., you do not have to allow for user input) but you must display the tuples in the result within the GUI. The resulting tuples should be presented in a clear manner (e.g., if you use a table to display the results, there should be column headings that give context to the displayed tuples).
Division: Find all AnimalID from Animal that are associated with all