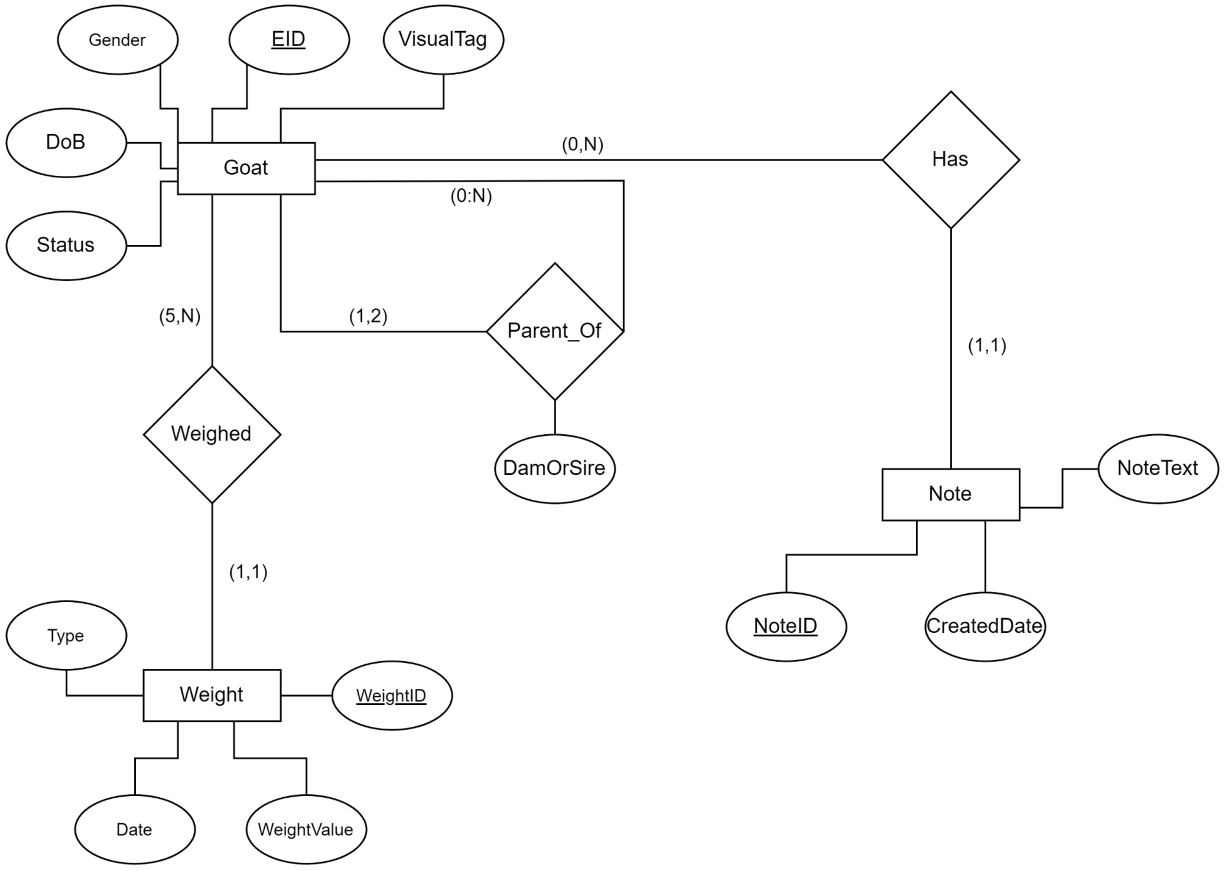
Phase III - Database Model
Group #7
3/7/2024

1. Develop a complete entity-relationship (ER) diagram.



Entities

- Goat
    - EID
    - Visual tag
    - Status (Current, Dead, Off Farm, Sold)
    - DoB
    - Gender

- Weight
    - WeightID
    - Type
    - WeightValue
    - Date
- Note
    - NoteID
    - CreatedDate

    - NoteText

Relations

- Parent_Of (Recursive between Goat)
    - DamOrSire (is Goat A the Dam or Sire of Goat B)
- Weighed (Goat – Weight)
- Has (Goat – Note

Note

- Types of Weight
    - Birth weight
    - Weaning weight
    - Winter weight
    - Last Weight (Current?)
    - Sale weight

2. Map the ER/EER diagram to a relational schema.

Step 1 - Mapping of Regular Entity Types:

- Goat
    - EID (PK)
    - VisualTag
    - Gender
    - DoB
    - Status
- Note
    - NoteID (PK)
    - NoteText
    - CreatedDate
- Weight
    - WeightID (PK)
    - WeightValue
    - Date
    - Type

Step 2 - Mapping of Weak Entity Types:

N/A

Step 3 - Mapping of Binary 1:1 Relation Types:

- HAS [NoteID (Note), EID (Goat)] (Note → Goat, (1:1))
- WEIGHED [WeightID (Weight), EID (Goat)] (Weight → Goat, (1:1))

Step 4 - Mapping of Binary 1:N Relationship Types:

- PARENT_OF [EID (Goat, ”Kid”), EID (Goat, “Parent”)] (Kid → Parent, (1:2))

Step 5 - Mapping of Binary M:N Relationship Types:

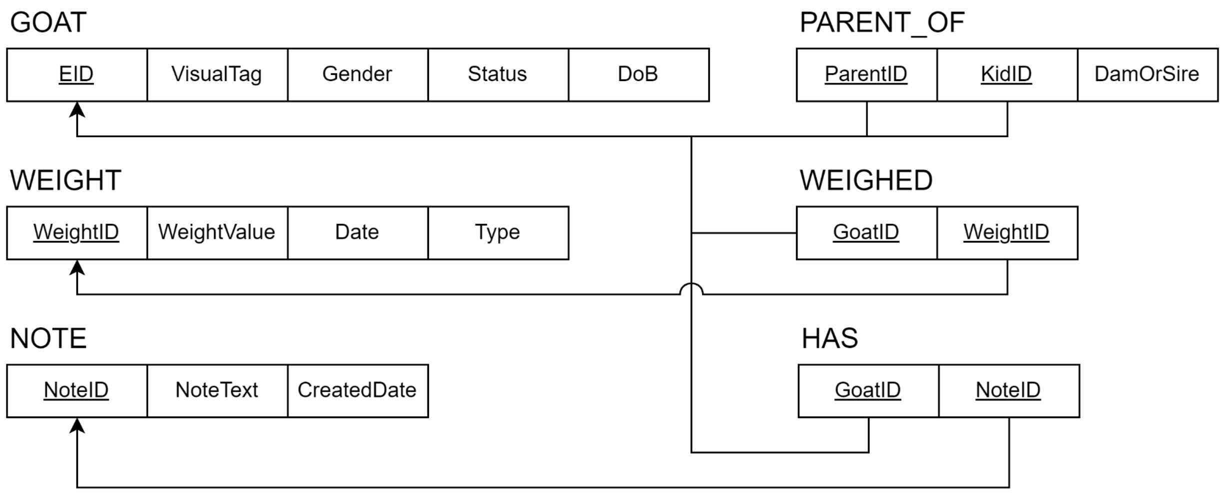
- WEIGHED [EID (Goat), WeightID (Weight)] (Goat → Weight, (5:N))
- PARENT_OF [EID (Goat,”Parent”), EID (Goat, “Kid”)] (Parent → Kid, (0:N))
- HAS [EID (Goat), NoteID (Note)] (Goat → Note, (0:N))

Step 6 - Mapping of Multivalued attributes:

N/A

Step 7 - Mapping of N-ary Relationship Types:

N/A


GOAT
EID
VisualTag
Gender
Status
DoB
PARENT_OF
ParentID
KidID
DamOrSire
WEIGHT
WeightID
WeightValue
Date
Type
WEIGHED
GoatID
WeightID
NOTE
NoteID
NoteText
CreatedDate
HAS
GoatID
NoteID

3. Estimate database size and types and average number of searches.

Database Size

If we are assuming a string takes at least 4 bytes of storage space and integers are 8 bytes (64 bit):

Relations:

- GOAT
    - Strings: EID, VisualTag, Gender, Status
    - Integers: DoB
    - Estimated space required $= 4(4) + 8(1) = 24\,bytes$
- NOTE
    - Strings: NoteID, NoteText
    - Integers: CreatedDate
    - Estimated space required $= 4(2) + 8(1) = 16\,bytes$
- WEIGHT
    - Strings: WeightID, Type
    - Integers: WeightValue, Date
    - Estimated space required $= 4(2) + 8(2) = 24\,bytes$
- PARENT_OF
    - Strings: ParentID, KidID,DamOrSire
    - Estimated space required $= 4(3) + 8(0) = 12\,bytes$
- HAS
    - Strings: GoatID, NoteID
    - Estimated space required $= 4(2) + 8(0) = 8\,bytes$
- WEIGHED
    - Strings: GoatID, WeightID
    - Estimated space required $= 4(2) + 8(0) = 8\,bytes$

In the reverse engineered data provided there are 8382 entries for goats meaning that with an estimated space required of 24 bytes per goat, they would take up 201,168 bytes.

Additionally, based on our ER Diagram, each goat has at least 5 weights (5 weight entities and 5 weighed relations) and 1 parent (1 parent_of relation). No Notes are required by default, but for the sake of this estimation we will include 1 note (1 HAS relation and 1 NOTE entity) for the goat's weaning group. This brings the per goat storage space estimation to $24 + 24(5) + 8(5) + 12(1) + 8(1) + 16(1) = 220\,bytes/goat$.

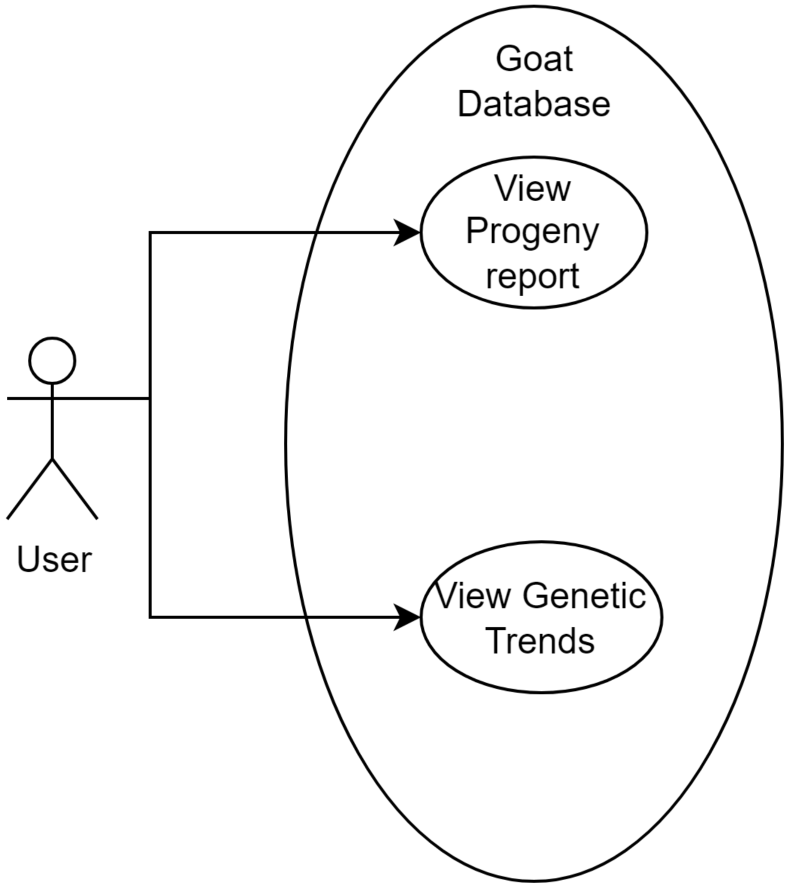
As such the estimated space required for this database is $220\,bytes * 8382 = 1,844,040\,bytes$ or $1.76\,MB$.

Search Types and Average Number of Searches per Query

The required search types for our database will mainly include just lookup operations for both string and integer keys. As of our current thought, there should be no need for any further or more complex search types as our means of allowing the user to search for goats will be done through either clicking on a node within the family tree or by entering the ID of a goat manually, with both only requiring a lookup operation.

For our progeny report a typical query will on average require 3 searches in order to lookup all the correct information from the GOAT, HAS, and WEIGHED tables. For the family tree, a query may require 7 searches in order to find a goat's parents as well as its own children.

4. **Develop a complete UML Use Case diagram.**



5. **Refine the textual use cases.**

Use Case 1 - Viewing the progeny report of a goat via the family tree

A user wants to view the progeny report of a goat as well as its Dam and kids. In order to accomplish this, the user makes use of the family tree to locate their desired goat and its relatives before viewing their progeny reports.

Precondition: The user must be logged in to a verified account.

Steps:

1. Starting from the family tree view, the user must locate the goat they would like to view the progeny report of.
2. Once the desired goat has been identified, the user may view the goat's progeny report by clicking on its node.
3. After viewing the progeny report of the goat, or had the user wanted to view a relative instead, the user may look for their new desired goat by following the lines stemming off of the current goat's node to view their Dam (and Sire, if known) by following the lines leading above, or their kids by following the lines leading below.
4. The user can repeat steps 2 and 3 as many times as they desire.

Alternative Steps:

1. Step 1 (and possibly 3) can be replaced if the user already knows the EID or visual tag of any of the goats they are looking for by instead using the search feature to quickly locate that goat within the family tree, from there they can continue as normal.

Open Issues:

1. Some goats do not have data for all of the attributes we are looking to store. If one such goat is to be displayed, how should the system handle displaying their information? Ideally the UI will indicate the lack of this information, but what happens if a goat has no valid information? Should the page be displayed with blank fields for each missing attribute, or should the UI prevent the user from viewing the progeny report?

2. What is the best way to display Notes? A list is a simple solution, but maybe there is a way to sort the results into categories? This would require some degree of natural language processing or the insertion of additional attributes to the Note entity.

Use Case 2 - Viewing genetic trends in the family tree

A user would like to view how a certain trait compares between members of a goat's family. To accomplish this, they will use a trait comparison filter to view the values of a trait for different goats related to a selected goat.

Precondition: The user must be logged in to a verified account.

Steps:

1. Starting from the family tree view, the user navigates to or searches for the goat they desire.
2. Once the user has located the goat they want to inspect, they can select a trait comparison filter for the trait they would like to compare.
3. The interface will then display each goat's value for that trait alongside their EID and tag information on their node.
4. The user can then either navigate to another part of the family tree to view the results for goats in that section, or they can repeat from step 2 with a different trait comparison filter.

Open Issues:

1. In the reverse engineered data, how are Dams mapped to EID (rfid)? Some seem to map to VisualTags (tag), but others do not. Are the ones that don’t map from outside the farm or is there an error in the data?
2. Most data we have is good for displaying a maternal line, but we do have data for the sires of some goats. Is it worth spending time creating a means to also view the family tree from the paternal line, assuming that there are enough connections to create a good alternative view. Additionally, is there a way that paternal connections, such as sire’s mother/other children, be displayed in some way that does not distract from the maternal line?
3. Based on our data, what trends are best to allow the user to view that give the best information without being confusing to look at or needing excessive explanation?

---

6. <u>**Describe the reasoning behind your database design, given the sustainability goals for the project.**</u>

Our project has two main components, the new progeny report with data about each goat, and our family tree of goats. In designing the database to accommodate the progeny report, it was important that we include as much relevant information as possible that can be displayed in the GUI. To accomplish this we created a central Goat entity that would store information about each goat. The attributes we gave to it were based on both the original description of the progeny report that came from the shareholder, as well as from analyzing the reverse engineered data. Some attributes listed in the latter’s Animal table could have also been included, but the decision was made to not include them as it appeared that only a small number of goats had entries for them and as such we could save time and space by omitting them. Additionally, we decided to split off Weight as a separate entity in order to simplify the many types of weight (weaning, winter, sale, birth, etc) included in the original suggestion. Under this new entity, the Type attribute will store which type of weight is being stored in a given tuple which will allow us to accommodate each type as it exists now in a simpler manner than having many different attributes to account for each different type. Notes are also an important source of some of the information that we want to display in our progeny report, as such we have included our Note entity to accommodate that data as well. For both our Weight and Note entities, we deemed it was necessary to add IDs to each to act as primary keys for each. The alternative was to attempt to use other attributes such as dates, but as it appears the date/time format used in the data does not include time data we may run into the likely issues of multiple notes being made or weights being recorded on the same date.

We extended our model by including a relation that would enable us to implement the goat family tree we are aiming for, that being a recursive relation Parent_Of that matches one goat as being the parent (Dam or Sire) of another. This relation is generic between DamOrSire, but includes an attribute that indicates whether the parent is a Dam or Sire. From this relation we can find all the information we need to construct our family tree including finding siblings or other relationships.