CS353 GROUP 32
PROJECT DESIGN REPORT

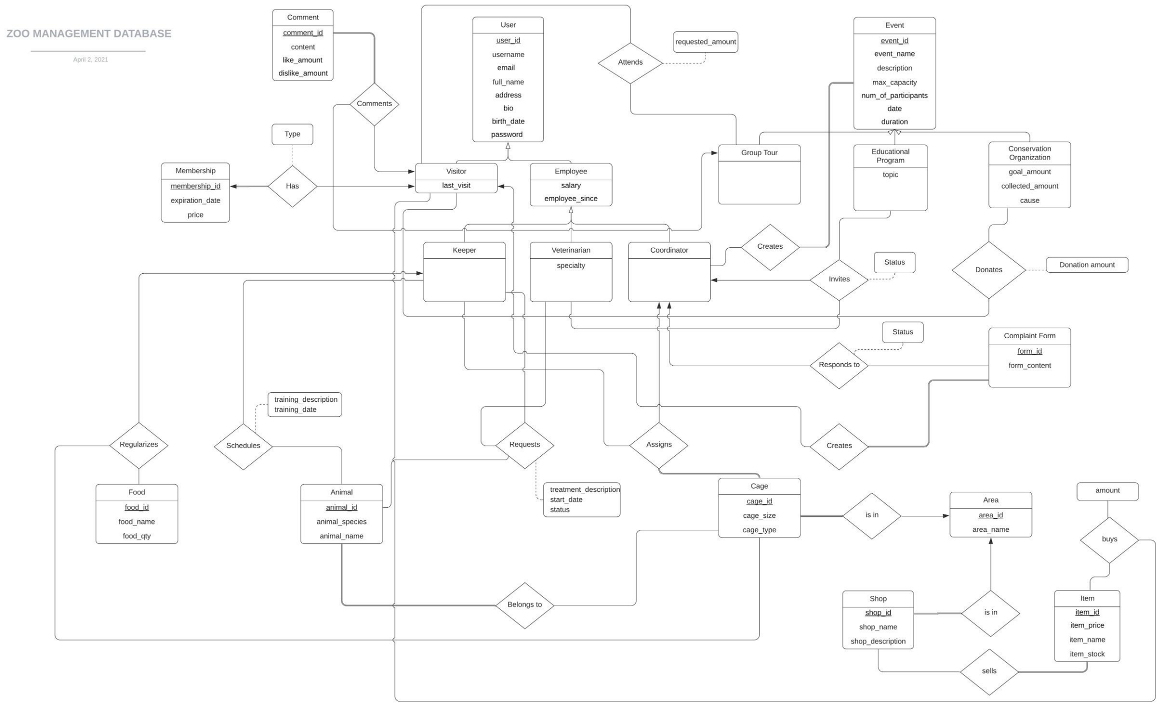
ZOO DATABASE MANAGEMENT SYSTEM

Group Members

Ramazan Melih DİKSU - 21802361
Aleyna SÜTBAŞ - 21803174
Berk TAKIT - 21803147
Sarp TEKIN - 21600813

Table Of Contents

1.Revised ER Model



URL:https://drive.google.com/file/d/1Eos9UcCn2DiG2I-L91aedpJz-4GtNwfA/view?usp=sharing

2.RELATIONAL SCHEMAS

2.1 User

- Relational Model:
  User(user_id, username, email, full_name, address, bio, birth_date,password)

- Functional Dependencies:
  user_id, username, email → full_name, address, bio, birth_date, password

- Candidate Keys:
  {(user_id), (username), (email)}

- Normal Form:
  BCNF

- Table Definition:

  ```
  CREATE TABLE User(
  user_id INT AUTO_INCREMENT,
  username VARCHAR(50) NOT NULL UNIQUE,
  email VARCHAR(50) NOT NULL UNIQUE,
  full_name VARCHAR(50) NOT NULL,
  address VARCHAR(50) NOT NULL,
  bio VARCHAR(100) NOT NULL,
  birth_date DATE NOT NULL,
  password VARCHAR(50) NOT NULL,
  PRIMARY KEY (user_id)
  );
  ```

2.2 Visitor

- Relational Model:
  Visitor(visitor_id, last_visit)

- Functional Dependencies:
  visitor_id → last_visit

- Candidate Keys:
  {(visitor_id)}

- Normal Form:
  BCNF

- Table Definition:
  ```
  CREATE TABLE Visitor(
  visitor_id INT,
  last_visit DATE,
  PRIMARY KEY (visitor_id),
  FOREIGN KEY (visitor_id) REFERENCES User(user_id)
  );
  ```

2.3 Employee

- Relational Model:
  Employee(employee_id, salary, employee_since)

- Functional Dependencies:
  employee_id → salary, employee_since

- Candidate Keys:
  {(employee_id)}

- Normal Form:
  BCNF

- Table Definition:
  CREATE TABLE Employee(
  employee_id **INT**,
  salary **INT NOT NULL,**
  employee_since **DATE NOT NULL,**
  PRIMARY KEY (employee_id),
  FOREIGN KEY (employee_id) **REFERENCES** User(user_id));

2.4 Keeper

- Relational Model:
  Keeper(keeper_id)

- Functional Dependencies:
  None

- Candidate Keys:
  {(keeper_id)}

- Normal Form:
  BCNF

- Table Definition:
  ```
  CREATE TABLE Keeper(
  keeper_id INT,
  PRIMARY KEY (keeper_id),
  FOREIGN KEY (keeper_id) REFERENCES Employee(employee_id));
  ```

2.5 Veterinarian

- Relational Model:
  Veterinarian(veterinarian_id, speciality)

- Functional Dependencies:
  veterinarian_id → speciality

- Candidate Keys:
  {(veterinarian_id)}

- Normal Form:
  BCNF

- Table Definition:
  CREATE TABLE Veterinarian(
  veterinarian_id **INT**,
  speciality **VARCHAR(50) NOT NULL**,
  PRIMARY KEY (veterinarian_id),
  FOREIGN KEY (veterinarian_id) **REFERENCES** Employee(employee_id));

2.6 Coordinator

- Relational Model:
  Coordinator(coordinator_id)

- Functional Dependencies:
  None

- Candidate Keys:
  {(coordinator_id)}

- Normal Form:
  BCNF

- Table Definition:
  CREATE TABLE Coordinator(
  PRIMARY KEY (coordinator_id),
  FOREIGN KEY (coordinator_id) **REFERENCES** Employee(employee_id));

2.7 Comment

- Relational Model:
  Comment(comment_id, content, like_amount, dislike_amount)

- Functional Dependencies:
  comment_id → content, like_amount, dislike_amount

- Candidate Keys:
  {(comment_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Comment(
comment_id INT AUTO_INCREMENT,
content VARCHAR(1000),
like_amount INT NOT NULL,
dislike_amount INT NOT NULL,
PRIMARY KEY (comment_id)
);
```

2.8 Membership

- Relational Model:
  Membership(membership_id,expiration_date, price)

- Functional Dependencies:
  membership_id → expiration_date, price

- Candidate Keys:
  {(membership_id)}

- Normal Form:
  BCNF

- Table Definition:

  ```
  create table Membership(
  membership_id INT AUTO_INCREMENT,
  expiration_date  DATE NOT NULL,
  price INT NOT NULL,
  PRIMARY KEY (membership_id)
  );
  ```

2.9 Comments

- Relational Model:
  Comments(comment_id, user_id, groupTour_id)

- Functional Dependencies:
  comment_id → user_id, groupTour_id

- Candidate Keys:
  {(comment_id)}

- Normal Form:
  BCNF

- Table Definition:
  create table Comment(
  comment_id **INT,**
  user_id **INT,**
  groupTour_id **INT,**
  PRIMARY KEY (comment_id),
  FOREIGN KEY (comment_id) **REFERENCES** Comment(comment_id),
  FOREIGN KEY (user_id) **REFERENCES** Visitor(user_id),
  FOREIGN KEY (groupTour_id) **REFERENCES** GroupTour(event_id));

2.10 Has

- Relational Model:
  Has(membership_id, user_id, type)

- Functional Dependencies:
  membership_id, user_id → type

- Candidate Keys:
  {(membership_id), (user_id)}

- Normal Form:
  BCNF

- Table Definition:
  ```
  create table Has(
  membership_id INT,
  user_id INT,
  type VARCHAR(50) NOT NULL,
  PRIMARY KEY (membership_id),
  FOREIGN KEY (membership_id) REFERENCES
  Membership(membership_id),
  FOREIGN KEY (user_id) REFERENCES Visitor(user_id)
  );
  ```

2.11 Event

- Relational Model:
  Event(event_id, description, max_capacity, num_of_participants, date, duration)

- Functional Dependencies:
  event_id → description, max_capacity, num_of_participants, date, duration

- Candidate Keys:
  {(event_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Event(
        event_id INT AUTO INCREMENT,
        event_name VARCHAR (25) NOT NULL,
        description VARCHAR(1000) NOT NULL,
        max_capacity INT NOT NULL,
        num_of_participants INT NOT NULL DEFAULT 0,
        date DATE NOT NULL, duration INT NOT NULL,
        PRIMARY KEY(event_id));
```

2.12 Group Tour

- Relational Model:
  Group_Tour(group_tour_id)

- Functional Dependencies:
  None

- Candidate Keys:
  {(group_tour_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Group_Tour(
        group_tour_id INT,
        PRIMARY KEY(group_tour_id),
        FOREIGN KEY (group_tour_id) REFERENCES Event(event_id));
```

2.13 Educational Program

- Relational Model:
  Educational_Program(edu_prog_id, topic)

- Functional Dependencies:
  edu_prog_id → topic

- Candidate Keys:
  {(edu_prog_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Educational_Program(
        edu_prog_id INT,
        topic VARCHAR(20) NOT NULL,
        PRIMARY KEY(edu_prog_id),
        FOREIGN KEY (edu_prog_id) REFERENCES Event(event_id));
```

2.14 Conservation Organization

- Relational Model:
  Educational_Program(con_org_id, goal_amount, collected_amount, cause)

- Functional Dependencies:
  con_org_id → goal_amount, collected_amount, cause

- Candidate Keys:
  {(con_org_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Conservation_Organization(
        con_org_id INT,
        goal_amount INT NOT NULL,
        collected_amount INT NOT NULL DEFAULT  0,
        cause VARCHAR(50) NOT NULL,
        PRIMARY KEY(con_org_id),
        FOREIGN KEY (con_org_id) REFERENCES Event(event_id));
```

2.13 Attends

- Relational Model:
  Attends(visitor_id, group_tour_id, requested_amount)

- Functional Dependencies:
  visitor_id, group_tour_id → requested_amount

- Candidate Keys:
  {(visitor_id, group_tour_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Attends(
        visitor_id INT,
        group_tour_id INT,
        requested_amount INT NOT NULL,
        PRIMARY KEY(visitor_id, group_tour_id),
        FOREIGN KEY (visitor_id) REFERENCES Visitor(visitor_id),
        FOREIGN KEY (group_tour_id)
        REFERENCES Group_Tour(group_tour_id));
```

2.14 Creates Event

- Relational Model:
  Creates(coordinator_id, event_id)

- Functional Dependencies:
  None

- Candidate Keys:
  {(coordinator_id, event_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Creates_Event(
        coordinator_id INT,
        event_id INT,
        PRIMARY KEY(coordinator_id, event_id),
        FOREIGN KEY (coordinator_id)
        REFERENCES Coordinator(coordinator_id),
        FOREIGN KEY (event_id) REFERENCES Event(event_id));
```

2.15 Donates

- Relational Model:
  Donates(visitor_id, con_org_id, donation_amount)

- Functional Dependencies:
  visitor_id, con_org_id → donation_amount

- Candidate Keys:
  {(visitor_id, con_org_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Donates(
        visitor_id INT,
        con_org_id INT,
        donation_amount INT NOT NULL,
        PRIMARY KEY (visitor_id, con_org_id)
        FOREIGN KEY (visitor_id) REFERENCES Visitor(visitor_id),
        FOREIGN KEY (con_org_id) REFERENCES
        Conservation_Organization(con_org_id));
```

2.16 Responds to

- Relational Model:
  Responds_to(form_id, coordinator_id, status)

- Functional Dependencies:
  form_id → coordinator_id, status

- Candidate Keys:
  {(form_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE RespondsTo(
        form_id INT,
        coordinator_id INT,
        status VARCHAR(8) NOT NULL,
        PRIMARY KEY (form_id),
        FOREIGN KEY (form_id) REFERENCES Complaint_Form(form_id),
        FOREIGN KEY (coordinator_id)
        REFERENCES Coordinator(coordinator_id));
```

2.17 Invites

- Relational Model:
  Responds_to(veterinarian_id, edu_prog_id, coordinator_id, status)

- Functional Dependencies:
  veterinarian_id, edu_prog_id → coordinator_id, status

- Candidate Keys:
  {(veterinarian_id, edu_prog_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Invites(
        veterinarian_id INT,
        edu_prog_id INT,
        coordinator_id INT,
        status VARCHAR(8) NOT NULL,
        PRIMARY KEY (veterinarian_id, edu_prog_id),
        FOREIGN KEY (coordinator_id )
        REFERENCES Coordinator(coordinator_id ),
        FOREIGN KEY (veterinarian_id )
        REFERENCES Veterinarian(veterinarian_id),
        FOREIGN KEY (edu_prog_id )
        REFERENCES Educational_Program(edu_prog_id));
```

2.18 Complaint Form

- Relational Model:
  Complaint_Form(<u>form_id</u>, form_content)

- Functional Dependencies:
  form_id → form_content

- Candidate Keys:
  {(form_id)}

- Normal Form:
  BCNF

- Table Definition:
  CREATE TABLE Complaint_Form(
  form_id **INT AUTO_INCREMENT,**
  form_content **VARCHAR(1000) NOT NULL**,
  PRIMARY KEY (form_id));

2.19 Schedules

- Relational Model:
  Schedules(keeper_id,animal_id,training_description,training_date)

- Functional Dependencies:
  keeper_id, animal_id, training_description → training_date

- Candidate Keys:
  {(keeper_id,animal_id, training_description)}

- Normal Form:
  3NF

- Table Definition:

```
CREATE TABLE Schedules(
        keeper_id             INT,
        user_id               INT,
        training_description VARCHAR(200),
        training_date         DATE,
        PRIMARY KEY (keeper_id, animal_id),
        FOREIGN KEY (keeper_id) REFERENCES Keeper(keeper_id),
        FOREIGN KEY (animal_id) REFERENCES Animal(animal_id));
```

2.20 Regularizes

- Relational Model:
  Regularizes(food_id, keeper_id, cage_id)

- Functional Dependencies:
  food_id → keeper_id, cage_id

- Candidate Keys:
  {(food_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Regularizes(
        food_id             INT PRIMARY KEY,
        keeper_id           INT,
        cage_id             INT,
        FOREIGN KEY (food_id) REFERENCES Food(food_id),
        FOREIGN KEY (keeper_id) REFERENCES Keeper(keeper_id),
        FOREIGN KEY (cage_id) REFERENCES Cage(cage_id));
```

2.21 Animal

- Relational Model:
  Animal(animal_id,animal_species,animal_name)

- Functional Dependencies:
  animal_id → animal_species,animal_name

- Candidate Keys:
  {(animal_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Animal(
        animal_id           INT PRIMARY KEY AUTO_INCREMENT,
        animal_species      VARCHAR(32) NOT NULL,
        animal_name         VARCHAR(32) NOT NULL));
```

2.22 Food

- Relational Model:
  Food(food_id,food_name,food_qty)

- Functional Dependencies:
  food_id → food_name,food_qty

- Candidate Keys:
  {(food_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Food(
        food_id             INT PRIMARY KEY AUTO_INCREMENT,
        food_name           VARCHAR(32) NOT NULL,
        food_qty            INT NOT NULL);
```

2.23 Requests

- Relational Model:
  Regularizes(keeper_id,veterinarian_id,animal_id,treatment_description,start_date,status)

- Functional Dependencies:
  keeper_id, veterinarian_id →
  animal_id,treatment_description,start_date,status

- Candidate Keys:
  {(keeper_id,veterinarian_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Requests(
        keeper_id                   INT,
        veterinarian_id             INT,
        animal_id                   INT,
        treatment_description       VARCHAR(200),
        start_date                  DATE,
        status                      ENUM(‘REQUESTED’, ’ACCEPTED’,
        ’REJECTED’, ’ONGOING’, ’FINISHED’),
        PRIMARY KEY(keeper_id, veterinarian_id),
        FOREIGN KEY (keeper_id) REFERENCES Keeper(keeper_id),
        FOREIGN KEY (veterinerian_id) REFERENCES
        Veterinarian(veterinarian_id),
        FOREIGN KEY (animal_id) REFERENCES Animal(animal_id)
        ));
```

2.24 Belongs_to

- Relational Model:
  Belongs_to(animal_id,cage_id)

- Functional Dependencies:
  None

- Candidate Keys:
  {(animal_id,cage_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Food(
        animal_id           INT,
        cage_id             INT,
        PRIMARY KEY(animal_id, cage_id),
        FOREIGN KEY (animal_id) REFERENCES Animal(animal_id),
        FOREIGN KEY (cage_id) REFERENCES Cage(cage_id));
```

2.25 Cage

- Relational Model:
  Cage(cage_id, cage_size, cage_type)

- Functional Dependencies:
  cage_id → cage_size, cage_type

- Candidate Keys:
  {(cage_id)}

- Normal Form:
  BCNF

- Table Definition
  CREATE TABLE Cage(
  cage_id **INT AUTO_INCREMENT,**
  cage_size **INT**,
  cage_type **VARCHAR(50) NOT NULL,**
  PRIMARY KEY (cage_id));

2.26 Area

- Relational Model:
  Area(area_id, area_name)

- Functional Dependencies:
  area_id → area_name

- Candidate Keys:
  {(area_id)}

- Normal Form:
  BCNF

- Table Definition
  ```
  CREATE TABLE Area(
  area_id INT AUTO_INCREMENT,
  area_name VARCHAR(50) NOT NULL,
  PRIMARY KEY (area_id));
  ```

2.27 Shop

- Relational Model:
  Area(shop_id, shop_name, shop_description)

- Functional Dependencies:
  shop_id → shop_name, shop_description

- Candidate Keys:
  {(shop_id)}

- Normal Form:
  BCNF

- Table Definition
  CREATE TABLE Shop(
  shop_id **INT AUTO_INCREMENT,**
  shop_name **VARCHAR(50) NOT NULL**,
  shop_description **VARCHAR(50) NOT NULL**,
  PRIMARY KEY (shop_id));

2.28 Item

- Relational Model:
  Area(item_id,item_name,item_stock)

- Functional Dependencies:
  item_id → item_name, item_stock

- Candidate Keys:
  {(item_id)}

- Normal Form:
  BCNF
- Table Definition

```
CREATE TABLE Item(
item_id INT AUTO_INCREMENT,
item_name VARCHAR(50) NOT NULL,
item_stock  INT,
PRIMARY KEY (item_id));
```

2.29 Is_In_C

- Relational Model:
  Is_In(area_id,cage_id)

- Functional Dependencies:
  None

- Candidate Keys:
  {(area_id,cage_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Is_In_C(
        area_id                INT,
        cage_id                INT,
        PRIMARY KEY (area_id, cage_id)
        FOREIGN KEY (areal_id) REFERENCES Area(areal_id),
        FOREIGN KEY (cage_id) REFERENCES Cage(cage_id));
```

2.30 Is_In_S

- Relational Model:
  Is_In(area_id,shop_id)

- Functional Dependencies:
  None

- Candidate Keys:
  {(shop_id,area_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Is_In_S(
        area_id             INT,
        shop_id             INT,
        PRIMARY KEY (area_id, shop_id),
        FOREIGN KEY (area_id) REFERENCES Areal(area_id),
        FOREIGN KEY (shop_id) REFERENCES Shop(shop_id));
```

2.31 Sells

- Relational Model:
  Belongs_to(shop_id,item_id)

- Functional Dependencies:
  None

- Candidate Keys:
  {(shop_id,item_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Sells(
        shop_id             INT,
        item_id             INT,
        PRIMARY KEY (shop_id, item_id),
        FOREIGN KEY (animal_id) REFERENCES Animal(animal_id),
        FOREIGN KEY (cage_id) REFERENCES Cage(cage_id));
```

2.32 Buys

- Relational Model:
  Buys(item_id,user_id, amount)

- Functional Dependencies:
  item_id, user_id → amount

- Candidate Keys:
  {(item_id,user_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Buys(
        item_id                 INT,
        user_id                 INT,
        amount                  INT,
        PRIMARY KEY(item_id, user_id),
        FOREIGN KEY (item_id) REFERENCES Item(item_id),
        FOREIGN KEY (user_id) REFERENCES User(user_id));
```

2.33 Assigns

- Relational Model:
  Assigns(keeper_id,cage_id, coordinator_id)

- Functional Dependencies:
  keeper_id,cage_id → coordinator_id

- Candidate Keys:
  {(keeper_id,cage_id)}

- Normal Form:
  BCNF

- Table Definition:

```
CREATE TABLE Assigns(
        keeper_id           INT,
        cage_id             INT,
        coordinator_id      INT,
        PRIMARY KEY(keeper_id, cage_id),
        FOREIGN KEY (coordinator_id) REFERENCES
        Coordinator(coordinator_id),
        FOREIGN KEY (keeper_id) REFERENCES Keeper(keeper_id),
        FOREIGN KEY (cage_id) REFERENCES Cage(cage_id));
```

3.FUNCTIONAL COMPONENTS

3.1 Algorithms

3.1.1 Item Purchase Algorithm

When an item is bought it stock amount must be updated. Customers should not be able to pay for an item that is out of stock. Therefore, an item with the stock amount 0 should not be displayed as an item sold in a shop. In order to ensure this, when a tuple is inserted to the Buys relation, stock amount of the corresponding item will be reduced by the amount attribute of the Buys relation (i.e. the amount of the same item visitor purchased). Then, the stock amount of the item will be checked. If the stock amount hits 0, all the tuples in the Sells relations with the corresponding item id will be deleted.

3.1.2 Donation Algorithm

Conservation Organizations are created by Coordinators. However, during or after the creation of the event, Coordinators should not be able to set nor alter the received donation amount. Collected amount of the event can only be altered through the donations made by the visitors. Therefore, during creation the collected amount of the conservation organization will be set to 0 as a default. When a visitor grants a donation, the corresponding conservation organization will be traced through its table and its collected amount will be incremented by the donation amount.

3.1.3 Logical Requirements

Our system should work with the minimised logical errors in its constitution. Therefore, there should be precautions against illogical information being presented or used as an alteration to the system. Therefore, ids, names or any other relevant attributes must be checked in terms of relevancy. For example, each type of user has their limited amount of access to certain information. A visitor should not be able to read the keeper information whereas a keeper should not have access to complaint forms. In order to prevent such inconsequential access problems, the id received during the login should be traced in the database, the identity of the actor must be found and only the pages that actor can access should be displayed.

3.2 Data Structures

Our database system will store numeric values such as ids, prices, salaries etc. as INT, dates as DATE and alphabetic values such as names, descriptions etc. as VARCHAR.

3.3 Use Cases

3.3.1 Visitor

Create an Account: Visitors can create accounts by entering their full names, usernames, emails, addresses, bios, birth dates and passwords.

Login: Visitors can login to their accounts with their emails and passwords.

Comment on Group Tours: Visitors can comment on group tours that they have attended.

Start a Gold / Silver Membership: Visitors can start a silver or gold memberships.

Cancel Membership: Visitors can cancel their memberships.

View Group Tours: Visitors can look at the available group tours.

Buy Tickets to Group Tours: Visitors can buy tickets to group tours.

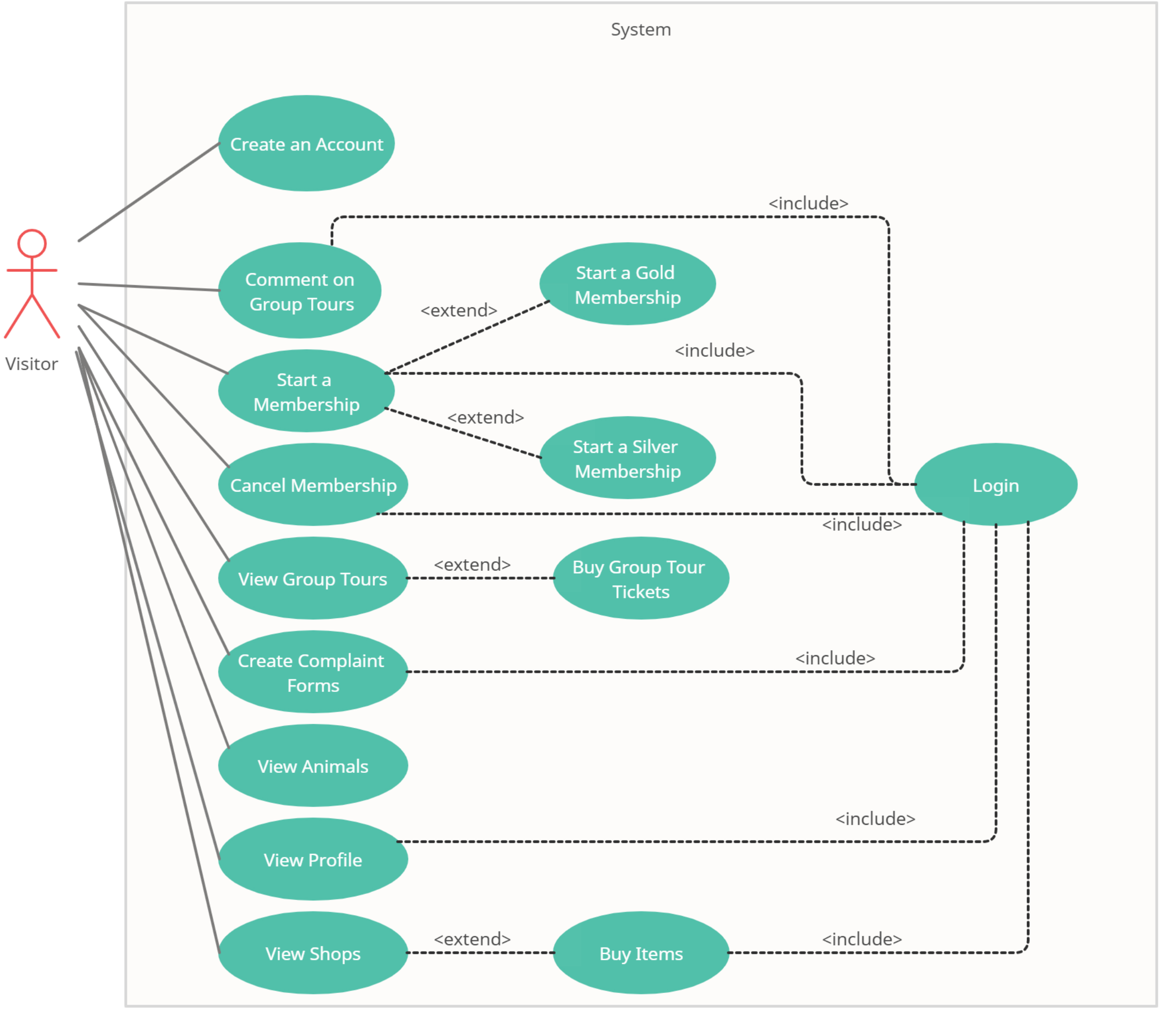
Create Complaint Forms: Visitors can create complaint forms about the zoo.

View Animals: Visitors can look at the list of the animals that are present in the zoo.

View Profile: Visitors can view their profile.

View Shops: Visitors can look at the list of shops in the zoo.

Buy Items: Visitors can buy items from the shops in the zoo.


System
Visitor
Create an Account
Comment on Group Tours
Start a Gold Membership
Start a Membership
Start a Silver Membership
Cancel Membership
View Group Tours
Buy Group Tour Tickets
Create Complaint Forms
View Animals
View Profile
View Shops
Buy Items
Login
<include>
<extend>
<include>
<extend>
<include>
<extend>
<include>
<include>
<extend>
<include>

3.3.2 Keeper

Request Treatment: Keepers can request Treatment from Veterinarians.

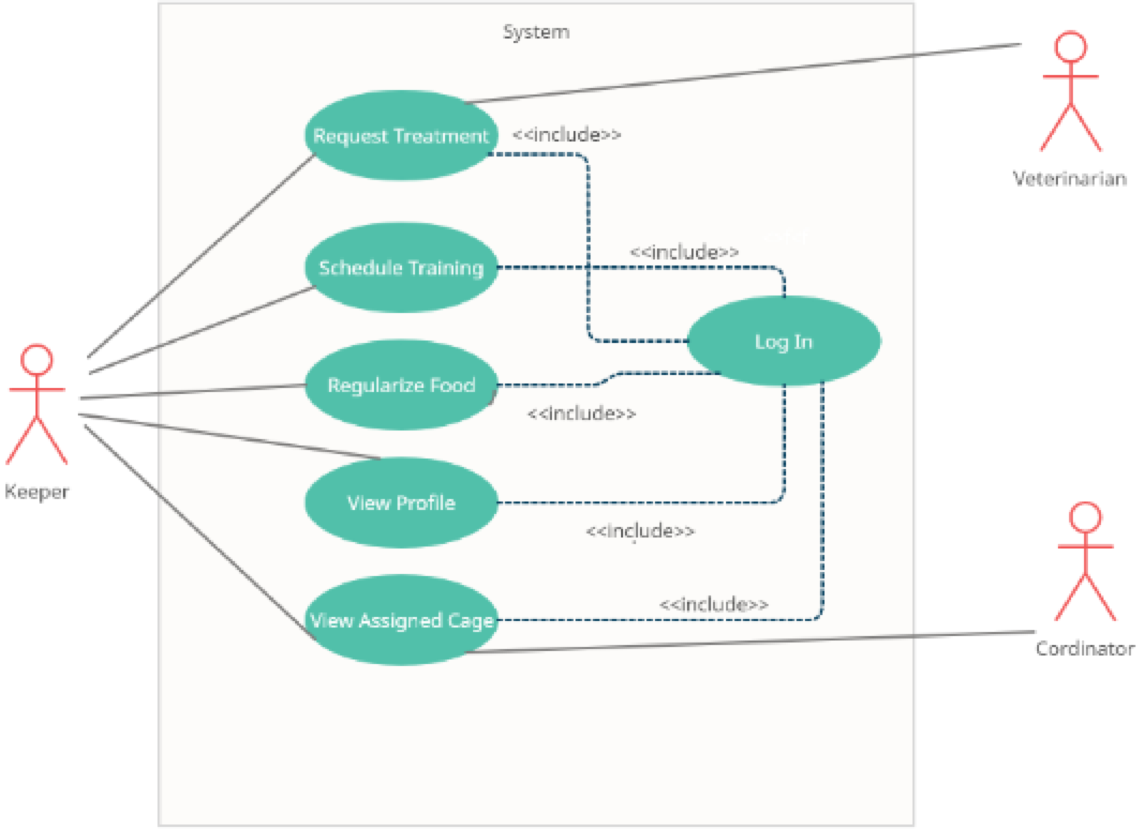
Schedule Training: Keepers can schedule training for animals.

Regularize Food: Keepers can regularize food for the animals.

View Profile: Keepers can view their profiles.

View Assigned Cage: Keepers can view the cage that they have been assigned by the coordinator.

Log In: Keepers can log in to their profile with their password and e-mail.



3.3.2 Veterinarian

View Request: Veterinarians can view treatment requests made by the keepers.

Accept Request: Veterinarians can accept the treatment requests made by the keepers.

Reject Request: Veterinarians can reject the treatment requests made by the keepers

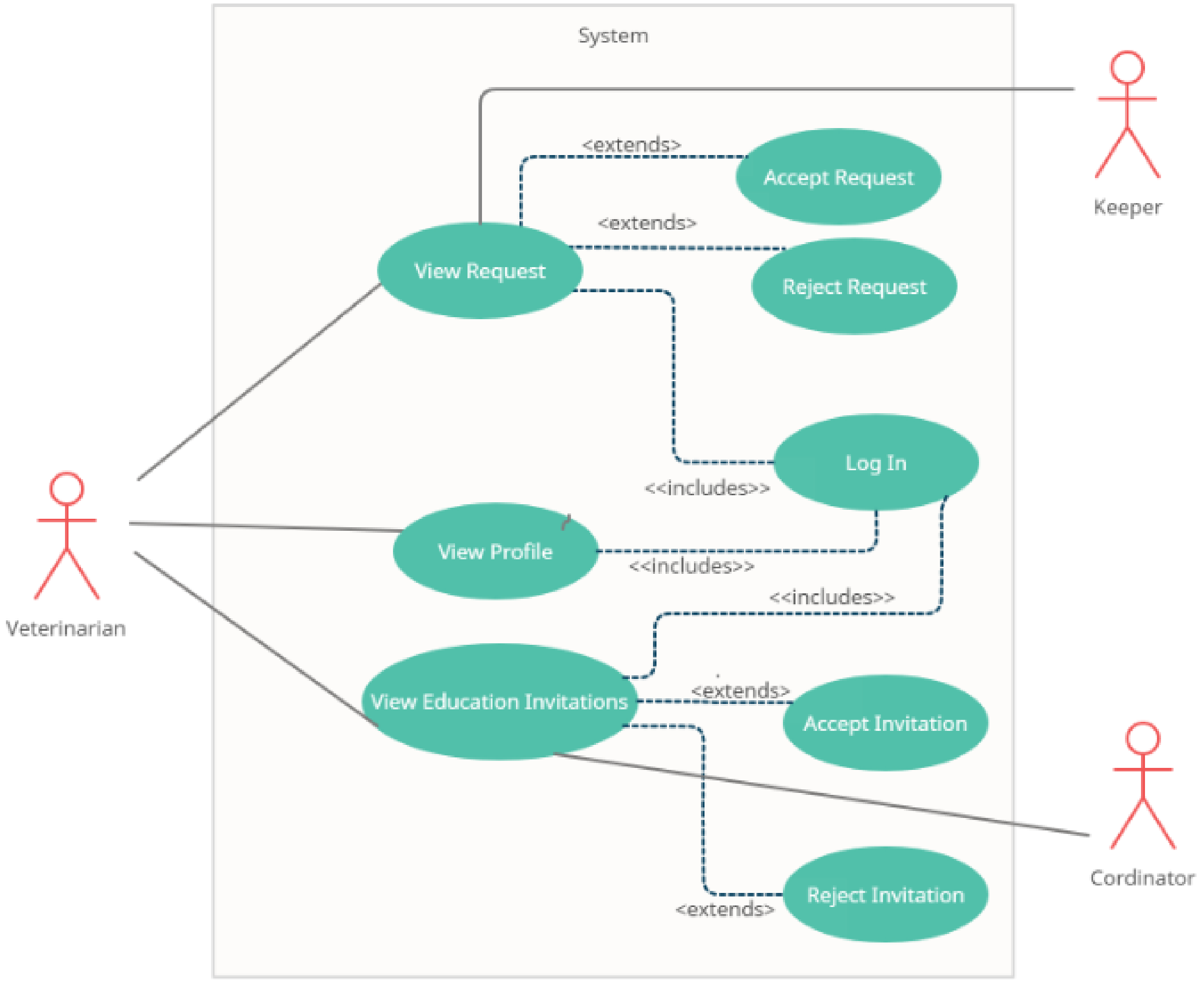
Log In: Veterinarians can log in using their email and password

View Profile: Veterinarians can view their own profile

View Education Invitation: Veterinarians can view education invitations made to them by the coordinator.

Accept Invitation: Veterinarians can accept the education invitation made to them by the coordinator.

Reject Invitation: Veterinarians can reject the education invitation made to them by the coordinator.


System
Keeper
<extends>
Accept Request
<extends>
View Request
Reject Request
Log In
<<includes>>
Veterinarian
View Profile
<<includes>>
<<includes>>
View Education Invitations
<extends>
Accept Invitation
Cordinator
Reject Invitation
<extends>

3.3.2 Coordinator

Login: Coordinators can login to their accounts using the email address and the password provided by the company.

View Events: Coordinators can view the list of the current events.

Create an Event: Coordinators can create new events. These events can be group tours, educational programs or conservation organizations in particular.

Send Invitation for Educational Program: Coordinators can send an invitation to a veterinarian for an educational program.

View Assigned Cages: Coordinator can view the list of the assigned cages.

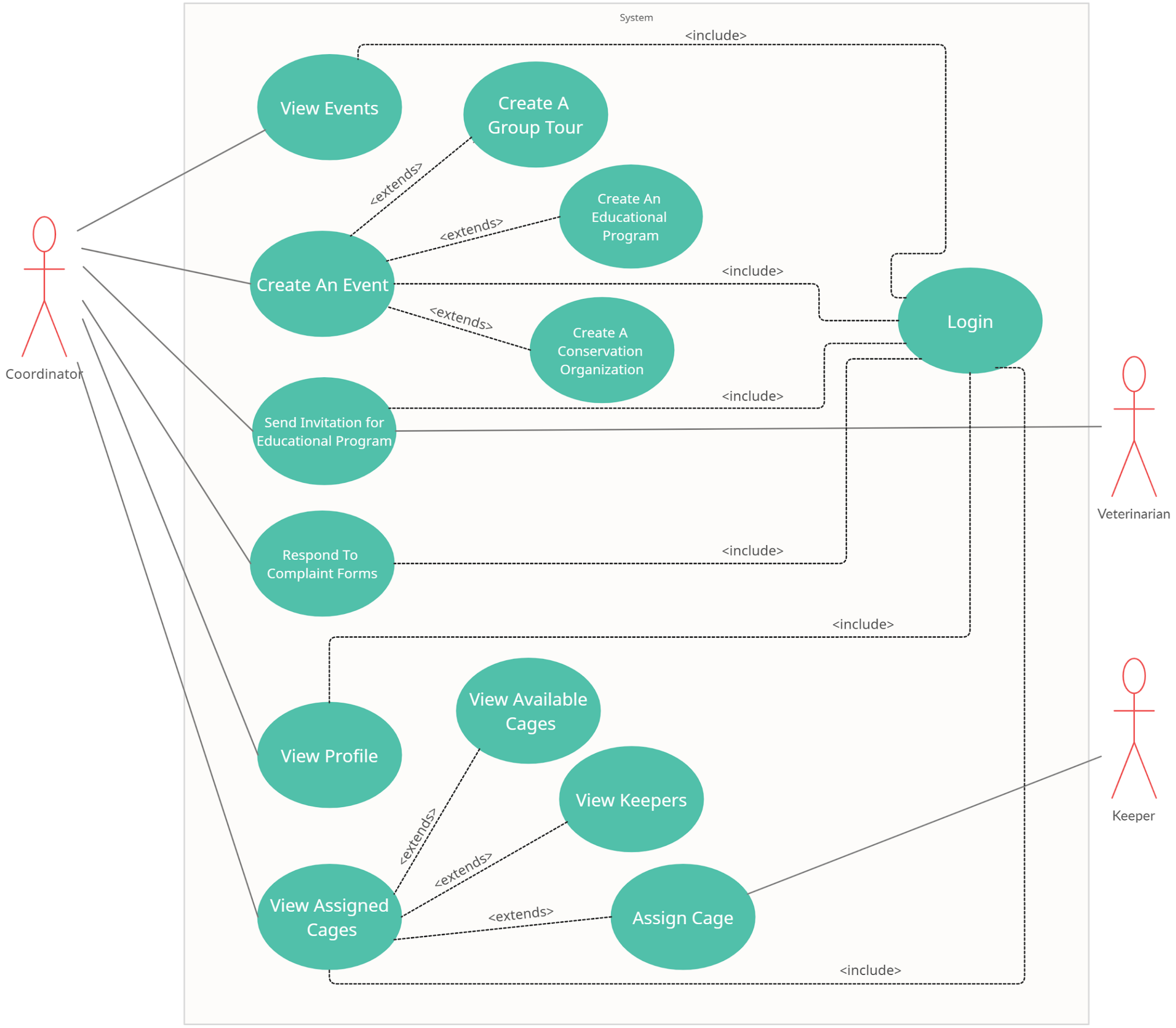
View Available Cages: Coordinators can view the list of available cages.

View Keepers: Coordinators can view the list of the keepers.

Assign Cage: Coordinator can pick an available cage and assign it to a keeper from the keepers list.

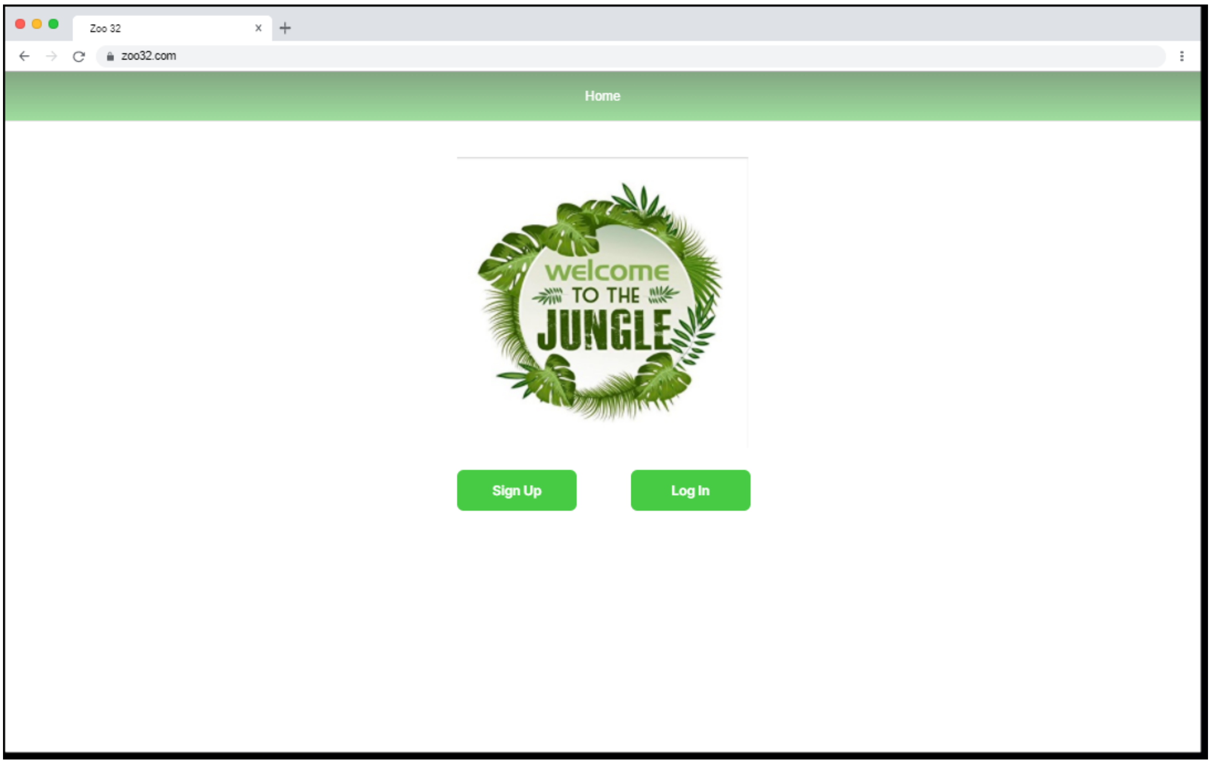
Respond to Complaint Form: Coordinators can respond to the complaint forms.

View Profile: Coordinators can view their own profiles.


System
<include>
View Events
Create A Group Tour
<extends>
Create An Educational Program
<extends>
Coordinator
Create An Event
<include>
Login
<extends>
Create A Conservation Organization
<include>
Send Invitation for Educational Program
Veterinarian
Respond To Complaint Forms
<include>
<include>
Keeper
View Available Cages
View Profile
View Keepers
<extends>
<extends>
View Assigned Cages
<extends>
Assign Cage
<include>

4. USER INTERFACES AND CORRESPONDING SQL STATEMENTS

4.1 Home Page (Not logged In)

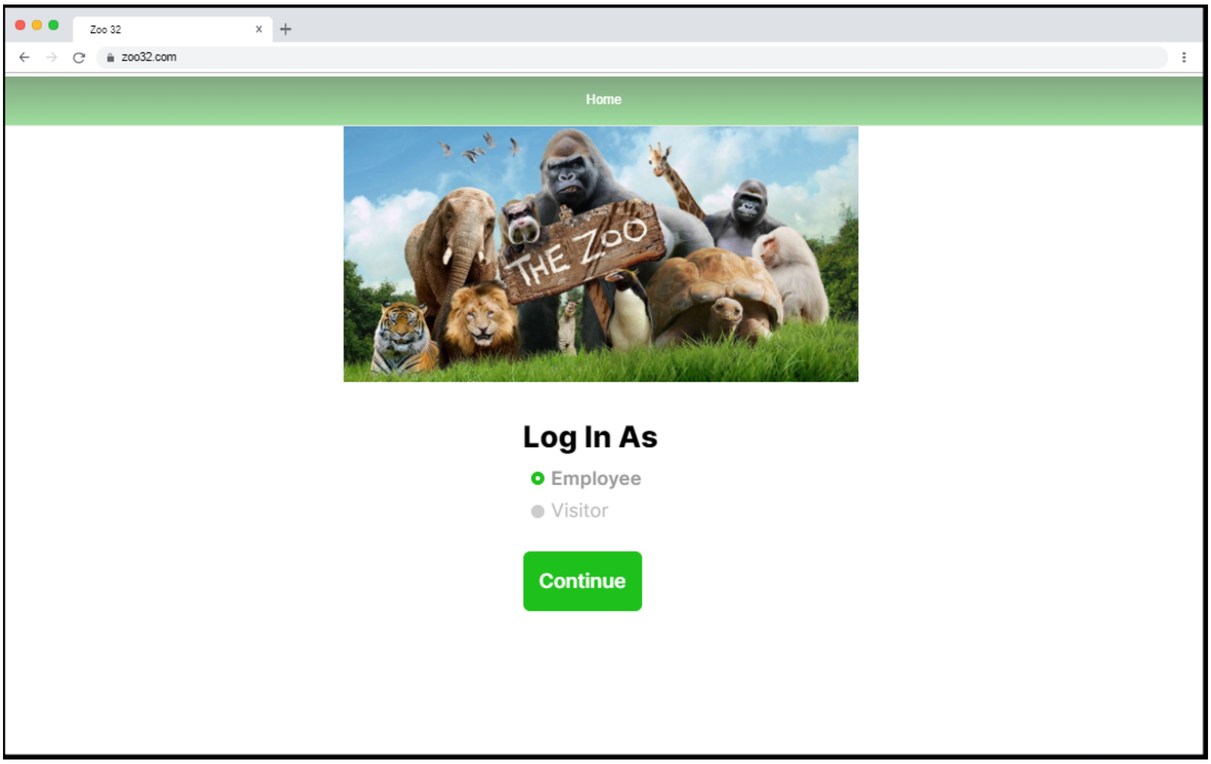


https://framer.com/share/id7GQxxQyTy4lz3AA4bJ/RqnUqcwbb

SQL Statements:

No SQL statement is needed here, the user will choose whether they want to log in or sign up and click on the corresponding button.

4.2 Log In Page



Log In As

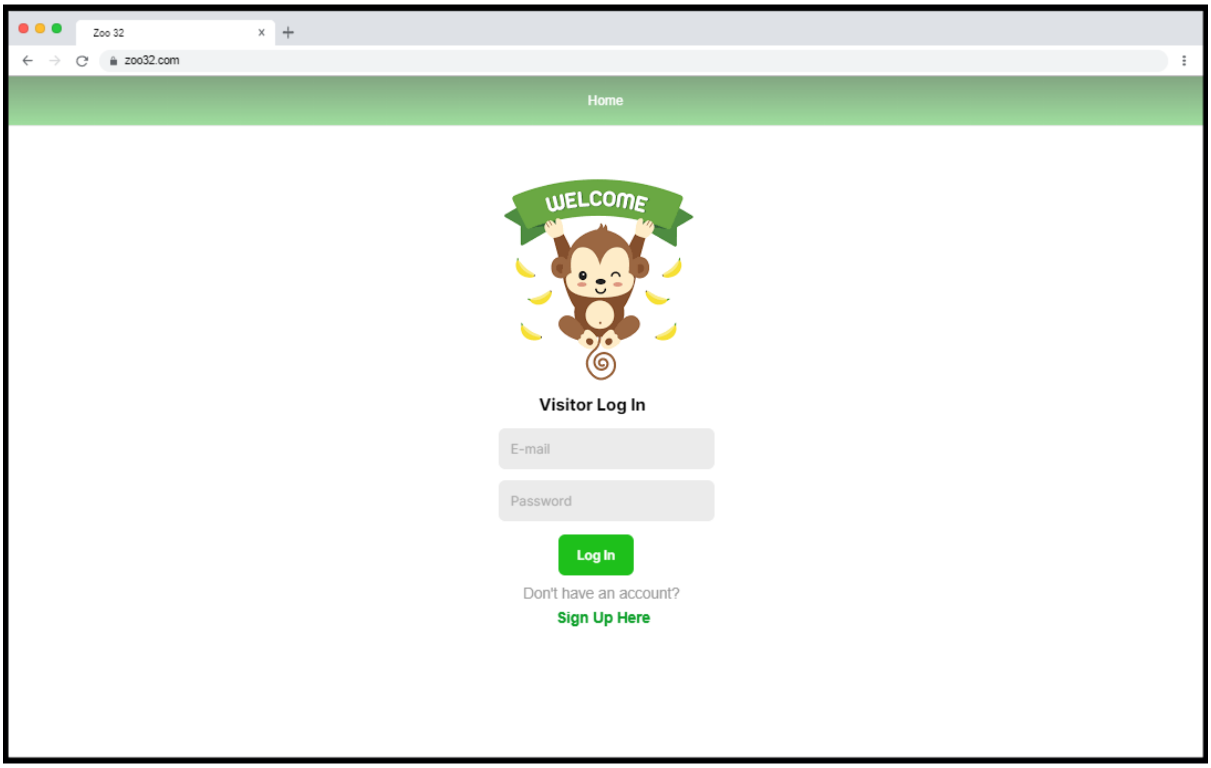
- ● Employee
- ○ Visitor

https://framer.com/share/id7GQxxQyTy4Iz3AA4bJ/aQNUvrGkp

SQL Statements:

No SQL statement is needed and the users will choose whether they want to log in as an employee or visitor and get directed to the corresponding pages when they click continue.

4.3 Log In Page(Visitor)

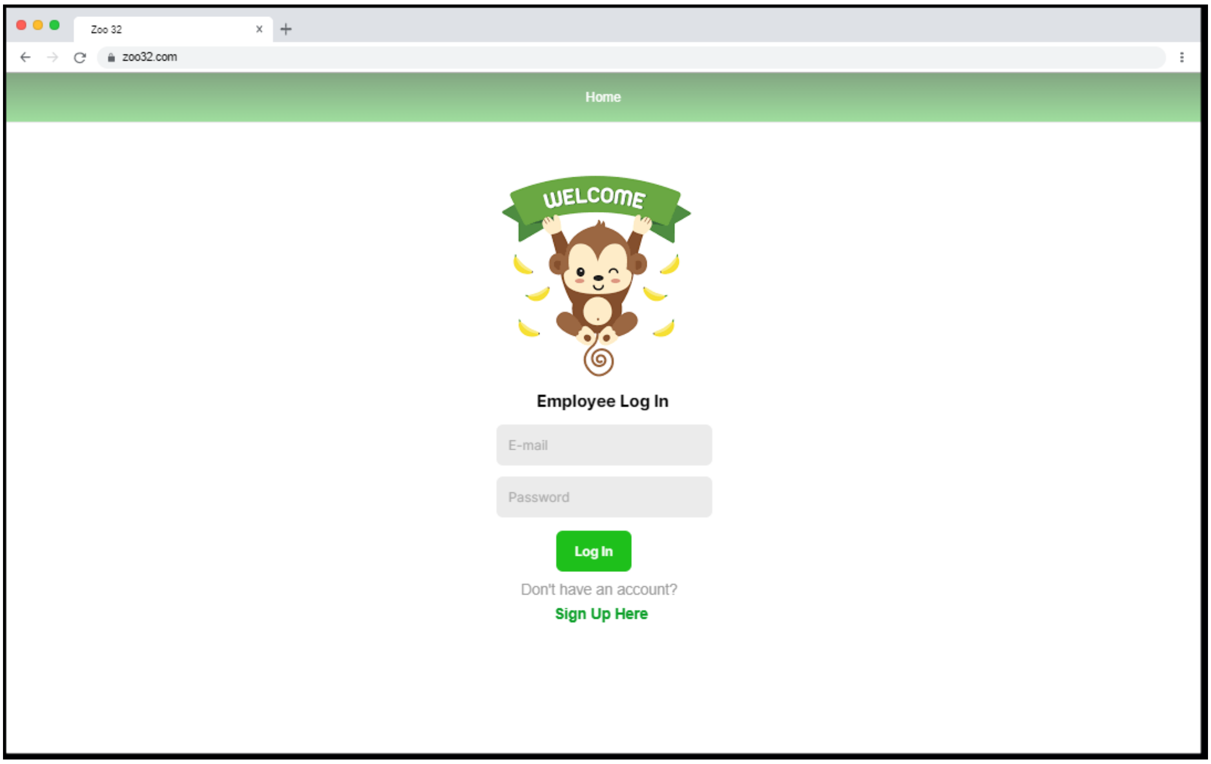


https://framer.com/share/id7GQxxQyTy4lz3AA4bJ/ETP4C99FV

SQL Statements:

```
SELECT*
FROM Visitor JOIN User ON Visitor.visitor_id=User.user_id
WHERE (email=@email) AND (password = @password)
```

4.4 Log In Page(Employee)

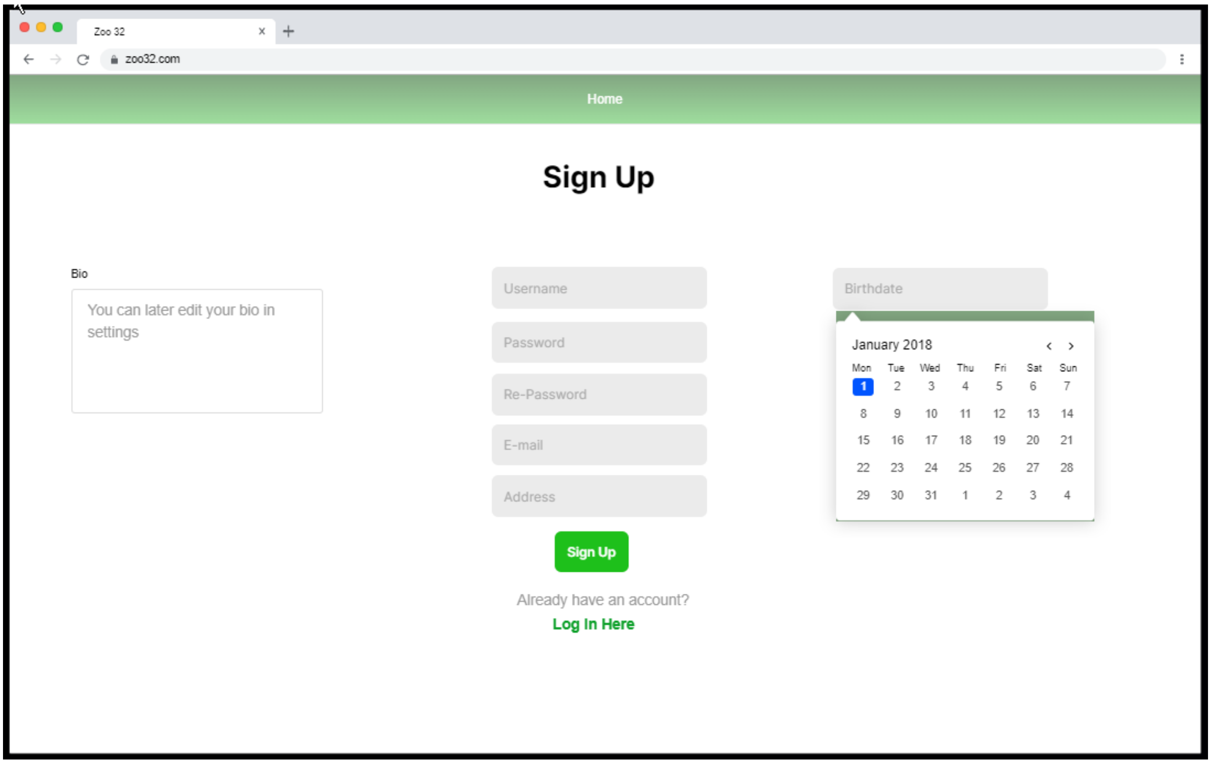


https://framer.com/share/id7GQxxQyTy4Iz3AA4bJ/B69RNynhw

SQL Statements:

```
SELECT *
FROM (SELECT *
          FROM User
          WHERE User.user_id NOT IN ( SELECT visitor_id
                                       FROM Visitor)
WHERE (email = @email) AND (password = @password)
```

4.5 Sign Up Page



January 2018

Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

https://framer.com/share/id7GQxxQyTy4lz3AA4bJ/XLoFvZV4R

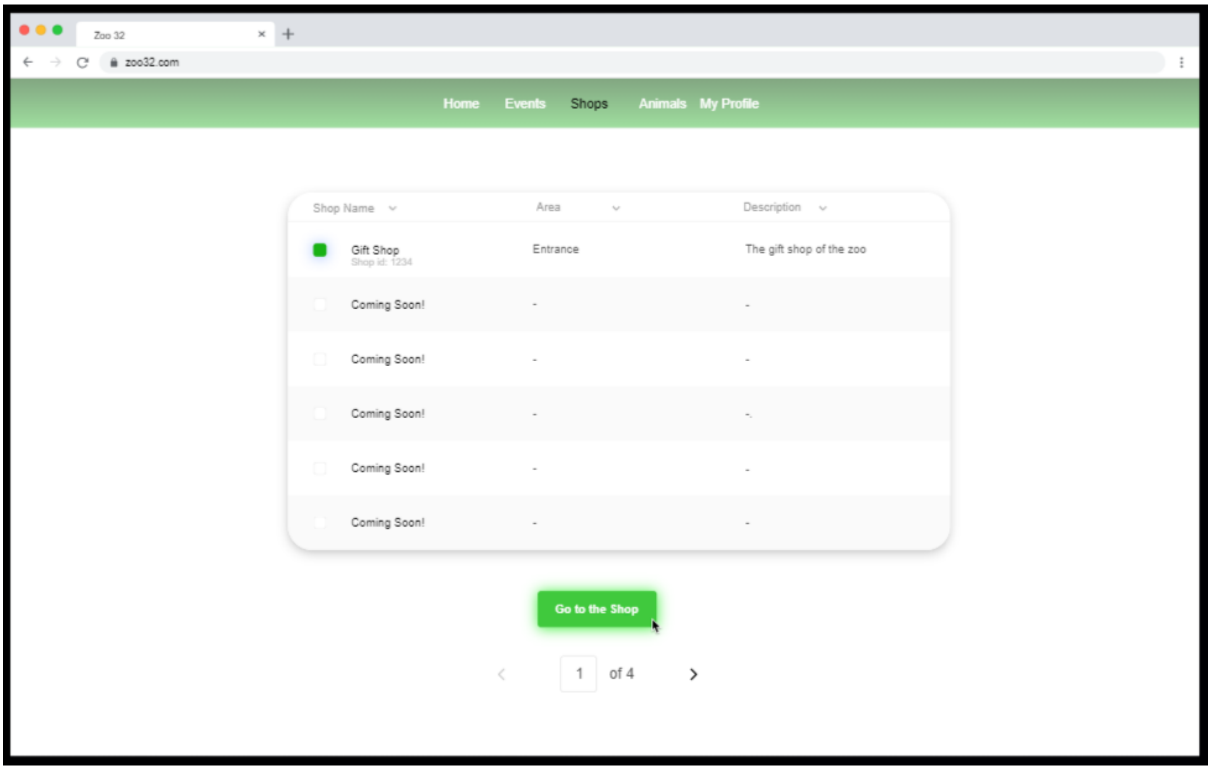
SQL Statements:

Inputs: @username, @password, @repassword, @email, @address, @birthdate, @bio

INSERT INTO User (username, password, email, address, birthday, bio)
VALUES (@username, @password, @email, @address, @birthdate, @bio)

INSERT INTO Visitor
VALUES (**LAST_INCREMENT_ID()**, last_visit);

4.6 Shops Menu Screen



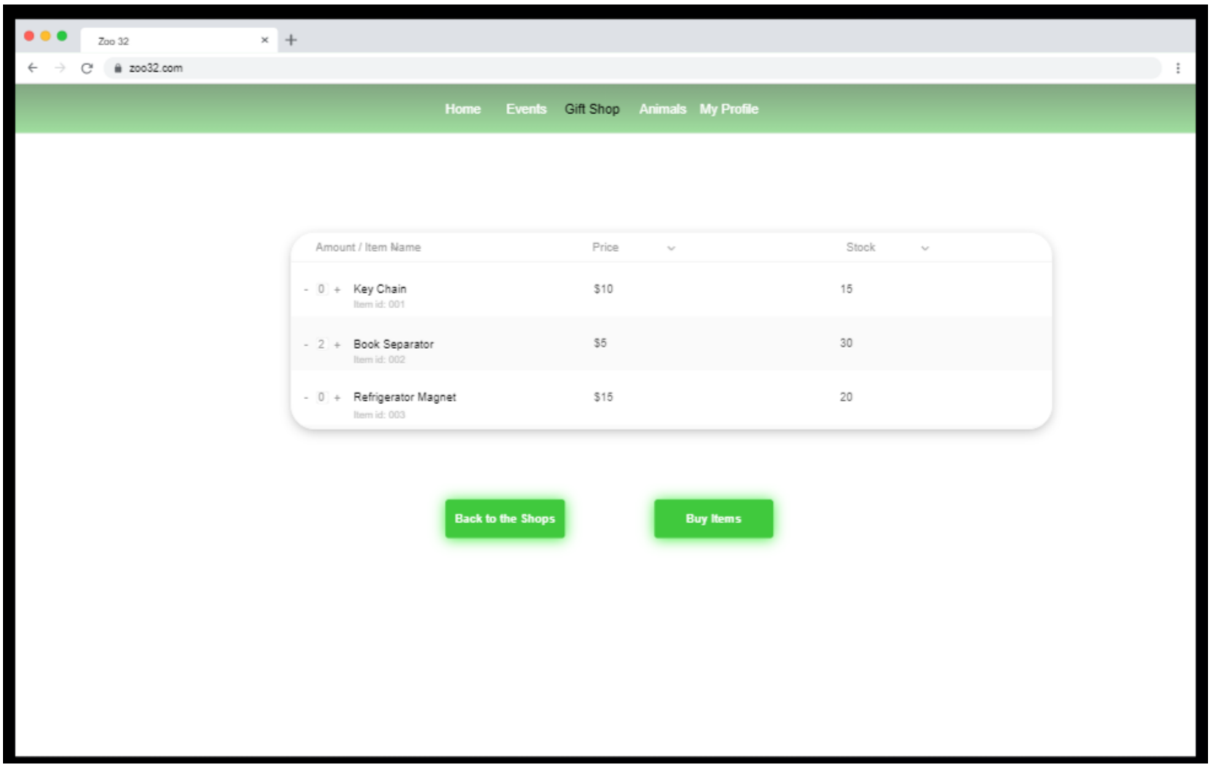
	Shop Name	Area	Description
■	Gift Shop Shop id: 1234	Entrance	The gift shop of the zoo
□	Coming Soon!	-	-
□	Coming Soon!	-	-
□	Coming Soon!	-	-.
□	Coming Soon!	-	-
□	Coming Soon!	-	-

https://framer.com/share/jAMkfRkOe9j2QDXPDaF0/qEPkgqN53?editor=1

SQL Statements:

- For listing the shop names, ids, areas and descriptions:
  SELECT shop_id, shop_name, shop_description, area_name
  FROM Shop **NATURAL JOIN** Is_in **NATURAL JOIN** Area

4.7 Items Menu Screen



Amount / Item Name	Price	Stock
- 0 + Key Chain Item id: 001	$10	15
- 2 + Book Separator Item id: 002	$5	30
- 0 + Refrigerator Magnet Item id: 003	$15	20

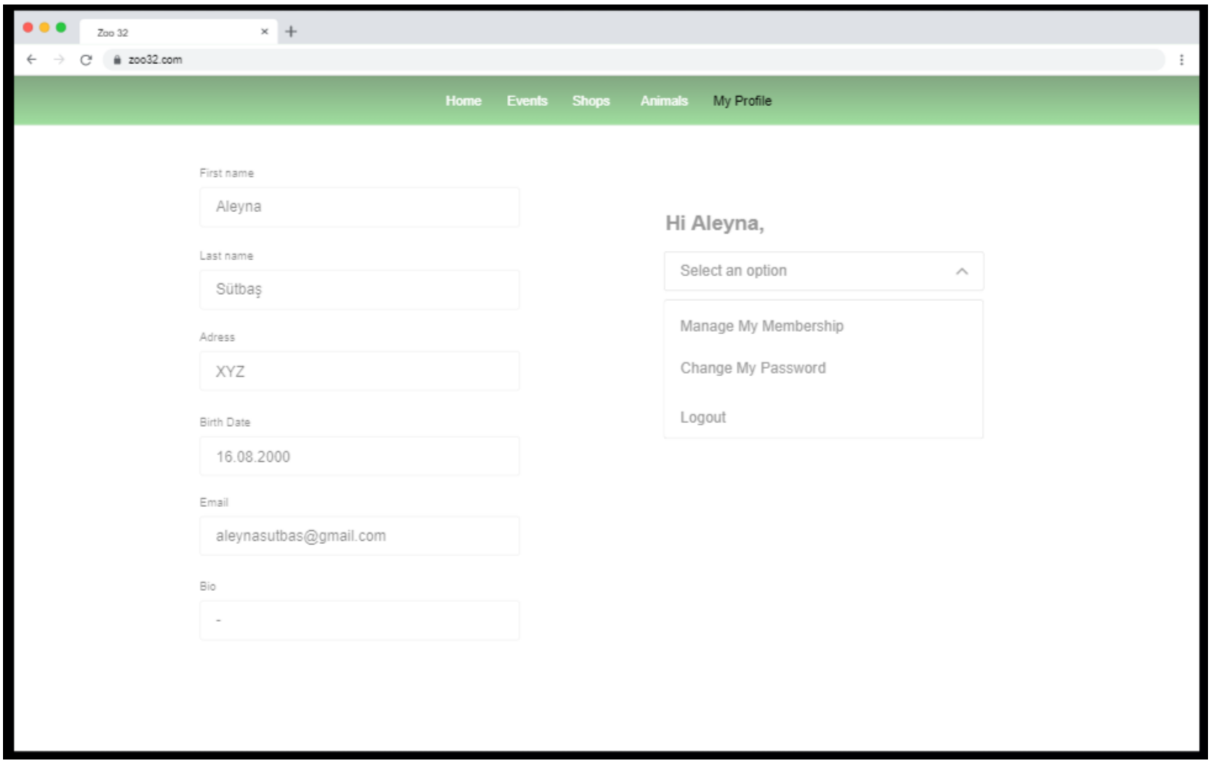
https://framer.com/share/jAMkfRkOe9j2QDXPDaF0/cm1Bj4b4U?editor=1

SQL Statements:

Inputs: @shopname, @visitor_id, @item_id, @amount

- For listing the item names, ids, and stocks of the shop:
  SELECT I.item_id, I.item_name, I.item_stock
  FROM Shop Sh, Item I , Sells S
  WHERE Sh.shop_name = @shopname **AND** Sh.shop_id = S.shop_id **AND** S.item_id = I.item_id
- Decreasing the stock of a sold item
  UPDATE Item
  SET item_stock = item_stock - @amount
  WHERE item_id = @item_id
- Adding the item that has been sold to the “Buys” relation
  INSERT INTO Buys **VALUES** (@visitor_id, @item_id, @amount)

4.8 Profile Screen



First name
Aleyna

Last name
Sütbaş

Adress
XYZ

Birth Date
16.08.2000

Email
aleynasutbas@gmail.com

Bio
-

Hi Aleyna,

Select an option

Manage My Membership
Change My Password
Logout

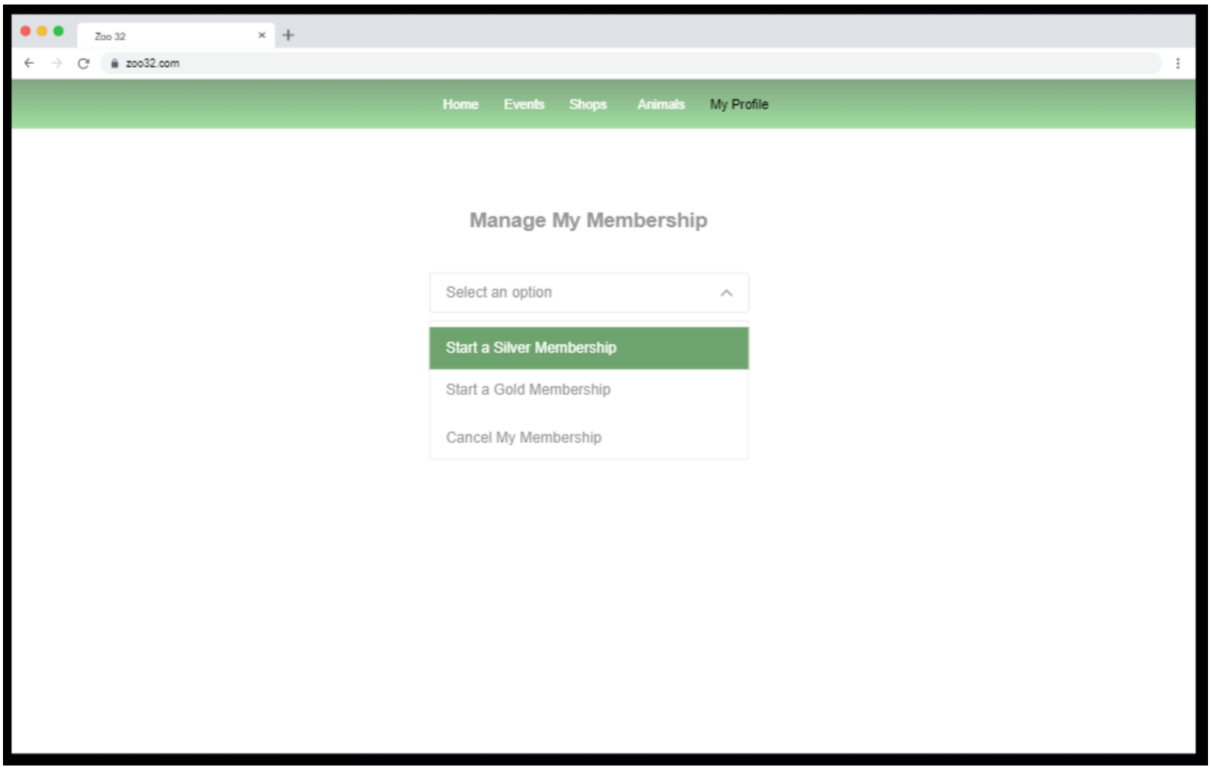
https://framer.com/share/jAMkfRkOe9j2QDXPDaF0/tL9DsxXSh?editor=1

SQL Statements:

Inputs: @visitor_id

- For listing the information about a visitor:
  SELECT full_name, address, birth_date, bio
  FROM Visitor
  WHERE visitor_id = @visitor_id

4.9 Membership Screen



Manage My Membership

Select an option

Start a Silver Membership

Start a Gold Membership

Cancel My Membership

https://framer.com/share/jAMkfRkOe9j2QDXPDaF0/F6CQ7IOse?editor=1

After selecting the “Manage My Membership” option from the profile screen, the membership screen opens. If a visitor does not have an active membership, cancel my membership button will be shown. If a visitor has a silver or gold membership, only cancel my membership button will be available.

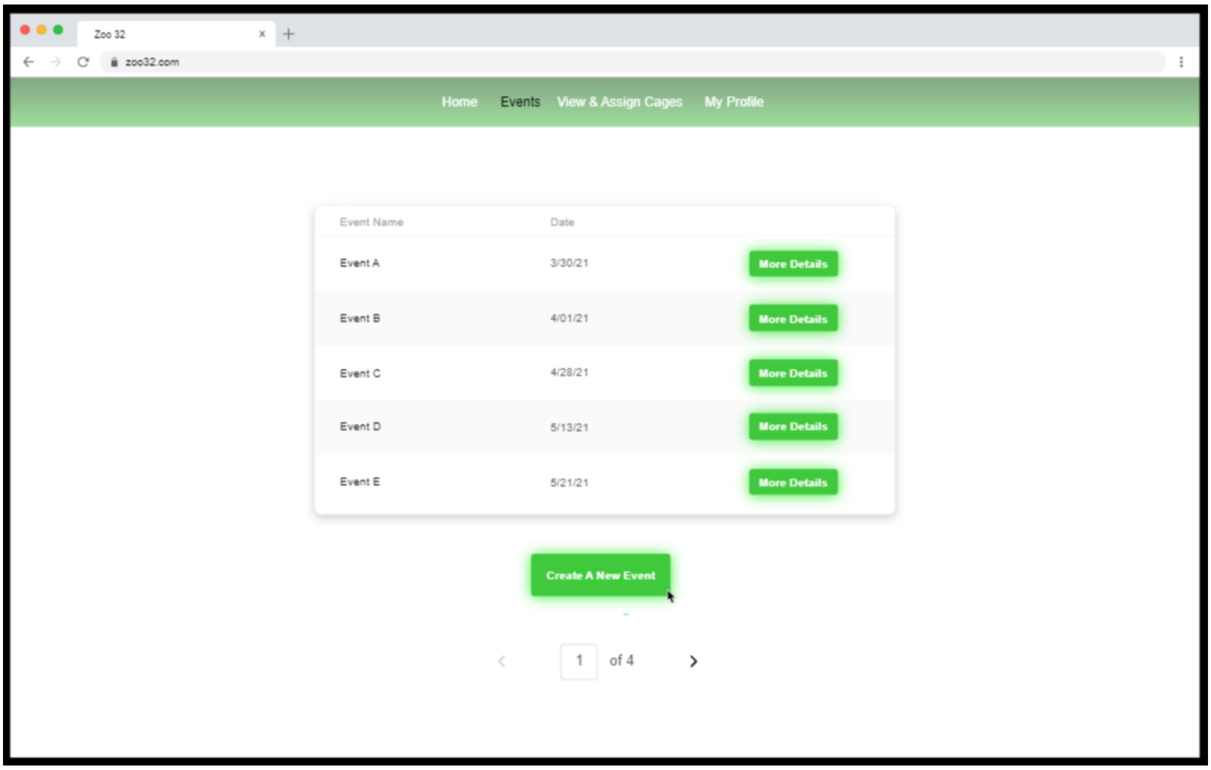
SQL Statements:

Inputs: @visitor_id, @m_type

- For canceling a membership:
  ```
  -WITH membership_id (id) AS (   SELECT membership_id
                                  FROM Has H
                                  WHERE H.visitor_id = @visitor_id)
  -DELETE FROM Membership
   WHERE membership_id IN (SELECT * FROM membership_id)
  -DELETE FROM Has
   WHERE membership_id IN (SELECT * FROM membership_id)
  ```

- For starting a membership:
  ```
  -WITH expiration_date (date) AS SELECT CURDATE() + 365
  -INSERT INTO Membership (expiration_date, price)
  -VALUES (SELECT * FROM expiration_date, 100)
  -INSERT INTO Has  VALUES (LAST_INSERT_ID(), @visitor_id, @m_type)
  ```
  **Normally, deletions and insertions in “Has” will be handled by a trigger.

4.10 Events Screen (Coordinator Account)



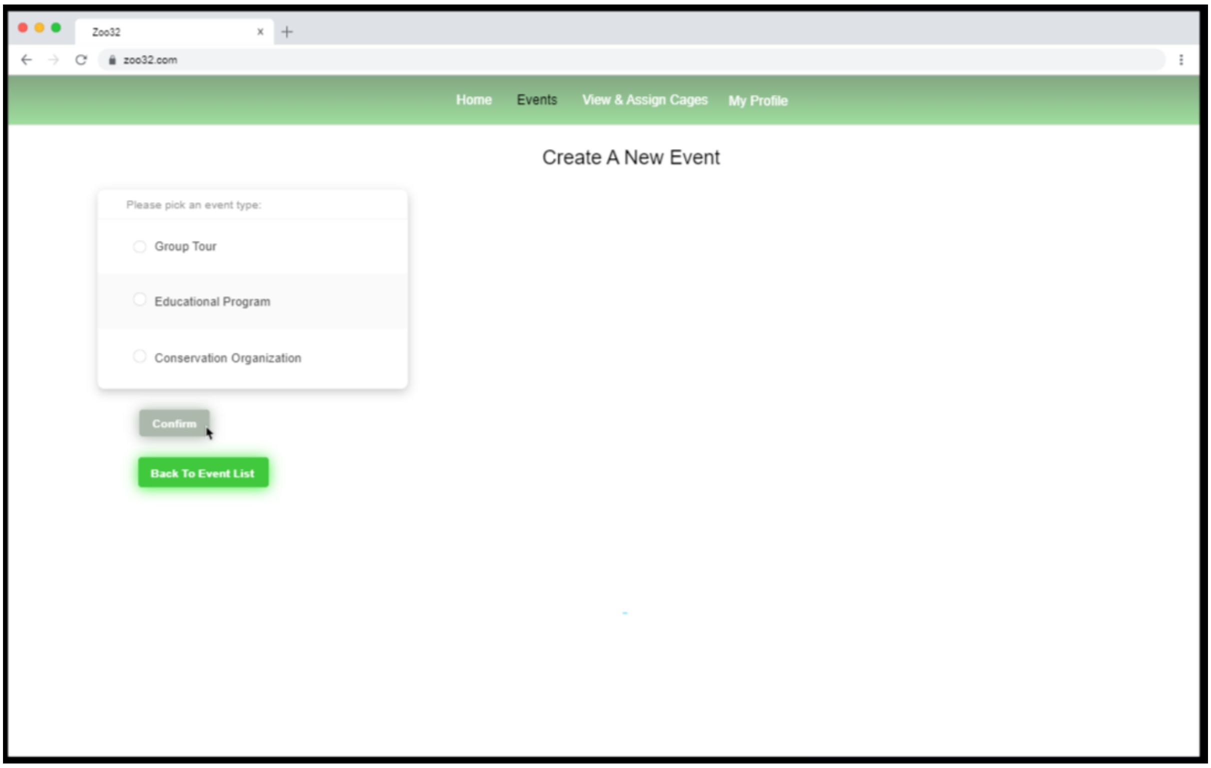
Event Name	Date	
Event A	3/30/21	More Details
Event B	4/01/21	More Details
Event C	4/28/21	More Details
Event D	5/13/21	More Details
Event E	5/21/21	More Details

https://framer.com/share/d76zfFVvixxWshWHWnpl/qEPkgqN53#qEPkgqN53

SQL Statements:

- To list the current events
  SELECT event_name, date **FROM** Event

4.11 Create Event Screen (Coordinator Account)



Create A New Event

Please pick an event type:

○ Group Tour

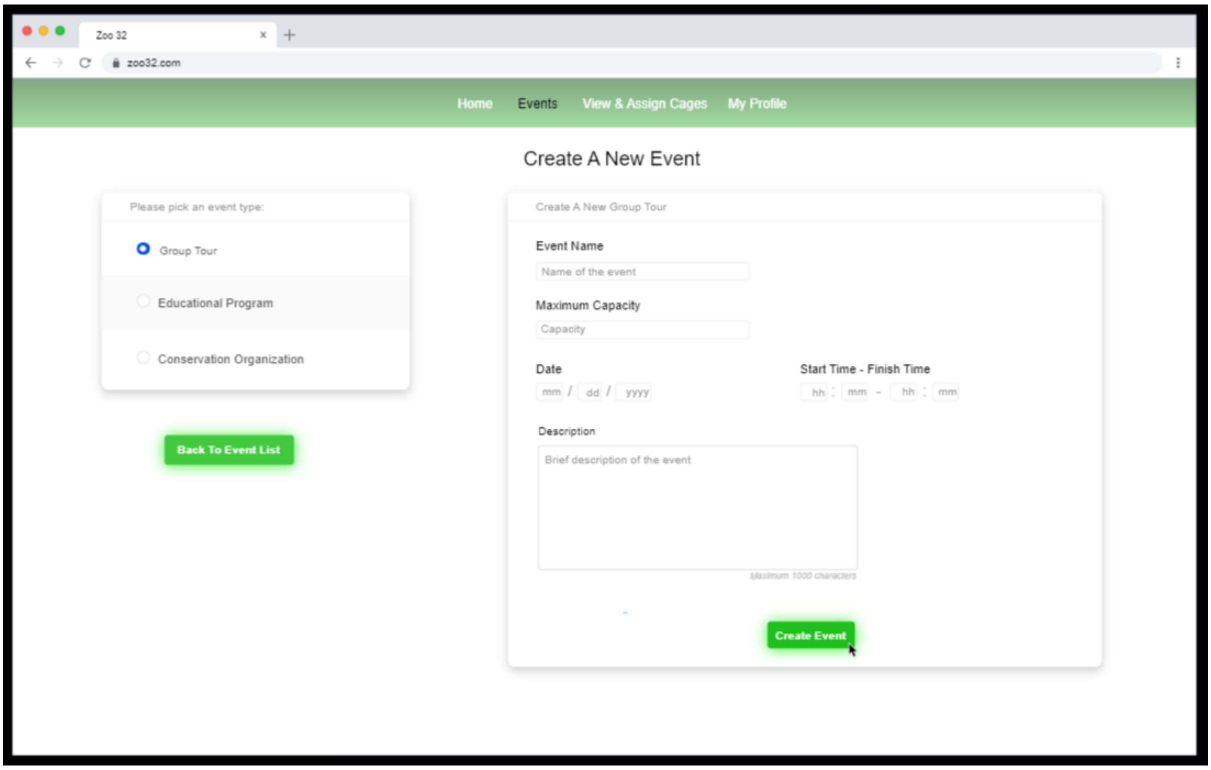
○ Educational Program

○ Conservation Organization

https://framer.com/share/9tqsuTm9aKtfloc4UYQJ/qEPkgqN53

No SQL statement needed for this section. When the coordinator picks an event type and confirms a form structure will appear at the blank area.

4.12 Create Group Tour (Coordinator Account)



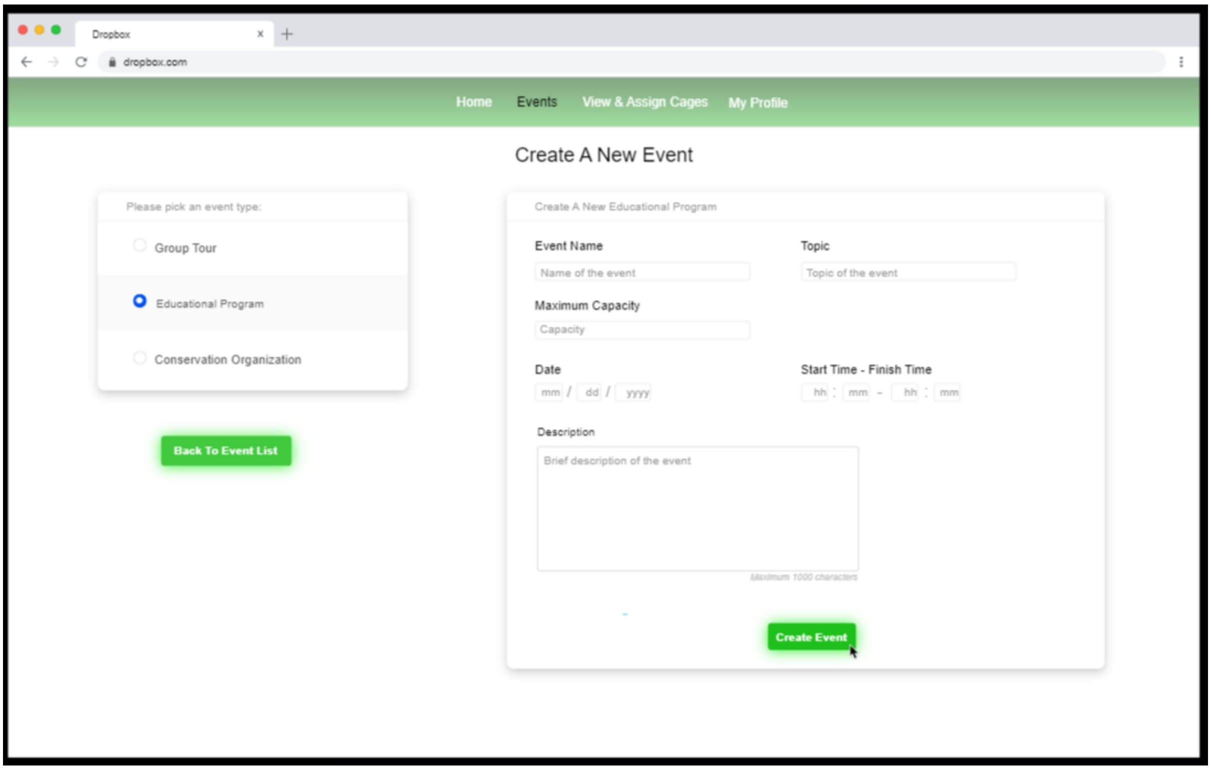
https://framer.com/share/u5J7UMrr6zAC5uMkU0T4/qEPkgqN53

SQL Statements:

Inputs: @description, @max_capacity, @date, @duration, @coordinator_id

- After the coordinator creates the event by clicking the button:
  INSERT INTO Event (description, max_capacity, num_of_participants,
  date, duration)
  VALUES (@description, @max_capacity, 0, @date, @duration);
- Inserting the corresponding group tour to the newly created event:
  INSERT INTO Group_Tour **VALUES** (**LAST_INSERT_ID()**);
- Inserting the new creation to the Creates_Event relation:
  INSERT INTO Creates_Event **VALUES** (@coordinator_id,
  LAST_INSERT_ID());

4.13 Create Educational Program (Coordinator Account)



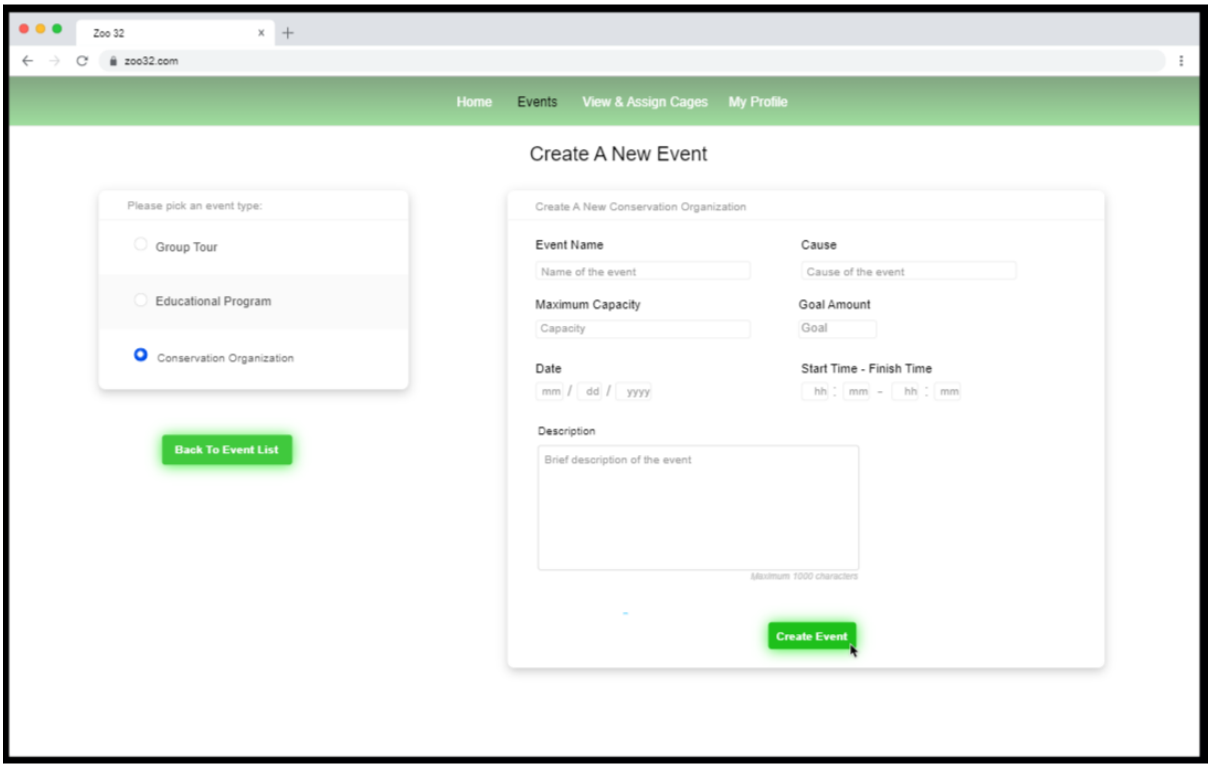
https://framer.com/share/unRheBXJHnYqWDFns7Ia/qEPkgqN53

SQL Statements:

Inputs: @description, @max_capacity, 0, @date, @duration, @coordinator_id, @topic

- After the coordinator creates the event by clicking the button:
  INSERT INTO Event (description, max_capacity, num_of_participants, date, duration)
  VALUES (@description, @max_capacity, 0, @date, @duration);
- Inserting the corresponding educational program to the newly created event:
  INSERT INTO Educational_Program **VALUES** (**LAST_INSERT_ID()**, @topic);
- Inserting the new creation to the Creates_Event relation:
  INSERT INTO Creates_Event **VALUES** (@coordinator_id, **LAST_INSERT_ID()**);

4.14 Create Conservation Organization (Coordinator Account)



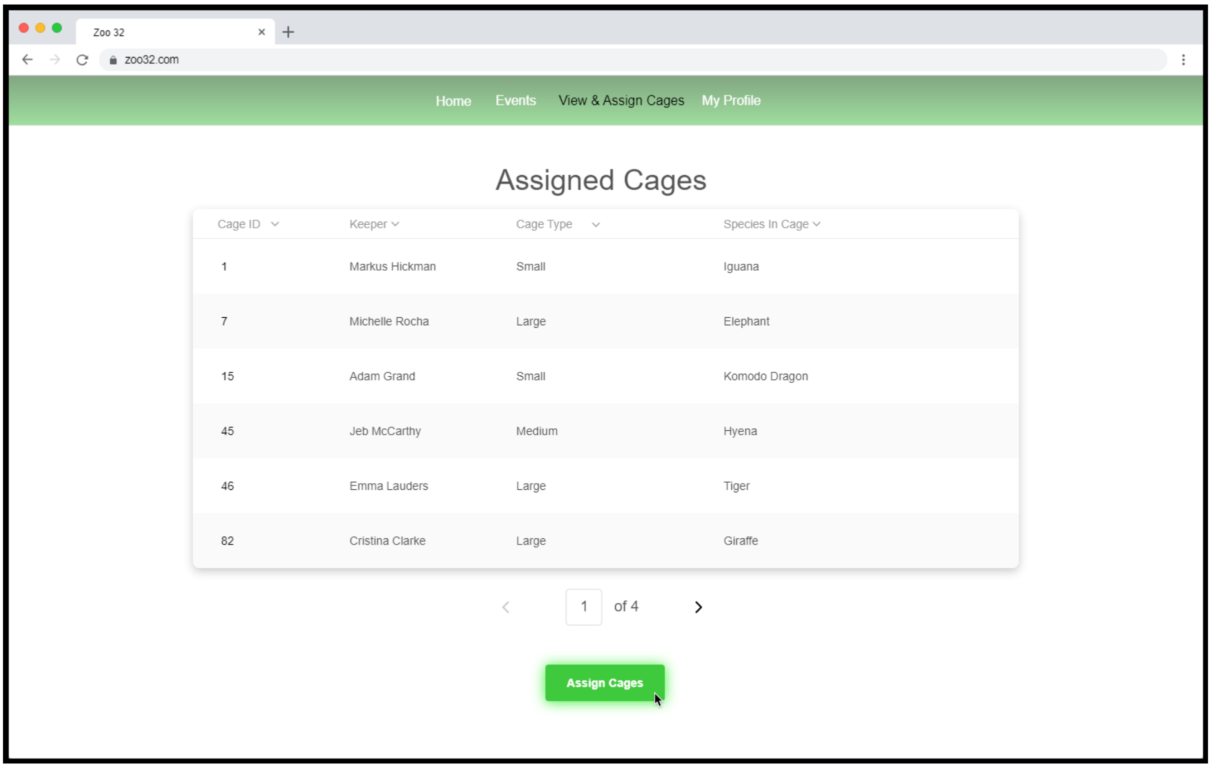
https://framer.com/share/wW2gaJdrrnSNksBcUVAp/qEPkgqN53

SQL Statements:

Inputs: @description, @max_capacity, 0, @date, @duration, @coordinator_id, @topic, @goal_amount, @cause

- After the coordinator creates the event by clicking the button:
  INSERT INTO Event (description, max_capacity, num_of_participants, date, duration)
  VALUES (@description, @max_capacity, 0, @date, @duration);
- Inserting the corresponding conservation organization to the newly created event:
  INSERT INTO Conservation_Organization
  VALUES (LAST_INSERT_ID(), @goal_amount, 0, @cause);
- Inserting the new creation to the Creates_Event relation:
  INSERT INTO Creates_Event **VALUES** (@coordinator_id, **LAST_INSERT_ID()**);

4.15 View Cages Assigned by This Coordinator (Coordinator Account)



Assigned Cages

Cage ID	Keeper	Cage Type	Species In Cage
1	Markus Hickman	Small	Iguana
7	Michelle Rocha	Large	Elephant
15	Adam Grand	Small	Komodo Dragon
45	Jeb McCarthy	Medium	Hyena
46	Emma Lauders	Large	Tiger
82	Cristina Clarke	Large	Giraffe

https://framer.com/share/Untitled-1-3--xceLeqSsASyhl4dLklpS/qEPkgqN53?editor=1

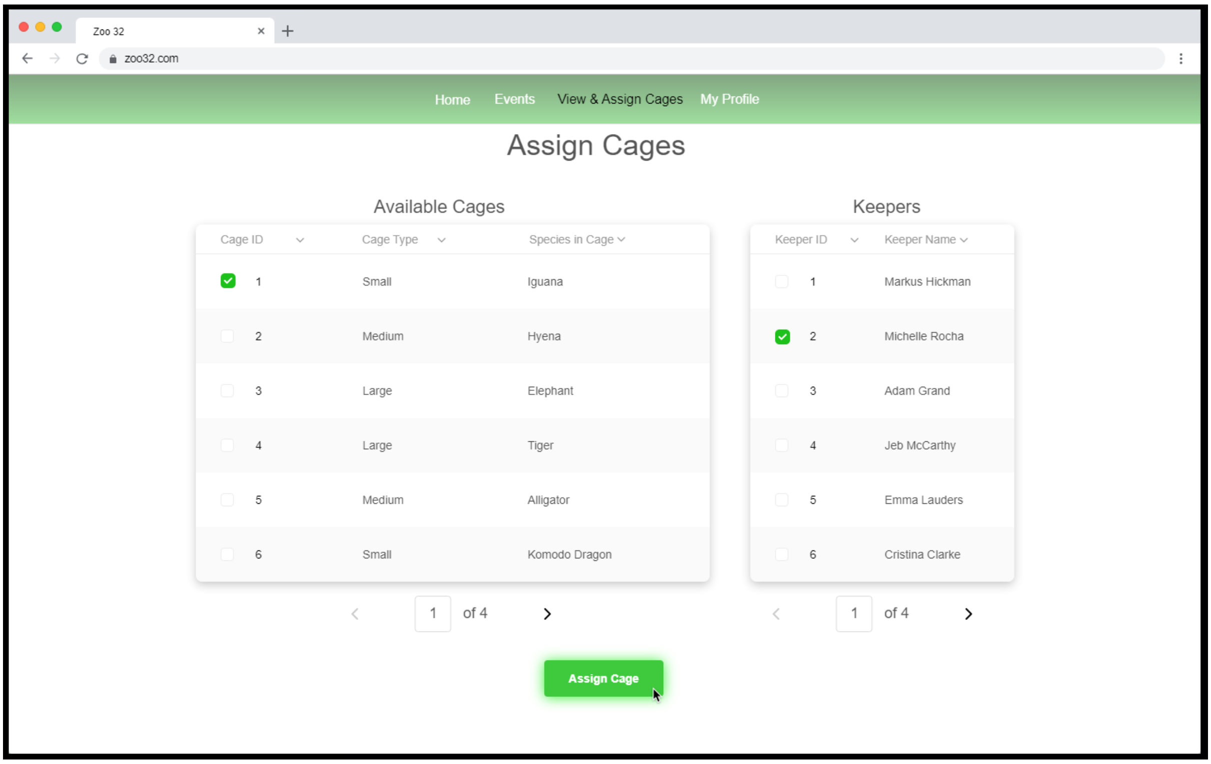
SQL Statement:

When the coordinator first loads the page, to display all cages assigned by them:

Input: @coordinator_id

- **SELECT** cage_id, full_name, cage_type, animal_species
  FROM Cage **INNER JOIN** Assigns **ON** Cage.cage_id=Assigns.cage_id
  INNER JOIN Belongs_to **ON** Belongs_to.cage_id=Cage.cage_id **INNER JOIN** Animal **ON** Animal.animal_id=Belongs_to.animal_id **INNER JOIN** User **ON** Assigns.keeper_id=User.user_id
  WHERE Assigns.coordinator_id=@coordinator_id

4.16 Assign a Cage to a Keeper (Coordinator Account)



Assign Cages

Available Cages

	Cage ID	Cage Type	Species in Cage
☑	1	Small	Iguana
□	2	Medium	Hyena
□	3	Large	Elephant
□	4	Large	Tiger
□	5	Medium	Alligator
□	6	Small	Komodo Dragon

1 of 4

Keepers

	Keeper ID	Keeper Name
□	1	Markus Hickman
☑	2	Michelle Rocha
□	3	Adam Grand
□	4	Jeb McCarthy
□	5	Emma Lauders
□	6	Cristina Clarke

1 of 4

https://framer.com/share/Untitled-1-3--xceLeqSsASyhl4dLklpS/h7tn3WiVK?editor=1

SQL Statements:

To get available cages (cages that do not have a keeper assigned):

- **SELECT** C.cage_id, C.cage_type, An.animal_species
  FROM Cage **AS** C, (**SELECT** A.animal_species, B.cage_id
  FROM Animal **AS** A **INNER JOIN** Belongs_to **AS** B **ON** A.animal_id=B.animal_id) **AS** An,
  WHERE An.cage_id=C.cage_id **AND** C.cage_id **NOT IN** (**SELECT** cage_id,
  FROM Assigns)

To get keepers:

- **SELECT** user_id, full_name
  FROM User
  WHERE User.user_id **IN** (**SELECT** keeper_id
  FROM Keeper)

To assign a cage to a keeper:

Inputs: @keeper_id, @cage_id, @coordinator_id

- **INSERT INTO** Assigns **VALUES** (@keeper_id,@cage_id,@coordinator_id)

5. Advanced Database Components

5.1 Reports

- Coordinators will be able to see the total number of comments made for a group tour in the past month, they may want to check the tours that attracted the most discussion.

  ```
  CREATE VIEW monthly_group_tour_comments
  AS SELECT event_id, event_name, Count(comment_id) AS num_comments
  FROM Group_Tour NATURAL JOIN Comments JOIN Event ON
  Event.event_id=Group_Tour.group_tour_id
  GROUP BY event_id
  WHERE DATEDIFF(CURDATE(), date) <= 30
  ```

- Coordinators will be able to see the number of items sold by shops, which may help in organizing which shop goes where or creating events to incentivize visitors to buy from underperforming shops.

  ```
  CREATE VIEW shop_items_sold_report
  AS SELECT shop_id, shop_name, SUM(amount)
  FROM Shop NATURAL JOIN Sells NATURAL JOIN Item JOIN Buys ON
  Buys.item_id=Item.item_id
  GROUP BY shop_id
  ```

5.2 Views

- View to show visitors the list of group tours.
  CREATE VIEW tour_view (name, description, date, duration, capacity, num_of_participants) **AS**
  SELECT event_name, description, date, duration, max_capacity, num_of_participants
  FROM Group_Tour

- View to show visitors the list of animals.
  CREATE VIEW animal_view (species, name) **AS**
  SELECT animal_species, animal_name
  FROM Animal

- View to show visitors the list of shops.
  CREATE VIEW animal_view (name, description, area_name) **AS**
  SELECT shop_name, shop_description, area_name
  FROM Shop **NATURAL JOIN** Is_in **NATURAL JOIN** Area

5.3 Constraints

- The collected money amount of a Conservation Organization is equal to the total money that people donated to that specific organization.
- A coordinator can not invite more veterinarians than the maximum capacity of an Educational Program.
- A veterinarian can be invited only once to an Educational Program.
- A cage can have a maximum equal to its size in animals that belong to it.
- A visitor can only have one membership active at a time.
- A visitor can only comment on group tours that they have attended.

5.4 Stored Procedures

We will be using stored procedures to:

1. Increment the number of attendees for a given event when a visitor is added to the attends relation.
2. Update the amount raised for a Conservation Organization when a new tuple is added to the donates relation.
3. Automatically change the status of complaint forms when a coordinator responds to it.

5.5 Triggers

- When a membership is cancelled, the corresponding row in “Has” will be removed too.
- When a membership is created, “Has” table will be updated with the required entities too.
- When a membership is expired, it will be removed from “Membership” and “Has” tables automatically.
- When an event is created, “Creates” table will be updated with the required entities too.
- When a complaint form is created, “Creates” table will be updated with the required entities too.
- When a comment is created, “Comments” table will be updated with the required entities too.

6. Implementation

While implementing our database system, we will use Bootstrap, HTML, CSS and JavaScript for the user interface. Also, we will use PHP and MySQL to meet the system functionalities.