Golden Gate Shelter

CMPS 3420-60 Database Systems

Spring 2021

Pascual Garcia

Antonio Millin

Ariel Espindola Mercado

Roger De La Rosa

Table of Contents

Phase 1: Conceptual Design

Phase 2: Conversion From Conceptual to Relational Database

Phase 3: Physical Implementation and Relational Normalization

Phase 4: Programming Logic for SQL

Phase 5: GUI Development

1.1 - Fact-Finding Techniques and Information Gathering

1.1.1 - Introduction to the Enterprise/Organization.

The business is an animal shelter with multiple locations that provide a database with all the animals available for adoption in the Golden Gates Shelter Collective.It will let users know which animals are available for adoption, along with some information about the animal. This Information can include: age, name, species, breed, medical history, alterations, training. The database will also offer information to the employees about

the animal such as what their role is, and what the needs of the animals are.

1.1.2 - Description of Fact-Finding Techniques.

Our group will model our service based on what is being offered by petfinder.com. To understand the internal needs for an employee at an animal shelter we will interview friends that have worked in the industry. Our group can also use their knowledge of owning pets as a reference source when it comes to the necessities of caring for them.

1.1.3 - The Miniverse of Interest.

This database will maintain information on animals in the shelters of our business, and also holds information on the employees of the shelter. The main entities are Animal, Animal_Medical_History(weak), Animal_Status(weak), Adopter, Employees, and Shelter. The database will need to be able to track who has adopted which animal, who is assigned to care for an animal, where an animal is located, and to keep track of what the employee hierarchy is.

1.1.4 - Itemized Description of Entity sets and Relationship sets.

ENTITIES:

Animals:

- Name (pets name, varchar)
- DOB (age of animal, Composite, int)
  - Date
  - Estimation, Yes or No
- Sex (M or F, varchar)
- Species (dog, cat, bird, reptile, etc. varchar)
- Breed (subspecies of a species, varchar)
- Color (primary color of animal, varchar)
- Pattern (solid, spotted, striped, etc. varchar)
- Altered (yes or no if neutered/spayed, varchar)
- Weight (weight in lbs of animal, float)
- ID (Animal Identification number varchar),
- Shelter_Section_ID (Shelter_Section ID the animal is staying in, varchar).
- Adoption Fee (the fee cost for adoption of animal, Composite, float)
  - Species Fee
  - Age Fee
  - Size Fee
  - Supply Fee
  - Medical Fee
- Euthanization (Was the animal euthanized, Composite, varchar)
  - Yes or No
  - Date
- Arrival Date (Date animal arrived to shelter, Composite, int)
- Adoption Date (Date animal was adopted, Composite, int)

Adopter:

- Name (First, Last, and Middle name,Composite, varchar)
  - First
  - Middle
  - Last

- Address (Where the adopter will be staying with the animal, Composite, varchar)
  - Street Name
  - House Number
  - Zip Code
  - City
  - State
- Phone Number (to contact the adopter, int)
- E-mail (adopters email for contact, varchar)
- Payment (How and what adopter paid, Composite, varchar)
  - Date
  - Method
  - Amount
- ID (Adopter ID number for distinction, varchar)

Employees:

- Name (First, Last, and Middle Name, Composite, varchar)
  - First
  - Middle
  - Last
- Address (Home address of employee, Composite, varchar)
  - Street Name
  - House Number
  - Zip Code
  - City
  - State
- Phone-Number (number to contact employee, int)
- ID (Employee Identification number, varchar)
- SSN (Social Security Number, varchar)
- E-mail (email for contact, varchar)
- Payment (How much the employee is paid, float)
- Start Date (Date employee started working, int)
- End Date (last day of employment, int)

Shelter:

- Name (Name of the shelter, varchar)
- Address (Shelter address, Composite, varchar)

        - Street Name
        - House Number
        - Zip Code
        - City
    - Phone-Number (number for contact, int)
    - E-mail (email for contact, varchar)
    - ID (shelter ID, varchar)

Shelter_Section:

- Section_ID (The name of the subsection, varchar)
- Animal_Type (The classification of the animal in a section, varchar)
- Room_Number (The number of the room, int)
- Capacity (The number of animal that can habitate a room, int)

Health_Record:

- Condition (current condition of animal, varchar)
- History (prior conditions of animal, varchar)
- Special Need (any special need of animal, varchar)
- Serial Number (Number to identify health record, varchar)

Status:

- Activity Level (is the animal very active or lazy, varchar)
- Adoptability (Can the animal be adopted, varchar)
- Temperament (animals temperament, varchar)
- Training (The training status of the animal, Composite, varchar)
    - Crate/Cage Training (is the animal crate trained)
    - Obedience Training (Does the animal know basic obedience commands)
    - Potty Trained (Has the animal been potty trained)
- Exercising (Exercise routine of animal, Composite, varchar)
    - Playtime (the amount of hours of playtime the animal needs)
    - Walks (the amount of time a walk is needed in hours)

- Feeding (Feeding of animal, Composite, varchar)
  - Type of food
  - Feeding time

RELATIONSHIP SETS:

Adopted

- Description - This relationship will relate Adopter with an Animal currently adopted or adopted in the past. Example: An adopter picks up a pet from a shelter and takes it home.
- Entities Involved - Adopter; Animal
- Attributes - Taken, Returned
- Cardinality - 1:M
- Participation/Constraints - Partial for Adopter; Partial for Animal

Cares_For

- Description - Will be for linking which Employee is assigned to take care of an Animal. Example: An employee would need to be able to clean, feed and maybe provide medical needs for the animal as well.
- Entities Involved - Employee; Animal
- Attributes - Start_date
- Cardinality - 1:M
- Participation/Constraints - Total for Employee; Partial for Animal

Located_In

- Description - This will link an Animal to the Shelter_Section that it resides in. Example: An animal is assigned to reside in the cat section of the shelter.
- Entities Involved - Animal; Shelter_Section
- Attributes - Entrance

- Cardinality - 1:N
- Participation/Constraints - Total for Animal; Partial for Shelter_Section

Within

- Description - This will link s Shelter_Section to the Pet Shelter that it resides in. Example: The cat shelter section is within the Pet Shelter.
- Entities Involved - Shelter section; Pet Shelter
- Attributes - Entrance
- Cardinality - 1:N
- Participation/Constraints - Total for Shelter_Section; Total for Pet Shelter

Manages

- Description - This will relate an Employee to the Shelter that they work at. Example: An employee is assigned to work as the manager of the shelter during a shift.
- Entities Involved - Employee; Shelter
- Attributes - N/A
- Cardinality - 1:N
- Participation/Constraints - Partial for employee; Total for Shelter

Supervises

- Description - This will give a status for how an Employee relates to the top manager. Example: An employee is assigned to manage the entire operation.
- Entities Involved - Employee
- Attributes - N/A
- Cardinality - 1:N
- Participation/Contraints - Partial

Health_Is

- Description - This will describe an Animal's Health. Example: A record of an animal's veterinary status is kept on record.

- Entities Involved - Animal; Health
- Attributes - N/A
- Cardinality - 1:1
- Participation/Constraints - Total for Animal; Total for Health

Care

- Description - This an Animal's current status with regard to care. Example: Currently fluffelopolis is recorded as being well fed, exercised, and receiving obedience training.
- Entities Involved - Animal; Status
- Attributes - N/A
- Cardinality - 1:1
- Participation/Constraints - Total for Animal; Total for Health

1.1.5 - User Groups, Data Views, and Operations.

USER GROUP:

- ❖ Adopter
  - ➢ This person will be able to adopt pets and maintain a profile on their history with the shelter. The customer will only be able to interact with the animals during an adoption appointment, once they adopt an animal, and when they return an animal.
- ❖ Employee
  - ➢ This person will provide care for an animal and interact with the Adopters. This user group will need the most access to the database.

DATA VIEWS:

- Adopter
  - They will see the database, likely through the internet, through a GUI that will tell them what animals are available for adoption and what the status of said animals is.
- Employee
  - They will see what work they are assigned to, what the employee profile is, and will have an ability to modify the database depending on rank.

OPERATIONS:

- Employee
  - Cleaning
  - Feeding
  - Vet checkups
  - Release and Accept animal
  - Putting Animal down
  - Maintenance
  - Archive each adopter in database and create accounts for new adopters
  - Maintain profiles for animal within shelter
  - Sterilization (Spayed or Neuter)
- Adopter
  - Adopt or return animal
  - Access online database to check availability and status of animal
  - Make appointment with shelter to meet animal

1.2 - Conceptual Database Design.

1.2.1 - Entity Type Descriptions

Entity: Animals

Candidate Key: Name, ID

Primary Key: ID

Strong or Weak: Strong

Attributes:

Attribute	Description	Domain	Value-Range	Default Value	Null Value	Unique	Single or Multiple	Simple or Composite
Name	Animal name	varchar	A-Z, a-z	N/A	No	No	Single	Simple
DOB	Date of Birth	int	0-9	N/A	Yes	No	Single	Composite
Sex	Gender of animal	varchar	M, F	N/A	No	No	Single	Simple
Species	Species of animal	varchar	A-Z, a-z	N/A	No	No	Single	Simple
Breed	Breed of animal	varchar	A-Z, a-z	N/A	Yes	No	Multiple	Simple
Color	Color of animal	varchar	A-Z, a-z	N/A	No	No	Multiple	Simple
Pattern	Color pattern of animal	varchar	A-Z, a-z	N/A	Yes	No	Single	Simple
Altered	Has the animal been spayed or neutered	varchar	Y, N	N/A	No	No	Single	Simple
Weight	Weight of animal in lbs	float	0.01 - 999.0	N/A	No	No	Single	Simple
ID	Identificatio	varcha	A-Z,	N/A	No	Yes	Single	Simple

	n Number of animal	r	0-9					
Shelter_ Section	Section animal is being kept	varcha r	A-Z, 0-9	N/A	No	No	Single	Simple
Adoption Fee	Price for adoption	float	0.01 - 999.0	N/A	Yes	No	Single	Simple
Euthani- zation	Was the animal Euthanized	varcha r	Y, N, 0-9	N	No	No	Single	Composite
Arrival Date	Date animal arrived to shelter	int	0-9	N/A	No	No	Single	Simple
Adoption Date	Date animal was adopted	int	0-9	N/A	Yes	No	Single	Simple

Entity: Adopter

Candidate Key: Name,

Primary Key: Name, E-mail, ID

Strong or Weak: Strong

Attributes:

Attribute	**Description**	**Domain**	**Value-Range**	**Default Value**	**Null Value**	**Unique**	**Single or Multiple**	**Simple or Composite**
Name	Name of adopter	varchar	A-Z, a-z	N/A	No	No	Single	Composit e
Address	Address where	varchar	A-Z,	N/A	No	No	Single	Composit

	adopter lives		a-z, 0-9					e
Phone Number	Phone number of adopter	int	0-9	N/A	No	No	Single	Simple
E-mail	E-mail of adopter	varchar	A-Z, a-z, 0-9, Special chars	N/A	Yes	No	Single	Simple
Payment	How adopter paid	varchar	A-Z, a-z	N/A	No	No	Multiple	Composite
ID	Identification number of adopter	varchar	A-Z, 0-9	N/A	No	Yes	Single	Simple

Entity: Employee

Candidate Key: Name, E-mail, ID

Primary Key: ID

Strong or Weak: Strong

Attributes:

Attribute	**Description**	**Domain**	**Value - Range**	**Default Value**	**Null Value**	**Unique**	**Single or Multiple**	**Simple or Composite**
Name	Name of Employee	varchar	A-Z, a-z	N/A	No	No	Single	Composite

Address	Address where employee lives	varchar	A-Z, a-z, 0-9	N/A	No	No	Single	Composite
Phone Number	Phone number of employee	int	0-9	N/A	No	No	Single	Simple
ID	Employee identification number	varchar	A-Z, 0-9	N/A	No	Yes	Single	Simple
SSN	Employee Social Security Number	varchar	0-9	N/A	Yes	Yes	Single	Simple
E-mail	E-mail of employee	varchar	A-Z, a-z, 0-9, Special chars	N/A	Yes	No	Single	Simple
Payment	Amount employee is paid hourly	float	0.01 - 999.0	N/A	No	No	Single	Simple
Start Date	Date employee started working	int	0-9	N/A	No	No	Single	Simple
End Date	Ending date employee worked	int	0-9	N/A	Yes	No	Single	Simple

Entity: Shelter

Candidate Key: Address, ID

Primary Key: ID

Strong or Weak: Strong

Attributes:

Attribute	**Description**	**Domain**	**Value-Range**	**Default Value**	**Null Value**	**Unique**	**Single or Multiple**	**Simple or Composite**
Name	Name of Shelter	varchar	A-Z, a-z	Golden Gate Shelter	No	No	Single	Simple
Address	Address of Shelter	varchar	A-Z, a-z, 0-9	N/A	No	Yes	Single	Composite
Phone Number	Phone number of shelter	int	0-9	N/A	No	Yes	Single	Simple
E-mail	E-mail address of shelter	varchar	A-Z, a-z, 0-9, Special chars	N/A	No	No	Single	Simple
ID	Shelter ID number	varchar	A-Z, 0-9	N/A	No	Yes	Single	Simple

Entity: Shelter_Section

Candidate Key: Section_ID

Primary Key: Section_ID

Strong or Weak: Weak

Attributes:

Attribute	**Description**	**Domain**	**Value-**	**Default**	**Null**	**Unique**	**Single**	**Simple**

			Range	**Value**	**Value**		**or Multiple**	**or Composite**
Section_ID	Sections within the shelter	varchar	A-Z, 0-9	N/A	No	Yes	Single	Simpe
Animal_Type	Describes the type of animal held here	varchar	A-Z, a-z	N/A	No	Yes	Multipl e	Simple
Room_Number	Number of the room that houses the animal	int	0-9	N/A	No	No	Single	Simple
Capacity	The number of animal that can be housed in a section	int	0-9	N/A	No	No	Single	Simple

Entity: Health_Record

Candidate Key: Serial Number

Primary Key: Serial Number

Strong or Weak: Weak

Attributes:

Attribute	Description	Domain	Value-Range	Default Value	Null Value	Unique	Single or Multiple	Simple or Composite
Condition	Current condition of animal	varchar	A-Z, a-z, 0-9	N/A	No	No	Multiple	Simple
History	Prior conditions of animal	varchar	A-Z, a-z, 0-9	N/A	Yes	No	Multiple	Simple
Special Need	Special needs of animal	varchar	A-Z, a-z, 0-9	N/A	Yes	No	Multiple	Simple
Serial Number	Vet record	varchar	A-Z, 0-9	N/A	No	Yes	Multiple	Simple

Entity: Status

Candidate Key: Serial Number

Primary Key: Serial Number

Strong or Weak: Weak

Attributes:

Attribute	Description	Domain	Value-Range	Default Value	Null Value	Unique	Single or Multiple	Simple or Composite
Activity	Describes animals activity level	varchar	A-Z, a-z, 0-9	N/A	Yes	No	Single	SImple
Adoptable	Is the animal ready for adoption	varchar	Y, N	N	No	No	Single	Simple
Temperament	What is the animals like	varchar	A-Z, a-z	N/A	No	No	Single	Simple
Training	Does the animal know and trained skills	varchar	Y,N	N/A	Yes	No	Single	Composite
Exercising	Amount of Exercise the animal needs	varchar	A-Z, a-z, 0-9	N/A	Yes	No	Single	Composite
Feeding	Feeding routine of animal	varchar	A-Z, a-z, 0-9	N/A	No	No	Single	Composite
Animal_ID	Animal Status serial number	varchar	A-Z, a-z, 0-9	N/A	No	Yes	Single	Simple

1.2.2 - Relationship Type Description

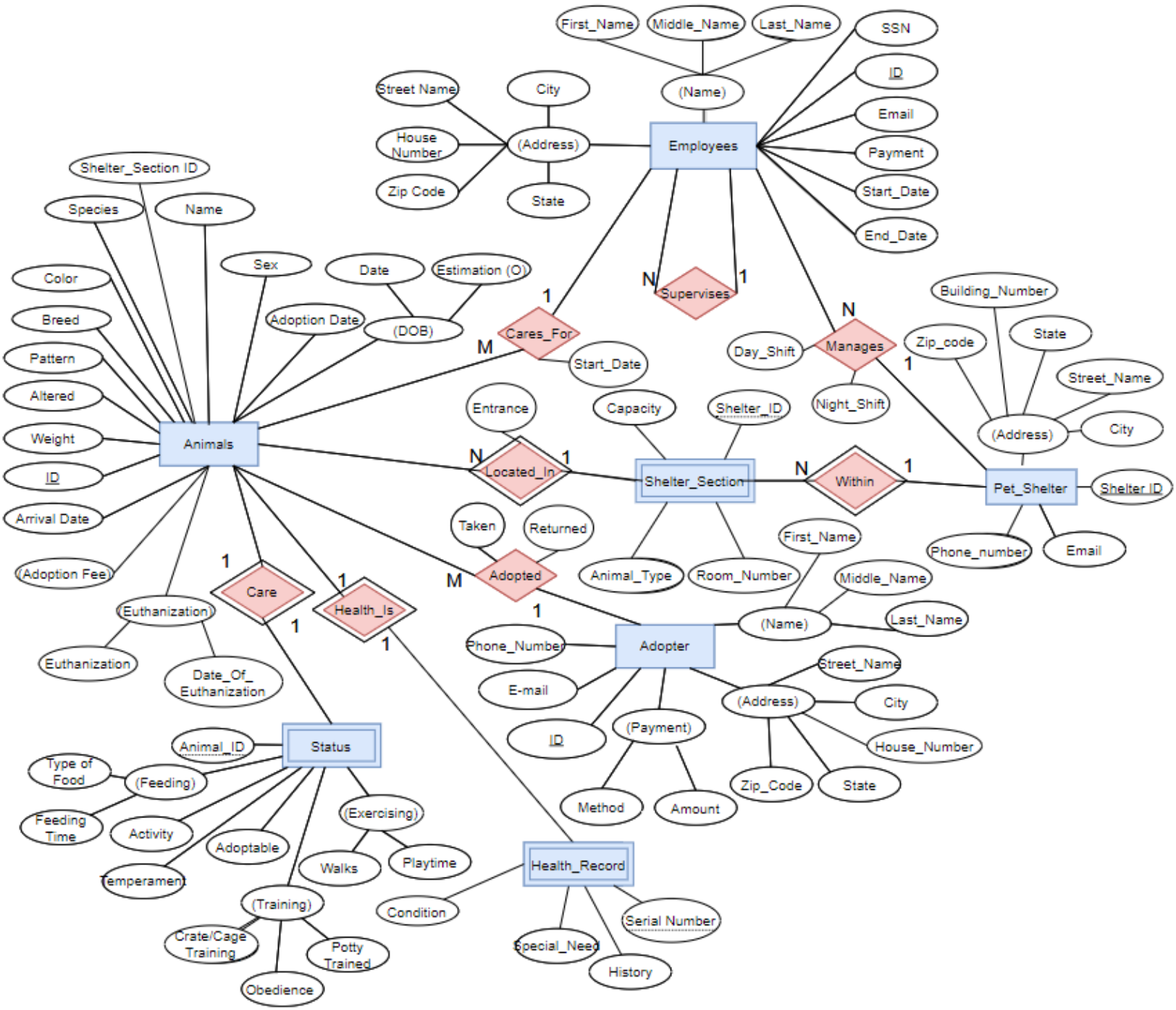
Relationship	Description	Entities Involved	Cardinality	Attributes	Participation Constraint

Adopted	Adopter adopted Animal	Adopter; Animal	1:M	Taken, Returned	Partial; Partial
Cares_For	Employee cares for animals	Employee; Animal	1:M	Start_Date	Total; Total
Within	Shelter_section is within the shelter	Shelter_section; Shelter	1:N	N/A	Total; Total
Located_In	Animal is located in a shelter section	Animal; Shelter_Section	1:N	Entrance	Total; Partial
Manages	An employee manages the Shelter	Employee; Shelter	1:N	N/A	Partial; Total
Supervises	One Employee is the supervisor of the whole operation.	Employee	1:N	N/A	Partial; Total
Health_Is	An Animals official health is based on the Health_Record	Animal; Health Record	1:1	N/A	Total; Total
Care	An Animal's care is measured by its status	Animal; Status	1:1	N/A	Total; Total

1.2.3 - Related Entity Types

There are no entities that will use features from the Enhanced ER Model at this time.

1.2.4 - ER Diagram



https://drive.google.com/file/d/1hE-pK1BuVUoCcXT8eCiiLdQhCKlPp8Sn/view?usp=sharing

2.1 - The ER and Relational Models

In this section a description of along with the history of the Entity-Relation and Relation models are given. We then compare both models with each other while going over their advantages and disadvantages.

2.1.1 - Descriptions of ER and Relational Models

ER Model

Description: Entity-Relation model is a high-level conceptual presentation of a network or database system. Data in the model is described as entities, relationships, and attributes. The diagrammatic notation associated with the model is called an ER Diagram (Fig 2.1).

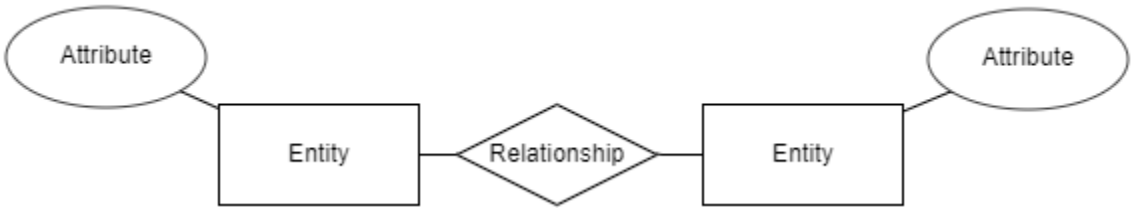


Fig 2.1

History: The Entity-Relationship modeling paradigm was first proposed by Peter Chen in 1976 in his paper titled "The Entity-Relationship Model - Toward a Unified View of Data". Formalization was done by Peter A. Ng in the paper “Further Analysis of the Entity-Relationship approach to Database Design” in 1981. Since then, many academics have worked toward refining the model such as adding semantic data modeling concepts. A nearly yearly conference related to ER modeling is held; the website for the 2021 conference is https://er2021.org/

Relational Model

Description: The relational model represents the database as a collection of relations. Often it is thought of as a flat file of records in the shape of a table of values. In the formal relational model terminology, a row is called a tuple, a column header is called an attribute, and the table is called a relation. The data type of the information in the relation are called the domain of possible values.

History: In 1970, while working for IBM, Tod Codd first introduced the relational model in the paper "A Relational Model for Large Shared Data Banks". It quickly became popular due to its simplicity and mathematical foundation. The first commercial implementations of the relations model became available in the early 1980s. Currently there is a plethora of relational database model systems in wide use.

2.1.2 - Model Comparisons

When comparing the ER model and the relational model, each has their own sets of advantages and disadvantages. However, both are useful for diagramming databases because they each possess different aspects of their designs that are beneficial.

An ER model is a design that uses visual representations of entities and relationships that are connected with lines. Each entity and relationship has a unique shape or design that indicates what each one represents in the database. This makes it easy for the reader to understand how each entity is connected with each other which is good for conceptual designs. Also, the ER model removes the need to have multiples of the same data because the diagram eliminates the need to have primary keys and foreign keys to connect everything. However, without this extra data, the ER model demonstrates its biggest weakness. The ER model does not tell the reader the keys that will be necessary for some of the relations to work properly. For example, an ER model will not show what the foreign key it will need to connect it to the strong entity it is derived from. Therefore, although the ER model is easier to understand because

of its graphic design, it comes at the cost of losing important information.

A relational model is designed as a set of relations. The advantage to using a relational model is that the data is easy to understand. Because the data is easier to understand through tables, it can be used to navigate through the database with queries no matter how complex. This differs from ER diagrams because while the ER model is focused on how each entity is connected, the relational model is focused on how the data interacts. In other words, the strengths of the ER diagram are the weaknesses of the relational model. The relational models design of tables for data makes it more difficult to immediately understand the connection between each of the relations.

2.2 - Conceptual to Logical Conversion Process

The following section will be the conversion of an ER diagram to a relational model. To do this we will be using a 9 step conversion process. The first step will be used to convert strong entities from the ER diagram to relations. The second step will be to convert weak entities to relations. The third step will be to convert 1:1 relationships and give them identifiers to determine which entities they are being connected to. The fourth step and fifth step are similar to the third step but for 1:N and M:N relationships, respectively. The sixth step is to create relations out of multivalued attributes and give them primary and foreign keys to help determine their relationship to other relations. The seventh step is to convert n-ary relationships into relations with foreign keys to determine which relation it belongs to.

2.2.1 - Converting Entity Types to Relations

To convert strong entities into a relation, a relation with the same attributes as the entity needs to be created with the entity name followed by the attributes inside a pair of parenthesis. A primary key is then selected from one of the key attributes and underlined in the model. For example, the Animals entity would be converted to the following relations model:

Animals(ID,Name,DOB,Sex,Species,Breed,Color,Pattern,...)

Similarly, a weak entity is created with a small change. Because a weak entity is derived from strong entities, the weak entity relation must contain the primary key of the strong entity it belongs to. For example, the weak entity Health_Record needs to contain the primary key, 'ID,' of the Animals entity because Health_Record is derived from Animals. So, the Health_Record entity would contain a primary key, 'Animal_ID' that matched 'ID' from the Animals entity which can be shown in the following relations model:

Health_Record(Serial Number,Animal_ID,Date,...)

This would be the way to convert entities with simple attributes. However, if any of the attributes were to be composite attributes, then the composite attributes would be separated into individual attributes in the relation. Also, if there are any multi-value attributes, then a new relation would need to be created for each of the values of the multi-value attribute with a foreign key connecting the new relations to the main relation.

2.2.2 - Converting Relationship Types to Relations

There are three methods for converting relationship types to relations. The most useful and most used method is foreign keys. A foreign key is a key used by a relation to link it to the primary key of another relation. The foreign key should have total participation which means that the key should be present in every instance of the relation. If the relation does not have total participation, another method should be used because there

will be entries whose foreign key will be null. This is a waste of storage.

The second method that can be used is the merged relationship method. For this method, if an entity and relationship have total participation in the relation, then the relationship and entity can be combined into just one relation. In other words, the two would be merged into one relation name with the attributes being a combination of all the attributes of both. The advantage to using this method is that it is an easy way to create and demonstrate a new relation that perfectly links its combined relations. The disadvantage would be that this method can only be used on relationships and entities that have a 1:1 relationship.

The third method is a cross-reference or relationship relation method. This method is used to create a third relation that has two foreign keys that each connect to another relation. This is called a relationship relation or lookup table. The primary key for the third relation is a combination of the two foreign keys. The advantage of this method is that all considerations are covered when converting relationships and entities into relations. In other words, there is no oversight when using this method. The disadvantage to using this method is that it could add unnecessary complexity to the model and must be used for M:N relationships

If an attribute is multivalued, a new relation needs to be created for each multivalued attribute. This new relation will have attributes corresponding to the attributes of the composite attribute the relation is being made from. One of these attributes will serve as a foreign key to the primary key of relation that the new relation belongs to.

If a relationship has more than two entities attached to it, then this n-ary relationship needs to be converted by creating a new relation that has foreign keys made up of the primary keys of the participating entity relations and attributes.

2.2.3 - Converting Extended Types to Relations

There are four methods for conversing superclass/subclass instances. The first option is to create multiple relations that map to a superrealation based on an inherited primary key that also acts as a foreign key. This works for any specialization be it total or partial, and disjoint or overlapping. The second option is to create relations that share a primary key but have it in this instance not map to a superrealation. This option will only work for total participation as every entity in the superclass must belong to at least one of the subclasses. Also, it is only recommended for situations where the specialization has the disjointedness constraint. The third option is to create a tuple where there is an attribute which defines which subclass the relation belongs to. This will only work for disjoint classes and it will produce many NULL values. The Fourth option is to give the relation Boolean type attribute values that indicate what subclass a relation belongs to. This is best used with overlapping classes but it can also work for disjoint classes.

In order to represent a Union type a surrogate value must be created. This surrogate value will act as a superclass for the relations and it will be mapped out using a foriegn key. If one of the relations mapped to the surrogate value is itself a superclass, its subclasses will not need a foriegn key corresponding to the surrogate value.

2.2.4 - Database Constraints

Constraints are rules placed on the type and the amount of data that can be placed in a database. Constraints generally fall into three main categories: implicit constraints, explicit constraints, and semantic constraints. The purpose of constraints is to increase the functionality, reliability,and accuracy of the database. The following are important constraints within a relational database.

Entity constraint: The definitive characteristic of this constraint is that it cannot have a NULL value as one of the primary key values. This is because the primary key value is used to identify the entity in a relation. A NULL for two entities will leave both indistinguishable from other relations.

Primary key and unique key constraints: These constraints make sure that at least one key is used to identify a tuple in a relation. A primary is a single key whose value can be used to make the identification. Other keys that had the potential to be classified as a primary key are designated as a unique key. Null values are not permitted for primary keys.

Referential constraints (foreign key): Referential constraints, also known as foreign key constraints,serve to maintain consistency among tuples in two relations whose primary keys are in the same domain. If an attribute in one relation references a value in a different relation then that value that was referenced must exist.

Check constraints and business rules (semantic constraints): These are constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced in some other way. An example of this is a programmed input restriction of heights between 0 and 10 feet for someone’s personal medical data since it is not possible to be a length beyond this domain.

2.3 - Results of ER to Relational Conversion

This section contains the results of converting your group's ER model to a relational model. Briefly describe how the process went and any challenges you had following the conversion procedure.

2.3.1 Relation Schema

Strong Entity Relations:

ANIMALS

Name | Domain: Varchar, A-Z, a-z | CANNOT BE NULL

DOB | Domain: 0-9, int | CAN BE NULL

Sex | Domain: varchar, M & F, | CANNOT BE NULL

Species | Domain: varchar, A-Z, a-z | CANNOT BE NULL

Breed | Domain: varchar, A-Z, a-z | CAN BE NULL

Color | Domain: varchar, A-Z, a-z | CANNOT BE NULL

Pattern | Domain: varchar, A-Z, a-z | CAN BE NULL

Altered | Domain: varchar, Y & N | CANNOT BE NULL

Weight | Domain: float, 0.01 - 999.0 | CANNOT BE NULL

ID (Primary Key) | Domain: varchar, A-Z, 0-9 | CANNOT BE NULL

Shelter-Section (Foreign Key) | Domain: varchar, A-Z, 0-9 | CANNOT BE NULL

Adoption Fee | Domain: float, 0.01 - 999.0 | CAN BE NULL

Euthanized | Domain: varchar, Y & N | CANNOT BE NULL

Date of Euthanization | Domain: int, 0-9 | CAN BE NULL

Arrival Date | Domain: int, 0-9 | CANNOT BE NULL

Adoption Date | Domain: int, 0-9 | CAN BE NULL

Constraints:-

Primary Key: ID

Foreign Key: Shelter-Section

Candidate Keys: ID

ADOPTER

First Name | Domain: varchar, A-Z, a-z | CANNOT BE NULL

Middle Name | Domain: varchar, A-Z, a-z | CANNOT BE NULL

Last Name | Domain: varchar, A-Z, a-z | CANNOT BE NULL

Street Name | Domain: varchar, A-Z, a-z | CANNOT BE NULL

House Number | Domain: int, 0-9 | CANNOT BE NULL

Zip Code | Domain: int, 0-9 | CANNOT BE NULL

City | Domain: varchar, A-Z, a-z | CANNOT BE NULL

State | Domain: varchar, A-Z, a-z | CANNOT BE NULL

Phone Number | Domain: int, 0-9 | CANNOT BE NULL

E-mail | Domain: varchar, A-Z, a-z, 0-9, Special chars | CANNOT BE NULL

Method | Domain: varchar, A-Z, a-z | CAN BE NULL

Amount | Domain: float, 0.00 - 99999.0 | CAN BE NULL

ID (Primary Key)| varchar, A-Z, 0-9 | CANNOT BE NULL

Constraints:-

Primary Key: ID

Candidate Keys: ID

EMPLOYEE

First Name | Domain: Varchar, A-Z, a-z | CANNOT BE NULL

Middle Name | Domain: Varchar, A-Z, a-z | CANNOT BE NULL

Last Name | Domain: Varchar, A-Z, a-z | CANNOT BE NULL

Street Name | Domain: varchar, A-Z, a-z | CANNOT BE NULL

House Number | Domain: int, 0-9 | CANNOT BE NULL

Zip Code | Domain: int, 0-9 | CANNOT BE NULL

City | Domain: varchar, A-Z, a-z | CANNOT BE NULL

State | Domain: varchar, A-Z, a-z | CANNOT BE NULL

Phone Number | Domain: int, 0-9 | CANNOT BE NULL

ID (Primary Key) | Domain: varchar, A-Z, 0-9 | CANNOT BE NULL

SSN | Domain: int 0-9 | CANNOT BE NULL

E-mail | Domain: varchar, A-Z, a-z, 0-9, Special chars | CANNOT BE NULL

Payment | Domain: float, 0.01-999.0 | CANNOT BE NULL

Start Date | Domain: int, 0-9 | CANNOT BE NULL

End Date | Domain: int, 0-9 | CAN BE NULL

Constraints:-

Primary Key: ID

Candidate Keys: SSN, ID

PET SHELTER

Name | Domain: varchar, A-Z, a-z | CANNOT BE NULL

Street Name | Domain: varchar, A-Z, a-z | CANNOT BE NULL

Building Number | Domain: int, 0-9 | CANNOT BE NULL

Zip Code | Domain: int, 0-9 | CANNOT BE NULL

City | Domain: varchar, A-Z, a-z | CANNOT BE NULL

State | Domain: varchar, A-Z, a-z | CANNOT BE NULL

Phone Number | Domain: int, 0-9 | CANNOT BE NULL

Phone Number | Domain: int, 0-9 | CANNOT BE NULL

E-mail | Domain: varchar, A-Z, a-z, 0-9, Special chars | CANNOT BE NULL

ID (Primary Key) | Domain: varchar, A-Z, 0-9 | CANNOT BE NULL

Constraints:

Primary Key: ID

Candidate Keys: ID, Name

Weak Entity Relations:

SHELTER_SECTION

Shelter_Name (Foriegn Key) | Domain: varchar, A-Z, a-z | CANNOT BE NULL

Section_ID (Primary Key) | Domain: varchar, A-Z, 0-9 | CANNOT BE NULL

Animal_Type | Domain: varchar, A-Z, a-z | CANNOT BE NULL

Room_Number | Domain: int, 0-9 | CANNOT BE NULL

Capacity | Domain: int, 0-9 | CANNOT BE NULL

Fill | Domain: int, 0-9 | CAN BE NULL

Constraints:-

Primary Key: Section_ID

Candidate Keys: Shelter_Name, Section_ID

HEALTH_RECORD

Date Administered | Domain: int, 0-9 | CAN BE NULL

Condition | Domain: Varchar, A-Z, a-z, 0-9 | CANNOT BE NULL

History | Domain: Varchar, A-Z, a-z, 0-9 | CAN BE NULL

Special Need | Domain: Varchar, A-Z, a-z, 0-9 | CAN BE NULL

Serial Number (primary number) | Domain: Varchar, A-Z,0-9 | CANNOT BE NULL

Animal_ID (foreign key)| Domain: | Domain: varchar, A-Z, 0-9 | CANNOT BE NULL

Constraints:-

Primary Key: Serial Number

Foreign Key: Animal_ID

Candidate Keys: Serial Number, Animal_ID

STATUS

Activity | Domain: varchar, A-Z, 0-9 | CAN BE NULL

Adoptable | Domain: varchar, Y & N | CANNOT BE NULL

Temperament | Domain: varchar, A-Z, a-z | CANNOT BE NULL

Training | Domain: varchar, Y & N | CAN BE NULL

Exercising | Domain: varchar, A-Z, a-z, 0-9 | CAN BE NULL

Feeding | Domain: varchar, A-Z, a-z, 0-9 | CANNOT BE NULL

Animal_ID (foreign key) | Domain: varchar, A-Z, a-z, 0-9 | CANNOT BE NULL

Constraints:-

Foreign key: Animal_ID

Candidate Keys: Animal_ID

Relationships:

Adopted

Adopter_ID (foreign key) | varchar, A-Z, 0-9 | CANNOT BE NULL

Animal_ID (foreign key and Primary key) | Domain: varchar, A-Z, 0-9 | CANNOT BE NULL

Taken | Domain: int, 0-9 | CAN BE NULL

Returned | Domain: int, 0-9 | CAN BE NULL

Constraints:-

Primary Key: Animal_ID

Foreign Keys: Adopter_ID, Animal_ID

Candidate Keys: Adopter_ID, Animal_ID

Cares_For

Employee_ID (foreign key) | Domain: varchar, A-Z, 0-9 | CANNOT BE NULL

Animal_ID (foreign key & primary key) | Domain: varchar, A-Z, 0-9 | CANNOT BE NULL

Start_Date | Domain: int, 0-9 | CANNOT BE NULL

End_Date | Domain: int, 0-9 | CAN BE NULL

Constraints:-

Primary Key: Animal_ID

Foriegn Keys: Animal_ID, Employee_ID

Candidate Keys: Animal_ID, Employee_ID

Manages

Employee_ID | Domain: varchar, A-Z, 0-9 | CANNOT BE NULL

Shelter_ID | Domain: varchar, A-Z, 0-9 | CANNOT BE NULL

Day_Shift | Domain: Text, A-Z | CANNOT BE NULL

Nigh_Shift | Domain: Text, A-Z | CANNOT BE NULL

Constraints:-

Primary Key: Shelter_ID

Foriegn Key: Employee_ID, Shelter_ID

Candidate Keys: Employee_ID, Shelter_ID

Supervises

Employee_ID | Domain: varchar, A-Z, 0-9 | CANNOT BE NULL

Superviser | Domain: varchar, A-Z, 0-9 | CAN BE NULL

Constraints:-

Primary Key: Employee_ID

Foriegn Key: Employee_ID

Candidate Keys: Employee_ID

2.3.2 Sample Data

ANIMALS

Name	DOB	Sex	Species	Breed	Color	Pattern
Hashim	7/10/2016 21:23	M	Dog	Chihuahua	Brown	Spotted
Tristam	2/18/2010 13:18	M	Dog	Pug	Tan	Solid
Aldus	10/9/2017 1:04	M	Cat	Ragdoll	white	Spotted
Giffie	4/1/2012 9:19	M	Dog	German Shepard	Brown	Patches
Janessa	10/22/2010 15:30	F	Lizard	Komoto Dragon	Green	Solid
Malcolm	1/25/2013 21:44	M	Cat	Brritish Shorthair	Grey	Solid
Rochella	2/21/2016 20:44	F	Ferret	Black Sable	Black	Solid
Cesar	4/30/2012 17:29	M	Lizard	Komoto Dragon	Green	Solid
Deane	4/20/2010 13:56	F	Cat	Persian	White	Solid
Amos	5/15/2016 16:36	M	Ferret	Blaze	Red	Striped

Altered	Weight	ID	Shelter_Section	Adoption_Fee	Euthanized
FALSE	4.3	e1d35a33	2	108	TRUE
FALSE	15	807cb21d	6	91	FALSE
TRUE	12	476f5c09	10	137	FALSE
TRUE	68	0f1f6580	5	15	FALSE
FALSE	150	fc29b8b1	5	122	TRUE
FALSE	10	feb847ac	2	93	FALSE
FALSE	2.5	3774615e	7	17	FALSE
FALSE	172	8d964852	8	36	FALSE
TRUE	9	48b52db0	7	93	FALSE
TRUE	3.2	48b52db0	10	113	FALSE

Date of Euthanization	Arrival Date	Adoption Date
2/25/2021 14:55	5/10/2019 23:38	NULL
NULL	12/8/2012 16:19	NULL
NULL	5/2/2020 19:47	9/23/2020 16:23
NULL	7/31/2014 18:41	NULL
4/15/2014 20:26	11/13/2011 16:40	NULL
NULL	11/4/2017 16:51	NULL
NULL	2/21/2019 6:00	2/20/2020 20:43
NULL	11/17/2016 11:28	9/13/2017 23:13
NULL	1/11/2012 4:39	NULL
NULL	4/16/2018 15:27	NULL

ADOPTER

First Name	Middle Name	Last Name	Street Name	House Number	Zip Code
Eleen	Noble	Sanbroke	Hooker	34	54569
Tamar	Lindy	Beeden	Bobwhite	24132	68745
Mason	Bud	Quartermaine	Hanson	6787	63779
Chickie	Annadiane	Leckey	Vahlen	95	11243
Jan	Emelyne	McIlvoray	Sugar	96919	61957
Jeannette	Kali	Moakes	Londonderry	65	67253
Hertha	Lazaro	Winchurst	Westridge	4815	47335
Evan	Tarrah	Lamps	Mendota	92293	44924
Dorita	Randolph	Muzzullo	Lakeland	84034	13537
Shoshanna	Florenza	Taffee	Superior	38843	63629

City	State	Phone Number	E-mail
Trenton	New Jersey	609-336-3831	nsanbroke0@gmpg.org
Omaha	Nebraska	402-927-4845	lbeeden1@epa.gov
Los Angeles	California	323-908-2875	bquartermaine2@stanford.edu
Pensacola	Florida	850-981-1507	aleckey3@time.com
Washington	District of Columbia	202-525-5899	emcilvoray4@dagondesign.com
Anniston	Alabama	256-988-7964	kmoakes5@icio.us
Herndon	Virginia	757-605-8721	lwinchurst6@ucoz.com
Nashville	Tennessee	615-353-5264	tlamps7@vkontakte.ru
Indianapolis	Indiana	317-463-0739	rmuzzullo8@cnbc.com
San Francisco	California	650-790-6926	ftaffee9@cafepress.com

Method	Amount	ID
Cash	50	f66fd56c
Cash	179	59cf6646
Cash	77	9e9b820a
Card	52	c8404263
Card	25	a3649a93
Cash	106	5ef0d2d3
Card	89	6ffbadea
Cash	57	9a93bf31
Cash	112	cd25fa5a
Card	175	f5be70ce

EMPLOYEE

First Name	Middle Name	Last Name	Street Name	House Number	Zip Code
Sebastian	Pepe	Bernardino	Hooker	8696	11822
Petronella	Isa	de Clerc	Westport	7894	53808
Shaylynn	Ario	Murkus	Oak	79	52380
Hillary	Cherida	Horley	Grayhawk	394	65601
Jay	Beverie	Haryngton	Riverside	694	42549
Orbadiah	Korie	Batterham	Arapahoe	8	43498
Ardra	Tadio	Scammell	Little Fleur	6	55154
Oriana	Ernestus	Gabbott	Corben	3	58118
Zed	Kirbee	Label	6th	4	49035
Frank	Kori	Cuchey	Prentice	19	63629

City	State	Phone Number	ID	SSN
Orange	California	858-815-1896	6dd330a5	331-91-3063
Bowie	Maryland	240-328-4047	90deb728	742-59-6375
Raleigh	North Carolina	919-768-9568	72e63041	863-94-6480
Washington	District of Columbia	202-987-8785	69ff2334	827-62-7816
Newark	New Jersey	201-283-5392	6cf6d551	173-64-2004
Evansville	Indiana	812-428-0302	860e64a8	314-47-1063
Lehigh Acres	Florida	863-277-6983	2f49929e	258-78-4450
Charlotte	North Carolina	704-335-0965	3c007748	874-15-7850
Washington	District of Columbia	202-811-3089	0e9aae59	112-53-4756
Aurora	Colorado	303-474-8778	4af64d6f	380-98-7582

E-mail	Payment	Start Date	End Date
pbernardino0@naver.com	19	8/25/2013	
ideclerc1@aboutads.info	20	8/4/2011	2/12/2020
amurkus2@ameblo.jp	23	2/22/2011	
chorley3@paginegialle.it	15	11/29/2014	
bharyngton4@epa.gov	20	6/17/2014	
kbatterham5@canalblog.com	17	6/21/2012	
tscammell6@phpbb.com	24	11/14/2013	8/2/2019
egabbott7@ucoz.ru	19	9/20/2011	
klabel8@123-reg.co.uk	24	2/13/2014	
kcuchey9@chron.com	23	12/31/2010	

PET SHELTER

Name	Street Name	Building Number	Zip Code	City
Golden Gate Pet Shelter	Carpenter	228	93309	Bakersfield

State	Phone Number	E-mail	ID
California	651-469-8885	goldengateshelter@hexun.com	04fccb6d

SHELTER_SECTION

Shelter_Name	Section_ID	Animal_Type	Room_Number	Max_Capacity
Golden Gate Pet Shelter	9c98ed50	Dogs	1	10
Golden Gate Pet Shelter	9af0fbd2	Dogs	2	10
Golden Gate Pet Shelter	3b0b99ba	Dogs	3	10
Golden Gate Pet Shelter	d26a69e4	Cats	4	15
Golden Gate Pet Shelter	678a431b	Cats	5	15
Golden Gate Pet Shelter	8641d39a	Birds	6	10
Golden Gate Pet Shelter	44985a14	Birds	7	10
Golden Gate Pet Shelter	56c1a344	Lizards	8	20
Golden Gate Pet Shelter	15cd2734	Lizards	9	20
Golden Gate Pet Shelter	5e132923	Rodents	10	30

Fill
10
10
5
15
0
5
0
15
5
20

HEALTH_RECORD

Vaccine_Type	Date Administered	Condition	History	Special Need
Parvo	11/18/2020	Good		Blind
Rabies	9/9/2020	Good		Deaf
Parvo	1/22/2021	Excellent		
Influenza	1/26/2021	Poor	Abused	
Covid	2/19/2021	Exellent		Hip Problems
Influenza	4/9/2020	Good		
Parvo	1/12/2021	Good		
Parvo	10/23/2020	Good		Cushings Disease
Rabies	8/18/2020	Poor	Abused	
Covid	2/9/2021	Good		

Serial Number	Animal_ID
dd338dcd-120b-4608-8cd0-f29064974bf0	f68351d7
f6967034-b862-4213-8a4c-d117225fdd52	3fb4d359
5da833f2-8d3e-40fd-a144-6328b78d8e9a	5955c0b8
96728ff2-cc3a-4bf5-b901-9572071f5d3a	879e920b
7c1bf283-ca84-4f17-b811-93ba3964e95f	99ae1406
b45518e3-34fa-40f6-b282-a58d5dbed023	f4e71f31
9183c9b2-ba1c-4dae-b117-ed84e5a99120	f3e46129
ad841147-2daf-4749-bf69-5e9b5b023a4c	2cfd6d3e
33fb7a56-c815-41d8-8faf-42a81e92fa5e	a6e75ef2
60ff5c78-1cb6-45bd-a192-86084cccc437	f0b19e39

STATUS

Activity	Adoptable	Temperament	Training	Exercising	Feeding	Animal_ID
high	FALSE	aggressive	30min	120min	3 times	c49b3ac5
high	TRUE	hyper	30min	120min	3 times	40227fe3
medium	TRUE	calm	15min	60min	2 times	bc5480be
medium	FALSE	calm	15min	60min	2 times	0f068b71
low	TRUE	lazy	10min	30min	2 times	2cf2c14e
high	FALSE	hyper	30min	120min	3 times	0f284bed
low	TRUE	lazy	10min	30min	2 times	ee49ff6e
low	FALSE	aggressive	30min	30min	2 times	abae6773
medium	TRUE	calm	15min	60min	2 times	3f682558
high	FALSE	hyper	30min	120min	3 times	55087048

Adopted

Adopter_ID	Animal_ID	Taken	Returned
7d5e18f7	ade0a4ad	12/14/2020 12:42	
7d5e18f8	d4afb4b8	3/29/2020 9:03	
96c23567	2efaf7c1	9/9/2020 4:58	
96c23568	3274h577		
96c23544	8a19211f		
96c23570	26cd13e4		
7d5e18f1	a90629c9		
69dfd34c	69dfd34c		
69dfd3fg	2ea9cae7	4/17/2020 16:42	
69dfd3ff	6b35c72a	9/3/2020 0:56	
69d455ff	bac36383	11/20/2020 0:52	
69dfd3bf	be697dc0	9/15/2020 5:58	
64dd20ab	8fd44a22	2/16/2021 15:17	
e80dc014	4b31185e	10/10/2020 10:36	
fhe12123	64b720ab	2/1/2021 14:23	
24544sdc	e80dc00c	10/24/2020 3:25	
12ewxd2	6cfeb41c	6/4/2020 6:01	1/9/2022 3:32
ajvdsf789	ced9c24f	7/18/2020 7:39	5/4/2021 2:50
dsvaqe32	6fb35b21	11/26/2020 19:44	
23r4dfa3	ac64d184	2/11/2021 11:12	
12fdws33	ccd2efca	10/12/2020 12:31	
221cccsq	2cfcb813	9/9/2020 19:14	
c44s32dd	905694ad	11/19/2020 9:31	7/29/2021 13:04
ddd32111	ed23444f	6/13/2020 13:45	4/11/2021 5:20
92ba8c6b	11f324de	1/18/2021 4:15	
99ba8c6a	8bb114f4	6/2/2020 4:16	
92558c6a	c0bbbaff	12/16/2020 9:43	
hhyyik77	ee136d63	2/4/2021 20:16	12/2/2021 8:44
92ba777a	7e00806d	2/25/2021 18:59	11/6/2021 23:59
hu78780h	bce4b643	12/11/2020 5:23	5/28/2021 1:36
5443fgyi	52bd85c3	6/8/2020 18:18	
iio9889v	6ac28482	7/28/2020 9:00	
71fc0lly	2f543a30	9/7/2020 13:21	
gtyu6bbc	71fc03b0	7/6/2020 8:40	6/27/2021 17:48
685nhy09	92ba8c6a	11/22/2020 8:48	8/18/2021 11:44
ji89654e	1c37f0ea	2/5/2021 7:41	4/17/2021 22:09
7865aaaa	2164cf62	3/24/2020 3:53	11/13/2021 0:17
ghy32856	fd670c3d	7/23/2020 4:56	
34e5rt61	24c03f5e	6/24/2020 3:56	
dqw23e45	62ec34cd	6/25/2020 4:56	

Cares_For

Employee_ID	Animal_ID	Start_Date	End_Date
1f23ea27	ade0a4ad	3/19/2014 20:21	11/24/2025 21:55
941408f1	d4afb4b8	5/21/2018 3:16	
1cab0f46	2efaf7c1	3/31/2016 1:45	
41eff9f4	3274h577	9/21/2011 0:43	
045eb818	8a19211f	9/5/2010 21:17	
a7a40012	26cd13e4	6/20/2012 19:00	
149h3052	a90629c9	3/29/2015 13:16	
0fe511cf	69dfd34c	9/19/2020 18:28	4/12/2021 1:47
dc8ce677	2ea9cae7	12/9/2019 18:50	6/30/2024 11:50
e49a3b41	6b35c72a	5/20/2017 18:58	2/22/2022 19:47
1a8f7e03	bac36383	7/4/2020 0:12	
943667f1	be697dc0	1/29/2021 1:25	
6cd8232a	8fd44a22	2/20/2015 14:40	
e6bdfa70	4b31185e	10/7/2016 16:31	3/26/2025 2:45
4d1ac922	64b720ab	11/3/2011 16:51	5/22/2021 2:02
2d7bcac0	e80dc00c	10/11/2018 0:44	9/18/2021 9:27
d667e068	6cfeb41c	7/30/2017 17:24	7/21/2021 13:04
8b01e350	ced9c24f	4/26/2017 10:37	11/14/2021 22:34
a4b51895	6fb35b21	12/24/2014 4:43	1/1/2022 17:23
d3a62e5a	ac64d184	10/29/2012 10:12	
db9473b1	ccd2efca	8/23/2014 17:13	
25b801b2	2cfcb813	3/2/2011 15:04	
91ddcf6c	905694ad	12/14/2012 7:22	
3b6057b0	ed23444f	9/9/2019 22:47	
e3bc5066	11f324de	4/12/2018 5:30	
dd18a604	8bb114f4	12/17/2018 17:45	
f198e198	c0bbbaff	1/12/2011 16:35	
d34d6416	ee136d63	4/21/2011 20:46	
0fdf6b4f	7e00806d	3/28/2013 7:53	
7589848a	bce4b643	12/19/2013 4:01	5/19/2021 20:37
d3cb5e57	52bd85c3	6/1/2012 14:41	6/12/2021 8:14
32625b1a	6ac28482	8/5/2012 9:32	7/18/2031 18:28
8f202f1d	2f543a30	12/4/2020 15:40	12/17/2021 14:57
f60d379c	71fc03b0	10/10/2018 14:56	3/30/2021 18:30
1047d69c	92ba8c6a	7/11/2018 3:48	4/20/2021 9:14
6c836357	1c37f0ea	3/16/2014 1:32	6/15/2028 13:33
g866374	2164cf62	11/15/2010 20:49	2/3/2022 16:17
e67c6d20	fd670c3d	7/9/2019 3:48	
bcd16eb9	24c03f5e	12/5/2010 1:36	
7edd314c	62ec34cd	12/12/2011 1:36	

Supervises

Employee_ID	Supervisor
1f23ea27	
941408f1	1f23ea27
1cab0f46	1f23ea28
41eff9f4	1f23ea29
045eb818	1f23ea30
a7a40012	1f23ea31
149h3052	1f23ea32
0fe511cf	1f23ea33
dc8ce677	1f23ea34
e49a3b41	1f23ea35
1a8f7e03	1f23ea36
943667f1	1f23ea37
6cd8232a	1f23ea38
e6bdfa70	1f23ea39
4d1ac922	1f23ea40
2d7bcac0	1f23ea41
d667e068	1f23ea42
8b01e350	1f23ea43
a4b51895	1f23ea44
d3a62e5a	1f23ea45
db9473b1	1f23ea46
25b801b2	1f23ea47
91ddcf6c	1f23ea48
3b6057b0	1f23ea49
e3bc5066	1f23ea50
dd18a604	1f23ea51
f198e198	1f23ea52
d34d6416	1f23ea53
0fdf6b4f	1f23ea54
7589848a	1f23ea55
d3cb5e57	1f23ea56
32625b1a	1f23ea57
8f202f1d	1f23ea58
f60d379c	1f23ea59
1047d69c	1f23ea60
6c836357	1f23ea61
g866374	1f23ea62
e67c6d20	1f23ea63
bcd16eb9	1f23ea64
7edd314c	1f23ea65

Manages

Employee_ID	Shelter_ID	Day_Shift	Night_Shift
1f23ea27	867648a6d53797e04fd7f2d991339e10f08988d8	Yes	No
941408f1	867648a6d53797e04fd7f2d991339e10f08988d8	Yes	No
1cab0f46	867648a6d53797e04fd7f2d991339e10f08988d8	No	Yes
41eff9f4	867648a6d53797e04fd7f2d991339e10f08988d8	No	Yes

2.4 - Sample Queries

Our database for Golden Gates Shelter features a large variety of different entities. This includes Animal data, Employee data, and Adoption data.In addition, our database features records that contain both animal care data and role assignment data; this allows us to keep track of how an animal is being cared for and by who is assigned to it. We also allow the ability to keep track of what person adopted the pet, and whether they returned it.

2.4.1 Design Of Queries

Our database revolves around Adopters, employees, and animals. Therefore, the queries in mind will be based on one or more of those relations. The following are 10 plain english queries that can be expected to be asked of our database.

1. What is the maximum capacity of the shelter section that holds cats.

2. List the employee ID of the day shift managers.

3. Return all the people who have adopted more than one animal.

4. Return all the adopters who have adopted dogs.

5. Which animals have been abused?

6. Return any Komodo Dragons that are in good condition.

7. Return the employees that are assigned to take care of dogs.

8. Return all adoptable German Shepherds.

9. Return Dogs that have been cared for by all employees.

10. Return employees who have cared for all cats.

2.4.2 Relational Algebra Expressions

Relational algebra is a mathematical theory that allows one to use a set of operations to issue a retrieval query from relations. The result of the query is a new relation that can be queried further. The following are formal relational algebra expressions based on the queries in section 2.4.1.

1. What is the maximum capacity of the shelter section that holds cats.

$$S \leftarrow \sigma_{Animal\_Type = cats}\ Shelter\_Section$$

$$C \leftarrow \Pi_{Max\_Capacity}\ S$$

2. List the employee ID of the day shift managers.

$$M \leftarrow \sigma_{Day\_shift = \text{“Day”}}\ (Manages)$$

$$SM \leftarrow \Pi_{Employee\_ID}(M)$$

3. Return all the people who have adopted more than one animal.

$$A \leftarrow \Pi_{Adopter\_ID}\ Adopted$$

$$A2 \leftarrow A \bowtie A$$

$$Adopter\_ID = Adopter\_ID \wedge Animal\_ID \neq Animal\_ID$$

$$P \leftarrow \sigma_{Adopter\_ID = ID}\ Adopter$$

4. Return all the adopters who have adopted dogs.

$$A \leftarrow \sigma_{Species=\text{“Dog”}}(Animals)$$

$A2 \leftarrow Adopted \bowtie_{Animal\_ID = Animal\_ID} A$

$A3 \leftarrow Adopter \bowtie_{Adopter\_ID = ID} A2$

5. Which animals have been abused?

$A \leftarrow \sigma_{History = \text{“Abused”}} (Health\_Record)$

$A2 \leftarrow \Pi_{Animal\_ID} A$

$A3 \leftarrow \sigma_{Animal\_ID = Animal\_ID} (Animals)$

6. Return any Komodo Dragons are in good condition

$A \leftarrow \sigma_{Species= \text{“Komodo Dragon”}} (Animals)$

$A2 \leftarrow \Pi_{Animal\_ID} (A)$

$A3 \leftarrow \sigma_{Condition = \text{“Good”}}(Health\_Record)$

$C \leftarrow A2 \bowtie_{Animal\_ID = Animal\_ID} A3$

7. Return the employees that are assigned to take care of dogs.

$A \leftarrow \Pi_{Animal\_ID}(\sigma_{Species = Dog} Animals)$

$C \leftarrow Cares\_for \bowtie_{Animal\_ID = Animal\_ID} A$

$E \leftarrow \Pi_{Employee\_ID} C$

$E2 \leftarrow Employee \bowtie E$

$$\text{Employee\_ID} = \text{Employee\_ID}$$

8. Return all adoptable German Shepherds.

$$GS \leftarrow \sigma_{\text{Breed} = \text{“German Shepard”}} (\text{Animals})$$

$$A \leftarrow P_{\text{Adopter\_ID} / \text{ID}}(\Pi_{\text{ID}}(GS))$$

$$AD \leftarrow A * \text{Status}$$

$$F \leftarrow \sigma_{\text{Adoptable} = \text{“True”}} AD$$

9. Return Dogs that have been cared for by all employees.

$$E \leftarrow \Pi_{\text{ID}} \text{Employee}$$

$$A \leftarrow \sigma_{\text{species} = \text{“Dog”}} \text{Animals}$$

$$C \leftarrow \Pi_{\text{Animal\_ID}, \text{ Employee\_ID}} (\text{Cares\_for} \bowtie_{\text{Animal\_ID} = \text{Animal\_ID}} A)$$

$$A2 \leftarrow C \div E$$

10. Return employees who have cared for all cats.

$$E \leftarrow \Pi_{\text{ID}} \text{Employees}$$

$$C \leftarrow \sigma_{\text{species} = \text{“Cat”}} (\text{Animals})$$

$$K \leftarrow \Pi_{\text{Employee\_ID}} (\text{Cares\_For})$$

$$AC \leftarrow K \bowtie C$$

$$F \leftarrow AC \div E$$

2.4.3 Relational Calculus Expressions

Relational Calculus provides a higher-level declarative language for specifying relational queries. The main difference between Relational Calculus and Relational Algebra is that the former does not have an order of operations to specify how to retrieve the query result. Relational calculus has two variations tuple and domain, but the following are only demonstrations of the tuple version.

1. What is the maximum capacity of the shelter section that holds cats.

{SS.Max_Capacity|Shelter_Section(ss) ^ SS.Animal_Type = "Cats"}

2. List the employee ID of the day shift managers.

{DS.employee_ID| Manages(DS) ^ Day_Shift = "Yes"}

3. Return all the people who have adopted more than one animal.

{a|(Adopter(a) ^ ∃b(Adopted(b)^ a.ID = b.Adopter_ID ^ (∃c(Adopted(c) ^ b.Animal_ID != c.Animal_ID))}

4. Return all the adopters who have adopted dogs.

{a|Adopter(a)^ ∃b(Adopted(b) ^ a.ID = b.Adopter_ID ^ ∃c(Animal(c) ^ b.Animal_ID = c.ID ^ c.species = "dog")) ^ ¬∃(d)(Adopted(d) ^ ∃(e)(Animal(e) ^ d.Animal_ID = e.ID ^ e.species != "dog"))}

5. Which animals have been abused?

{a | Animal(a) ^ ∃h(Health_Record(h) ^ h.History="Abused" ^ h.Animal_ID = a.ID)}

6. Return any Komodo Dragons that are in good condition

{a | Animal(a) ^ a.species = "Komodo Dragon" ^ ∃b(Health_Record(b) ^ b.Animal_ID = a.ID ^ b.Condition = "good")}

7. Return the employees that are assigned to take care of dogs.

{e | Employee(e) ^ ∃c(Cares_for(c) ^ c.Employee_ID = e.ID ^ ∃a(Animals(a) ^ a.Species = "dog" ^ a.ID = c.Animal_ID))}

8. Return all adoptable German Shepherds.

{a | Animals(a) ^ a.breed = “German Shepherds” ^ ∃s(Status(s)^ s.Animal_ID = a.ID ^ s.adoptable = “TRUE”}

9. Return Dogs that have been cared for by all employees.

{a | Animals(a) ^ a.species = ”dog” ^ □b(~Cares_For(B) v ∃c(Employee(c))}

10. Return employees who have cared for all cats.

{a | Employees(a)^ □b(~Cares_For(b)v ∃c(Animals(c) ^ c.species = cat))}

3.1 - Relational Normalization

3.1.1 Normalization Process

Normalization is a process of analysing data in a database based on their foreign dependencies and primary keys in order to minimize dependencies and minimize the insertion, deletion, and update anomalies. Lacking a normalization will cause a database to be difficult to survey.

The first normal form is considered to be part of the formal definition of the basic relational model. It is designed to disallow multivalued attributes, composite attributes and their combinations. Attributes can only have atomic values and it must also be singular. The second normal form is based on the concept of full functional dependency, visualized by X -> Y. Therefore, if an attribute was removed from X the function will no longer hold. A relation schema R is in the third normal for if, whenever a non-trivial functional dependency X -> A holds in R, either X is a superkey of R, or A is a prime attribute of R. The Boyce-Codd normal form is a stricter version of the third normal form. A relation is in Boyce-Codd normal form if whenever a non-trivial functional dependency X -> A holds in R, then X is a superkey of R.

There are three types of anomalies that can occur if the relational model is not updated. The anomalies are classified as Insertion, deletion and modification. Together they are categorized as update anomalies. An insertion anomaly can issue a result in one of two scenarios. The first scenario is when there is an attempt to insert a tuple into a relation but all the attribute values are not consistent with the other tuples in the relation. The second scenario occurs when one tries to create a new table that lacks data that should contain a primary key. A deletion anomaly is related to the second insertion

anomaly. It occurs when we delete the last tuple of a table, so that no primary key remains. Lastly the modification anomaly occurs when we fail to update the forign key of tuples when the attribute that they were referring to has been changed.

Essentially, normalization acts as the foil to anomalies. Leaving a relation unnormalized will run the risk of unsound dependencies making relations impossible to survey, constraints not being enforced, and redundancies to occur.

3.1.2 Application to Relational Model

STRONG ENTITY RELATIONS			
Relation	**Normal form**	**Modification anomalies**	**Changes?**
Animals	1NF	No	No
Adopter	1NF	No	No
Employee	1NF	No	No
Shelter	1NF	No	No

WEAK ENTITY RELATIONS			
Relation	**Normal form**	**Modification anomalies**	**Changes?**
Shelter Section	1NF	No	No
Health Record	1NF	No	No
Status	1NF	No	No

RELATIONSHIPS

Relation	Normal form	Modification anomalies	Changes?
Adopted	1NF	No	No
Cares_for	1NF	No	No
Within	1NF	No	No
Manages	1NF	No	No
Located_In	1NF	No	No
Supervises	1NF	No	No
Health_Is	1NF	No	No
Care	1NF	No	No

3.2 - Database Implementation

3.2.1 Background Information

The main purpose of using a relational database management software, or RDBMS, is to easily implement a database by not having to handle lower-level considerations which would need to be managed more strictly. One way of managing a database is through a client-server RDBMS which remotely handles a database on a server and clients would need permission for access. Advantages of using a client-server RDBMS includes easy data maintenance, better data integrity, and better security. Data maintenance is easier because data can be shared with multiple administrators. Also, data can be backed up onto multiple local machines. This leads into better data integrity because having an easier time maintaining data means that the chances of compromising data is significantly lowered. Security is better because in a client-server paradigm there are several layers of authentication like passwords and permissions. This helps control who has access to the database directly. Despite the

advantages of a client-server RDBMS, there is one major disadvantage. The disadvantage of using a client-server RDBMS is the level of complexity to implement. It is more complicated to implement this method than a database that is stored locally.

With these advantages and disadvantages in mind, the Golden Gate Pet Shelter will be using the RDBMS MariaDB on the CSUB server Delphi. Although it is more complex to implement, the advantages of using MariaDB outweigh the disadvantages. It can be easily accessed by all of the members of the team and contains an extensive amount of helpful documentation online.

3.2.2 Schema and Hosting

(1). CREATE TABLE table_name ...;

The CREATE TABLE statement is used to create a new table in a database. It's column parameters specify the names of the columns of the table while the datatype parameter specifies the type of data the column can hold (Ex. varchar, integer, date, etc.).

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
    ....
);
```

(2). CREATE VIEW view_name ...;

The CREATE VIEW statement creates a view which, for SQL, is a virtual table. The view is based on the result-set of the SQL statement. The view contains rows and columns with fields from one or more tables in the database.

```
CREATE VIEW view_name AS
```

```
SELECT column1, column2, …
FROM table_name
WHERE condition;
```

(3). CREATE INDEX idx_name ...;

The CREATE INDEX statement is used to create indexes in tables. Indexes are used to get data from a database more quickly. Indexes cannot be seen.

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

(4). INSERT INTO …

The INSERT INTO statement is used to insert new records in a table. INSERT INTO can be written in two ways. One is to specify both the column names and the values to be inserted.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

The second is that you do not need to specify the column names, but the order of the values need to be in the same order as the columns in the table.

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

(5). DROP TABLE ... ;

The DROP TABLE statement is used to drop an existing table from a database.

```
DROP TABLE table_name;
```

(6). DROP VIEW ... ;

The DROP VIEW command deletes a view.

```
DROP VIEW [view name];
```

(7). COMMIT;

The COMMIT statement is used to save changes invoked by a transaction to the database. It saves all the transactions since the last COMMIT or ROLLBACK.

```
COMMIT;
```

(8). ROLLBACK;

The ROLLBACK statement is a command that causes all data changes since the last BEGIN WORK or START TRANSACTION to be discarded by the RDBMS.

```
ROLLBACK;
```

(9). SELECT

The SELECT statement is used to select data from a database. The returned data is stored in a result table, called result-set.

```
SELECT column1, column2, …
FROM table_name;
```

(10). CREATE or REPLACE FUNCTION …

The CREATE or REPLACE FUNCTION will either create a new function or replace an existing definition.

```
CREATE [or REPLACE] FUNCTION function_name
```

(11). CREATE or REPLACE PROCEDURE ..

The CREATE or REPLACE PROCEDURE statement will either create or replace a procedure.

CREATE [or REPLACE] PROCEDURE *procedure_name*

(12). CREATE or REPLACE TRIGGER …

The CREATE or REPLACE TRIGGER statement will either create or replace a database trigger which is type of stored procedure that automatically runs when an event occurs in a database server.

CREATE [or REPLACE] TRIGGER *trigger_name*

(13). DROP PROCEDURE | FUNCTION …

The DROP PROCEDURE statement that removes one or more stored procedures or procedure groups from a database.

DROP { PROC | PROCEDURE } [IF EXISTS] { [*schema_name.*] *procedure* }

(14). Your own batch data insertion/automation.

We created .csv files for each of the tables that keeps track of the data we already have from Section 2. We then created a python script that creates a .sql file with that data and the correct SQL syntax. Then by calling the SOURCE statement in the database we can run the .sql files. This makes creating all the tables with all the appropriate data easy and quick.

SOURCE *filename.sql*

3.3 - Query Implementation

1. What is the maximum capacity of the shelter sections that holds cats.

```
SELECT Max_Capacity FROM Shelter_Section

WHERE Animal_Type = “Cats”;
```

```
MariaDB [pgarcia]> SELECT Max_Capacity FROM Shelter_Section
    -> WHERE Animal_Type = "Cats";
+--------------+
| Max_Capacity |
+--------------+
|           15 |
|           15 |
+--------------+
2 rows in set (0.00 sec)
```

2. List the names of the day shift managers.

```
SELECT Employee.firstName FROM Employee

INNER JOIN Manages ON Manages.Employee_ID = Employee.ID

WHERE Manages.Day_Shift = “Yes”;
```

```
MariaDB [pgarcia]> SELECT Employee.firstName FROM Employee
    -> INNER JOIN Manages ON Manages.Employee_ID = Employee.ID
    -> WHERE Manages.Day_Shift = "Yes";
+------------+
| firstName  |
+------------+
| Sebastian  |
| Petronella |
+------------+
2 rows in set (0.00 sec)
```

3. Return all the names and IDs of people who have adopted more than one animal.

```
SELECT Adopter.First_Name, Adopter.ID FROM Adopter

INNER JOIN Adopted ON Adopter.ID = Adopted.Adopter_ID

HAVING COUNT(*) > 1;
```

```
MariaDB [pgarcia]> SELECT Adopter.First_Name, Adopter.ID FROM Adopter
    -> INNER JOIN Adopted ON Adopted.Adopter_ID = Adopter.ID
    -> HAVING COUNT(*) > 1
    -> ;
+------------+----------+
| First_Name | ID       |
+------------+----------+
| Tamar      | 59cf6646 |
+------------+----------+
1 row in set (0.00 sec)
```

4. Return all the adopters who have adopted dogs.

```
SELECT Adopter.First_Name, Adopter.ID FROM Adopter

INNER JOIN Adopted ON Adopter.ID = Adopted.Adopter_ID

INNER JOIN Animals ON Adopted.Animal_ID = Animals.ID

WHERE Animals.species = “Dog”;
```

```
MariaDB [pgarcia]> SELECT Adopter.First_Name, Adopter.ID FROM Adopter
    -> INNER JOIN Adopted ON Adopter.ID = Adopted.Adopter_ID
    -> INNER JOIN Animals ON Adopted.Animal_ID = Animals.ID
    -> WHERE Animals.species = "Dog";
+------------+----------+
| First_Name | ID       |
+------------+----------+
| Jeannette  | 5ef0d2d3 |
| Chickie    | c8404263 |
+------------+----------+
2 rows in set (0.00 sec)
```

5. Which animals have been abused?

```
SELECT Animals.Name FROM Animals

INNER JOIN Health_Record ON Health_Record.Animal_ID = Animal.ID

WHERE Hist = “Abused”;
```

```
MariaDB [pgarcia]> SELECT Animals.Name FROM Animals
    -> INNER JOIN Health_Record ON Health_Record.Animal_ID = Animals.ID
    -> WHERE Hist = "Abused";
+--------+
| Name   |
+--------+
| Deane  |
| Giffie |
+--------+
2 rows in set (0.00 sec)
```

6. Return any Komodo Dragons that are in good condition.

```
SELECT Animals.ID, Animals.Name, Animals.Breed FROM Animals

INNER JOIN Health_Record ON Health_Record.Animal_ID = Animals.ID

WHERE Health_Record.Cond = “Good”

AND Animals.Breed = “Komodo Dragon”;
```

```
MariaDB [pgarcia]> SELECT Animals.ID, Animals.Name, Animals.Breed FROM Animals
    -> INNER JOIN Health_Record ON Health_Record.Animal_ID = Animals.ID
    -> WHERE Health_Record.Cond = "Good" AND Animals.Breed = "Komodo Dragon";
+----------+-------+---------------+
| ID       | Name  | Breed         |
+----------+-------+---------------+
| 8d964852 | Cesar | Komodo Dragon |
+----------+-------+---------------+
1 row in set (0.00 sec)
```

7. Return the employees that are assigned to take care of dogs.

```
SELECT DISTINCT Employee.firstName, Employee.ID FROM Employee
INNER JOIN Cares_For ON Cares_For.Employee_ID = Employee.ID
INNER JOIN Animals ON Animals.ID = Cares_For.Animal_ID
WHERE Animals.species = “Dog”
```

```
MariaDB [pgarcia]> SELECT DISTINCT Employee.firstName, Employee.ID FROM Employee
    -> INNER JOIN Cares_For ON Cares_For.Employee_ID = Employee.ID
    -> INNER JOIN Animals ON Animals.ID = Cares_For.Animal_ID
    -> WHERE Animals.Species = "Dog";
+-----------+----------+
| firstName | ID       |
+-----------+----------+
| Ardra     | 2f49929e |
| Oriana    | 3c007748 |
| Frank     | 4af64d6f |
| Hillary   | 69ff2334 |
| Sebastian | 6dd330a5 |
| Shaylynn  | 72e63041 |
| Orbadiah  | 860e64a8 |
| Petronella | 90deb728 |
+-----------+----------+
8 rows in set (0.00 sec)
```

8. Return all adoptable German Shepherds.

```
SELECT Animals.Name, Animals.ID, Animals.Breed FROM Animals
INNER JOIN Status ON Status.Animal_ID = Animals.ID
WHERE Status.Adoptable = 1 AND Animals.breed = “German Shepherd”
```

```
MariaDB [pgarcia]> SELECT Animals.Name, Animals.ID, Animals.Breed From Animals
    -> INNER JOIN Status ON Status.Animal_ID = Animals.ID
    -> WHERE Status.Adoptable = 1 AND Animals.Breed = "German Shepherd";
+--------+----------+-----------------+
| Name   | ID       | Breed           |
+--------+----------+-----------------+
| Giffie | 0f1f6580 | German Shepherd |
| Rex    | 1c37f0ea | German Shepherd |
| Kai    | 9999abcd | German Shepherd |
+--------+----------+-----------------+
3 rows in set (0.01 sec)
```

9. Return Dogs that have been cared for by all employees.

```
SELECT Animals.Name, Animals.ID, Animals.Species FROM Animals

INNER JOIN Cares_For ON Cares_For.Animal_ID = Animals.ID

GROUP BY Cares_For.Animal_ID

HAVING COUNT(*) = 10;
```

```
MariaDB [pgarcia]> SELECT Animals.Name, Animals.ID, Animals.Species FROM Animals
    -> INNER JOIN Cares_For ON Cares_For.Animal_ID = Animals.ID
    -> GROUP BY Cares_For.Animal_ID
    -> HAVING COUNT(*) = 10;
Empty set (0.01 sec)
```

*There are no animals at all that are cared for by all employees so the set came back empty.

10. Return employees who have cared for all cats.

```
SELECT Employee.firstName, Employee.ID FROM Employee

INNER JOIN Cares_For ON Employee.ID = Cares_For.Employee_ID

GROUP BY Cares_For.Employee_ID

HAVING COUNT(*) = (SELECT COUNT(*) FROM Animals WHERE
Animals.Species = 'Cat');
```

```
MariaDB [pgarcia]> SELECT Employee.firstName, Employee.ID FROM Employee INNER JO
IN Cares_For ON Employee.ID = Cares_For.Employee_ID GROUP BY Cares_For.Employee_
ID HAVING COUNT(*) = (SELECT COUNT(*) FROM Animals WHERE Animals.Species = 'Cat'
);
Empty set (0.02 sec)
```

*There are no employees who take care of all the cats so the set came back empty.

4.1 - Introduction

SQL is a programming language that is specifically designed and used to handle databases. This differs from other languages which are object-oriented or procedure-oriented. SQL is centered around databases and tables.

Some important tools in SQL include views, functions, stored procedures, and triggers. A “view” is a virtual table based on the result-set of an SQL statement. They are used to display tables and to query the database without changing anything. For example, a view can be created to combine the data of two tables to display it as a single table. Functions and stored procedures are mechanisms that allow a user to query a database without having to use a database management system directly. For example, a procedure could be created to run the same SQL query multiple times without having to type it out

everytime. Although functions and stored procedures are similar, there are some distinctions between them in some SQL languages. For example, in MySQL functions return a result while procedures do not. Triggers are mechanisms that are designed to happen automatically when there is some change that has occurred in a specified table. For example, a trigger can be created to delete rows in a table that are dependent on another row in a different table. So, when the row is removed from the second table, the trigger will delete all the rows that are dependent on it. This eliminates the need for a database administrator to have to go back and delete those dependent rows manually.

4.2 - Syntax of Programming Logic

RDBMS: MariaDB

VIEWS

General syntax to create view:

```
CREATE VIEW [view_name] AS
[Select statement];
```

view_name: The given name of the view being created.

Select statement: Any valid SQL SELECT statement.

FUNCTIONS

General syntax to create functions:

```
CREATE FUNCTION [function_name]([function parameters])
RETURN [type]
RETURN [function_body];
```

function_name: The given name of the function being created.

function_parameters: The parameters that will be passed into the function.

type: Any valid data type in MariaDB.

function_body: Any valid SQL procedure statement.

TRIGGERS

General syntax to create triggers:

```
CREATE TRIGGER [trigger_name]
[trigger_time] [trigger_event] ON [table_name]
FOR EACH ROW [trigger_statement];
```

trigger_name: The given name of the trigger being created.

trigger_time: The trigger action time. Use the statements BEFORE or AFTER to indicate if the trigger is activated before or after each row being modified.

trigger_event: Indicated the kind of statement that activates the trigger. It can use the following statements:

INSERT - trigger is activated when a new row is inserted into the table.

UPDATE - trigger is activated when a row is modified.

DELETE - trigger is activated when a row is deleted from the table.

table_name: The name of the table that the trigger is attached to.

trigger_statement: Any valid SQL procedure.

Similarities between stored procedures and functions is that they both reduce the compilation cost for repeated executions. They are also similar in their construction, they function relatively the same and are created in almost identical ways.

There are few differences between those two. Stored procedures have input and output parameters, while functions have only input parameters. Functions are limited to only one return value, while stored procedures can return multiple values. Stored procedure is more flexible to write any code that you want, while functions have a rigid structure and functionality.

Advantages of Stored procedures are that they are useful if a database program is needed by several applications, it can be stored at the server and invoked by any of the application programs. This reduces duplication of effort and improves software modularity. As mentioned before they can return

multiple values and are also possible to return any kind of datatype.

4.3 - Implementation

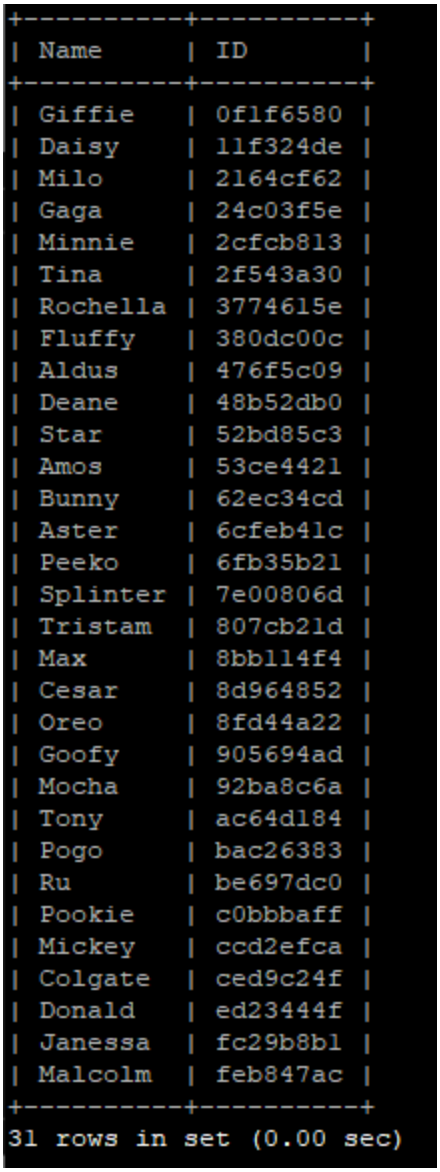
4.3.1 - Views

1. This view shows all the animals that are currently in the animal shelter by excluding the animals in the Animals table that have been adopted or euthanized.

```
CREATE VIEW Animals_In_Shelter AS

SELECT Animals.* FROM Animals

WHERE (ID) NOT IN

(SELECT Adopted.Animal_ID FROM Adopted) AND

Animals.Date_of_Euthanization IS NULL;
```

```
MariaDB [pgarcia]> CREATE VIEW Animals_In_Shelter AS
    -> SELECT Animals.* FROM Animals
    -> WHERE (ID) NOT IN
    -> (SELECT Adopted.Animal_ID FROM Adopted) AND
    -> Animals.Date_of_Euthanization IS NULL;
Query OK, 0 rows affected (0.00 sec)
```

```
SELECT Name, ID FROM Animals_In_Shelter;
```



Name	ID
Giffie	0f1f6580
Daisy	11f324de
Milo	2164cf62
Gaga	24c03f5e
Minnie	2cfcb813
Tina	2f543a30
Rochella	3774615e
Fluffy	380dc00c
Aldus	476f5c09
Deane	48b52db0
Star	52bd85c3
Amos	53ce4421
Bunny	62ec34cd
Aster	6cfeb41c
Peeko	6fb35b21
Splinter	7e00806d
Tristam	807cb21d
Max	8bb114f4
Cesar	8d964852
Oreo	8fd44a22
Goofy	905694ad
Mocha	92ba8c6a
Tony	ac64d184
Pogo	bac26383
Ru	be697dc0
Pookie	c0bbbaff
Mickey	ccd2efca
Colgate	ced9c24f
Donald	ed23444f
Janessa	fc29b8b1
Malcolm	feb847ac

31 rows in set (0.00 sec)

2. This view joins the Animals_In_Shelter table and Status table to show adoptable animals along with some important information about them.

```
ALTER VIEW Adoptable_Animals AS

SELECT a.ID, a.Name, a.Sex, a.Breed,

a.Species, Status.Temperament, a.Adoption_Fee

FROM Animals_In_Shelter AS a
```

```
INNER JOIN Status ON a.ID = Status.Animal_ID

WHERE Status.Adoptable=1;
```

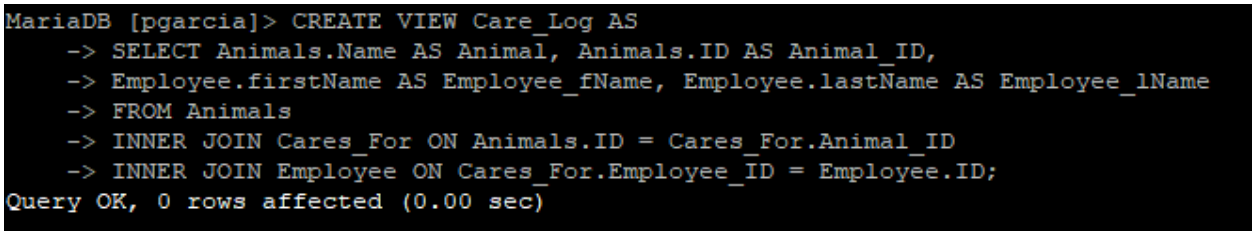
```
MariaDB [pgarcia]> ALTER VIEW Adoptable_Animals AS
    -> SELECT a.ID, a.Name, a.Sex, a.Breed,
    -> a.Species, Status.Temperament, a.Adoption_Fee
    -> FROM Animals_In_Shelter AS a
    -> INNER JOIN Status ON a.ID = Status.Animal_ID
    -> WHERE Status.Adoptable=1;
Query OK, 0 rows affected (0.01 sec)
```

```
SELECT * FROM Adoptable_Animals;
```

```
MariaDB [pgarcia]> SELECT * FROM Adoptable_Animals;
+----------+--------+---------------------+---------+-------------+--------------+
| Name     | Weight | Breed               | Species | Temperament | Adoption_Fee |
+----------+--------+---------------------+---------+-------------+--------------+
| Giffie   |     68 | German Shepherd     | Dog     | lazy        |          108 |
| Rex      |     50 | German Shepherd     | Dog     | calm        |           50 |
| Gaga     |     10 | Parrot              | Bird    | calm        |          100 |
| Minnie   |     45 | Mouse               | Rodent  | lazy        |           50 |
| Tina     |      5 | Chihuahua           | Dog     | calm        |           50 |
| Fluffy   |     15 | Tabby               | Cat     | lazy        |           10 |
| Aldus    |     12 | Ragdoll             | Cat     | calm        |           37 |
| Deane    |      9 | Persian             | Cat     | calm        |          9.3 |
| Polly    |     10 | Parrot              | Bird    | hyper       |          120 |
| Bunny    |     80 | Greyhound           | Dog     | lazy        |          100 |
| Nugget   |     20 | Ragdoll             | Cat     | calm        |           50 |
| Geico    |      1 | Gecko               | Lizard  | hyper       |            2 |
| Aster    |      2 | Parakeet            | Bird    | calm        |           10 |
| Tuna     |     10 | Dachshund           | Dog     | lazy        |           20 |
| Splinter |      5 | Rat                 | Rodent  | calm        |            2 |
| Tristam  |     15 | Pug                 | Dog     | hyper       |           91 |
| Max      |     60 | Great Dane          | Dog     | hyper       |           50 |
| Goofy    |    100 | Great Dane          | Dog     | calm        |          100 |
| Mocha    |     11 | Shorthair           | Cat     | calm        |           20 |
| Kai      |     50 | German Shepherd     | Dog     | lazy        |          100 |
| Tony     |     10 | Tabby               | Cat     | hyper       |           20 |
| Pogo     |     15 | Pug                 | Dog     | calm        |           50 |
| Odin     |     40 | Mastiff             | Dog     | calm        |          100 |
| Ru       |     20 | Australian Shepherd | Dog     | calm        |           60 |
| Pookie   |     20 | Pomeranian          | Dog     | calm        |          100 |
| Mickey   |     50 | Mouse               | Rodent  | calm        |           50 |
| Colgate  |      3 | Rat                 | Rodent  | calm        |            5 |
| Donald   |     10 | Duck                | Bird    | calm        |           20 |
| Boba     |     25 | Shiba Inu           | Dog     | lazy        |          200 |
| Beyonce  |     10 | Ragdoll             | Cat     | hyper       |          100 |
| Malcolm  |     10 | British Shorthair   | Cat     | calm        |          9.3 |
+----------+--------+---------------------+---------+-------------+--------------+
31 rows in set (0.00 sec)
```

3. This table joins the Animals table, Employees table, and Cares_For table to show which Employee is taking care of which Animal.

```
ALTER VIEW Care_Log AS

SELECT Animals.Name AS Animal, Animals.ID AS Animal_ID,

Employee.FirstName AS Employee_FName,

Employee.LastName AS Employee_lName,

Employee.ID AS Employee_ID

FROM Animals

INNER JOIN Cares_For ON Animals.ID  = Cares_For.Animal_ID

INNER JOIN Employee ON Cares_For.Employee_ID = Employee.ID;
```




```
SELECT * FROM Care_Log;
```

```
MariaDB [pgarcia]> SELECT * FROM Care_Log
    -> ;
+----------+-----------+---------------+---------------+-------------+
| Animal   | Animal_ID | Employee_FName | Employee_lName | Employee_ID |
+----------+-----------+---------------+---------------+-------------+
| Gaga     | 24c03f5e  | Zed           | Label         | 0e9aae59    |
| Deane    | 48b52db0  | Zed           | Label         | 0e9aae59    |
| Peeko    | 6fb35b21  | Zed           | Label         | 0e9aae59    |
| Splinter | 7e00806d  | Zed           | Label         | 0e9aae59    |
| Milo     | 2164cf62  | Ardra         | Scammell      | 2f49929e    |
| Rochella | 3774615e  | Ardra         | Scammell      | 2f49929e    |
| Aster    | 6cfeb41c  | Ardra         | Scammell      | 2f49929e    |
| Pookie   | c0bbbaff  | Ardra         | Scammell      | 2f49929e    |
| Cesar    | 8d964852  | Oriana        | Gabbott       | 3c007748    |
| Colgate  | ced9c24f  | Oriana        | Gabbott       | 3c007748    |
| Bunny    | 62ec34cd  | Frank         | Cuchey        | 4af64d6f    |
| Tony     | ac64d184  | Frank         | Cuchey        | 4af64d6f    |
| Giffie   | 0f1f6580  | Emp           | Yee           | 637482a     |
| Milo     | 2164cf62  | Emp           | Yee           | 637482a     |
| Giffie   | 0f1f6580  | Hillary       | Horley        | 69ff2334    |
| Donald   | ed23444f  | Hillary       | Horley        | 69ff2334    |
| Daisy    | 11f324de  | Jay           | Haryngton     | 6cf6d551    |
| Mocha    | 92ba8c6a  | Jay           | Haryngton     | 6cf6d551    |
| Janessa  | fc29b8b1  | Jay           | Haryngton     | 6cf6d551    |
| Star     | 52bd85c3  | Sebastian     | Bernardino    | 6dd330a5    |
| Pogo     | bac26383  | Sebastian     | Bernardino    | 6dd330a5    |
| Mickey   | ccd2efca  | Sebastian     | Bernardino    | 6dd330a5    |
| Hashim   | e1d35a33  | Sebastian     | Bernardino    | 6dd330a5    |
| Tina     | 2f543a30  | Shaylynn      | Murkus        | 72e63041    |
| Aldus    | 476f5c09  | Shaylynn      | Murkus        | 72e63041    |
| Oreo     | 8fd44a22  | Shaylynn      | Murkus        | 72e63041    |
| Goofy    | 905694ad  | Shaylynn      | Murkus        | 72e63041    |
| Fluffy   | 380dc00c  | Orbadiah      | Batterham     | 860e64a8    |
| Max      | 8bb114f4  | Orbadiah      | Batterham     | 860e64a8    |
| Malcolm  | feb847ac  | Orbadiah      | Batterham     | 860e64a8    |
| Minnie   | 2cfcb813  | Petronella    | de Clerc      | 90deb728    |
| Tristam  | 807cb21d  | Petronella    | de Clerc      | 90deb728    |
| Ru       | be697dc0  | Petronella    | de Clerc      | 90deb728    |
+----------+-----------+---------------+---------------+-------------+
33 rows in set (0.00 sec)
```

4.3.2 - Stored procedures/functions

1. This stored procedure will insert a new animal entry into the Animals table.

```
DELIMITER //

CREATE PROCEDURE New_Animal (a_name text, a_dob datetime,
a_sex text, a_species text, a_breed text, a_color text,
a_pattern text, a_altered text, a_weight float, a_id
varchar(20), a_sheltersection int, a_fee float,
a_euthanized text, a_dateofeuthanization datetime,
a_arrivaldate datetime)

BEGIN

INSERT INTO Animals (Name, DOB, Sex, Species, Breed, Color,
Pattern, Altered, Weight, ID, Shelter_Section,
Adoption_Fee, Euthanized, Date_of_Euthanization,
Arrival_Date)

VALUES (a_name, a_dob, a_sex, a_species, a_breed, a_color,
a_pattern, a_altered, a_weight, a_id, a_sheltersection,
a_fee, a_euthanized, a_dateofeuthanization, a_arrivaldate);

END;

//

DELIMITER ;
```

Before calling procedure:

```
SELECT Name, ID, Species FROM Animals

WHERE Name = “New Animal”;
```

```
MariaDB [pgarcia]> SELECT Name, ID, Species FROM Animals
    -> WHERE Name = "New Animal";
Empty set (0.00 sec)
```

Calling procedure:

```
CALL New_Animal("New Animal", '2010-02-01 00:00:00', "M",
"Dog", "Pug", "Tan", "Solid", "T", 15.0, '123456', 1, 25.0,
"F", NULL, '2021-05-04 00:00:00');
```

```
MariaDB [pgarcia]> CALL New_Animal("New Animal", '2010-02-01 00:00:00', "M", "Do
g", "Pug", "Tan", "Solid", "T", 15.0, '1234567', 1, 25.0, "F", NULL, '2021-05-04
 00:00:00');
Query OK, 1 row affected (0.00 sec)
```

After calling procedure:

```
SELECT Name, ID, Species FROM Animals

WHERE Name = "New Animal";
```

```
MariaDB [pgarcia]> SELECT Name, ID, Species From Animals
    -> WHERE Name = "New Animal";
+------------+--------+---------+
| Name       | ID     | Species |
+------------+--------+---------+
| New Animal | 123456 | Dog     |
+------------+--------+---------+
1 row in set (0.00 sec)
```

2. This stored procedure is used for deleting an existing animal in the Animals table.

```
DELIMITER //

CREATE PROCEDURE Remove_Animal (a_id varchar(20))

BEGIN

DELETE FROM Animals WHERE ID = a_id;

END;

//

DELIMITER ;
```

Before calling procedure:

```
SELECT Name, ID, Species FROM Animals

WHERE Name = “New Animal”;
```

```
MariaDB [pgarcia]> SELECT Name, ID, Species From Animals
    -> WHERE Name = "New Animal";
+------------+--------+---------+
| Name       | ID     | Species |
+------------+--------+---------+
| New Animal | 123456 | Dog     |
+------------+--------+---------+
1 row in set (0.00 sec)
```

Calling procedure:

```
CALL Remove_Animal(123456);
```

```
MariaDB [pgarcia]> CALL Remove_Animal(123456);
Query OK, 1 row affected (0.01 sec)

MariaDB [pgarcia]>
```

After calling procedure:

```
SELECT Name, ID, Species FROM Animals

WHERE Name = “New Animal”;
```

```
MariaDB [pgarcia]> SELECT Name, ID, Species FROM Animals
    -> WHERE Name = "New Animal";
Empty set (0.00 sec)
```

3. This stored procedure returns the amount of adoption fees accumulated within a specified date.

```
DELIMITER //


CREATE PROCEDURE Yearly_Fees (s_date datetime, e_date
datetime)
```

```
BEGIN
SELECT SUM(Amount) FROM Adopter
INNER JOIN Adopted ON Adopter.ID = Adopted.Adopter_ID
WHERE Adopted.Taken >= s_date AND Adopted.Taken <= e_date;
END;
//
DELIMITER ;
```

List of fees between the years 2010 and 2015:

```
SELECT Adopter.Amount, Adopted.Taken FROM Adopter
INNER JOIN Adopted ON Adopter.ID = Adopted.Adopter_ID
WHERE Adopted.Taken >= '2010-01-01 00:00:00' AND
Adopted.Taken <= '2015-01-01 00:00:00';
```

```
MariaDB [pgarcia]> SELECT Adopter.Amount, Adopted.Taken FROM Adopter INNER JOIN Adopt
ed ON Adopter.ID = Adopted.Adopter_ID WHERE Adopted.Taken >= '2010-01-01 00:00:00' AN
D Adopted.Taken <= '2015-01-01 00:00:00';
+--------+---------------------+
| Amount | Taken               |
+--------+---------------------+
|    149 | 2010-12-20 12:31:00 |
|     41 | 2010-02-28 03:25:00 |
|    196 | 2012-06-18 09:43:00 |
|     54 | 2011-01-02 09:31:00 |
+--------+---------------------+
4 rows in set (0.00 sec)
```

Calling procedure:

```
CALL Yearly_Fees('2010-01-01 00:00:00', '2015-01-01
00:00:00);
```

```
MariaDB [pgarcia]> CALL Yearly_Fees('2010-01-01 00:00:00', '2015-01-01 00:00:00'
);
+-------------+
| SUM(Amount) |
+-------------+
|         440 |
+-------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

4.3.3 - Triggers

1. This trigger gets rid of all connecting information attached to an animal if the animal is deleted from the Animals table

```
DELIMITER //

CREATE TRIGGER Animal_Deleted

BEFORE DELETE ON Animals

FOR EACH ROW

BEGIN

DELETE FROM Adopted WHERE Adopted.Animal_ID = old.ID;

DELETE FROM Health_Record WHERE Health_Record.Animal_ID =
old.ID;

DELETE FROM Cares_For WHERE Cares_For.Animal_ID = old.ID;

DELETE FROM Status WHERE Status.Animal_ID = old.ID;

UPDATE Shelter_Section SET Shelter_Section.Spaces_Left =

Shelter_Section.Spaces_Left+1 WHERE

Shelter_Section.Room_Number = old.Shelter_Section;

END

//
```

```
DELIMITER;
```

BEFORE TRIGGER:

```
MariaDB [pgarcia]> SELECT Name, ID From Animals WHERE ID='1234567';
+------------+---------+
| Name       | ID      |
+------------+---------+
| New Animal | 1234567 |
+------------+---------+
1 row in set (0.00 sec)
```

Showing animal named New Animal exists in Animals table

```
MariaDB [pgarcia]> SELECT * FROM Health_Record WHERE Animal_ID='1234567';
+------+--------+---------------+-----------------------+-----------+
| Cond | Hist   | Special_Needs | Serial_Number         | Animal_ID |
+------+--------+---------------+-----------------------+-----------+
| Good | Abused | NULL          | 11111-111111-11-111-11 | 1234567   |
+------+--------+---------------+-----------------------+-----------+
1 row in set (0.00 sec)
```

Showing that there is an entry in Health_Record with New Animals ID

TRIGGER ACTIVATION:

```
MariaDB [pgarcia]> DELETE FROM Animals WHERE ID='1234567';
Query OK, 1 row affected (0.00 sec)
```

Deleting the New Animal entry in Animals to activate trigger

AFTER TRIGGER:

```
MariaDB [pgarcia]> SELECT Name, ID From Animals WHERE ID='1234567';
Empty set (0.00 sec)
```

Showing New Animal is no longer in Animals table

```
MariaDB [pgarcia]> SELECT * FROM Health_Record WHERE Animal_ID='1234567';
Empty set (0.00 sec)
```

Showing Health_Record entry for New Animal is also deleted.

2. This trigger updates the Shelter_Section tables when a new animal is added to the Animals table.

```
DELIMITER //

CREATE TRIGGER Animal_Update Animals

AFTER INSERT ON Animals

FOR EACH ROW

BEGIN

UPDATE SHelter_Section SET Shelter_Section.Spaces_Left =
Shelter_Sextion.Space_Left-1 WHERE
Shelter_Section.Room_Number = new.Shelter_Section;

END

//

DELIMITER;
```

BEFORE TRIGGER:

```
MariaDB [pgarcia]> SELECT Name, ID From Animals WHERE Name="Animal";
Empty set (0.01 sec)
```

Showing animal named Animal does not exist in Animals table

```
MariaDB [pgarcia]> SELECT Spaces_Left FROM Shelter_Section WHERE Room_Number = 1;
+-------------+
| Spaces_Left |
+-------------+
|           3 |
+-------------+
1 row in set (0.00 sec)
```

Showing the number of spaces left in room number 1

TRIGGER ACTIVATION:

```
MariaDB [pgarcia]> CALL New_Animal("Animal", '2010-02-01 00:00:00', "M", "Dog", "Pug", "Tan", "Solid", "T", 15.
0, '1234567', 1, 25.0, "F", NULL, '2021-05-04 00:00:00');
Query OK, 1 row affected (0.01 sec)
```

Inserting Animal into Animals to activate trigger

AFTER TRIGGER:

```
MariaDB [pgarcia]> SELECT Name, ID, Shelter_Section FROM Animals WHERE Name="Animal";
+--------+---------+-----------------+
| Name   | ID      | Shelter_Section |
+--------+---------+-----------------+
| Animal | 1234567 |               1 |
+--------+---------+-----------------+
1 row in set (0.00 sec)
```

Showing Animal is now in Animals table in shelter_section 1

```
MariaDB [pgarcia]> SELECT Spaces_Left FROM Shelter_Section WHERE Room_Number=1;
+-------------+
| Spaces_Left |
+-------------+
|           2 |
+-------------+
1 row in set (0.00 sec)
```

Showing number of spaces left in room 1 has gone down by 1

3. For this trigger, when a new employee is added to the Employee table, a new Supervisor entry will also be inserted

```
DELIMITER //

CREATE TRIGGER New_Employee

AFTER INSERT ON Employee

FOR EACH ROW

BEGIN

DECLARE id varchar(20);

DECLARE s_id varchar(20);

SET id = new.ID;

INSERT INTO Supervisor(Employee_ID, Supervisor) VALUES (id,
NULL);

END

//

DELIMITER ;
```

BEFORE TRIGGER:

```
MariaDB [pgarcia]> SELECT firstName, lastName, ID FROM Employee WHERE firstName="Janna";
Empty set (0.01 sec)
```

Showing employee named Janna does not exist in Employees table

```
MariaDB [pgarcia]> SELECT * FROM Supervisor;
+-------------+------------+
| Employee_ID | Supervisor |
+-------------+------------+
| 0e9aae59    | 6dd330a5   |
| 2f49929e    | 6dd330a5   |
| 3c007748    | 6dd330a5   |
| 4af64d6f    | 6dd330a5   |
| 69ff2334    | 6dd330a5   |
| 6cf6d551    | 6dd330a5   |
| 6dd330a5    |            |
| 72e63041    | 6dd330a5   |
| 860e64a8    | 6dd330a5   |
| 90deb728    | 6dd330a5   |
+-------------+------------+
10 rows in set (0.00 sec)
```

Showing that there are currently 10 rows in Supervisor

TRIGGER ACTIVATION:

```
MariaDB [pgarcia]> DELIMITER ;
MariaDB [pgarcia]> INSERT INTO Employee (firstName, middleName, lastName, street
Name, houseNumber, zipCode, city, state, phoneNumberl, ID, SSN, email, payment,
startDate, endDate, pass, authority) VALUES("Janna", "Little", "Smith", "Hazelme
re", 1500, 93311, "Bakersfield", "California", "661-397-4813", "1234567", "123-4
5-6789", "fake@email.com", 15, "NULL", "NULL", "passwordlll", 0);
Query OK, 1 row affected, 2 warnings (0.00 sec)
```

Inserting a new employee into the Employee table

AFTER TRIGGER:

```
MariaDB [pgarcia]> SELECT firstName, ID FROM Employee WHERE firstName="Janna";
+-----------+---------+
| firstName | ID      |
+-----------+---------+
| Janna     | 1234567 |
+-----------+---------+
1 row in set (0.00 sec)
```

Showing Animal is now in Animals table in shelter_section 1

```
MariaDB [pgarcia]> SELECT * FROM Supervisor;
+-------------+------------+
| Employee_ID | Supervisor |
+-------------+------------+
| 0e9aae59    | 6dd330a5   |
| 1234567     | NULL       |
| 2f49929e    | 6dd330a5   |
| 3c007748    | 6dd330a5   |
| 4af64d6f    | 6dd330a5   |
| 69ff2334    | 6dd330a5   |
| 6cf6d551    | 6dd330a5   |
| 6dd330a5    |            |
| 72e63041    | 6dd330a5   |
| 860e64a8    | 6dd330a5   |
| 90deb728    | 6dd330a5   |
+-------------+------------+
11 rows in set (0.00 sec)
```

Showing that there is a new entry in Supervisor with the new employee's ID

5.1 - GUI Functionalities and User Groups

5.1.1 - Itemized Descriptions of the GUI

Groups:

Adopters: Adopters will have the least amount of access. They will only be able to access the view Adoptable_Animals along with other pages of the website that are not required to be logged in.

Employees: An employee must be logged in to access their controls. An employee can see their own information and edit it as they please. They can also see all the

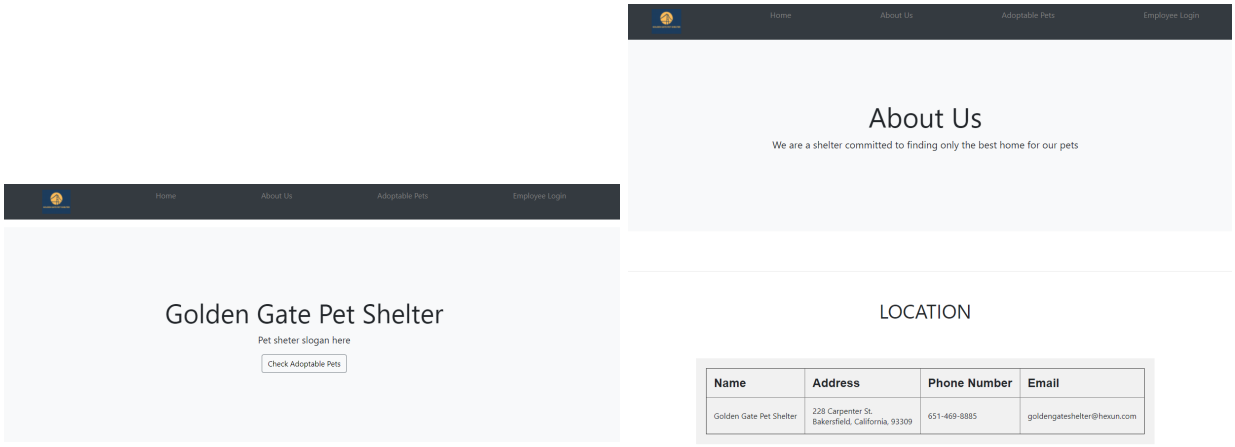
information of the animals which includes the information on the Status, Health_Record, and Shelter_Section tables. An employee will also be able to see which animals they are in charge of and see the information of the adopters. Employees can also insert new entries into the Animals table.

Manager: Managers must also be logged in to access their controls. A manager can do anything an employee can and more. A manager can assign which employee takes care of which animal. They also have access to limited information about all employees.

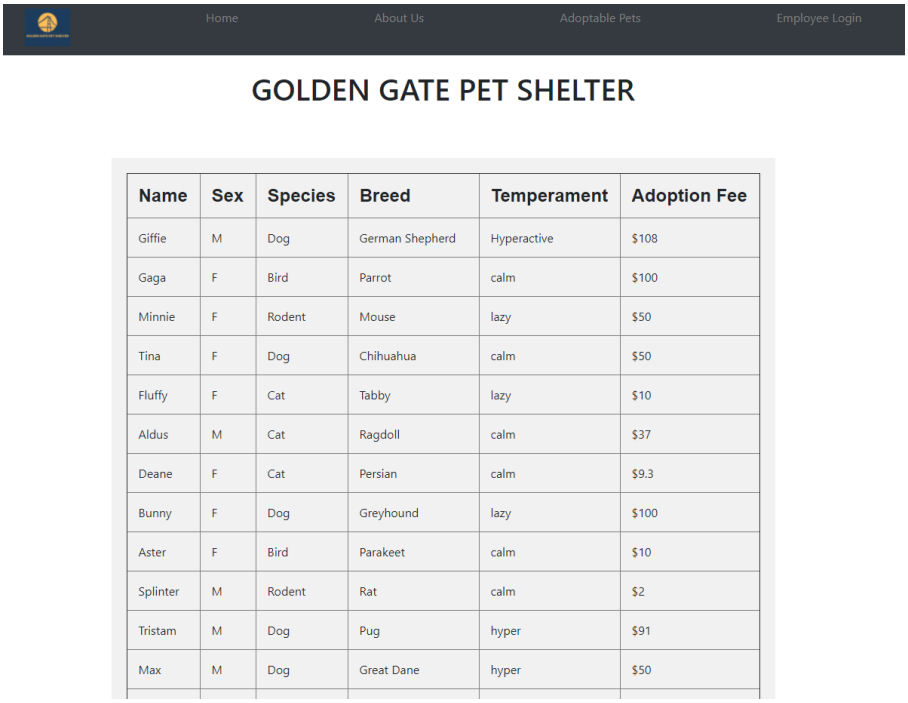
Supervisor: Supervisor must also be logged in to access their controls. A supervisor will be able to do anything the manager is able to do and more. A supervisor will also be able to promote employees and insert new employees into the employees table. He will also be able to make invoices of the money made through adoption fees.

5.1.2 - Screenshots and Walkthrough

Adopter:



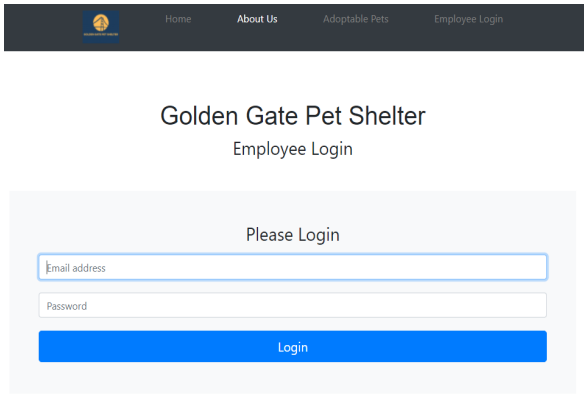
These are the home page and About Us page. Anybody can access these pages. The About Us page displays the info about the location of the pet shelter from the database.

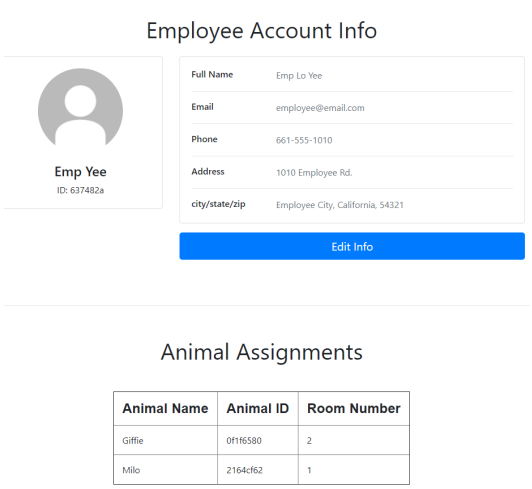


Name	Sex	Species	Breed	Temperament	Adoption Fee
Giffie	M	Dog	German Shepherd	Hyperactive	$108
Gaga	F	Bird	Parrot	calm	$100
Minnie	F	Rodent	Mouse	lazy	$50
Tina	F	Dog	Chihuahua	calm	$50
Fluffy	F	Cat	Tabby	lazy	$10
Aldus	M	Cat	Ragdoll	calm	$37
Deane	F	Cat	Persian	calm	$9.3
Bunny	F	Dog	Greyhound	lazy	$100
Aster	F	Bird	Parakeet	calm	$10
Splinter	M	Rodent	Rat	calm	$2
Tristam	M	Dog	Pug	hyper	$91
Max	M	Dog	Great Dane	hyper	$50

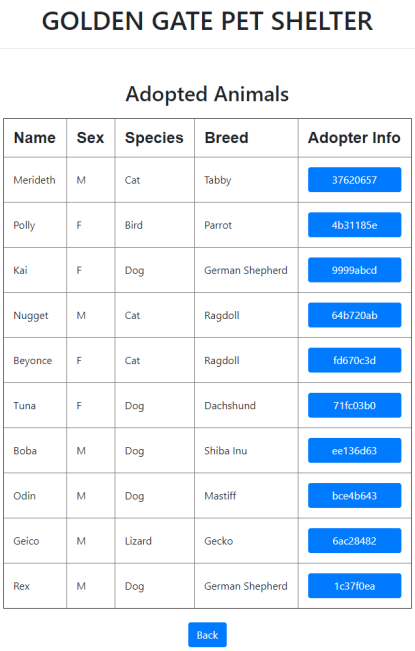
*Everybody also has access to the Adoptable Animals page. This shows the animals in the shelter that are adoptable by using the view that was in the database called Animals_In_Shelter.

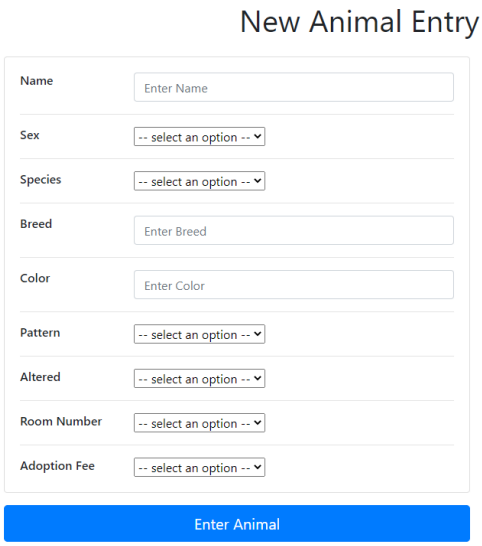
Employee:

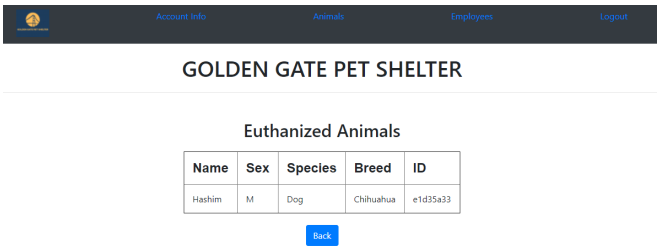


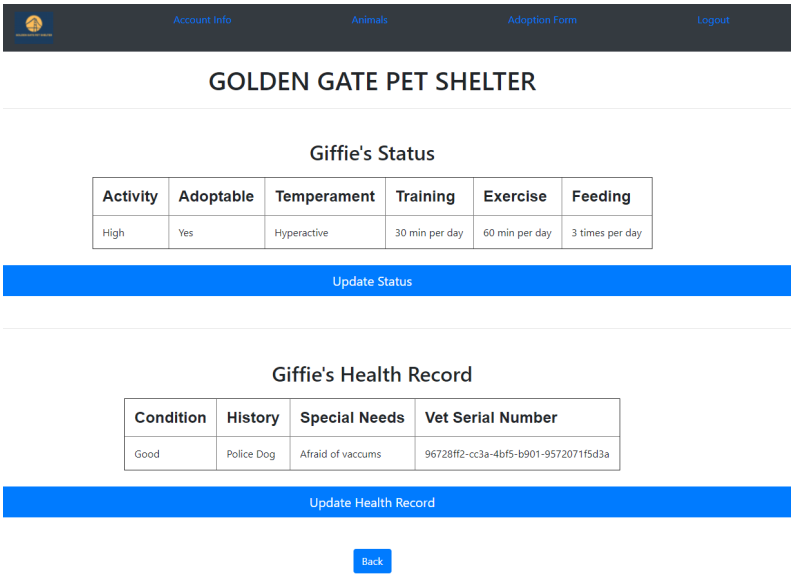


*These are the login page and the main page after being logged in. Everyone can see the login page but only the employee, manager, or supervisor can log in. The main page displays the information of the person that has logged in along with the animals in the shelter they are assigned to take care of. The employee may also edit their information.

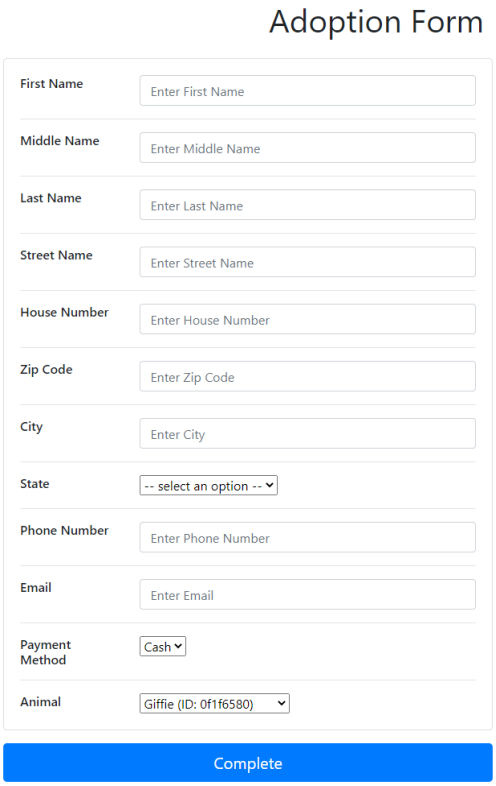


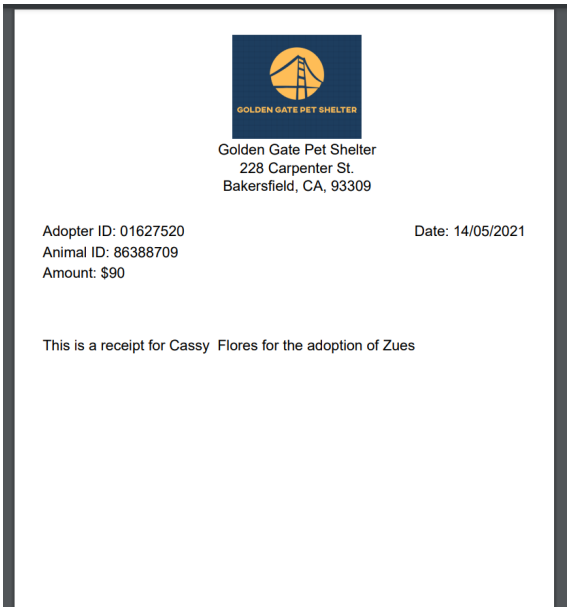






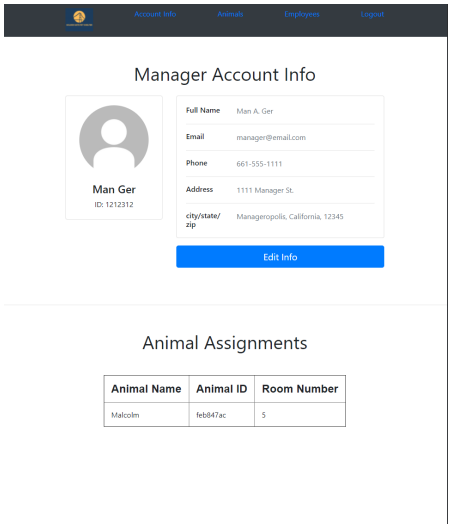
*Employees will be able to access the Animals page which displays all the animals in the shelter regardless if they are adoptable or not. From here they can access the New Animal Entry page, Adopted Animals page, Euthanized Animals page, and the page that displays information about a specified animal. In the Information page the employee can update the information within those tables.

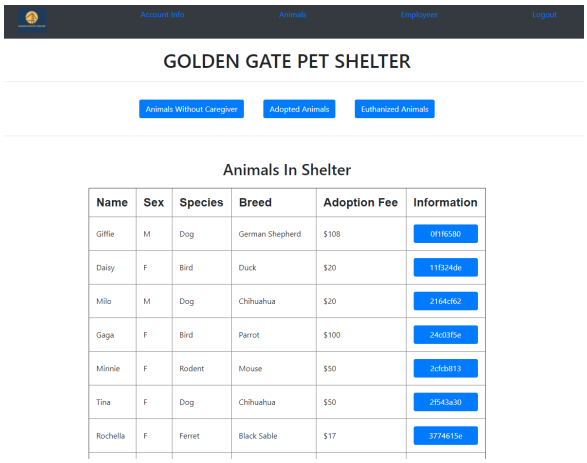


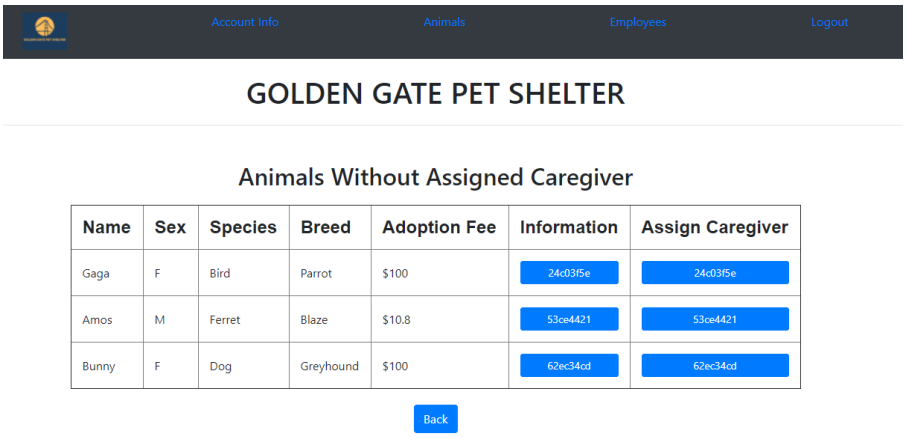


*An employee can also fill out adoption forms which enter new information to the adopter and adopted table and create receipts for the adopter.

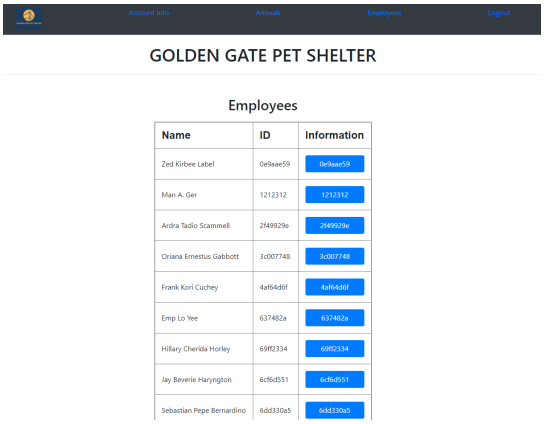
Manager:

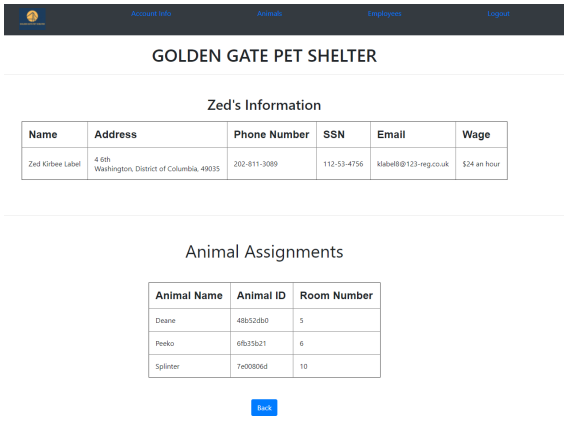






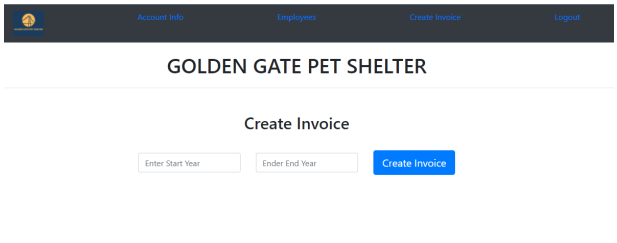
*Similar to the employee, a manager has access and change their own information and can check the animals in the shelter, animal information, adopted animals, and euthanized animals. The difference here is that they cannot enter a new animal but they can see which animals do not have an assigned caregiver. The manager can then assign a caregiver to those animals.





*A manager can also view the list of employees and view the information of a specified employee. The information page not only displays the employee information but also displays which animals they are assigned to care for.

Supervisor:



* The supervisor also has access to his own information like the employee and manager and the supervisor has access to the

employees list like the manager. The supervisor can also create invoices of the amount of money earned through adoptions between specified years.

5.1.3 - Demonstration of Programming Logic

Adopter:

The adopter view uses the following:

Adoptable_Animals View - used in the adoptable animals page to view adoptable animals.

Employee:

The employee view uses the following:

Animals_In_Shelter View - Used in the animals page to view animals in shelter.

Adoptable_Animals View - used in the adoption page to be able to choose an animal that is considered adoptable.

Care_Log View - Used in the home page to view animals users are assigned to care for.

Animal_Update Trigger - Used after an employee adds a new animal into the database.

Manager:

The manager view uses the following:

Animals_In_Shelter View - Used in the animals page to view animals in shelter.

Care_Log View - Used in the home page and employee information page to view animals the user and employees are assigned to care for.

Needs_Care View - Used in the Animals without an assigned caregiver page so that the manager can see what animals need to be assigned a caregiver.

Supervisor:

The supervisor uses the following:

Care_Log View - Used in the home page and employee information page to view animals the user and employees are assigned to care for.

Yearly_Fees Procedure - Used in the invoice page to generate the amount of money made from adoptions for the pdf.

New_Emplotee Trigger - Used after the supervisor enters a new employee into the database.

5.2 - GUI Programming

5.2.1 - Server-side Programming

Views:

Adoptable Animals - This view was made so that every user of the database can see what animals are available for adoption. It is used by the user groups Adopter and Employee through the adopterPets.php and adoption.php files respectively.

```php
$query = "SELECT * FROM Adoptable_Animals";
```

```php
if($result = $db->query($query)){
    while($row = $result->fetch_assoc())
    {
    $name = $row["Name"];
    $sex = $row["Sex"];
    $species = $row["Species"];
    $breed = $row["Breed"];
    $temperament = $row["Temperament"];
    $fee = $row["Adoption_Fee"];

    echo '<tr>
    <td>'.$name.'</td>
    <td>'.$sex.'</td>
    <td>'.$species.'</td>
    <td>'.$breed.'</td>
    <td>'.$temperament.'</td>
    <td> $'.$fee.'</td>
    </tr>';

    }
    $result->free();
}
```

```php
<?php
    $sql=mysqli_query($db, "SELECT Name, ID FROM Adoptable_Animals");
    while($row = $sql->fetch_assoc())
    {
        $oaid = $row['ID'];
        echo "<option value='".$oaid."'>".$row['Name']." (ID: ".$oaid.")</option>";
    }
?>
```

Animals_In_Shelter: This view shows what animals are currently in the shelter. It is used by the Employee and Manager user groups through animals.php.

```php
$query = "SELECT * FROM Animals_In_Shelter;";
```

```php
if($result = $db->query($query)){
```

```
    while($row = $result->fetch_assoc())
    {
    $name = $row["Name"];
    $sex = $row["Sex"];
    $species = $row["Species"];
    $breed = $row["Breed"];
    $fee = $row["Adoption_Fee"];
    $aid = $row["ID"];

    echo '<tr>
    <td>'.$name.'</td>
    <td>'.$sex.'</td>
    <td>'.$species.'</td>
    <td>'.$breed.'</td>
    <td> $'.$fee.'</td>
        <td> <input class="btn btn-md btn-primary btn-block" type="submit"
name="info" value='.$aid.'> </td>
    </tr>';

    }
    $result->free();
```

Care_log - Used in the home page and employee information page to view animals the user and employees are assigned to care for. It is used by the manager and supervisor user groups in the employee_info.php file.

```
$queryA = "SELECT Care_Log.Animal, Care_Log.Animal_ID, Animals.Shelter_Section
FROM Care_Log INNER JOIN Animals ON Care_Log.Animal_ID=Animals.ID WHERE Employee_ID='".$eid."'";
```

```
if($res = $db->query($queryA)){
    while($row = $res->fetch_assoc())
    {
    $name = $row["Animal"];
    $aid = $row["Animal_ID"];
    $rNum = $row["Shelter_Section"];

    echo '<tr>
        <td>'.$name.'</td>
        <td>'.$aid.'</td>
        <td>'.$rNum.'</td>
        </tr>';
    }
    $res->free();
}
```

Needs_Care - The purpose of this view is to help a manager assign animals caregivers. It is used by the manager group.

```
$query = "SELECT * FROM Needs_Care;";
```

```
if($result = $db->query($query)){
    while($row = $result->fetch_assoc())
    {
    $name = $row["Name"];
    $sex = $row["Sex"];
    $species = $row["Species"];
    $breed = $row["Breed"];
    $fee = $row["Adoption_Fee"];
    $aid = $row["ID"];

    echo '<tr>
    <td>'.$name.'</td>
    <td>'.$sex.'</td>
    <td>'.$species.'</td>
    <td>'.$breed.'</td>
    <td> $'.$fee.'</td>
    <td> <input type="submit" class="btn btn-sm btn-primary btn-block" name="info" value='.$aid.'> </td>
        <td> <input type="submit" class="btn btn-sm btn-primary btn-block" name="assign" value='.$aid.'></td>
    </tr>';

    }
```

Procedures:

Yearly_Fees: The purpose of this procedure is to generate the amount of money made from adoptions so that it can input in an invoice. It is used by the supervisor user group in create_invoice.php.

```
$result = mysqli_query($db, "CALL Yearly_Fees($sdate,$edate);");
if(mysql_num_rows($result)==0)
{
    $total = "NULL";
} else {
    $row = $result->fetch_assoc();
    $total = $row["Sum(Amount)"];
}
```

5.2.2 - Middle-tier Programming

Connecting to the database:

```
include("connect.php");

    if(mysqli_connect_errno())

    {

        printf("Connect failed: %s\n", mysqli_connect_error());

        exit();

    }
```

```
session_start();

    if(isset($_SESSION['id']) && $_SESSION['loggedin']==true &&
$_SESSION['auth']==0)

    {

    }else {

        header('Location: ../login.php');

    }


    include("connect.php");
```

```
<?php

$server = 'localhost';

$user = 'pgarcia';

$password = 'wif^3Fin';

$database = 'pgarcia';

$db = new mysqli($server, $user, $password, $database);

?>
```

Calling the Yearly_Fees procedure:

```
$result = mysqli_query($db, "CALL Yearly_Fees($sdate,$edate);");

    if(mysql_num_rows($result)==0)

    {

        $total = "NULL";

    } else {

        $row = $result->fetch_assoc();

        $total = $row["Sum(Amount)"];

    }
```

5.2.3 - Client-side Programming

This code searches for all instances of Animals in Animals_In_Shelter and returns the desired rows to be displayed.

```
if($result = $db->query($query)){
    while($row = $result->fetch_assoc())
    {
    $name = $row["Name"];
    $sex = $row["Sex"];
    $species = $row["Species"];
    $breed = $row["Breed"];
    $fee = $row["Adoption_Fee"];
    $aid = $row["ID"];

    echo '<tr>
    <td>'.$name.'</td>
    <td>'.$sex.'</td>
    <td>'.$species.'</td>
    <td>'.$breed.'</td>
    <td> $'.$fee.'</td>
        <td> <input class="btn btn-md btn-primary btn-block"
type="submit" name="info" value='.$aid.'> </td>
    </tr>';

    }
    $result->free();
```

Using the FPDF API:

```php
<?php
    session_start();
    include("../connect.php");
    require('../FPDF/fpdf.php');

    $date=date("d/m/Y");

    $sdate = $_SESSION['sdate'];
    $edate = $_SESSION['edate'];

    $sdate = $sdate."-01-01";
    $edate = $edate."-01-01";

    $time = "00:00:00";

    $sdate = $sdate." ".$time;
    $edate = $edate." ".$time;

    $sdate = strtotime("$sdate");
    $sdate = date('Y-m-d H:i:s', $sdate);

    $edate = strtotime("$edate");
    $edate = date('Y-m-d H:i:s', $edate);

    $result = mysqli_query($db, "CALL Yearly_Fees($sdate,$edate);");
    if(mysql_num_rows($result)==0)
    {
        $total = "NULL";
    } else {
        $row = $result->fetch_assoc();
        $total = $row["Sum(Amount)"];
    }

    class PDF extends FPDF
    {
        function Header()
        {
            $this->Image('../Images/GGPS.png',85,10,50);
            $this->SetFont('Arial','B',15);
            $this->Cell(80);
                $this->Ln(41);
        }
        function Footer()
        {
            $this->SetY(-15);
            $this->SetFont('Arial','',16);
            $this->Cell(0,10,'Employee ID: '.$eid.'',0,1);
        }
    }

    $pdf = new PDF();
    $pdf->AddPage();
    $pdf->SetFont('Arial','',16);
    $pdf->Cell(200,7,'Golden Gate Pet Shelter',0,1,'C');
    $pdf->Cell(200,7,'228 Carpenter St.',0,1,'C');
    $pdf->Cell(200,7,'Bakersfield, CA, 93309',0,1,'C');
    $pdf->Ln(10);
    $pdf->Cell(0,8,'Start Date: '.$sdate.'',0,0);
    $pdf->Cell(0,8,'Date: '.$date.'',0,1,'R');
    $pdf->Cell(0,8,'End Date: '.$edate.'',0,1);
    $pdf->Cell(0,8,'Total: $'.$total.'',0,1);
    $pdf->Ln(20);
    $pdf->Cell(0,8,'The total amount of money earned through the dates:',0,1);
        $pdf->Cell(0,8,''.$sdate.' and '.$edate.'',0,1);
        $pdf->Cell(0,8,'is $'.$total.'',0,1);
    $pdf->Output();
?>
```

The code is validating that the insertion of the newAnimal is valid when the header is reached.

```
if($_SERVER["REQUEST_METHOD"] == "POST")

    {

        $name = $_POST['name'];

        $species = $_POST['species'];

        $sex = $_POST['sex'];

        $breed = $_POST['breed'];

        $color = $_POST['color'];

        $pat = $_POST['pat'];

        $alt = $_POST['alt'];

        $rnum = $_POST['rnum'];

        $fee = $_POST['fee'];

        $DOB = NULL;

        $weight = NULL;

        $euth = "No";

        $arrive = NULL;

        $id="";

        for($i=0; $i<8; $i++)

        {

        $id .= rand(0,9);

        }

        mysqli_query($db, "INSERT INTO Animals SET Name='".$name."',
DOB='".$DOB."',  Sex='".$sex."', Species='".$species."',
Breed='".$breed."', Color='".$color."', Pattern='".$pat."',
Altered='".$alt."', Weight='".$weight."', ID='".$id."',
Shelter_Section='".$rnum."', Adoption_Fee='".$fee."',
Euthanized='".$euth."', Date_of_Euthanization=NULL,
Arrival_Date='".$arrive."'");

        header('Location: success.php');
```

```
}
```